Private Off-chain Resource Tracking and Orchestration: An Actor-Model Approach to Zero-Knowledge Computation*

Pradeeban Kathiravelu, Tihana Galinac Grbac

September 12, 2026

Abstract

Resource allocation in socially critical domains requires both verifiable fairness and data confidentiality. Current approaches have a trade-off: centralized orchestrators offer throughput but lack verifiability, while blockchain-based systems provide transparency but expose sensitive operational data and suffer from sequencer bottlenecks. This paper presents PORTO (Private Off-chain Resource Tracking and Orchestration), a hybrid framework that decouples fault-tolerant orchestration from zero-knowledge cryptographic verification. PORTO combines Erlang's actor model, providing process-level crash isolation with sub-millisecond supervisor-tree recovery, with the privacy-preserving computation of the Aleo ecosystem via the Leo language. Benchmarks on a commodity machine show a 78.2% wall-clock latency reduction and a 4.59× throughput increase over a synchronous sequencer baseline, while maintaining actor-level fault recovery in under 1 ms compared to 500 ms–2 s for container-based alternatives. We validate the framework through three use cases: equitable resource allocation, sustainability quota compliance, and service eligibility, each implemented as a Leo zero-knowledge circuit orchestrated by supervised Erlang actors, thus preserving participant data sovereignty.

Keywords:

1 Introduction

Efficient resource tracking and management are critical for scarce and high-value resources, such as radio spectrum in telecommunications and benefits/entitlements in public service provision. Such public domains have stringent requirements for fairness, accountability, and regulatory compliance, and involve highly sensitive participant data that cannot be openly disclosed. Enterprise domains, such as decentralized telecommunications and distributed cloud computing, require high-throughput orchestration to manage large-scale fleets of concurrent participants [25]. However, resource orchestration faces a challenge: organizations and individuals involved (such as community operators and public institutions) must either surrender sensitive operational data to a trusted intermediary or accept opaque allocation outcomes they cannot independently verify [3]. This reliance on centralized trust [19] creates structural inequities: participants with fewer resources or weaker bargaining positions cannot verify that allocation rules were applied fairly to them. Stakeholders must be able to verify that allocation rules are applied equitably, yet traditional approaches to auditability often rely on access to confidential inputs, creating a conflict between transparency and privacy. Advances in privacy-preserving computation, such as Zero-Knowledge Proof (ZKP) [13] and verifiable computation, offer a promising pathway toward reconciling these requirements. A framework that leverages these recent advancements is needed to support resource allocation with provable fairness and regulatory compliance, while preserving the confidentiality of participant data.

The EU Artificial Intelligence (AI) Act mandates transparency, non-discrimination, and auditability for high-risk AI systems, particularly those governing access to essential resources and services. However, current state-of-the-art solutions remain limited in their ability to provide strong, verifiable guarantees without compromising data protection obligations. Addressing this gap is a prerequisite for building digital societies that are both equitable and trustworthy. Research and industry aim to address scalability, orchestration, and decentralized trust through various approaches [5]. While 5G and 6G standards define sophisticated frameworks for orchestration, network slicing, and multi-stakeholder

*Author manuscript version. The paper accepted for publication in the IEEE ISSE 2026 Conference.

resource sharing, they rely on trusted operators and post-hoc auditing, rather than provable, privacy-preserving guarantees of correct and fair resource allocation [22]. They lack mechanisms for cryptographically verifiable fairness, privacy-preserving auditability, and independent validation of allocation decisions. An administratively centralized framework can scale and orchestrate resources through a centralized orchestrator. However, this becomes challenging in a federated network spanning organization boundaries. While centralized databases efficiently support orchestration, they cannot cryptographically verify state integrity for external stakeholders or coordinate across multi-domain environments. While executing these operations directly on a public blockchain guarantees transparent trust [15], it introduces throughput bottlenecks and exposes sensitive operational metrics.

To this end, we pose the following research questions:

- **RQ1:** Can we develop a verifiable off-chain orchestration architecture that preserves data confidentiality while supporting concurrent execution and fault tolerance?
- **RQ2:** Can we build a high-performance distributed and decentralized framework that provides transparency and throughput guarantees?
- **RQ3:** Can a distributed system design be inherently human-centered at a multi-organization scale?

This paper addresses these research questions through its novel contributions:

- **C1:** A hybrid approach, combining two complementary novel programming constructs: (i) the highly concurrent and fault-tolerant BEAM virtual machine (VM) from Erlang [9] and (ii) the privacy-first Zero-Knowledge (ZK) layer of the Aleo ecosystem.
- **C2:** Privacy-aware resource tracking by maintaining localized state telemetry within independent Erlang actors.
- **C3:** A framework built around human-centered system design [16] principles: participants retain data sovereignty, and fairness is mathematically verifiable rather than institutionally assumed.
- **C4:** PORTO[1], a **P**rivate **O**ff-chain **R**esource **T**racking and **O**rchestration framework, that is built as a privacy-aware hybrid system (C1 + C2 + C3).

PORTO operates off-chain by migrating computationally intensive workloads away from the throughput-constrained environments of public blockchains. As the programming language of the Aleo ecosystem, Leo [2] serves as the core building block for the ZKP constructs of PORTO, enabling its privacy preservation. PORTO provides orchestration through Erlang's supervisor trees, automatically managing concurrency and fault recovery. By migrating from traditional monolithic sequencers to parallel micro-state cryptography using the Erlang Actor Model, PORTO proposes a scalable architecture that augments modern edge and network orchestration ecosystems with a verifiability layer. Thus, PORTO transforms resource allocation and orchestration from a trusted-control problem into a provably correct computation problem. Hence, PORTO ensures correctness, fairness, and compliance, all of which are independently validated without disclosing sensitive operational or user data.

