Supra's Blockchain Infrastructure Stack

Supra Research

Tuesday 19 November, 2024

Abstract

We present an overview of Supra's blockchain infrastructure technology in this whitepaper.

We connect our multiple innovations, which were published over the years, into a resilient, extreme performance architecture that *vertically integrates* MultiVM smart contract support and native services: such as price feed oracles, on-chain randomness, cross-chain communications, and automation.

We describe the end-to-end transaction flow of Supra's blockchain and show how we mitigate censorship and Maximal Extractable Value (MEV) attack vectors. We showcase Supra network's capabilities in terms of our transaction throughput and commit latency through experimental studies.

The Supra network provides a wide range of services and features on a shared security platform. These include capabilities such as our novel Distributed Oracle Agreement (DORA), Distributed Verifiable Random Functions (DVRF), a zero-block delay automation network, AppChain inspired Containers, MultiVM support, and optimized block execution via parallel transaction processing. Additionally, our bridge designs—HyperLoop and HyperNova—position Supra as the world's first "IntraLayer" linking multiple chains to and from Supra through logic executed by Supra's smart contract platform.

1 Native Vertical Integration of Blockchain Services

Blockchains are transforming the world by creating a new paradigm through decentralization. We are merely at the beginning of recognizing the potential of a truly global village. While this is particularly evident in the finance sector, the influence and reach of blockchains are gradually extending into all areas of human activity. Supra aims to support and enhance this revolution through our groundbreaking research and robust engineering capabilities. We have an ambitious mission poised to bring to market a highly efficient Layer 1 blockchain that is vertically integrated with various blockchain-related services, offering clients a comprehensive solution and a seamless experience for both users and developers. In this paper, we illustrate how Supra's technology embodies the insightful words of the ancient Greek philosopher Aristotle: "The whole is greater than the sum of its parts."

Guided by our philosophy of vertical integration, we offer a comprehensive suite of features and blockchain services tailored for both individual and institutional users, as well as developers. The services available natively for dApps on the Supra blockchain are outlined in Figure 1 and include:

- A Layer 1 blockchain that supports various types of assets, including native tokens, programmable fungible and non-fungible tokens, and standard cross-chain tokens.
- Multiple on-chain Turing complete smart contract [8, 21] execution environments. A smart contract language platform is essential for implementing various Decentralized Finance (DeFi) protocols, games, lotteries, blockchain-based supply chain tracking, and other applications on a public blockchain.
- On-demand (pull-based) and streamed (push-based) off-chain data services, such as cryptocurrency price feeds, foreign exchange rates, indexed equities, and weather data, delivered to the Supra blockchain and other blockchains through our Distributed Oracle Agreement (DORA) protocol [10].

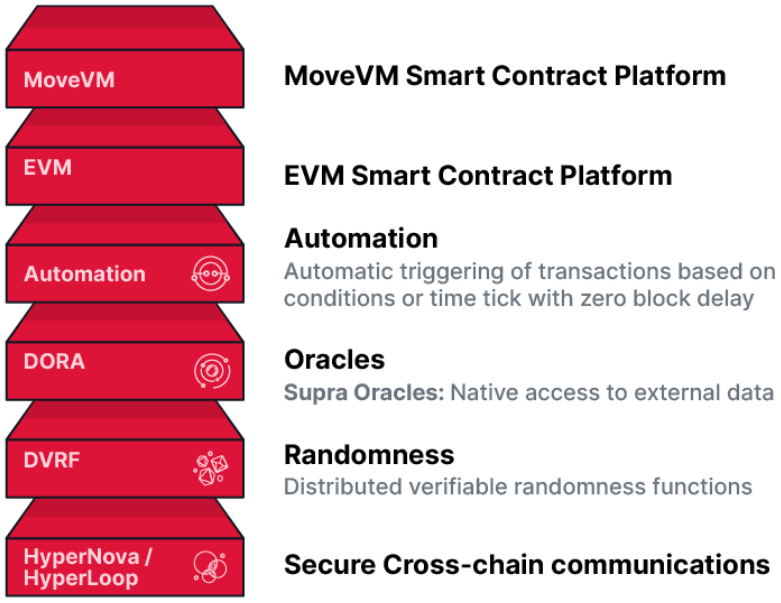

Figure 1 Vertical Integration

- Push and Pull on-chain randomness services [15, 13, 16], generating and providing distributed verifiable random functions (dVRF) for both Web 2.0 and Web 3.0 users, available in both streamed and on-demand formats.
- Automation network for scheduling transaction execution based on specific time ticks, on-chain events, or off-chain events (provided through DORA). For example, a client might request: "On June 1st, 2025 at precisely 12:00 am EST, if ETH is above $4000, then sell my DOGE."
- Application-specific chains (AppChains) are represented as Containers [2] on Supra. These Containers provide the flexibility and autonomy of an AppChain, while being significantly cheaper to deploy and furthermore benefit from the high performance, shared security, and unified liquidity of the Supra network.

2 IntraLayer: Connecting Blockchains to Supra and Beyond

While Supra offers a range of native services, we acknowledge the multi-chain reality and its value. To enhance interoperability, we propose a star topology (see Figure 2) where Supra's Layer 1 blockchain, along with its integrated services, serves as an IntraLayer network, connecting to other Layer 1 and Layer 2 chains through our interoperability solutions: HyperLoop and HyperNova. While Supra functions as an independent MultiVM smart contract platform, it also aspires to play a crucial role as a layer between or "within" other networks. This includes facilitating value exchange through native smart contracts, automation, and oracles, and enabling communication with other chains in a crypto-economically secure manner.

HyperLoop [4] is based on a traditional multi-signature bridge protocol but has been rigorously analyzed and proven to be game-theoretically secure, the first of its kind in our industry. HyperNova [5] is a trustless interoperability solution that safeguards the security

Figure 2 Supra's IntraLayer

of the connected chains without imposing security assumptions of traditional bridge or relay nodes. We outline the specific scenarios in which HyperLoop and HyperNova are most applicable.

- We find that our HyperLoop bridging solution is particularly advantageous for connecting optimistic rollups to Supra, as it eliminates the fraud-proof challenge period required for true finality.
- HyperNova is suitable for connecting any Proof of Stake (PoS) Layer 1 chains to Supra, as it maintains the security of the connected chains by simply recomputing the consensus of the chains Supra interacts with, thereby preserving L1 to L1 security. Supra Layer 1 blockchain is specifically designed to facilitate secure and efficient cross-chain communications.
- Our plans include using HyperNova for bridging Bitcoin to Supra while employing HyperLoop for the reverse connection. We are also exploring the possibility of implementing atomic swap methods in this context.

The following list showcases just a few capabilities enabled by the Supra IntraLayer technology stack:

- Decentralized Finance (DeFi) IntraLayer [6], which acts as a platform of platforms, encapsulates a range of well-known DeFi protocols, such as Automated Market Makers [7].
- Dapp creators can effortlessly access assets and information across various chains.
- A cross-chain automation service that enhances user experience by enabling consumers to set up automation tasks based on events or information from multiple chains.

