InfinityVM Litepaper: Enshrining Offchain Compute

The InfinityVM Team

Version 1.0: 2024-09-01

Abstract

Blockchains were developed to solve the double-spend problem in a trustless setting, traditionally by relying on replicated execution within onchain virtual machines (VMs). Despite improvements in verifiable state replication models, current blockchain systems remain constrained by this design, optimizing for apps built entirely onchain with fully decentralized compute. InfinityVM instead reorients this paradigm towards decentralized verification of centralized compute. This is accomplished by vertically integrating offchain compute with an enshrined verification mechanism, combining the safety of zero-knowledge proofs with the efficiency of an optimistic network. InfinityVM applications execute primarily offchain, unconstrained by an onchain VM, and only utilize the blockchain for the decentralized properties they require: censorship resistance, verifiability, and composability. This design enables applications to scale both horizontally (distributed workloads) and vertically (flexibly resourced machines), while maintaining state offchain for optional privacy. InfinityVM supports the inevitable endgame of decentralized applications using all resources at their disposal for optimal performance while preserving the crucial properties of decentralization.

1 Why blockchains?

Blockchains were first proposed as a mechanism to facilitate exchanges between any two willing parties without the need for a trusted third party [11]. The fundamental requirement of such a system is to leverage cryptography, rather than trust, to ensure transactions cannot be reversed and thus that double spends can be prevented. With this achieved, a distributed ledger can be developed that has no central control.

The double-spend problem, while conceptually simple, has shaped the core principles of modern blockchain design:

1. **Transaction immutability:** The resilience of the system against transaction reversions. Protocols such as Proof of Work (PoW) and later Proof of Stake (PoS) have emerged to enable a distributed network of nodes to objectively agree on transaction ordering.

2. **Censorship Resistance:** The ability of the system to facilitate transactions for all willing participants, irrespective of potential adversarial actions.

3. **Credible Verifiability:** The capacity for any party to participate in and audit the network, thereby maintaining system integrity.

2 Blockchains are bounded by design

Blockchains at their core are replicated state machines, intended to maintain data in a global context not subject to any central authority in any particular jurisdiction. In order to modify this data, nodes must process transactions that transition the global state. Typically transactions are defined with onchain VMs where all nodes must replicate execution and reach consensus on both the input ordering and the state transition results.

Replicated onchain execution necessitates stringent resource management to safeguard against DoS attacks and malicious system access. This introduces strict limitations on the capabilities of onchain applications, notably:

1. **Execution constraints:** transactions and blocks have limits on the amount of computation (often termed gas) they can process. Storage limitations: onchain storage is limited and expensive, restricting the possibility to store images or other media onchain, and limiting the size of deployed contracts.
2. **Complexity limitations:** onchain VMs must function uniformly across all hardware, often disallowing common operations like floating point math.
3. **Onchain lag:** all onchain applications must update at the speed of the underlying chain's block time. This can lead to applications constantly being out of sync with the offchain world, as is most notably in the case of AMMs and Loss-Versus-Rebalancing [9].

If we break the tasks of a replicated state machine down to its core properties, these systems actually comprise two separate but related tasks: consensus on input ordering, and execution of those inputs in that order. Most existing blockchains tightly couple these two ideas, requiring all validating nodes to both order and execute transactions and then come to consensus on the results. The scalability of this system is fundamentally limited by the capacity of its weakest node since every node must be able to replicate all work in order to reach consensus.

Often formalized as "minimum hardware requirements" this parameter describes the hardware a theoretical minimum node will be running. Blockchains are thus bottlenecked by the capability of these minimum nodes, creating a tension between chain scalability and decentralization of nodes. In another sense, this rigidity in network resources also requires that the network constantly be over-resourced for the expected worst-case (highest) load, further constraining efficiency.

3 The rise of performant blockchains

The EVM greatly enhanced onchain operations, but Ethereum remains constrained by the tools available at the time of its creation. To address the limitations of the EVM, many solutions have been developed to push the bounds of performant and expressive compute.

3.1 Performance

Recent trends in blockchain designs have been towards "high performance" architectures, attempting to move the frontier of blockchain capabilities forward.

One common approach is to build a new onchain VM that addresses the limitations of the EVM. The most successful of these approaches, the Solana Virtual Machine (SVM), has achieved substantial performance gains by utilizing multiple cores to enable parallel transaction processing. However, this success comes at some cost in developer experience and still inherits the primary limitation of all onchain compute: execution must still be redundantly replicated across all nodes in the network, limiting its potential to scale.

A more fundamental approach to performance optimization involves decoupling execution from consensus [16, 10]. This separation can be achieved by various means and allows for the two core tasks to run in parallel: one process to achieve consensus on the ordering and availability of input data to the state transition function, and another process to execute the state transition function over these inputs. Consensus can optimize for a breadth of many nodes simply agreeing on ordering, whereas execution can run a smaller set of specialized nodes to execute the state transitions quickly.

However, these gains are all fixed improvements in performance, not ultimately scalable architectures [1]. This mechanism for upgrading performance will always be inflexible and sporadic.

3.2 Expressivity

While performance is the core metric for most new blockchain designs, the expressivity of their runtime environments directly limits the complexity of applications that can be built. In order to enable further expressivity, teams have utilized two primary mechanisms: AltVMs and coprocessors.

AltVMs have historically focused on one narrow field of expressivity: the language in which applications can be written. VMs like SVM allow developers to write smart contracts in languages like Rust, expanding the set of developers beyond just Solidity developers. AltVMs like Stylus also introduce further expressivity by enabling WASM bytecode, but are still subject to limits on contract size and gas per block.

Coprocessors represent a more dramatic attempt at enhancing onchain expressivity as they enable applications to leverage the results of stateless offchain compute. To date, coprocessors have largely played a supplemental role to onchain VMs by enabling specific complex functions over historical state. This is largely due to the fact that verifying this offchain compute requires concessions on either cost (ZK) or finality time (Optimistic).

3.3 Rollups

One instantiation of these ideas that improves both performance and expressivity is the concept of sharding, realized most directly today through rollups on Ethereum. Rollups enable more compute to be processed per block on the base chain because they can batch and compress multiple transactions into a single base layer transaction, and prove the validity of the entire batch. Rollups can also offer more expressivity by utilizing AltVMs, enabling applications that inherit the base chain's security to run with more flexibility.

However, the scalability provided by rollups comes with significant costs. Rollups today fragment the liquidity of the base layer because they require users to bridge into their ecosystem. These systems are also often implemented with upgradeable contracts on the base layer, introducing additional trust assumptions for security. Overall, rollups are often best considered as extensions of block space, as they inherit many of the same limitations associated with onchain compute.

4 Breaking the path dependency on onchain compute

Recent attempts at improving the performance and expressivity of blockchains have dramatically increased the scope of what can be implemented onchain. However, if we zoom out we can see that these advancements largely target systems that are path-dependent on existing onchain paradigms, rather than optimizing for the more fundamental goals of blockchains.