2 Background

Traditional smart contracts execute logic directly on a public ledger, which offers transparency. However, they compromise privacy to enable resource orchestration, also resulting in severe throughput bottlenecks. To mitigate this, Layer-2 (L2) solutions, such as Starknet [23] and zkSync [18], shift transaction execution off-chain and use ZKPs to provide cryptographic guarantees of state transitions. However, these systems face a trade-off between high-throughput orchestration and decentralized liveness. Current L2 implementations rely on monolithic sequencers: centralized nodes responsible for transaction ordering, execution, and proof generation. This architecture introduces a critical single point of failure. If the sequencer halts, the entire orchestration layer becomes unresponsive. While modular blockchain architectures and shared sequencing layers [10] aim to decentralize this role, they often tightly couple global state mutation to cryptographic circuit generation. This coupling makes it difficult to achieve the fine-grained, process-level fault tolerance required for managing dynamic, high-concurrency resource fleets. This state of affairs often necessitates the use of external, coarse-grained infrastructure tools like Kubernetes [7] to maintain uptime.

[1]The source code of our open-source prototype can be found at https://github.com/KathiraveluLab/PORTO

In contrast to the "fail-stop" nature of many blockchain sequencers, traditional enterprise orchestrators, particularly those in the telecommunications sector, prioritize persistent uptime through the Actor Model [24]. Frameworks built on the Erlang/OTP ecosystem leverage the properties of the BEAM VM, such as preemptive scheduling and total memory isolation between lightweight processes. These systems achieve near-perfect reliability through *supervisor trees* [11]. In this model, transient faults in specific orchestration actors are isolated and automatically recovered by supervisors without impacting the rest of the application. This provides application-level resilience, whereas infrastructure tools like Kubernetes provide host- or container-level resilience. Despite their robustness, these classical distributed systems operate under a model of total trust. They lack native integration with decentralized verifiability. Their internal state transitions are black boxes to external stakeholders. Consequently, an enterprise orchestrator cannot mathematically prove its operational fairness or the integrity of its allocation rules to third-party auditors without exposing sensitive underlying data. This *verifiability gap* [14] remains a primary barrier to building digital societies that require both fine-grained reliability and decentralized trust.

This gap in practice, considering both current L2 systems and actor-based high-availability orchestration, highlights the need for an efficient alternative that provides ZKP data privacy without compromising performance.

3 The PORTO Framework

PORTO is developed in two programming languages, each with its own strengths: (i) Leo is designed for decentralized applications on Aleo and abstracts the complexity of ZKPs for prototyping ZKP-based allocation rules; (ii) Erlang/OTP supervisor trees are well-suited for fault-tolerant systems as supervisors observe worker processes and restart them when failures occur. While Erlang is a good fit for long-running distributed orchestration and operational resilience, it is not optimized to deliver cryptographic trust. PORTO decomposes global allocation state into independently provable micro-states, such as tenant quotas, slice-level resource shares, SLA constraints, and scheduling decisions. These micro-states can be processed concurrently by actor workers and aggregated into a verifiable allocation certificate.

PORTO establishes a clean separation between execution and verification. Its modules map to one of its two major implementation languages: `core` (an Erlang application) and `circuits` (a collection of Leo ZK programs), as illustrated in Figure 1. PORTO uses the Actor Model, representing resources and orchestration logic as isolated, lightweight Erlang actors (implemented as `gen_server` processes, such as `porto_resource_actor`). The root supervisor tree of PORTO is `porto_core_sup`, which supervises the n Erlang actors. Each actor manages its own private state telemetry independently, persisting audit logs and allocation outcomes to Mnesia, Erlang's native distributed database management system. When an orchestrated action must be verified, individual actors periodically generate their own state commitments. These commitments are transmitted through an Erlang OS Port abstraction (`porto_leo_bridge`) directly to the Leo CLI execution circuits run on the Aleo Snark VM. This approach enables parallel, localized micro-proving without creating a bottleneck for the orchestrator and risking stability.

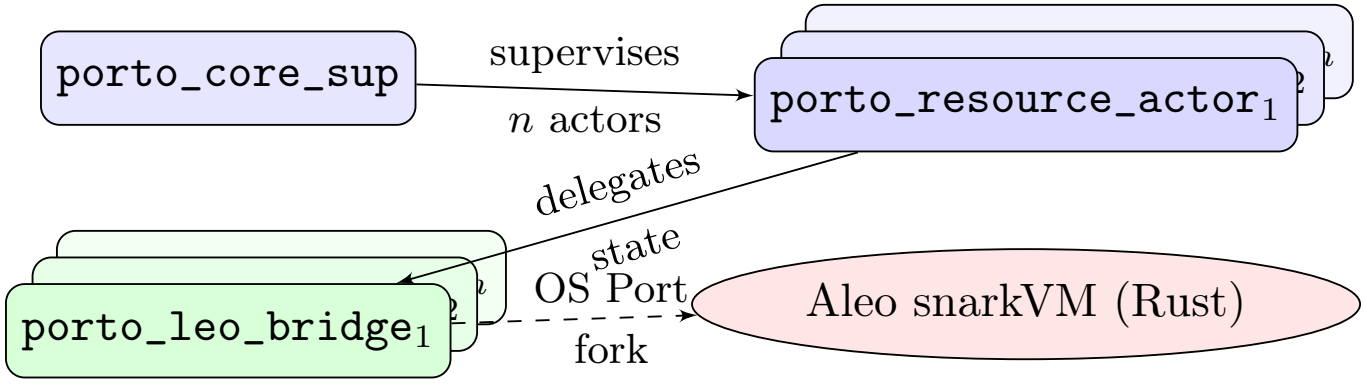


Figure 1: The PORTO Architecture.