We believe that the native vertical integration of multiple services on a high-performance Layer 1 works hand-in-hand with our aspirations to be the world's first cross-chain IntraLayer. As demand for our services from other blockchains rises, the utility of our network — including

our token and the assets on our blockchain — *may fluctuate* in value, motivating clients to adopt our infrastructure. Likewise, as our IntraLayer architecture gains greater utility across different ecosystems, more developers will discover our native services and performance, further encouraging them to onboard and build directly on the Supra blockchain. This will enhance our DeFi layer, encompassing both our in-house DeFi protocol offerings and third-party protocols within our network. Consequently, the Supra network may increase adoption and traction. We believe that our extreme performance, all-in-one offering of diverse services will facilitate the next wave of Web 3.0 adoption among the growing developer community.

3 Supra's Core Insight - Tribes and Clans

Supra's fundamental philosophy is vertical integration, providing most, if not all, blockchain-related services on a single platform. This approach ensures that users and developers enjoy an integrated and seamless experience.

Supra employs a modular architecture where individual components can be upgraded independently without disrupting the entire system. This architectural approach differs fundamentally from modular Layer 2 solutions, which distribute core functionalities like consensus, execution, and data availability across separate networks. While Layer 2s claim modularity as an advantage, their approach introduces significant drawbacks: increased latency due to cross-network communication, fragmented economic security because of disparate networks with their own token, and higher overall complexity.

In contrast, Supra's vertically integrated design unifies all components under a single blockchain with shared economic security and a unified token economy. This cohesive architecture delivers superior performance through reduced latency, enhanced security through unified incentives, and lower operational costs compared to fragmented Layer 2 solutions.

Research in distributed systems has shown that protocols such as Byzantine Fault Tolerant (BFT) State Machine Replication (SMR), BFT reliable broadcast, distributed oracles, and distributed key generation can tolerate up to one-third of nodes being Byzantine in asynchronous or partially synchronous network environments. Supra's core insight [22] takes a leap from this setting and allows a distributed protocol to tolerate up to one-half Byzantine nodes.

Supra's Core Insight

Run only the ordering service in a network that can tolerate up to one-third Byzantine nodes, and then base all other services on this ordering service. This will improve the Byzantine fault tolerance of these services from one-third to one-half.

The ordering service is currently handled by a consensus protocol that forms the foundation of Supra's Layer 1 blockchain. As we explain later in this paper, because the transaction data dissemination is separated from the ordering service, our blocks do not contain the actual transactions. Instead, they only include metadata for batches of transactions, primarily consisting of digests and data availability proofs. As a result, the blocks in the Supra blockchain are quite small. Since the ordering service is so central and all other services rely on it, we refer to our blockchain as the "Chain of Integrity".

The pertinent question is: what exactly is the benefit of increased tolerance of the number of Byzantine nodes for a protocol? The answer is that it allows the protocol to operate with a smaller committee. For instance, to tolerate 50 Byzantine nodes, a committee of 101 members could be used instead of 151. This reduces the number of nodes that need to be

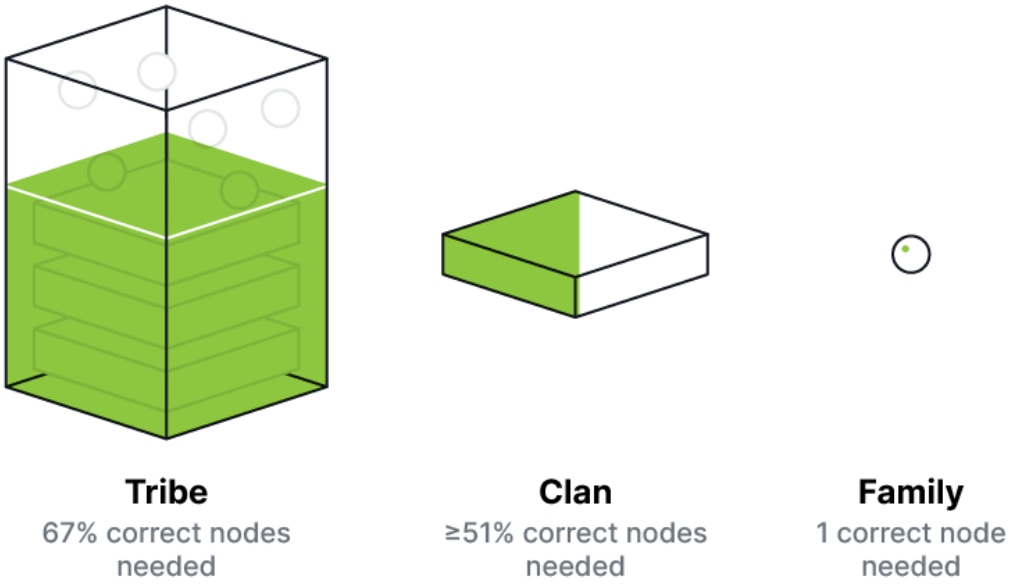

Figure 3 Tribes, Clans, and Families

compensated for their work, resulting in lower user fees. Moreover, it enhances execution speed, as fewer nodes are involved in reaching consensus, all while preserving the protocol's strong security properties.

Tribes, Clans, and Families

This core insight enables and facilitates performant vertical integration of multiple services on the Supra blockchain. Supra offers a novel network framework to enable executions of a variety of different algorithms that may require computing on the tribe, clan, or family level.

As shown in Figure 3,

- a tribe is a committee of nodes tolerating at most one-third Byzantine nodes,
- a clan is a committee of nodes tolerating at most one-half Byzantine nodes, and
- a family is a committee of nodes requiring at least one correct node.

To achieve optimal performance and robust security, we structure our network as follows: active nodes are grouped into a tribe that runs our consensus protocol, providing the ordering service and tolerating up to one-third Byzantine nodes. A key insight from our core design is that the entire Supra tribe is not necessary for transaction execution or for maintaining the full Supra state. Instead, a smaller subset called a "clan," is sufficient to manage the state, execute transactions, compute the post-state of a block, and sign state commitments. As a result, transaction data dissemination initially occurs only at the clan level before it is broadcasted further.

This architecture is well-suited for efficient state sharding, where different clans manage distinct state shards, potentially using separate virtual machines. This provides scalability, allowing us to adjust throughput by adding more clans (or shards) as needed. Therefore, we run all protocols—except for consensus—such as data dissemination, sharded execution,

oracle services, and distributed randomness services, in smaller committees (clans) that require only a simple majority of correct nodes for operation.

We organize our ordering tribe and multiple service clans in such a way that these clans actually form a partition of the tribe using a randomized selection of nodes into these clans. Consensus, or global ordering, is run on the tribe, while a variety of verifiable compute services are executed in clans. This random selection of nodes allows us to execute at the clan level and introduces a probabilistic element to the system. For example, suppose the tribe consists of 625 nodes, with no more than 208, Byzantine nodes. If the tribe is divided into 5 clans of 125 nodes each, the probability that any given clan contains more than half, i.e., 62 Byzantine nodes, is approximately 35×10^{-6}. In other words, the likelihood of having a *bad* clan is just 35 in a million draws when the entire time the tribe is 33% Byzantine. These probabilities are extremely low in our view. We will revisit these probabilities when discussing epochs and cycles, and show that they become virtually negligible in practice.