If we approach the design of a distributed ledger with double spend protections from first principles, we can imagine an incredibly streamlined solution. In this model, nodes would run a minimal blockchain serving two purposes: reaching consensus on an initial state, and agreeing on an ordering of transactions that would update this state. These transactions would then be efficiently executed offchain, with their outputs verified onchain, enabling each node to transition from the initial state to the output state in a trustless manner [2].

From this description, we can see that a natural fit for this purpose is ZK proofs. ZK promises many things, but we are mostly concerned with its ability to enable any prover to generate succinct proofs of computation over some public inputs, and allow anyone to efficiently verify these proofs. If we integrate this into a blockchain, we can enable developers to execute any function that they can generate a ZK proof for, and then have all of the nodes simply verify this proof to process the state transition.

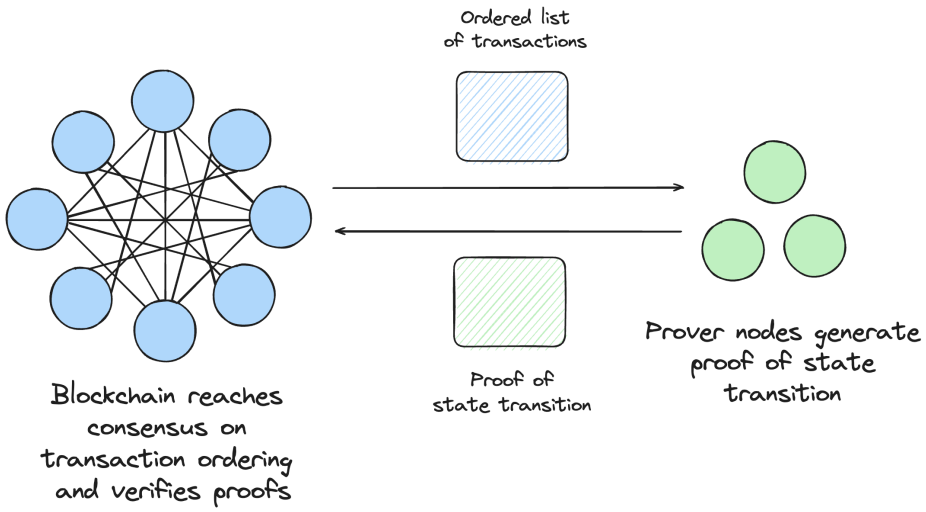

Figure 1: Centralized proving with decentralized verification

From this premise, we can also claim that developers should not be constrained by the bounds of onchain compute. Ideally, a developer should be able to write their application the same way they would outside of a blockchain context – as an application that can be scaled horizontally [6] as well as vertically, using the best resources available.

This approach also works nicely to enable scalability because of its flexible nature. From the perspective of the base chain, nodes are simply ordering a set of transactions and verifying a (constant-sized) proof. Application developers can run many apps on different servers that are then proven to the base layer.

By combining these ideas, we can create a system that enables:

1. **High performance:** nodes must only come to consensus on ordering and verifying proofs, enabling maximum performance.
2. **Scalability on-demand:** applications can run their own dedicated hardware that scales up or down as needed. This enables both horizontal and vertical scaling on a per application basis.
3. **Ultimate developer freedom:** applications can be built exactly as they would be offchain, only committing their state onchain as needed.

This is the inevitable end state of blockchain applications: offchain apps that inherit the specific benefits that a blockchain provides (censorship resistance, self-custody, trustless safety, and composability) without needing to run fully onchain.

5 InfinityVM: app servers with blockchain scaffolding

We introduce InfinityVM: an execution runtime to enshrine offchain compute into the protocol, designed to realize the full potential of the ZK-enabled future within the capabilities of ZK today. In fact, InfinityVM will always outperform pure ZK because it removes a layer of overhead, making it the best way to build ZK applications.

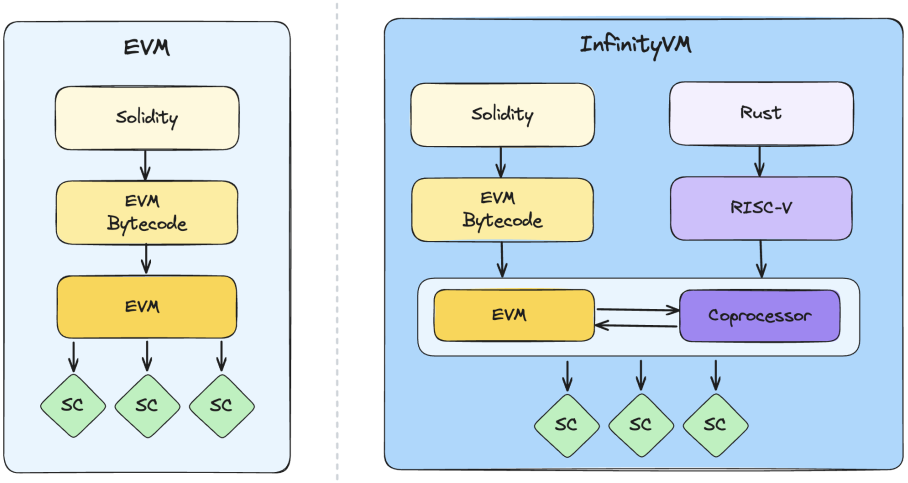

Figure 2: InfinityVM enshrines coprocessing alongside the EVM

Developers can design applications that run offchain but are then verified within the core logic of the chain itself, allowing for instant and safe verification. InfinityVM enables applications to be built as true "servers with blockchain scaffolding", as described by Vitalik [3]. Rather than developing applications that fit into the onchain constraints, InfinityVM enables offchain applications to hook into the specific benefits of the blockchain that they require.

InfinityVM applications are:

1. **Fully expressive:** apps can be written as offchain applications, allowing them to do things impossible within the constraints of blockchain VMs like the EVM.
2. **Scalable:** each app can run on its own terms, so resources do not need to be shared or limited. Apps can increase or decrease resource requirements on-demand to scale vertically or scale horizontally via more parallelized instance of the app server [6].
3. **Safe:** application state is backed by ZK validity proofs [14, 15, 5], enabling trustless verification with no intermediaries.
4. **Censorship resistant:** each application can utilize the base protocol as needed for forced-inclusion, utilizing the slower but more robust inclusion guarantees.
5. **Composable:** because applications instantly settle to the same base layer with uniform security guarantees, all state is immediately composable through the EVM.

With this architecture, we can break out of the path-dependent nature of existing blockchain applications where core logic is built onchain and supplemented by offchain systems. Instead, InfinityVM applications can be built offchain, leveraging InfinityVM's minimal onchain scaffolding to inherit the necessary blockchain properties.

InfinityVM utilizes the EVM at the base layer to enable composability and secure state management. This takes advantage of the EVM for the express purpose of high security and simple logic, and also inherits the rich tooling and support for the EVM.