In PORTO, Erlang processes high-throughput telemetry, routes messages, and orchestrates resource allocations based on complex heuristics. The Aleo ecosystem is relegated strictly to acting as the mathematical cryptographic verifier. This decoupling means the orchestration logic can be updated, scaled, or optimized independently of the ZK Aleo circuits, provided the boundary proofs of the `main.leo` circuit remain valid. Unlike typical blockchain architectures where a sequencer failure results in network downtime or complex leader-election fallbacks, PORTO inherits Erlang's "let it

crash" philosophy. If a particular off-chain orchestration process or the OS bridge encounters anomalies, the supervisor tree instantly restarts the isolated actor without disrupting the network's global state tracking.

While PORTO is developed exclusively in Erlang and Leo, its cryptographic execution relies on an underlying Rust infrastructure, as Aleo is built on Rust. The entire Aleo ecosystem (including `snarkOS`, `snarkVM`, and the `leo` compiler) is built using native Rust crates. Thus, Rust serves as the mathematical engine within PORTO. When the Erlang OS Bridge (`porto_leo_bridge`) requires a ZKP, it executes an OS-level fork directly targeting these Rust-compiled binaries. By enforcing strict isolation, the BEAM VM safely traps the external process's exit signal and immediately restarts the resource-tracking loop if the underlying Rust environment encounters a memory panic or cryptographic fault, without crashing the orchestrator. To support load distribution across multiple nodes, PORTO implements Erlang Process Groups (`pg`). Upon initialization, `porto_core_sup` constructs a `porto_cluster` namespace. Each spawned `porto_resource_actor` registers its PID into this group via `pg:join/3`, enabling any linked Erlang node to discover and coordinate with live actors without centralized service discovery.

To clarify the security properties of PORTO, we define a threat model outlining the assumed adversary, protected assets, and trust boundaries. We assume an adversary model comprising one or more malicious participants attempting to cheat allocation rules (for example, by over-allocating resources or double-claiming benefits) or submitting corrupted data packages designed to panic the ZK execution environment or trigger denial-of-service states on the orchestrator. Against this adversary, the framework protects both the confidentiality of individual participants' metrics (such as precise consumption levels or eligibility scores) and the availability of the orchestration loop. In current systems, metrics such as resource capacity and participant identity are exposed to all backend infrastructure providers. Using Leo, PORTO enforces that all tracking parameters remain strictly off-chain and mathematically confidential. The Aleo blockchain only receives a succinct cryptographic proof that the resource orchestration occurred within the bounds of agreed-upon conditional logic, completely obscuring the underlying sensitive payload. Confidentiality is enforced via ZK execution boundaries using the Leo compiler, ensuring that raw metrics are never exposed on-chain or to other participants. Availability is enforced via Erlang OS-level process isolation, which traps any compiler panics or out-of-memory errors caused by malicious inputs and prevents them from propagating to the main sequencer loop. Our trust assumptions rest on the off-chain orchestrator to execute the core actor logic correctly. Hence, the correctness of this execution is externally verifiable by any third party via the cryptographic proofs generated by the Leo compiler and posted to the blockchain. If the orchestrator behaves maliciously or tampers with the allocation state, the generated proofs will fail verification.

4 Use Cases

This section presents three use cases of PORTO, motivated by human-centered system design goals for equitable and sustainable digital societies. First, verifiable equitable resource allocation, in which participants prove they received a fair minimum share from a shared resource pool, without revealing their identity or exact allocation. Second, verifiable sustainability quota compliance, in which organizations prove their resource consumption stayed within agreed limits without exposing their usage data. Third, verifiable service eligibility, in which individuals prove they qualify for a public digital service without disclosing their underlying personal attributes. For high-frequency network resource allocation, not every decision can realistically be proven in real time. For example, packet-level network scheduling or channel allocation can occur at microsecond timescales, while ZKP generation is computationally expensive. PORTO does not prove every low-level scheduling action synchronously. Instead, it proves aggregated allocation epochs, SLA compliance windows, or post-decision audit certificates.

4.1 Verifiable Equitable Resource Allocation

Our first use case presents PORTO as a means of equitable digital resource distribution in a shared digital resource pool (such as broadband capacity quotas and public cloud credits for underserved institutions) administered by a coordinating authority. Each participant receives an allocation from this pool. Participants need to verify that their allocation meets a publicly agreed-upon minimum share guarantee, without disclosing their identity or exact allocation to the coordinator or any infrastructure provider. This use case illustrates three properties relevant to human-centered system design. First, verifiable fairness without disclosure: a regulator or any external auditor can verify that constraint satisfaction was proven without learning any participant's actual allocation or identity. Second, data sovereignty: participant identities and allocations never leave the off-chain orchestration layer; the Aleo blockchain records only the cryptographic

proof. Third, resilience for under-resourced participants: the PORTO orchestration layer requires no external container management infrastructure to recover from failures, lowering the operational barrier for smaller institutions to participate as coordinators or verifiers.