4 Transaction Flow

The schematic in Figure 4 shows the end-to-end transaction flow in Supra's blockchain:

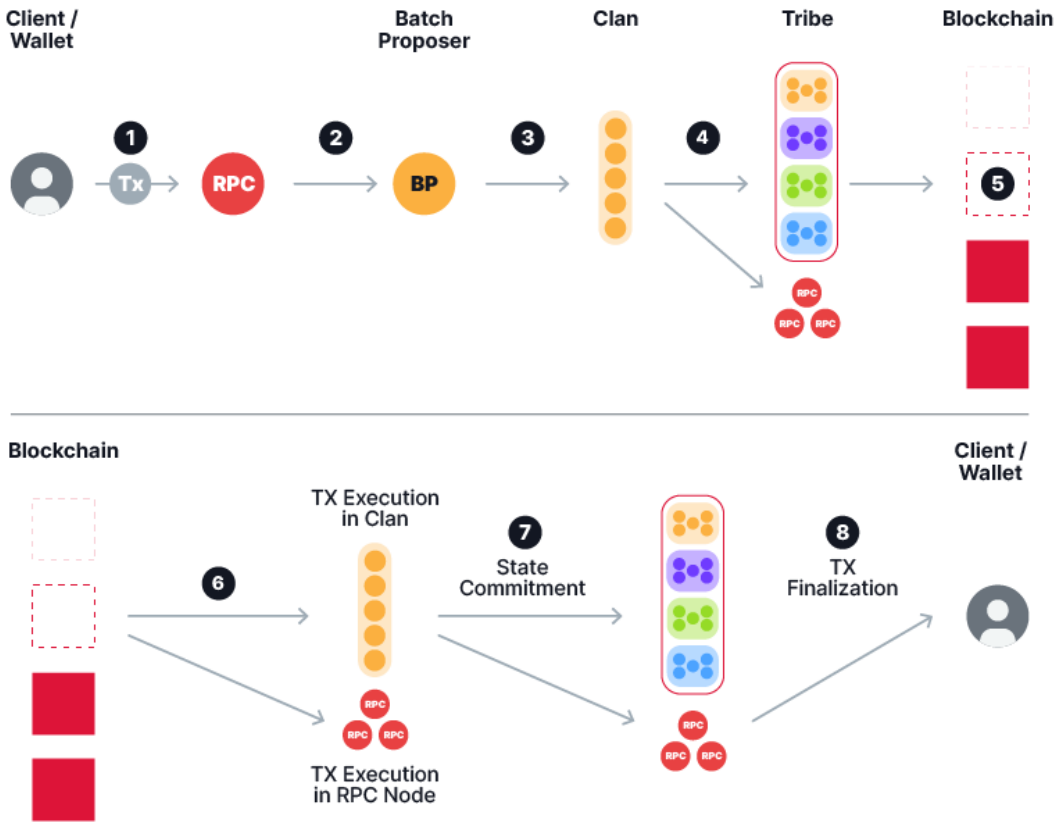

Figure 4 End-to-End Transaction Workflow

At a high level, the transaction flow is as follows. We describe the steps in detail in later sections.

1. A user submits a transaction t via Supra's Starkey wallet [3] or a dApp. The wallet is connected to a *gateway* RPC node, to which t is sent.

2. The gateway RPC node sends t to a specific batch proposer's primary bucket based on a basic validation and classification of t by the RPC node. The RPC node also sends t to the secondary buckets of some other batch proposers.
3. The batch proposers construct batches by packing transactions from their primary buckets as well as the timed-out transactions from their secondary buckets. More details are given in Section 5.
4. The batches are disseminated to the respective execute clan nodes using xRBC protocol and must form a data availability quorum certificate (DAQC) before the batch is included in a block. These certificates are disseminated to the entire tribe. The batches are also disseminated to the gateway RPC nodes.
5. The tribe nodes run Supra's Moonshot [12] consensus protocol. The block proposers construct blocks by packing these DAQCs and any required metadata of these batches.
6. The consensus protocol orders the blocks, thus indirectly ordering the transactions across batches. When a block is finalized it is seen by all the nodes of all the clans as they are all part of the same tribe running the consensus protocol. The finalized blocks are also disseminated to the gateway RPC nodes.
7. The clan nodes execute transactions from their respective batches from the finalized block next to its current tip of the blockchain. They then compute the post-state, Merkelize it, and compute the new Merkle root. Note that different clans execute different batches in parallel. The clan nodes sign this Merkle root and disseminate it to the entire tribe, as well as to the gateway RPC nodes.
8. The gateway RPC nodes execute the block, compute the post-state and its Merkle root, validate it against the received signed Merkle root, and notify the wallet once the transaction is complete.

Stages of Finality

In our workflow, a transaction t progresses through three distinct stages of finality, each providing stronger guarantees about its status on the blockchain:

Pre-confirmation When a batch b containing transaction t obtains its Data Availability Quorum Certificate (DAQC), this guarantees that t will be included in the blockchain. At this stage, while inclusion is guaranteed, the transaction's position and execution results are not yet determined.

Ordering Finality This stage is reached when the consensus protocol finalizes a block containing the DAQC of batch b. At this point, the position of transaction t in the blockchain becomes fixed and immutable, determining its execution order relative to other transactions.

Execution Finality This represents the ultimate stage of finality. Here, the clan nodes execute batch b, update the blockchain state with t's execution results, create a Merkelized version of the new state, sign the resulting Merkle root, and broadcast it to form the *state commitment*. Once this stage is complete, the transaction's execution results are final and irreversible.

Supra Layer 1 supports fast finality times. The observed times for these different stages of finality in our experiments are reported in Section 7.

The next sections expand on the important steps of the aforementioned transaction workflow.

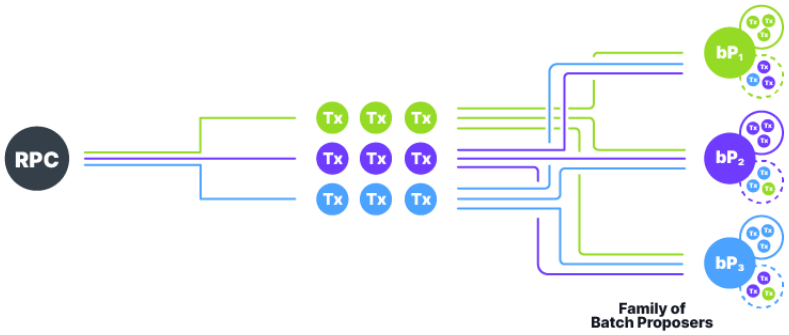

Figure 5 Mempool-less Bucketing Scheme

5 Mempool-less Bucketing Scheme

We now discuss the existing mempool protocols of Ethereum and Solana, and then we show how Supra improves over both.