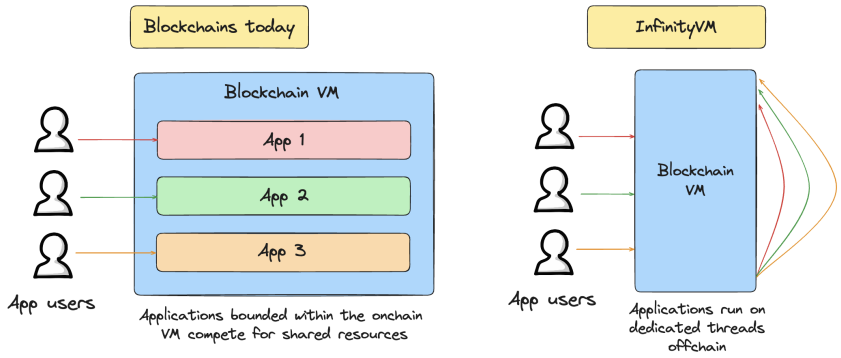

Figure 3: Comparison of blockchain app scaling

6 Challenges with offchain compute

Naively, we could accomplish each of these properties by utilizing ZK to enable any server to run an application and post proofs regularly for onchain verification. This quickly runs into the limitations of ZK: zkVMs today can express practically any application, but are still not performant enough to provide the latency and low costs expected for modern applications.

Alternatively, we can utilize optimistic techniques: applications can elect proposers that can attest to a state transition without any accompanying proof. Anyone can then run a "challenger" service to verify this transition offchain and submit a challenge onchain if the proposer maliciously submitted a fraudulent result.

However, optimistic systems must account for the possibility of reversions in the case of fraud. For a system like Arbitrum [8], this is minimized because the entire system is internally consistent and so this reversion risk is only relevant for any L2 state access on the L1 or for inter-chain actions.

When we imagine a system like this on an application-basis, where each application can post its result and is subject to a fraud period, this internal consistency goes away. Now each application can be reverted independently, which leads to each app needing to implement logic to handle fraud and reversions. In practice, this reversion logic at an application level is extremely complex, often requiring days of offchain social coordination to determine which actors were harmed and what the "right" reversion path is. Application-level reversion logic is a massive increase in complexity not only for developers to implement but also for users to understand. This only gets worse when considering composition of many applications with distinct reversion conditions.

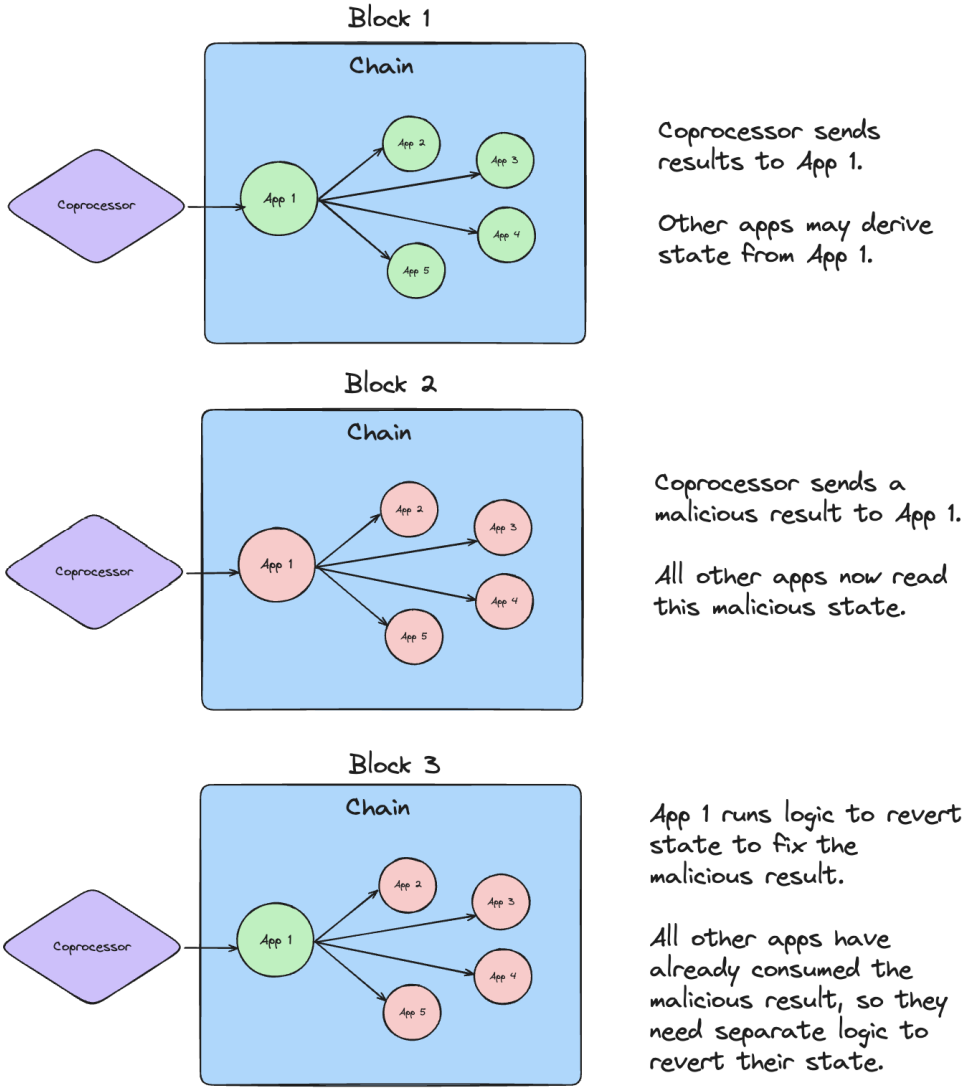

Figure 4: How co-dependent optimistic coprocessor apps introduce systemic complexity

7 Enshrining offchain compute

InfinityVM resolves these challenges fundamentally by enshrining offchain compute into the protocol, specifically into the fork-choice rule of the chain itself. We accomplish this by utilizing ZK fraud proofs and tying honest execution of coprocessing into the definition of the valid chain. The InfinityVM protocol guarantees that the canonical view of an InfinityVM chain will include honest execution of any offchain compute.

	In-protocol Finality	Out-of-Protocol Finality	Latency	Cost	DevX
Optimistic	High	High	Low	Low	Poor – devs must handle application level reorgs
Zk	Low	Low	High	High	Great – No reversions, no trust assumptions
Enshrined OP-ZK	Low	High	Low	Low	Good – Reversion logic is abstracted

Figure 5: Trade-off comparison between offchain compute integration techniques. Latency here refers to the perceived onchain time to receive results (including prover times).

Whereas the previous section detailed approaches to add supplemental offchain compute to an onchain system; InfinityVM elevates offchain compute into the protocol as a first-class citizen alongside the EVM. Fraud proofs run with a designated challenge period: any user can audit the output of a coprocessing job and submit a challenge onchain to dispute the result. Once challenged, the challenger is given time to run the job and generate a ZK validity proof of the correct execution. Upon a successful challenge, the chain itself will reorg by invalidating any blocks built off of the incorrect coprocessor result.