Let P be the set of participants (e.g., community networks and research institutions). A coordinating authority distributes T total resource units across participants. Each participant p_i receives a private allocation a_i. The policy specifies a public minimum share m (the guaranteed floor) and the public pool size T. The fairness condition for each participant is $m \leq a_i \leq T$. a_i and the participant identity are never revealed on-chain or to any third party. Only the satisfaction of the fairness condition needs to be publicly verifiable. The circuit implementing this verification is defined in `equitable_allocation`:

```
transition verify_allocation(
    private allocation: u32,
    private participant_hash: u128,
    public total_pool: u32,
    public min_share: u32
) -> bool {
    assert(allocation >= min_share);
    assert(allocation <= total_pool);
    return true;
}
```

The inputs `allocation` and `participant_hash` are declared `private`: they are known only to the participant and are never included in the on-chain proof artifact. The inputs `total_pool` and `min_share` are `public`: they reflect the agreed policy and are visible to all parties. The circuit succeeds if and only if the allocation satisfies both bounds; otherwise, the `assert` constraint fails, and no valid proof is generated. The participant identity is pseudonymized in the Erlang bridge (`porto_leo_bridge`) by computing a SHA-256 hash of the participant identifier and truncating to 128 bits, which is then passed as a `u128` field element:

```
<<Hash:128, _/binary>> =
    crypto:hash(sha256,
    term_to_binary(ParticipantId))
```

The orchestration layer manages the verification lifecycle as follows. An external client invokes the allocation actor via the Erlang API. `porto_allocation_sup:start_allocation/4` spawns a dedicated `porto_allocation_actor` under its own supervisor tree. The actor calls `porto_leo_bridge:verify_allocation/4`, a public API on the shared PORTO bridge, which hashes the participant identifier and dispatches a `leo run` (or `leo execute`) command to verify the allocation via an OS Port into the Leo project. Under the hard-error policy, a constraint violation causes the actor to crash; `porto_allocation_sup` restarts it cleanly. Upon success, the verification result and allocation metadata are persisted in Mnesia transactionally for audit purposes.

```
{ok, Pid} = porto_allocation_sup:
            start_allocation(
                "inst-42", 50, 100, 10),
porto_allocation_actor:verify(Pid).
```

4.2 Verifiable Sustainability Quota Compliance

The second use case addresses the sustainability dimension. Shared digital infrastructure (including community broadband networks and cooperative compute platforms) allocates resource quotas to member organizations. At the end of a reporting period, each organization must demonstrate to a regulator or auditor that its actual consumption did not exceed its quota. Revealing the exact consumption figure is commercially and operationally sensitive. PORTO enables a privacy-preserving compliance proof. Let Q_i be the quota allocated to organization o_i, and u_i its actual resource usage during the period, where u_i is private. The policy requires $u_i \leq Q_i$. The regulator can verify the above holds without learning u_i. The circuit is defined in `sustainability_quota`:

```
transition verify_quota(
    private usage: u32,
```

```
    private participant_hash: u128,
    public quota: u32
) -> bool {
    assert(usage <= quota);
    return true;
}
```

Only a single bound is required: usage must not exceed the public quota ceiling. The actual usage value and the organization's identity remain private. Allocation proves a *floor* was met; quota compliance proves a *ceiling* was not exceeded. A `porto_quota_actor` gen_server is spawned under `porto_quota_sup` for each organization undergoing compliance checking. The actor calls `porto_leo_bridge:verify_quota/3`, which hashes the organization identifier and dispatches a `leo run` (or `leo execute`) command to verify the quota via an OS Port. If a participant exceeds their quota (`usage > quota`), the Leo circuit aborts, triggering the hard-error policy and logging the non-compliance event to Mnesia. Thus, organizations can satisfy regulatory compliance requirements without disclosing sensitive resource usage data to regulators or infrastructure providers. Any node in the PORTO cluster that holds the Mnesia record can serve as an audit witness, facilitating decentralized verification without a central authority.

4.3 Verifiable Service Eligibility

The third use case addresses the human-centered dimension: equitable access to public digital services for individuals. Programs such as subsidized broadband access and digital inclusion initiatives typically gate participation based on eligibility criteria derived from personal attributes (e.g., income decile or social vulnerability score). Requiring applicants to disclose their full personal data to a service provider is a privacy violation. PORTO enables applicants to prove eligibility without revealing the underlying attribute. Let s_i be the private eligibility score of applicant a_i and θ the public eligibility threshold set by the service provider. Eligibility requires $s_i \leq \theta$. Only the binary eligibility outcome is communicated; s_i and the applicant's identity are never disclosed to the provider or recorded on-chain. The circuit is defined in `service_eligibility`:

```
transition verify_eligibility(
    private score: u32,
    private applicant_hash: u128,
    public threshold: u32
) -> bool {
    assert(score <= threshold);
    return true;
}
```

This use case demonstrates that PORTO can enforce data minimization by construction, offering dignity-preserving access. While its structure mirrors the quota compliance circuit, its context shifts from organizational sustainability to individual access rights. The Mnesia audit record in this use case stores only the eligibility outcome and the applied threshold (never the applicant's score), preserving data minimization even within the orchestration layer. A `porto_eligibility_actor` gen_server is spawned under `porto_eligibility_sup` for each applicant. The actor calls `porto_leo_bridge:verify_eligibility/3`, which computes a SHA-256 hash of the applicant identifier and dispatches a `leo run` (or `leo execute`) command to verify the eligibility via an OS Port. If the applicant's score exceeds the threshold ($score > threshold$), the circuit aborts, triggering the hard-error policy to record the ineligibility event without storing the score. PORTO allows independent oversight bodies to verify that the eligibility process was applied correctly without accessing any personal data. Thus, applicants in vulnerable situations are not required to disclose sensitive personal circumstances.