Ethereum's mempool The traditional Ethereum-style *mempool* protocol is as follows. The client (wallet) sends a transaction to an RPC node which then gossips that transaction so that it reaches all the consensus validators. When a validator becomes a block proposer it tries to pack the transactions that have been received. Block proposers could attempt to censor a transaction by not including it in a block. However because transactions are available with all the consensus validators, and when another validator becomes a block proposer, missed or censored transactions will be included, and hence this protocol is censorship-resistant.

In a typical architecture like in Ethereum we have the clients (Web 3 wallets) sending transactions to an advertised RPC node as the first step. Then that RPC node gossips the transaction to the network of RPC nodes as well as the consensus validators using a peer to peer gossip protocol. The RPC nodes and the validators maintain a data structure called a "mempool" housing such transactions sent by the clients. These transactions are then picked up by a *Block builder* and packed in a block and processed by a consensus protocol. When the block achieves finality, then the transactions inside that block are also deemed final.

Given that the number of unconsumed transactions in the mempools of Bitcoin and Ethereum are high, the transactions do spend a considerable amount of time in the mempool on an average before getting picked up by a block builder. Also there is an added factor of latency due to peer to peer gossiping of transactions.

Solana's mempool-less approach Motivated to minimise this mempool delays, Solana uses a *mempool-less* protocol, which works as follows at a high level. The block production happens at a fixed schedule and the block proposers are known in advance. The client (wallet) sends a transaction to an RPC node. Then the RPC node packs auxiliary information such as the read-write-access information to the transaction and sends it to a few of the next block proposers, so that if one misses a transaction others will include it in a later block. It is important to note that Solana requires 32 block confirmations to achieve full finality, and despite its design, transaction duplication issues still occur occasionally, leading to network outages [1]. Compared to Ethereum's mempool protocol, here is a targeted communication to expected block proposers rather than gossiping, hence optimizing the end-to-end latency.

Supra's approach Like Solana, Supra implements a mempool-less protocol that uses targeted communication to validators rather than network-wide gossip. However, our

approach differs in two key aspects. First, unlike Solana's time-slot based system, we use a classical Byzantine Fault Tolerant (BFT) PBFT [9] style consensus protocol called Moonshot (detailed in Section 7). Here, blocks achieve instant finality upon receiving a *commit* Quorum Certificate (QC), rather than requiring Solana's 32 block confirmations. This results in significantly faster transaction finality. Second, our consensus blocks do not directly contain transactions, but instead contain DAQCs and metadata for batches of transactions (detailed in Section 6). In our mempool-less protocol, transactions from client wallets are placed into *buckets* maintained by designated batch proposers, as illustrated in Figure 5 and as described below. The identities of these batch proposers are made known to RPC nodes in advance for each cycle (see Section 9). Our scheme works as follows:

Mempool-less Bucketing Scheme

- Each batch proposer maintains two buckets: a primary bucket and a secondary bucket.
- For each transaction t, we select a family of batch proposers based on a unique, unpredictable identifier (hash) of t[1].
- Within this family, one node is designated as the primary batch proposer for t, while the remaining nodes serve as secondary batch proposers.
- Upon receiving a transaction t from a wallet or dApp front-end, the RPC node forwards t to its associated family of batch proposers, determined by t's identifier.
- The primary batch proposer stores t in its primary bucket, while secondary batch proposers store t in their respective secondary buckets.
- When constructing a new batch, batch proposers must include all transactions from their primary bucket.
- To protect against Byzantine batch proposers who may attempt to censor transactions by withholding them from their primary bucket, we employ secondary batch proposers as a fallback mechanism.
- Once a batch is finalized and observed by a batch proposer, that proposer removes the batch's transactions from both its primary and secondary buckets.
- Each transaction t has a configurable *timeout* period. If t times out without appearing in a finalized batch, secondary batch proposers in t's family must include it from their secondary buckets when constructing their next batch.

De-duplication Our design assigns exactly one primary batch proposer per transaction, who bears the responsibility for including that transaction in a batch. This single-responsibility model naturally prevents duplicate transactions from entering the blockchain.

Redundancy and Censorship Resistance To guarantee transaction inclusion, we employ multiple secondary batch proposers as a backup mechanism. If the primary batch proposer fails to include a transaction, secondary batch proposers will include it after a timeout period. While this may result in duplicate transactions when multiple secondary proposers act simultaneously, our system maintains consistency by executing only the first occurrence of each transaction. Any subsequent duplicates are automatically aborted during execution due to invalid transaction sequence numbers.

[1] This family is selected in such a way that the probability of having at least one correct node in this family is extremely high. In other words, the probability of having a family with no correct nodes is practically zero.

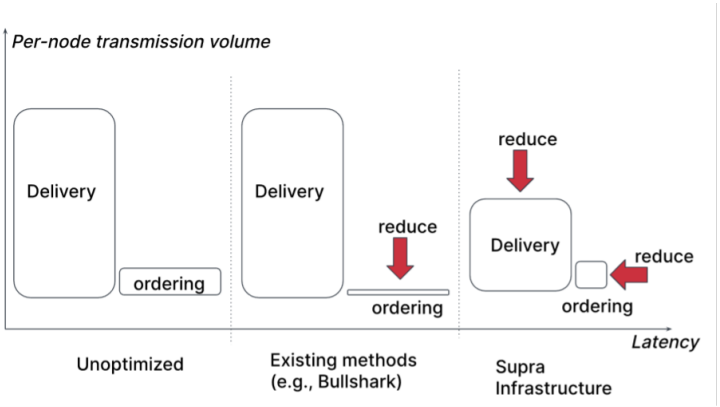

Figure 6 Conceptual Comparison with previous solutions

This dual-layer approach strikes an **elegant balance** between efficient de-duplication and robust censorship resistance. Furthermore, when a secondary batch proposer successfully includes a transaction, it creates an immutable record of the primary proposer's failure to act. This evidence can be used for subsequent analysis and enforcement of penalties against batch proposers who engage in transaction censorship or unnecessary delays.

6 Transaction Data Dissemination

We now describe how our tribe-clan architecture enables efficient decoupling of transaction data dissemination from transaction ordering.

Several recent protocols, notably Narwhal/Tusk [11], Bullshark [20] recognize the importance of transaction data delivery, and follow a new design paradigm of decoupling delivery and ordering, i.e., fulfilling these two objectives through separate subprotocols that are run concurrently. Note that without the ordering requirement, data delivery reduces to the reliable broadcasting (RBC) primitive, which has been studied extensively in the distributed systems literature. In particular, nodes keep proposing and disseminating new blocks via an RBC subprotocol; these blocks are later finalized and output by an ordering subprotocol, which operates concurrently to transaction data delivery. As a result, these novel designs lead to better utilization of bandwidth resources, and, thus, improved system throughput compared to previous approaches. Supra too follows the same *decoupling delivering and ordering principle*. However, we observe that even in the state-of-the-art consensus protocols, the focus still largely lies in optimizing the ordering subprotocol, which is counter-intuitive. For instance, a major selling point of Tusk [11] and Bullshark [20] is that they perform ordering in a zero-message fashion, i.e., they clamp down the transmission overhead of ordering all the way to zero. While this is remarkable, as explained before, it is the overhead of data delivery that dominates the total communication cost, not ordering.