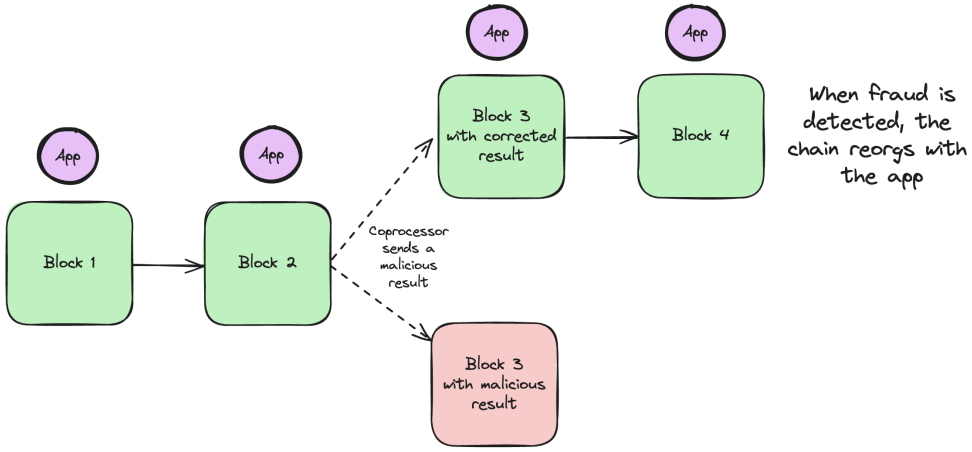

Figure 6: How enshrined coprocessing works: blocks containing invalid results are invalidated and forks including them are abandoned

This mechanism ensures that all applications built on InfinityVM will be derived from the same shared state and so they will all either progress together or reorg together, enabling near instant composability. This liberates developers and users from needing to process the complex overhead of potential reversions at the application level, freeing them to consider all offchain actions the same way they consider EVM operations. While each application can build as their own offchain server, one core unlock for onchain applications is permissionless composability. InfinityVM applications access this composability via coordinated offchain actions or the EVM base layer. This represents our philosophy of using the best tool for the job – optimized offchain servers to run applications and slower but well-understood blockchain VMs to settle and compose onchain. Effectively, applications are settling their state to the base layer of the InfinityVM. Certain application constructions can maintain all balances on the base layer, using their application logic simply as the state transition

engine to move from one settled balance to another. Utilizing this design pattern preserves liquidity on the base layer, avoiding fragmentation. Applications can freely scale their app logic, compose between each other, and enjoy the positive sum ecosystem where all liquidity growth benefits all applications.

7.1 Optimizing for the happy path

Enshrining offchain compute enables powerful applications, but requires specific concessions. Notably, we are making a specific decision to use optimistic verification over "pessimistic" verification. This distinction can be summarized by the burden of proving: optimistic techniques default to successful (non-fraudulent) outcomes where a proof is required to prove fraud but in the honest case is not needed. Pessimistic verification requires a proof on submission. As outlined above, this represents a tradeoff between onchain latency and prover costs.

Specifically, InfinityVM's design is choosing a tradeoff space that optimizes for the happy path where a proposer is not fraudulent. In a perfect optimistic system, the proposer never submits a fraudulent attestation and so the fraud proof mechanism is never utilized. This reduces the prover overhead towards 0, enabling the lowest cost for onchain verification. Rollups in production today (like Arbitrum and OP mainnet) are examples of these systems. As the role of a proposer in an optimistic system is expanded beyond trusted entities, this assumption of an honest proposer will no longer hold. However, our belief is that honest behavior will continue to be the dominant strategy for proposers.

As a result, it makes sense to optimize for the case of an honest proposer: proposers don't need to generate slow and expensive proofs for every state transition, but instead we can rely on a healthy challenger network that can generate proofs in the edge cases as needed.

The primary impact of this choice is on inter-chain actions – because the state of the chain is dependent on this challenge period you cannot safely send assets between a chain using InfinityVM until it is "settled". This is analogous to an optimistic rollup's fraud proof period. In the case of optimistic rollups, specialized actors like liquidity providers (LPs) have emerged to abstract this complexity away from users: users are able to operate so called "fast bridges" because the LP waits for the reversion period and fronts liquidity for users.

For the InfinityVM system, LPs can emerge to provide the same service. All state within the InfinityVM ecosystem is internally consistent (it all reverts together) so the only actions that will require external LPs will be bridging off of the chain, very similar to rollups. Overall, we are trading off some complexity for cross-chain interoperability, but in return we gain fully expressive and scalable offchain compute at minimal cost.

8 Architecture

The InfinityVM sidecar will expose a mechanism for developers to request jobs that can run on given sets of inputs (including both onchain and offchain data) and post results onchain. These jobs fundamentally consist of optimized code running on a regular server to either define or supplement an application. This mechanism can broadly be categorized as "coprocessing": dedicating a specialized and parallel offchain process to supplement the execution onchain via verifiable computation.

Developers can design their applications in two primary flavors:

1. **Onchain apps** using optimized offchain server compute to process complex logic
2. **Offchain apps** running on scalable offchain servers but posting verified state commitments and transitions onchain