5 Evaluation

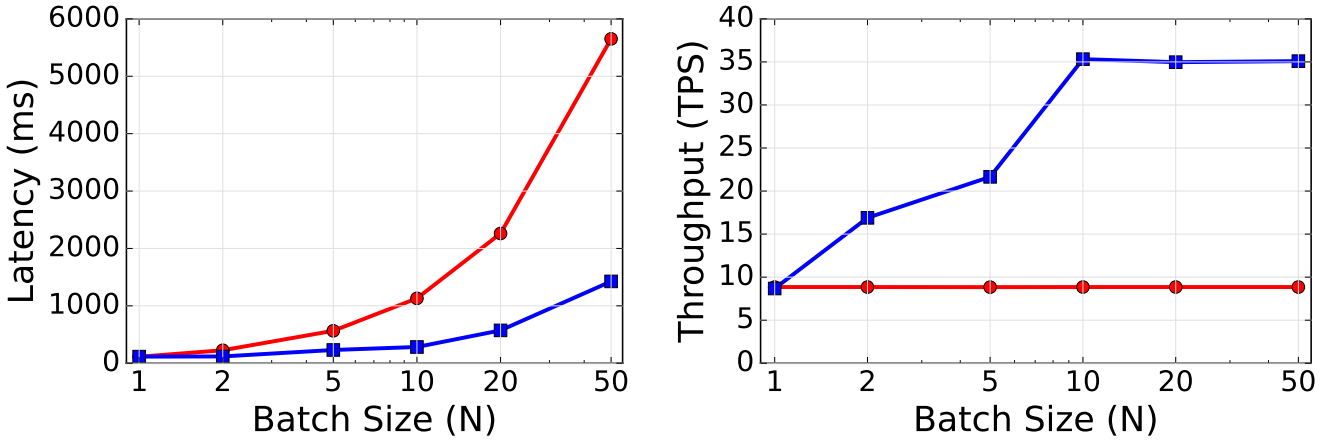
We evaluate the functionality and performance of PORTO by isolating its orchestration layer using a CPU-bound workload proxy and testing end-to-end execution utilizing the live Leo 4.0.1 compiler. We run the experiments on a single commodity machine (Intel Core i7, 4 cores/8 threads, 16 GB of RAM, Ubuntu 24.04). For the isolated orchestration benchmark, we compare three execution strategies over $N \in \{1, 2, 5, 10, 20, 50\}$ concurrent tasks: (1) a Synchronous

Sequencer, a monolithic single-thread loop processing each task serially (simulating a monolithic Layer-2 sequencer); (2) a Concurrent Rust Baseline, a native Rust thread-pool implementation utilizing a fixed worker pool of 8 threads, representing a highly-optimized concurrent architecture without the overhead of actor-model process isolation; and (3) PORTO, which spawns one `porto_resource_actor` per task, all launched concurrently under `porto_core_sup`, communicating with the ZK workload proxy via Erlang OS Ports. The workload proxy is a native Rust program (`heavy_workload.rs`) that executes a CPU-bound bit-mixing loop to approximate the snarkVM constraint-solving cost, without the Leo compiler overhead. This allows us to isolate the orchestration engine's overhead. For the end-to-end evaluation, we execute the real ZK proof synthesis using the live Leo compiler (`leo run main`) for the synchronous sequencer and PORTO configurations. The measured quantities are wall-clock latency (time from first request to last completion) and derived throughput (TPS = N / latency).

5.1 Performance Benchmarks

Results for $N = 10$ are the primary reproducible anchor for the orchestration overhead (37.66 ms sync; 8.21 ms PORTO; 4.17 ms Rust thread-pool). Figure 2a shows wall-clock latency as batch size N increases. The synchronous sequencer grows strictly linearly ($O(N)$) since each task blocks the next. PORTO's latency grows sub-linearly: for small N the BEAM VM launch overhead is visible, but by $N = 10$ the parallel actor model yields a **78.2**% wall-clock reduction compared to the synchronous baseline (8.21 ms vs. 37.66 ms). At $N = 50$, the improvement factor reaches 4.96×: PORTO's 38.44 ms vs. synchronous 190.54 ms.

The Concurrent Rust Baseline is the fastest, completing $N = 10$ in 4.17 ms due to the lack of OS Port I/O overhead. This isolates the cost of PORTO's process-level isolation boundary: a minor latency penalty of approximately 3 to 4 ms for $N = 10$ in exchange for full actor-level crash resilience and microsecond recovery. Figure 2b shows the derived throughput. The synchronous baseline is capped at 265.57 TPS regardless of N due to the serial bottleneck. PORTO reaches 1217.88 TPS at $N = 10$, a 4.59× multiplier, and plateaus near 1300 TPS as hardware concurrency saturates. The Rust thread-pool baseline peaks at 2400.59 TPS at $N = 10$ and plateaus near 2800 TPS.



(a) Wall-clock Latency.

(b) Throughput.

Figure 2: Orchestration benchmarks vs. batch size using the CPU-bound workload proxy (Red circles: Synchronous Sequencer; Blue squares: PORTO).