Figure 6 compares our approach with previous solutions on a conceptual level. As shown in the figure, we aim to reduce the per-node transmission volume (i.e., the maximum amount of transmissions in any node) in the data delivery process. In practice, nodes' network bandwidth is usually a major limiting factor for system throughput.

Specifically, we reduce the communication cost of data delivery by exploiting the fact that there is a separate ordering service, run by the same set of nodes. In previous methods, data delivery is performed through reliable broadcasting (RBC). RBC requires a supermajority

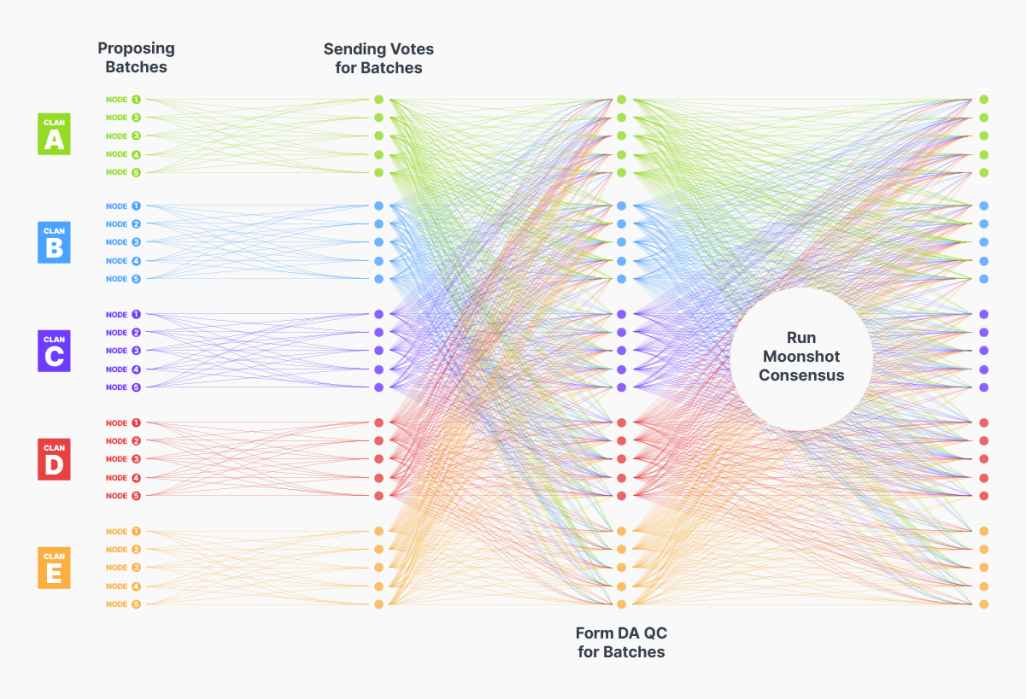

Figure 7 xRBC protocol

(i.e., more than 2/3) of honest nodes in the system, in order to defend against sender equivocation, i.e., the broadcaster sends different data batches to different nodes. Leveraging our core insight, we observe that with the help of a separate ordering service, data delivery can be instead performed through a relaxed RBC primitive, called xRBC, which can be done in a setup with a simple majority (i.e., more than 1/2) of honest nodes. In particular, unlike the case of traditional RBC, in xRBC the participating nodes alone cannot prevent sender equivocation. Instead, the protocol utilizes an ordering service to certify batches broadcast by the sender, eliminating disagreement among honest nodes. Note that the ordering service still requires an honest supermajority. Nevertheless, there are performance advantages of using xRBC compared to RBC.

Since transactions are executed only within clans (randomized sub-committees of tribes), transaction data initially needs to be disseminated only to nodes within the executing clan. This significantly reduces network bandwidth load compared to disseminating data to the entire tribe. Within each execution clan, we designate certain nodes as *batch proposers*. These batch proposers are responsible for constructing transaction batches from their *buckets* and disseminating them to all nodes in their clan. Importantly, batch proposers across different clans continuously and independently disseminate transaction batches in parallel, without synchronizing or waiting for protocol steps. This parallel, asynchronous dissemination maximizes bandwidth utilization efficiency when pushing data to nodes.

We may later disseminate the transaction data to the entire tribe for the purpose of state synchronization so that the reshuffling of nodes across cycles (see Section 9) happens smoothly.

The steps are shown in Figure 7 and is described as follows:

1. Batch proposer constructs a batch and sends it to all the clan nodes.

2. Upon receiving a valid batch, clan nodes send vote on that batch's data availability to all the clan nodes, as well as to the rest of the nodes in the tribe, and designated gateway RPC nodes.
3. The aggregation of a simple majority of the votes results in a *data availability certificate (DAQC)* of a batch that guarantees that at least one honest node in the clan has the full transaction data. If a node misses any part of the batch data it may pull that data by making a request to at most a simple majority of the nodes in the clan. The DAQC guarantees that such a request will always be fulfilled. All the nodes in the tribe form these DAQCs independently.

For batches exceeding a size threshold, and in cases where not all clan nodes serve as batch proposers, we employ erasure coding to ensure uniform bandwidth utilization across all clan nodes. This approach helps distribute the network load evenly throughout the clan.

Blocks contain batch certificates. When a consensus protocol validator (a tribe node) becomes a *block proposer* it constructs a block packing in all the DAQCs of the batches it has received. When a block achieves finality and is seen by a validator then that validator drops all the DAQCs from its *cache* as these DAQCs have already found a place in a block that is finalized and is not required anymore. When a block proposer is proposing a block then it optimistically skips the DAQCs (but does not delete from its cache) that are seen so far in previous proposals even though the previous blocks may not have been finalized. Because the blocks do not contain transactions and contain only the certificates of batches, the Supra blockchain is actually a very thin blockchain.

7 Consensus Protocol

A Byzantine fault tolerant (BFT) consensus protocol is the heart of a blockchain. It provides a canonical ordering of blocks and hence a canonical order for transactions inside blocks. We have innovated a novel BFT consensus protocol called Moonshot SMR [12] inspired by the classical Practical Byzantine Fault Tolerance (PBFT) [9] protocol and tuned towards maximal performance.

As mentioned before, because the consensus protocol is executed in a tribe but transactions are executed only inside clans, we have an elastic design to accommodate the required demand for transaction throughput. Note that blocks do not contain transactions but only the certificates of batches.

Moonshot achieves an optimistic consecutive-proposal latency (the minimum latency between two block proposals) of 1 message delay (md) and a commit latency of 3 md. Batch dissemination and data availability certificate generation cumulatively add 2 md. Since blocks are proposed every network hop, data availability certificates must be queued until the next block proposal, introducing an average queuing latency of 0.5 md. Consequently, the overall end-to-end latency of our system is 5.5 md.