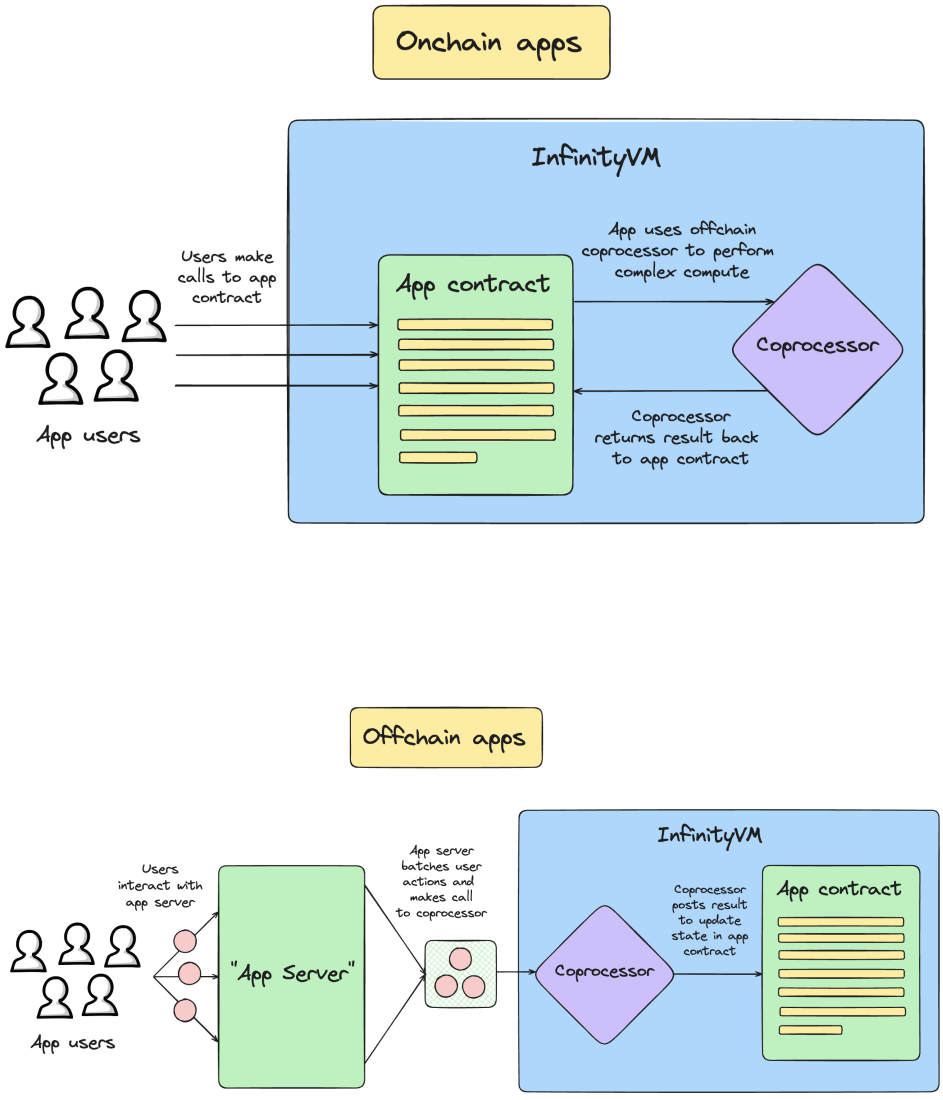

Figure 7: Two alternative app designs: differing on where the bulk of computation is processed

Regardless of the application's nature, all InfinityVM enabled applications will make standard requests for offchain work.

8.1 Requests

Utilizing the coprocessor begins with developers submitting `Programs` to the **Coprocessor Network** (a network of nodes running the InfinityVM sidecar). These Programs look like any other program developed today to run on a server. They include optimized code to accomplish a particular purpose without needing to adhere to a strict framework of blockchain semantics (block/tx limitations). A Program will define the code it runs and the interface of the inputs and outputs it expects.

Once the Program is submitted, a user can request a `Job` to run this program over a set of inputs. This data is posted to the coprocessor network and also replicated on an external DA service to provide strong availability and verifiability guarantees.

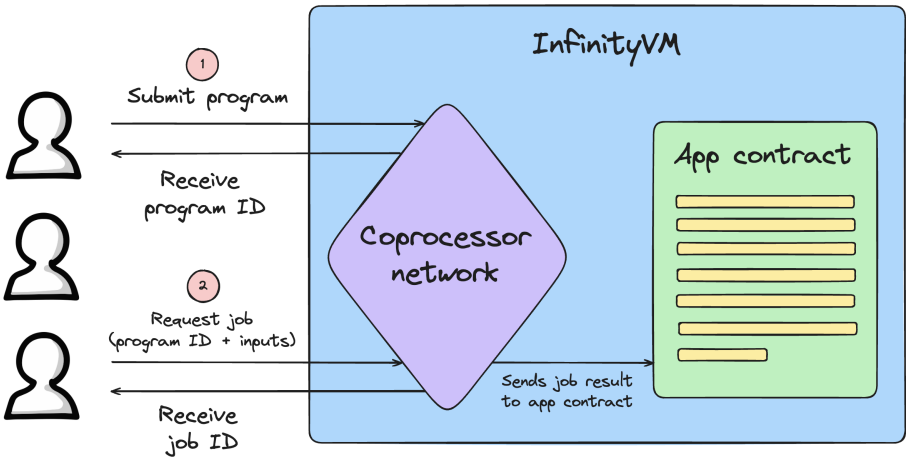

Figure 8: System architecture with an enshrined coprocessor receiving requests

Validators will then run this Job via the InfinityVM sidecar which operates as a parallel process to regular node consensus. This parallelization applies on a per-job basis, enabling these jobs to fully optimize execution via both horizontal and vertical scaling on-demand per job. Jobs can be requested in two primary modalities:

First is **asynchronous requests**: these requests can be made either onchain or offchain and are a commitment from the user of their intention for a coprocessing job to run a specific Program with a given set of inputs and to post the result to a given smart contract's callback.

Alternatively, users may make **synchronous requests** to the coprocessor. This is a unique capability inherent to InfinityVM enshrining the coprocessor into the chain. A user in this case can send a transaction requesting a coprocessing result to be committed within the same transaction as the request. This works for execution that has fixed cycle limits in order to not affect average block times. The proposer then commits to the result in the same transaction which may then use that result within that block to continue further operations.

Regardless of the approach taken, a result is then posted to the requested smart contract's callback function with an associated signature from the proposer of the result and the commitment to the job.

8.2 Validators

Once a developer submits a request to the coprocessor network, anyone can run the requested job with the committed set of inputs. This enables any actor to validate what the correct result of an offchain request is. In the event that an incorrect result to an offchain request was posted onchain, validating nodes would detect this and reject any blocks including this fraudulent result as invalid. Validating nodes are therefore able to maintain full trustless safety, although it does come at the cost of needing to run all onchain and offchain compute themselves. [1]

8.3 Proposers

While anyone can view and audit the correctness of job results, not just anyone can post a result onchain. The specific nodes that can post results onchain are called Proposers and they must post

[1] It is expected that various service providers in the ecosystem will run full validating nodes. This enables them to have a trustless view of the correct state of the chain at all times, allowing services like liquidity providers to provide fast bridging securely. These nodes are analogous to full nodes on an optimistic rollup like Arbitrum [8].

an economic bond with the coprocessor network. This economic bond will be slashed in the event that a proposer posts a fraudulent response, incentivizing the proposer to act honestly.

The key to InfinityVM's ability to provide instant and composable offchain compute is enshrining the coprocessor into the fork-choice of the chain. However, this means that a malicious proposer can introduce a fork by maliciously posting fraud. In practice, this attacker's gain is actually quite limited because of the ability for validating nodes to detect fraud quickly and stop transacting with the fraudulent network state.

That said, instances of fraud still come at significant cost monetarily and socially, and so our system takes all actions to minimize them. Specifically, the role of proposer in InfinityVM will follow a similar path to the proposer of state roots in optimistic rollup systems. Initially, this will be a permissioned action by a set of trusted actors, utilizing both economic bonds and social reputation to incentivize good behavior. In the longer term, though, a network of proposers will be built that can be rotated to choose the next proposer.

8.4 Challengers

Challengers are critical actors in the system who identify instances of fraud and submit a proof of incorrect execution. A challenger must necessarily run a full validating node in order to determine the correct state and detect a malicious proposal. As mentioned above, ecosystem service providers will likely play the role of validators and challengers because the services they provide require instant finality. They are incentivized to provide this service by user fees.