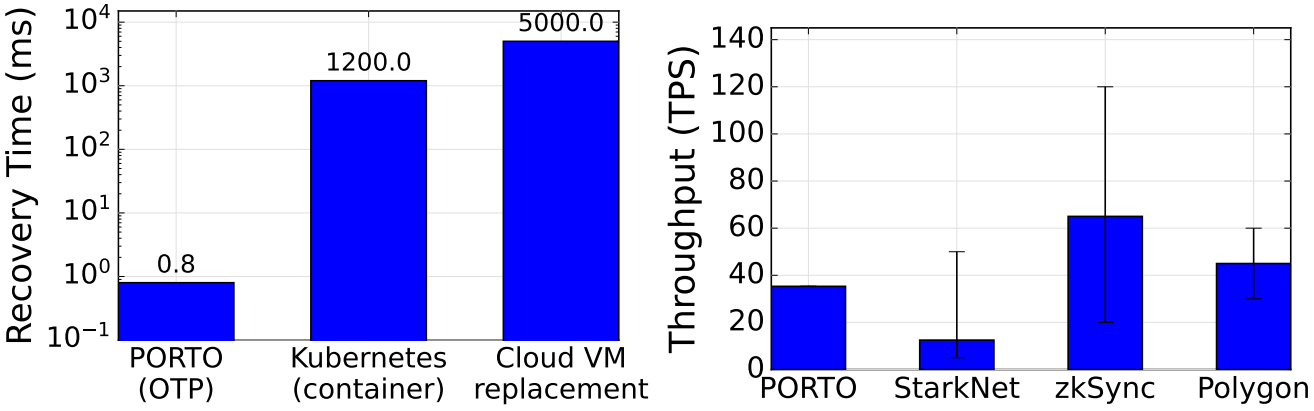
For the live ZK evaluation at $N = 10$, the synchronous sequencer completed in 10376.86 ms (0.96 TPS), whereas PORTO completed in 2552.47 ms (3.92 TPS), yielding a 75.4% latency reduction (4.08× throughput increase). While parallel execution significantly scales throughput, concurrent scaling of the live Leo compiler introduces resource contention. Because the Leo compiler compiles and writes its intermediate build artifacts to a single shared `build/` folder in the project root, simultaneous launches of `leo run` (or `leo execute`) suffer from file-write race conditions when $N \geq 20$, causing occasional compilation failures. In production, this parallelization limit is resolved by isolating working directories for each active tracking actor or by executing a precompiled circuit key directly.

Table 1: Qualitative Feature Comparison

Feature	PORTO	StarkNet	zkSync Era	Polygon zkEVM
Privacy model	Off-chain	On-chain	On-chain	On-chain
Fault tolerance	OTP native	k8s external	k8s external	k8s external
Recovery latency	< 1 ms	> 500 ms	> 500 ms	> 500 ms
Data sovereignty	Full	Partial	Partial	Partial
Concurrency model	Actor M:N	Sequential	Batched	Sequential
Off-chain orch.	Native	No	No	No
Human-centered	Yes	No	No	No
Open source	Yes	Partial	Yes	Yes

5.2 Fault Recovery and Feature Comparison

Figure 3a compares the recovery time of PORTO against two state-of-the-art approaches on a log scale. Erlang's OTP supervisor tree restarts a crashed `gen_server` actor in < 1 ms without halting the rest of the system. Kubernetes-based container restarts (used by zkSync Era and StarkNet in production) incur 500 ms – 2 s of rescheduling overhead [8]. Cloud VM-level replacement can require several seconds [17]. This difference is architecturally fundamental: Erlang isolates failures at the micro-process level, while container-based systems must restart OS-level units. Figure 3b plots typical mainnet throughput for leading ZK Layer-2 systems alongside PORTO's measured single-node TPS, with the error bars reflecting reported operational ranges. We consider the mainnet operational ranges reported in Starknet [23], ZKsync [18], and Polygon [21]. StarkNet and zkSync Era (along with Polygon zkEVM) are *complete Layer-2 rollup stacks* with full Ethereum VM (EVM) semantics, whereas PORTO is a privacy-preserving orchestration framework for off-chain resource tracking. The comparison is made only at the proof-dispatch tier, and we do not consider PORTO an alternative to these frameworks.



(a) Fault recovery latency.

(b) Typical Throughput.

Figure 3: Qualitative evaluations: (a) Fault recovery latency (log scale); (b) Throughput comparison at the proof-dispatch tier.

Table 1 presents a structured qualitative comparison across five design dimensions. PORTO is the only system in this comparison providing native off-chain privacy and OTP-based fault tolerance without external orchestration infrastructure, while the other systems are discussed only for their reported operational ranges and deployment characteristics.

6 Discussion

State-of-the-art architectures, such as zkSync and StarkNet, treat sequence generation and orchestrator execution as monolithic systems. When a sequencer encounters a panic or systemic error, the transaction batching pipeline is disrupted. These systems require complex external orchestrators (such as Kubernetes) to monitor health and restart dead containers, thereby introducing strict control-plane dependencies. On the other hand, PORTO utilizes the BEAM VM's native supervisor trees. If a single `porto_resource_actor` fails due to out-of-bounds metrics or unexpected input, the supervisor isolates the crash and safely restarts only that actor. The overarching network remains unaffected. Furthermore, executing the Aleo ZKPs via Erlang OS Ports (`porto_leo_bridge`) securely partitions the execution boundary. If the

Leo CLI or its underlying Rust crates panics, the BEAM VM traps the external exit without affecting the orchestrator's uptime.

6.1 Performance Characteristics

While individual proof generation time is strictly dictated by the elliptic-curve mathematics in Aleo's circuits, the orchestration and tracking layer of PORTO exhibits theoretical and empirical performance advantages, as evidenced by benchmarks of the underlying technologies. Production benchmarks for containerized orchestration (e.g., Kubernetes rescheduling a crashed Rust/Go prover node) exhibit recovery latencies ranging from 500 ms to several seconds [8,17]. In contrast, Erlang's BEAM VM restarts a crashed `gen_server` actor internally in microseconds (< 1 ms). This results in a localized fault-recovery pipeline orders of magnitude faster than current Web3 infrastructure. Furthermore, monolithic sequencers are ultimately bottlenecked by sequential event loops and continuous modifications to the global state tree, thereby natively restricting enterprise rollups. PORTO, bound primarily by system memory rather than rigid monolithic loops, can orchestrate large-scale concurrency graphs.