Formal verification Distributed protocols exhibit complex behaviors with unbounded state spaces, making it exceptionally challenging to prove their correctness. Formal verification stands as the gold standard for guaranteeing protocol safety, as it can mathematically prove the absence of errors. Recognizing this, we formally verified the safety property of our Moonshot consensus protocol that it never forks using Microsoft's IvY verifier [18], providing rigorous mathematical assurance of its correctness.

7.1 Experiments

We conducted extensive evaluations on the Google Cloud Platform (GCP), distributing nodes evenly across five distinct regions: us-east1-b (South Carolina), us-west1-a (Oregon), europe-north1-a (Hamina, Finland), asia-northeast1-a (Tokyo), and australia-southeast1-a (Sydney).

In our setup, a client is co-located with the consensus node. Each transaction consists of 512 random bytes, and the batch size is set to 500KB. Each experimental run lasts 180 seconds. For latency measurements, we calculated the average time from transaction creation to its commit by all non-faulty nodes to determine end-to-end latency. Throughput is assessed based on the number of finalized transactions per second.

We evaluated our architecture with a network of 300 nodes, partitioned into 5 clans, each comprising 60 nodes (12 from each GCP region). In this configuration, clans of size 60 in a 300-node network have a failure probability of 0.0107 concerning the risk of a dishonest majority within a clan. Nonetheless, our objective was to demonstrate that our architecture can maintain high throughput and low latency even at larger system sizes.

For this experimental evaluation, we used e2-standard-32 machines, each equipped with 32 vCPUs, 128 GB of memory, and up to 16 Gbps of egress bandwidth.

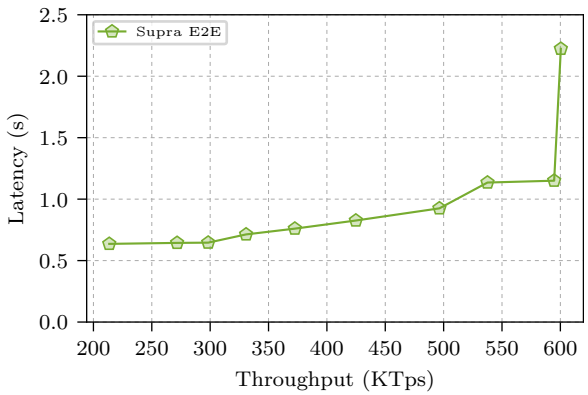

Figure 8 Throughput vs. End-to-end Latency

We observed throughput and end-to-end latency, as shown in Figure 8. We achieved a sub-second end-to-end latency with a throughput of 500 KTps. Additionally, we reached a throughput of 300 KTps with a latency of approximately 650 ms, closely aligning with the theoretical limits of our architecture. We also observed that the DAQC formation takes around 160 ms. Hence we have transaction **pre-confirmation** to be in the order of 160 ms, and the **ordering finality** is under a second.

7.2 DAG Consensus

Inspired by the literature on the DAG-based consensus protocols we have invented a novel DAG-based consensus protocol called Sailfish [19] that betters the commit latency of the current state-of-the-art DAG-based protocols without compromising on the throughput by achieving a commit latency of 1 reliable broadcast + 1 md. We have also created a variant of Sailfish that leverages our tribe clan architecture to achieve a higher throughput. We are conducting extensive experiments on this design and if found to outperform the Moonshot results on large network sizes then we plan to switch our core consensus protocol to Sailfish.

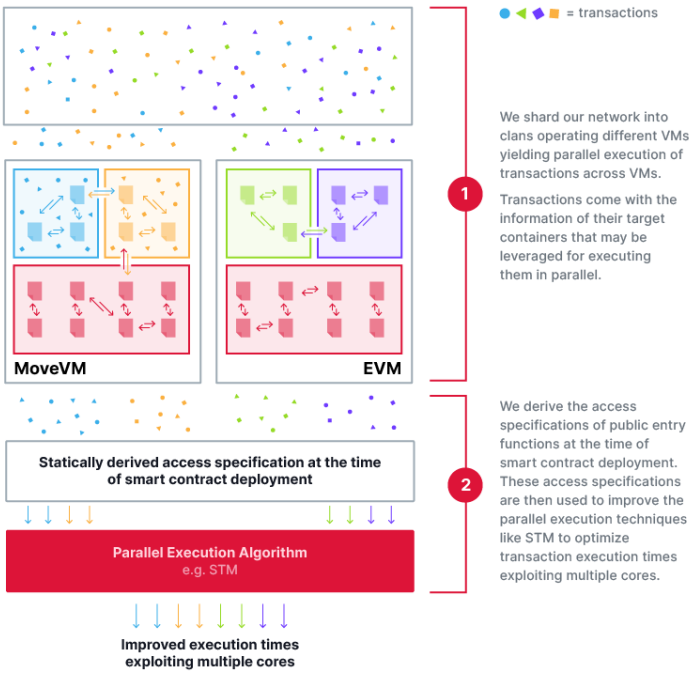

Figure 9 Supra's parallel execution approaches

8 Execution

The current state of a blockchain holds the information of all the assets and their ownership, and more generally the latest data of all the smart contracts. Execution of a finalized block of transactions entails loading the appropriate part of the state, modifying the state as per the semantics of the transactions in the order mandated by that block, and then persisting the modified state to the storage. With the modern blockchains which can process a high throughput of transactions, execution time is no longer a negligible factor in the end to end latency of a transaction finalization. The trend of parallelizing execution by Solana, Sui, Aptos, Monad etc is a testimony to this factor.

Our tribe-clan architecture already facilitates parallel execution of transactions at the network level because the execution happens at the clan level, and different clans execute different batches of transactions as shown in Figure 9.

8.1 Executing Transactions in Parallel

Beyond network-level parallelism, we have extensively researched efficient techniques to execute transactions in parallel by leveraging multi-core processors within individual nodes. The approaches considered in the literature and employed in our industry may be classified into two kinds: optimistic/speculative execution techniques, and deterministic techniques that pre-determine the dependencies.

Software Transactional Memory (STM) STM techniques are well known in the traditional space for executing programs utilizing multiple cores. Aptos' BlockSTM [14] adapted this to the context of executing blockchain transactions. The core idea is to

optimistically execute transactions in parallel on any available core and then to validate if the execution of a transaction resulted in a conflict owing to a dependency with respect to another transaction. If so, such conflicting transactions are re-executed again using a collaborative scheduler. We have innovated a novel STM-based scheduler-less parallelized execution algorithm that is comparable to Aptos' BlockSTM. We are performing extensive evaluations on this design.

Access specification based parallelization Some blockchains employ various approaches leveraging the specified transaction access patterns. Solana's Sealevel requires transactions to explicitly declare which accounts they will read and write. Similarly, Sui also requires that the transaction specifies the exact read and write objects (succinctly via object identifiers) as well as the information if the specified objects are exclusively owned or shared. In Polygon's approach, a block proposer computes the dependency information for a given block by executing it using BlockSTM technique and then includes this information within this block so that other validators may read this additional information and execute independent transactions in parallel. It is important to note that including explicit access specifications within transactions or dependency information in blocks can significantly increase block size. This increased data overhead leads to faster bandwidth saturation, ultimately constraining the system's maximum achievable throughput.