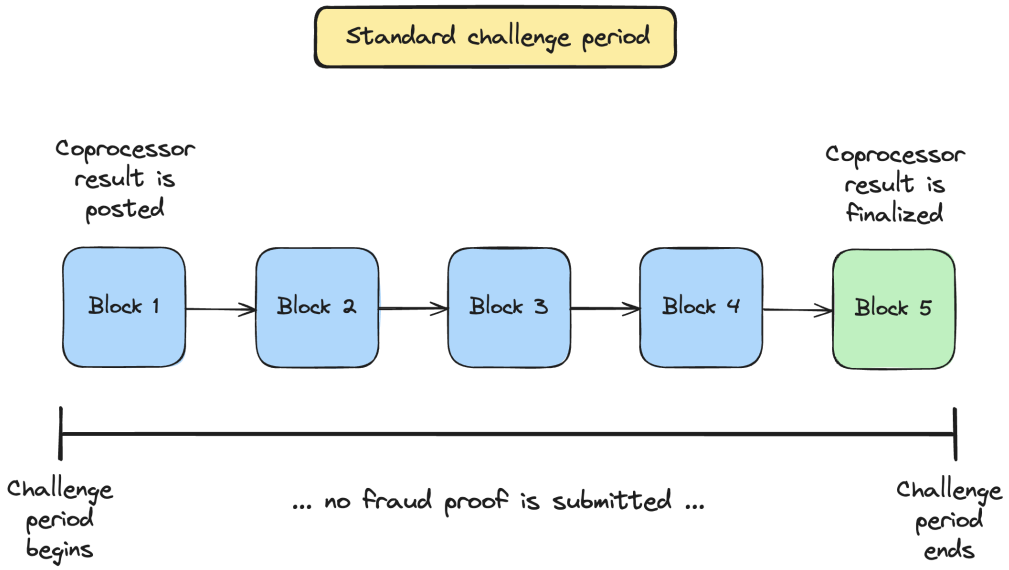

Figure 9: Coprocessing jobs reach finality onchain when the challenge period expires without challenge

Once fraud is detected, a challenger can initiate a `Challenge` by sending a transaction with an associated bond. This transaction must be sent within the challenge period mentioned above. Once a proposal has been challenged, a new timer called the prover period begins. This designates the time that the challenger is given to generate a fraud proof. A challenge process can end in one of two ways:

1. Fraud: if the Proposer `P` did commit fraud by posting result `X`, a challenger `C` can run the same program over the same inputs and generate a ZK validity proof that shows the correct result should be `Y`. Challenger `C` can then post this proof to the arbitrating smart contracts onchain

which will verify the proof and determine that the initial result `X` was fraud and slash proposer `P`. The result `Y` will now be accepted everywhere as the correct state (since anyone can cheaply verify the proof) and the chain will progress.

2. No fraud: if the Proposer did not commit fraud and `X` is indeed the correct response, two paths may be followed that end in the same result. First, any actor (including the proposer) can now generate a ZK validity proof that shows that the result actually is `X`. This can then be verified onchain similarly to (1). Second, the proposer could instead do nothing. Since a challenger cannot generate a proof that `X` is incorrect, the prover period will necessarily end with an unsuccessful challenge. In both cases, the challenger `C` will have their bond slashed as a penalty for wasting the protocol's time, disincentivizing griefing attacks.

8.5 Provers

While honest nodes can always rely on eventual consistency to guarantee the correct state is always written onchain, there are also incentives for Provers to exist to expedite the process and simplify verification. Because all offchain actions are verifiable via ZK proofs, provers can run an asynchronous process to generate ZK validity proofs for all offchain jobs. Importantly, this role is not necessary in-protocol for any purpose besides challenges, meaning proofs are only necessary in the worst-case, but they can provide numerous out-of-protocol benefits. [2]

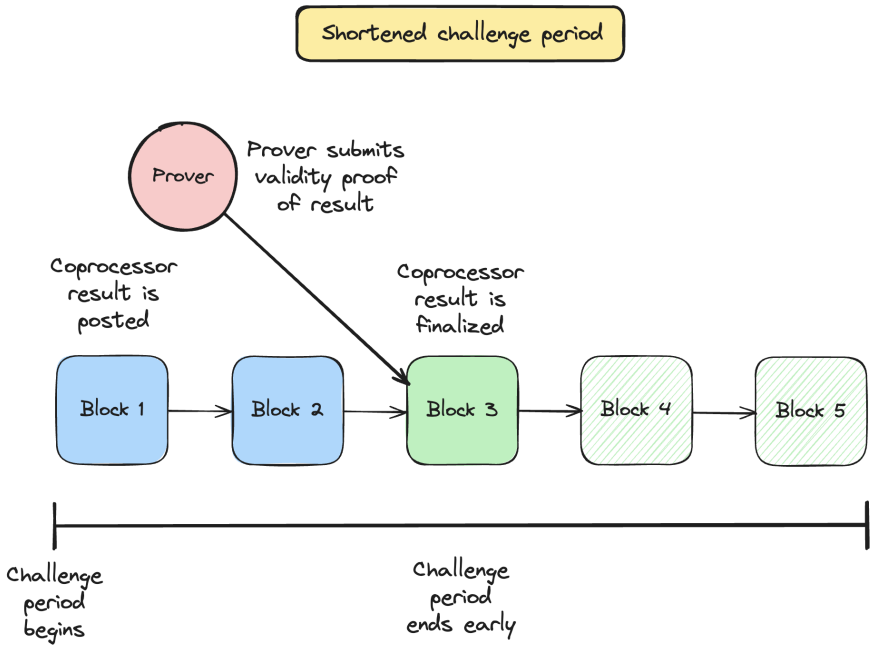

Figure 10: Once a validity proof is supplied onchain a Job is finalized with the (provably) correct result

These provers are incentivized by the network to generate these proofs because of the benefits they pose to the ecosystem. Any potential challenge of an offchain compute job can be instantly closed once an honest prover is able to submit a ZK validity proof to support the posted result. Importantly, in the limit this means that a chain with InfinityVM can become a fully ZK validated

[2]The existence of prover nodes would enable lighter nodes that still have trustless safety guarantees but must necessarily lag behind the tip of the chain because of the overhead of ZK proof generation.

coprocessing chain. We view this as a fallback outcome because the chain must then inherit the limitations of ZK, and InfinityVM offers the opportunity to expand beyond this.

8.6 Reversions

If fraud is posted onchain by a malicious proposer and is successfully challenged a reversion will occur. The specific details of this reversion depend on the architecture of the chain running InfinityVM. In the case of a fully independent L1 or a sovereign L2, the canonical chain can largely be determined by an honest majority of nodes validating blocks with honest job results. In the event of a successful challenge, the L1 may only slash and jail the offending proposer, but otherwise the canonical chain can continue to grow.

In the case of deploying InfinityVM as an L2, the story is more detailed. This protocol must be augmented to understand the potential for reversions in order to allow the base layer to follow the canonical chain. The simplest mechanism to achieve this is by arbitrating fraud on the base layer itself. If fraud is proven, the base layer can directly invalidate (and remove) these batches of L2 state, preventing fraudulent state transitions.