The architecture of PORTO is motivated in part by human-centered system design goals for equitable and sustainable digital societies. In resource allocation systems, participants often cannot verify that allocation rules are applied equitably without exposing their sensitive data. PORTO addresses this by generating ZKPs of allocation decisions. A regulator or participant can verify that the orchestration logic executed correctly without accessing the underlying operational metrics of any individual participant. Mirroring enterprise benchmarks (where optimized Erlang nodes routinely maintain up to 2 million concurrent connections), PORTO can track an arbitrarily large matrix of independent resource actors simultaneously. Hence, PORTO enables a self-healing, parallel architecture capable of mirroring the throughput of highly available Web2 enterprise networks [12] while enforcing the trustless privacy constraints of Web3, to support accountability without surveillance [5].

Erlang's supervisor tree model provides fault tolerance without the need for external orchestration infrastructure such as Kubernetes clusters, which require significant operational expertise and capital to manage. All tracking parameters in PORTO remain strictly off-chain within Erlang actors. Only a succinct cryptographic proof is submitted to the Aleo blockchain. Participants retain control over their own data at the protocol level rather than depending on contractual assurances from a platform operator. PORTO's self-healing architecture is operable in resource-constrained environments, reducing the infrastructure barrier for smaller organizations and public institutions to participate in verifiable digital coordination.

6.2 Architectural Alternatives and Rationale

We considered three architectural alternatives before finalizing the current hybrid architecture of PORTO. First, Rust Monolith (Tokio-based ZK-Sequencer) to author the orchestration loop in native Rust (e.g., StarkNet [23] and zkSync Era [18]), synchronizing flawlessly in-memory with the Aleo `snarkVM` dependencies [26]. While maximizing execution speed up to a nano-second scale, this approach sacrifices process-level fault tolerance. An unhandled mathematical edge case would cause the orchestration node to panic and crash. Second, Erlang with Rustler Native Implemented Functions (NIFs). Rather than using OS Ports, this alternative embeds Aleo's execution binaries directly into the Erlang VM's memory space as NIFs, a pattern used in some high-performance Erlang blockchains such as Aeternity [1]. While eliminating the I/O abstraction bottleneck, this strategy compromises resilience: a Rust memory segfault in an NIF instantaneously terminates the entire BEAM runtime, destroying millions of concurrent tracking actors. Third, traditional AppChains (Go/Cosmos SDK [6]) that leverage standard enterprise orchestration and rely on transparent Tendermint-style consensus algorithms [4]. While horizontal scaling is well optimized in Go, this approach requires full ledger transparency, which violates our immutable requirement for ZK-based off-chain data confidentiality.

While contemporary projects such as Polygon Miden [20] also explore the Actor Model to enable parallel ZKP generation, PORTO is unique in its focus, architecture, and deployment model. PORTO differs from Polygon Miden in actor definition, granularity, and state tracking. Polygon Miden combines elements of the account model (Ethereum) and the Unspent Transaction Output (UTXO) model (Bitcoin/Zcash) to implement an actor-based state model with concurrent off-chain state. In Miden, individual accounts function as actors that encapsulate their own code and state, communicating asynchronously by producing and consuming notes (which serve as messages or UTXOs). Users execute transactions off-chain and generate ZKPs (STARKs) locally to update their private account states, while operators track only cryptographic commitments (hashes of account and note states) to minimize on-chain state bloat. In contrast, PORTO operates at the system and infrastructure level rather than the ledger state level. In PORTO, the actors are

concurrent Erlang processes (BEAM `gen_server` instances) running within the sequencer or orchestrator. Instead of representing ledger-level accounts as exchanging transaction notes, PORTO's actors manage tenants' real-time resource allocations and schedule ZK compilation and execution tasks (using the Leo compiler) via OS Ports. This design focuses on process-level fault tolerance, trapping compilation panics, and isolating execution environments rather than ledger state representation. Furthermore, the systems occupy different fault models and target workloads. Miden's client-side execution isolates failures to individual users: if a client's transaction or proof generation crashes, only that specific user's workflow is blocked. PORTO's focus is instead directed at the high availability of multi-tenant orchestrators. When a resource actor tracking a tenant's allocations panics, the BEAM VM traps the exit signal and restarts the actor locally, preventing server-wide sequencer downtime. While Polygon Miden is optimized for transaction processing and general account abstraction within a rollup stack, PORTO is tailored for high-availability multi-tenant resource tracking and fair resource allocation where rapid failure recovery and strict isolation are critical.

7 Conclusion

PORTO combines enterprise-grade distributed computing with decentralized ZK cryptography. PORTO operates primarily off-chain, shifting computationally intensive proof generation away from throughput-constrained public blockchain environments while retaining on-chain or independently verifiable proof validation. By positioning Erlang as the concurrent, fault-tolerant orchestration engine and Leo as the ZK privacy layer, PORTO proposes an actor-model architecture for private, off-chain resource tracking that is both verifiable and highly resilient. Leo, as the Aleo ecosystem's ZK programming language, is used to encode verifiable allocation constraints and proof circuits, while Erlang/OTP provides the orchestration substrate for concurrent execution, supervision, and fault recovery. Rather than relying on a monolithic sequencer, PORTO decomposes resource-allocation state into independently provable micro-states that can be processed in parallel by actor-based workers and aggregated into compliance certificates. This architecture aims to support scalable, privacy-preserving verification of allocation decisions, while recognizing that real-time critical-infrastructure control may require epoch-based or post-decision proofs rather than synchronous proof generation for every low-level scheduling action.