Drawing inspiration from these approaches, we are developing a deterministic parallel execution algorithm that leverages transaction access specifications. Our novel approach differs by not requiring RPC nodes or wallets to provide these specifications. Instead, we derive them through static analysis of smart contracts during deployment and store them in the blockchain state. These access specifications enable us to transform the linear transaction order from finalized blocks into a partial (relaxed) order, allowing us to execute independent transactions in parallel.

Hybrid approach We believe a hybrid approach that improves an optimistic technique like an STM algorithm by utilizing these statically derived access specifications is the end game of optimized execution techniques (shown in Figure 9). We are currently conducting comprehensive evaluations of this design.

8.2 Execution Order Fairness

To ensure fair transaction ordering, we employ a two-step process. First, the consensus protocol provides an initial randomized seed ordering of transactions. We then use this seed to derive a final deterministic ordering through our in-house *beacon*-based on-chain randomness protocol [15] leveraging BLS threshold signature on the blocks by the Supra tribe. This derived order determines the actual execution sequence of transactions. By introducing this additional randomization layer, we effectively mitigate Maximum Extractable Value (MEV) attacks, as neither batch proposers nor block proposers can predict or influence the ultimate randomized execution order of transactions. To mitigate spamming, we posit localized fee markets can make it exponentially more expensive to transact on contentious states.

8.3 Multi-VM

Ethereum introduced EVM [21], a Turing complete on-chain programming language that unlocked the creativity of dApp development and among many things enabled Decentralized Finance applications. It adopted Solidity from the paradigm of Web 2.0 programming

languages. However, the blockchain community eventually realized the need for a programming language that is tailored for asset transfers and on-chain asset management. The language Move [8] was designed by the Diem group of Meta precisely for such a reason. Soon the variants of Move emerged – Aptos Move, Sui Move, etc. Supra recognizes the value of having a programming language tailor-made for on-chain transactions and is adopting Move as its first-choice smart contract platform. We are launching our Layer 1 blockchain with the Move smart contract platform.

However, we recognize the value of multiple virtual machines (VMs) and believe that, for a thriving blockchain ecosystem and a vibrant community, developers working across different VMs can complement each other. Together, they drive progress and advance the industry as a whole. Hence we are designing an efficient Multi-VM architecture. Because our tribe clan architecture naturally yields to a sharding design where each clan hosts a state shard, the natural direction for us is to host different VMs as shards on different clans. Following our Move smart contract platform support, we plan to integrate the Ethereum Virtual Machine (EVM), enabling compatibility with the vast Ethereum ecosystem. Our next phase involves integrating the Solana Virtual Machine (SVM), which enables smart contract development in widely-used languages like Rust, C, and C++. This integration significantly expands our platform's reach to developers from traditional programming backgrounds. Following this, we plan to incorporate CosmWasm support, allowing us to seamlessly connect with the vibrant Cosmos ecosystem.

Cross-VM communication. Typically users could have accounts on multiple VMs and would want to transfer assets from an account on one VM to another account on another VM. Also, Supra's native token, $\$SUPRA$, has to be managed integrally across all the VMs. This is analogous to the Bridge problem, where a simple 3 step workflow solves such cross-VM asset transfers:

1. A user submits the cross-VM asset transfer transaction t. Such transfers are split into two parts: a debit transaction t_d on the source VM and a credit transaction t_c on the destination VM. So, the transaction t itself is considered as t_d and is executed on the source VM, and an appropriate event is emitted.
2. We randomly pre-select a family (requires at least one honest node) of nodes from the source VM clan and designate them to be watching for the debit event upon which t_c is submitted by them.
3. The t_c is executed by the destination VM clan crediting the asset and completing the transfer.

9 Epochs and Cycles

In Supra's architecture, new nodes may be added to the tribe and some of the existing nodes can be removed from the tribe only at the *epoch* boundaries. These epoch durations are configurable and are typically set in terms of the number of blocks or wallclock time. We plan for epoch duration in the order of a week. All the nodes of the tribe shall be running the *Ordering* service by executing a consensus protocol.

As shown in Figure 10, an epoch is further split into cycles typically on the order of 1 day. The nodes of the tribe are randomly shuffled (using Supra's VRF) to clans every cycle. For example, in epoch 1 cycle 2 (e1c2) a node may be performing a DORA validator role, and the same node in e1c3 may be performing EVM execution of transactions. This random shuffling of roles every cycle boosts the security of the system against an attacker who wishes

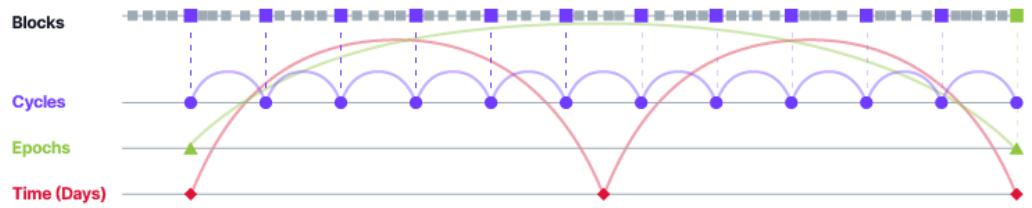

Figure 10 Epochs and Cycles

to attack a specific service, say S, by targeting to bring down some nodes of a clan offering S.

This reshuffling requires running a Distributed Key Generation (DKG) protocol every cycle. We can afford to do such reshuffling only because of our innovations in the space of DKG by developing a highly performant class-group-based DKG protocol [17].

Chances of a bad clan As discussed in Section 3, let us analyze the security implications for a tribe of 625 nodes divided into 5 clans of 125 nodes each. The probability of forming a *bad* clan – where Byzantine nodes constitute a simple majority – is approximately 35 in one million random draws. With clan reshuffling happening once a day (suppose cycle is of a day's length), this translates to an expected occurrence of one bad clan every 78 years only in the extreme scenario where 33% of the entire tribe is compromised by Byzantine actors the entire period, which is practically negligible. This demonstrates the robust security guarantees provided by our clan formation mechanism.

State Synchronization

The shuffling of nodes into different roles requires state synchronization. A node has to synchronize the state of a service, say S, before the beginning of the cycle enlisting this node for S. A state synchronization protocol is also a necessity when a node goes offline and comes back online to resume the services the node is enlisted in the latest cycle.

There are two options. Firstly, the simple option, where all the tribe nodes maintain the state of all the clans (or shards). But only the clan that is primarily responsible for that state signs the execution commitment. There is no separate synchronization protocol necessary here, as the nodes are ready to take on any role in any cycle. The second option is to make the mapping of nodes to clans available at least one cycle in advance so that the nodes synchronize a *base* state from an RPC node and then try and execute the transactions in the blocks of the previous cycle so that by the start of the targeted cycle the node is prepared to take on the role. This is similar to Ethereum where Ethereum Sync committee validators are made known in one Sync committee period in advance.

Note that many of our Oracle services, such as DORA and dVRF, are stateless protocols. When nodes are randomly assigned to these services, they don't need to synchronize their full state before beginning execution.