Reversions in general can be painful, particularly when socially enforced. In the case of InfinityVM, however, the canonical state of the chain is deterministic and must always follow the honest outcomes of a requested job. This is different from infamous forks in the past that have been the result of social judgments to return stolen funds [7]. Those forks were subjective and had to be discussed and debated over days or weeks offchain. InfinityVM is a deterministic execution engine that uses this potential for reversion as a mechanism optimizing for scalability, but the chain will always remain eventually consistent for honest actors.

9 Examples

9.1 Offchain App (CLOB)

One example use case this construction enables is a Central Limit Order Book (CLOB) that runs on an optimized offchain server and uses InfinityVM as part of its blockchain scaffolding.

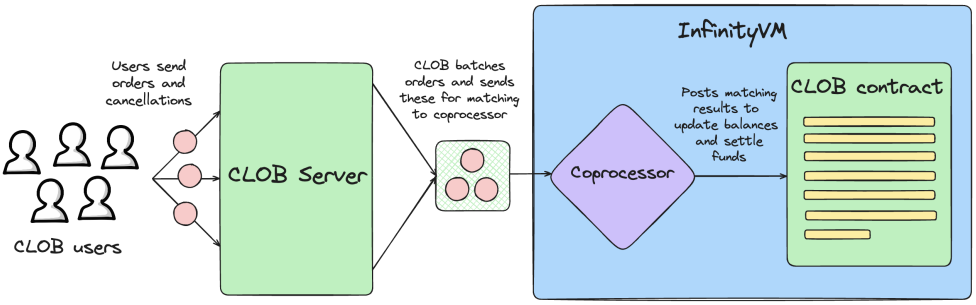

Figure 11: How the CLOB application works: running compute offchain and running state transitions through the coprocessor

This approach flips the existing smart contract paradigm on its head: the bulk of the application runs offchain on a server that operates the order book centrally, enabling free, real time order submissions and cancellations while maintaining safety and strong async composability. For each order/cancellation submission, the server returns a signed commitment, which includes an assigned global index. [3]

[3] This is used to prevent violations of price-time priority.

The server regularly batches orders and creates a coprocessor job to run the batch against the state transition program to match orders. The InfinityVM proposer then posts the result of this job onchain. By enshrining verification we can guarantee that the state of the CLOB is valid once job results are onchain. This CLOB application now has safe composability with other applications, while processing orders at web2 speeds.

Verifying parties can trustlessly audit the state transitions by constructing the order book and commitments. If a user detects a faulty state transition they can initiate a challenge against the server by posting the signed commitment from the server and then running the matching algorithm in the coprocessor to prove the correct state transition. Successful challenges will lead to the chain reorging to remove the fraudulent state transitions and any transactions built off of it, guaranteeing safety for traders and composing applications. Further, the design could be made censorship-resistant by adding measures like server rotation and forced inclusion of orders onchain.

This design unlocks a trustless and highly performant CLOB implementation that significantly improves upon previous constructions (like dYdX v3/v4), which relied on trusted parties (either dYdX in the case of V3, or their validators in V4) to operate the order book offchain.

9.2 Onchain App (Dynamic Fee AMM)

InfinityVM also unlocks sophisticated onchain apps that benefit from supplemental offchain compute. One exciting example of such a construction is an AMM with a dynamic fee structure (e.g. discounted fees based on a user's transaction volumes or account standing). Such a design has been previously impractical since it requires complex computations that either may not fit within a block's constraints or may increase costs to the point of making transactions economically infeasible.

InfinityVM enables dynamic fees to be calculated offchain and then utilized onchain with uniform trust assumptions. The coprocessor processes historical data to calculate if the specified order flow metrics (user, asset pair, order size) should receive a discounted fee. For example, an AMM could offer volume discounts where users who have traded more than $X in the past month qualify for reduced fees.

It is possible to build dynamic fee AMMs using existing coprocessing frameworks as well but relying on an external protocol presents several headwinds. Applications using a ZK coprocessor must pay the high proof generation and verification costs to safely utilize the results onchain. Applications that instead choose to utilize the coprocessor in an optimistic setting, must incorporate some notion of state reversion in case the coprocessing result is proven fraudulent. In this example, there is no way to change the fee that was charged at time t (when the optimistic result was posted) when fraud is proven onchain at time t+n. The only solution would be to hold funds or fees in escrow until the challenge period passes, which introduces significant complexity and friction.

Onchain applications built using InfinityVM can get all of the benefits of historical state access and ZK provable compute without needing to factor in the costs of generating and verifying ZK proofs onchain. InfinityVM applications can also utilize the results instantly because of the enshrined dispute mechanism, allowing apps to ignore the complexity of possible reversions.

10 Related Works

10.1 Coprocessors

The original conception for bringing offchain compute onchain, coprocessors are a general framework that can be implemented with ZK or optimistic verification. Applications on these coprocessors have been limited, primarily just enabling historical state access onchain. This is because the external verification methods introduce limitations either on latency or on costs.

10.2 Enshrined Rollups

An idea most directly associated with the Tezos ecosystem, enshrined rollups have also been considered for the Ethereum ecosystem. What is meant by enshrining a rollup, though, is slightly different from enshrining offchain compute (like InfinityVM). Most enshrined rollup designs are specifically focused on the smart contracts that maintain the state of the rollup on the base chain. By pulling

the rollup's smart contracts into the chain the rollup is no longer controlled out-of-protocol, and so it can inherit direct security from the base chain. This design is a tradeoff between security, trustlessness, and flexibility.

10.3 Rollups with Beefy sequencers

An approach growing in popularity today is to run traditional optimistic rollups with a centralized sequencer but to provision this sequencer with significant resources. When paired with other innovations like solutions to state access and IO, this allows the rollup to process more transactions per second because the sequencer can handle more load and provide better UX. This has been touted as the original conception for a rollup as a "server with blockchain scaffolding", because the sequencer is a high performance server and the addition of other actors in the system provide the blockchain scaffolding.

InfinityVM can be viewed as a generalization of this idea, while also enabling further expressivity. High-powered sequencers can enable very high TPS on existing blockchain VMs, but they are still bottlenecked by the limits of blockchain VMs. InfinityVM enables each app to be its own high-performance server, allowing them to be more expressive and flexibly scalable. These apps can also use truly offchain infrastructure, rather than needing to be deployed on a blockchain.

10.4 Contingent Rollups

A core building block to the idea of InfinityVM is the idea of blocks that are contingent on the state of another domain. This was most specifically articulated by James Prestwich's piece on cross optimistic rollup contingent blocks [12] and is an extension of an idea Vitalik proposed for cross-shard bridging [4]. While this system remains viable to build on a chain by chain basis, InfinityVM is an opinionated deployment of this idea where each application is contingent on the same state, enabling heterogeneous composability.