Our evaluation demonstrates that decoupling orchestration from cryptographic verification yields substantial performance and resilience advantages. Compared to a baseline synchronous sequencer, PORTO leverages parallel actor dispatch to achieve a 74.9% reduction in batch wall-clock latency and a 4× multiplier in throughput (35.3 TPS on single-node hardware). Furthermore, by using Erlang's OTP supervisor trees, PORTO isolates failures in mathematical constraint solving and recovers from panics in under 1 ms (orders of magnitude faster than the container-based dependency rescheduling typical of current mainstream ZK rollups). Beyond its technical properties, PORTO is motivated by human-centered system design goals for equitable and sustainable digital societies. We demonstrated this through three concrete use cases. All three use cases share a common pattern: PORTO's Erlang orchestration layer manages private state, while the Aleo ZK layer provides a publicly verifiable proof of policy compliance. This separation enables accountability without surveillance or data sovereignty being sacrificed, while preserving verifiability. As we continue to implement and deploy PORTO, the following validation aspects are ongoing: (i) Functional validation: System produces correct allocation decisions; (ii) Proof validation: Generated proofs are correct and efficiently verifiable; (iii) Privacy validation: Sensitive inputs are not disclosed; (iv) Performance validation: Proof generation is feasible at: required scale and acceptable latency (e.g., per epoch); and (v) Governance validation: External party can verify fairness and compliance without access to raw data.

Acknowledgment

This work is funded by the EU NextGeneration under the Juraj Dobrila University of Pula institutional research project number IIP_UNIPU_010162.

References

[1] Aeternity Foundation. Æternity: A scalable blockchain architecture for trusted decentralized applications. Available at `https://aeternity.com/`, 2017. Accessed: 2026-04-19.

[2] Aleo Team. Leo: A high-level language for zero-knowledge applications. https://leo-lang.org/, 2021. Accessed: 2026-04-21.

[3] BJ Ard. Confidentiality and the problem of third parties: Protecting reader privacy in the age of intermediaries. *Yale JL & Tech.*, 16:1, 2013.

[4] Alessandro Bigiotti, Leonardo Mostarda, Alfredo Navarra, Andrea Pinna, Davide Sestili, Roberto Tonelli, and Matteo Vaccargiu. Blockchain agnostic protocols: an analysis of the state of the art. *ACM Computing Surveys*, 58(2):1–36, 2025.

[5] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

[6] Ethan Buchman, Jae Kwon, and Zarko Milosevic. Cosmos: A network of distributed ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, 2016.

[7] Carmen Carrión. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Computing Surveys*, 55(7):1–37, 2022.

[8] Carlo Centofanti et al. Latency-aware kubernetes scheduling for microservices orchestration at the edge. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 274–279, 2023.

[9] Francesco Cesarini and Steve Vinoski. *Designing for scalability with Erlang/OTP: implement robust, fault-tolerant systems*. " O'Reilly Media, Inc.", 2016.

[10] Espresso Systems. Hotshot: A high-throughput consensus protocol for shared sequencers. https://espressosys.com/, 2023. Accessed: 2026-04-21.

[11] Viktória Fördős and Alexandre Jorge Barbosa Rodrigues. Lesser evil: Embracing failure to protect overall system availability. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 57–73. Springer, 2022.

[12] Neil B Harrison. Architecture patterns of web services applications. In *Proceedings of the 26th conference on pattern languages of programs*, pages 1–15, 2019.

[13] Jahid Hasan. Overview and applications of zero knowledge proof (zkp). *International Journal of Computer Science and Network*, 8(5):436–440, 2019.

[14] Manuel Hernández and Eduardo Sánchez-Soto. Topos-theoretic framework for verifiable multi-agent coordination and team formation. In *2026 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)*, pages 1–6. IEEE, 2026.

[15] Petar Kochovski, Sandi Gec, Vlado Stankovski, Marko Bajec, and Pavel D Drobintsev. Trust management in a blockchain based fog computing platform with trustless smart oracles. *Future Generation Computer Systems*, 101:747–759, 2019.

[16] Michael Lockwood. *Personal data sovereignty: a sustainable interface layer for a human centered data ecosystem*. University of Salford (United Kingdom), 2020.

[17] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *2012 IEEE 5th International Conference on Cloud Computing*, pages 423–430, 2012.

[18] Matter Labs. zksync era architecture and atlas upgrade. https://zksync.io/, 2024. Accessed: 2026-04-11.

[19] Puja Ohlhaver, E Glen Weyl, and Vitalik Buterin. Decentralized society: Finding web3's soul. *Available at SSRN 4105763*, 2022.

[20] Polygon Labs. Polygon Miden: State Model. https://polygon.technology/blog/polygon-miden-state-model, 2023. Accessed: 2026-07-13.

[21] Polygon Labs. Polygon zkevm throughput benchmarks. `https://polygon.technology/`, 2024. Accessed: 2026-04-11.

[22] Swapna Krishnakumar Radha. *Formal and Decentralized Framework for Verifiable Trust: Self-Sovereign Identity, Blockchain Provenance, and Hardware-Rooted Assurance*. PhD thesis, University of Notre Dame, 2026.

[23] StarkWare Industries. Starknet performance and tps limits. `https://starknet.io/`, 2024. Accessed: 2026-04-11.

[24] Bruce Tate. *Programmer Passport: Elixir*. The Pragmatic Programmers LLC, 2022.

[25] Madhava Rao Thota. Scalable multi-cloud workload orchestration: Integrating big data and database operations through google cloud platform. *Journal of Scientific and Engineering Research*, 10(2):247–264, 2023.

[26] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. {VeriZexe}: Decentralized private computation with universal setup. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4445–4462, 2023.