10 Containers

One of the key features appealing to the developer community is Supra Containers. Supra Containers are inspired by the appchains of today.

We recognize the merits of appchains, especially as a provider of sovereignty to the dApp creators and the associated community. Moreover, they facilitate an application-based business model. Layer 2s, sidechains, Polkadot Parachains, Cosmos Zones, and Avalanche Subnets are the state-of-the-art solutions for appchains. We realize that these solutions are not optimal, especially in the context of a high throughput blockchain like Supra Layer 1. Appchains provide gated deployment of smart contracts, custom gas tokens, and custom gas pricing. When these appchains are hosted on secondary chains such as Parachains, Zones, Subnets, etc, their gas prices do not compete and do not vary with respect to the gas price of the corresponding primary chain such as respectively, Polkadot Relay chain, Cosmos Hub, and Avalanche C-Chain. With such a localized fee market, they provide a desirable predictable transaction fee experience to their users.

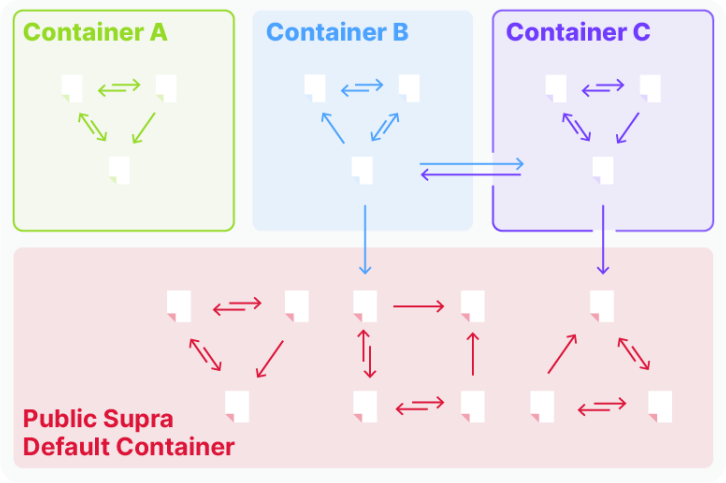

Figure 11 Containers

Supra Containers [2] are a new concept in the blockchain space. We define them to be a group or a bundle of smart contracts servicing a single or a group of related dApps. We modify the runtime framework of Supra to support and offer the same appchain experience to dApp creators with custom gas tokens, custom gas pricing, and gated deployment of smart contracts, through these Supra Containers, at an inexpensive price, completely bypassing the hassle of bootstrapping up a whole new secondary network of incentivised validators with stakes. More importantly, we overcome a significant problem of appchains, which is liquidity fragmentation by leveraging the atomic composability of cross-container method calls.

Facilitates Parallel Execution Because in the Ethereum Virtual Machine (EVM) model each smart contract comes with its own exclusive storage, Supra Containers applied to EVM result in a partition of the blockchain state. This is a boon especially to leverage towards optimizing the execution times of intra-container transactions via parallel execution. The insight here is that an intra-container transaction targeting one container does not access the state of any other container. We leverage this information to build a simple work-sharing queue for parallel execution optimization.

11 Closing Words

Supra's ambitious vision as a vertically integrated IntraLayer has been taking shape for many years now through rigorous vetted research behind the scenes. We have published many research papers at top academic conferences such as: IEEE Security & Privacy, Network and Distributed Security Symposium, ACM Computer and Communications Security, etc, and have also released many whitepapers of individual components – Moonshot and Sailfish consensus, Distributed Oracle Agreement, Distributed Verifiable Random Functions, efficient class group DKG, Supra Containers, HyperLoop, HyperNova, etc. For the first time, Supra is presenting our comprehensive vision for a vertically integrated blockchain infrastructure stack, bringing together multiple novel protocols developed over the years into an all-in-one extreme-performance blockchain.

References

1 Solana outage due to duplicate transactions. `https://solana.com/news/02-25-23-solana-mainnet-beta-outage-report`.

2 Supra containers (app-chains) and their benefits towards parallel execution. `https://supra.com/documents/Supra-Containers-Whitepaper.pdf`.

3 Supra's starkey wallet. `https://starkey.app/`.

4 Hyperloop: Rationally secure cross-chain bridge. `https://supra.com/docs/Supra-Hyperloop-Whitepaper.pdf`, 2023.

5 Hypernova: Efficient, trustless cross-chain solution. `https://supra.com/docs/Supra-HyperNova-Whitepaper.pdf`, 2023.

6 Supra as a defi intralayer. , 2024.

7 Arman Abgaryan and Utkarsh Sharma. Dynamic function market maker. *arXiv preprint arXiv:2307.13624*, 2023.

8 Sam Blackshear, John Mitchell, Todd Nowacki, and Shaz Qadeer. The move borrow checker, 2022.

9 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

10 Prasanth Chakka, Saurabh Joshi, Aniket Kate, Joshua Tobkin, and David Yang. Oracle agreement: From an honest super majority to simple majority. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 714–725, 2023.

11 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 34–50, New York, NY, USA, 2022. Association for Computing Machinery.

12 I. Doidge, R. Ramesh, N. Shrestha, and J. Tobkin. Moonshot: Optimizing block period and commit latency in chain-based rotating leader bft. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 470–482, Los Alamitos, CA, USA, jun 2024. IEEE Computer Society.

13 Sanjam Garg, Aniket Kate, Pratyay Mukherjee, Rohit Sinha, and Sriram Sridhar. Insta-pok3r: Real-time poker on blockchain. Cryptology ePrint Archive, Paper 2024/1061, 2024.

14 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, page 232–244, New York, NY, USA, 2023. Association for Computing Machinery.

15 Jacob Gorman, Lucjan Hanzlik, Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Pratik Sarkar, and Sri AravindaKrishnan Thyagarajan. VRaaS: Verifiable randomness as a service on blockchains. Cryptology ePrint Archive, Paper 2024/957, 2024.

16 Aniket Kate, Easwar Vivek Mangipudi, Siva Maradana, and Pratyay Mukherjee. Flexirand: Output private (distributed) vrfs and application to blockchains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1776–1790, New York, NY, USA, 2023. Association for Computing Machinery.

17 Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive vss using class groups and application to dkg. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2024.

18 M. Praveen, Raghavendra Ramesh, and Isaac Doidge. Formally Verifying the Safety of Pipelined Moonshot Consensus Protocol. In Bruno Bernardo and Diego Marmsoler, editors, *5th International Workshop on Formal Methods for Blockchains (FMBC 2024)*, volume 118 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

19 Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of dag-based bft. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025 (To appear).

20 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2705–2718, New York, NY, USA, 2022. Association for Computing Machinery.

21 Gavin Wood. Ethereum: A secure decentralized generalised transaction ledger. `https://ethereum.github.io/yellowpaper/paper.pdf`, 2014.

22 Jianting Zhang, Zhongtang Luo, Raghavendra Ramesh, and Aniket Kate. Optimal sharding for scalable blockchains with deconstructed smr, 2024.