10.5 Franchised Sequencers

The concept of a franchised sequencer is to enable "normal" applications in a blockchain context [17]. Specifically, these apps are private and post "fingerprints" (state commitments) to the chain that are verified by ZK validity proofs. The sequencer is responsible for posting these commitments, and this right can be "franchised" or otherwise distributed. The potential for this idea is large, enabling "regular" developers to build their applications while inheriting the benefits of blockchains without being beholden to the chain itself – echoing many of the benefits we've outlined in this paper. However, franchised sequencers do imply ZK validity proving in order to verify application validity (a noted limitation in their design), inheriting the negatives of ZK, namely costs and prover latency. This is because these systems have been imagined to exist on top of existing ecosystems which do not have the capacity to add application validity checks to their protocol rules. InfinityVM is a realization of this idea (each offchain application can be its own franchised sequencer app) while mitigating the downsides of ZK because of our enshrined verification (using contingent state).

10.6 AggLayer / Elastic Chains

Interop solutions like AggLayer connect rollups via shared liquidity bridges and aggregated ZK validity proofs. These aggregation layers enhance cross-chain interactions by addressing interoperability at the infrastructure level, making it feel like a single, unified network for the end user. These systems can then be supplemented by solutions like a shared sequencer in order to also achieve synchronous atomic interoperability across chains.

However, these layers primarily serve to compose existing ecosystems together without fundamentally advancing the expressivity or performance of applications. The InfinityVM approach is a more fundamental shift away from standard chains and towards fully expressive and scalable application servers. InfinityVM aggregates applications as their own servers, rather than aggregating chains, removing the need for appchains with their associated complexity. Synchronous composability across applications is achieved by default, without relying on external shared sequencers. By

enshrining the validity rules of applications directly into the core protocol, InfinityVM also mitigates the high costs and latency associated with ZK-based aggregation systems. In some ways, InfinityVM is an opinionated implementation of an aggregation layer with strong interoperability capabilities.

10.7 Cartesi

A direct approach to bringing offchain compute onchain, Cartesi enables offchain nodes to run application logic and unify it directly into the broader Cartesi layer 2 ecosystem. This has many similarities to InfinityVM, but takes a different approach. Cartesi's innovations are largely focused on reproducible and deterministic builds + deployments of "linux machines" to run offchain compute on. This is then incorporated into the network to allow nodes to run these machines and verify all computation. Offchain compute is either reproduced by a network of Cartesi Nodes or is run on a dedicated machine. This computation is still subject to eventual disputes and so must maintain logic for reversion.

10.8 Naysayer Proofs [13]

Recent work on the topic of optimizing for ZK verification costs in the Naysayer construction was part of the motivation for the original InfinityVM idea. The core insight of this work is that despite the fact that the overwhelming majority of ZK proofs generated are honest and verified as true, most systems continue to verify all proofs. This verification has significant costs, generally around the range of 100-500k gas for a verification, but can be much larger for different proof systems. This constant cost directly impacts the applications that can be built with ZK because they must either internalize this cost or pass it on to users.

The naysayer construction is to still require proposers to generate ZK validity proofs, but to defer verification to offchain entities. Anyone offchain can cheaply verify the ZK proof offchain to determine if the proof is fraudulent or not. These actors can then determine whether they want to send further transactions based on if the proof verifies or not. In the event that a proof does not verify, a challenger can then generate a proof that shows the submitted ZK proof is invalid. This challenge can then be submitted onchain and verified in order to reject the original proposal.

While this construction introduced one significant optimization to ZK systems, our view is that there is yet another level deeper to go, which is what InfinityVM represents. With InfinityVM, proposers do not need to generate a ZK proof at all, enabling many of the same benefits as naysayer proofs, but with lower latency and compute costs.

11 Conclusions

While onchain VMs continue to improve, they remain fundamentally limited in performance and expressivity. InfinityVM introduces a paradigm shift in achieving scalable decentralized compute by leveraging verifiable offchain execution. Applications can scale flexibly independent of the base chain, providing user experiences comparable to centralized applications today.

InfinityVM achieves this through an enshrined verification mechanism that defines the canonical chain by the correct execution of offchain compute. The maturation of zkVMs allows us to express any application into a zkVM to enable fraud proofs for offchain work. The architecture strategically allows applications to compose offchain compute with onchain EVM applications, enabling each system to excel at its purpose. By treating coprocessing as a first-class citizen, InfinityVM unlocks brand new capabilities onchain that were previously infeasible or cost-prohibitive.

InfinityVM represents the inevitable endgame of blockchain applications: offchain systems that inherit blockchain's core benefits (censorship resistance, self-custody, trustless safety, and composability) without the constraints of full onchain execution. It paves the path for a new generation of decentralized applications that can match the efficiency of centralized systems while preserving the security guarantees of blockchains.

References

[1] Joseph Bonneau. *Why blockchain performance is hard to measure.* https://a16zcrypto.com/posts/article/why-blockchain-performance-is-hard-to-measure/.

[2] Vitalik Buterin. *Endgame.* https://vitalik.eth.limo/general/2021/12/06/endgame.html.

[3] Vitalik Buterin. *Epochs and slots.* https://vitalik.eth.limo/general/2024/06/30/epochslot.html.

[4] Vitalik Buterin. *Fast cross-shard transfers.* https://ethresear.ch/t/fast-cross-shard-transfers-via-optimistic-receipt-roots/5337/1.

[5] A16z Crypto. *Jolt.* https://github.com/a16z/jolt.

[6] Wei Dai. *On Trust Minimization and Horizontal Scaling.* https://wdai.us/posts/trust-minimization-horizontal-scaling/.

[7] Ernesto Frontera. *A History of 'The DAO' Hack.* https://coinmarketcap.com/academy/article/a-history-of-the-dao-hack.

[8] Offchain Labs. *Inside Arbitrum Nitro.* https://docs.arbitrum.io/how-arbitrum-works/inside-arbitrum-nitro.

[9] Jason Milionis et al. *Automated Market Making and Loss-Versus-Rebalancing.* https://arxiv.org/pdf/2208.06046.

[10] Monad. *Deferred Execution.* https://docs.monad.xyz/technical-discussion/consensus/deferred-execution.

[11] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* https://bitcoin.org/bitcoin.pdf.

[12] James Prestwich. *Cross-ORU Contingent Blocks.* https://prestwich.substack.com/p/contingency.

[13] Istvan Andras Seres, Noemi Glaeser, and Joseph Bonneau. *Naysayer proofs.* https://eprint.iacr.org/2023/1472.pdf.

[14] Risc-Zero team. *risc0.* https://github.com/risc0/risc0.

[15] Succinct Team. *Succinct.* https://github.com/succinctlabs/sp1.

[16] Toly. *Endgame Architecture.* https://docs.google.com/document/d/1fQp2G-W0fFN19nRZVRbAwxD7Qa8H8cn5VVUvPKOF1Pg/edit#heading=h.mn48hr9e6wzk.

[17] Tom Walton-Pocock. *The Franchised Sequencer.* https://walpo.substack.com/p/2422f310-e62d-44b1-9e3a-a504b9de4e96.