Introduction

We introduced the motivation, design principles and high-level requirements of Espresso Shared Decentralized Sequencer in an earlier note. In this series of notes, we will dive deeper into understanding the design of the two key components of our sequencer network: HotShot Consensus and the Tiramisu Data Availability layer. We will cover this in three sections:

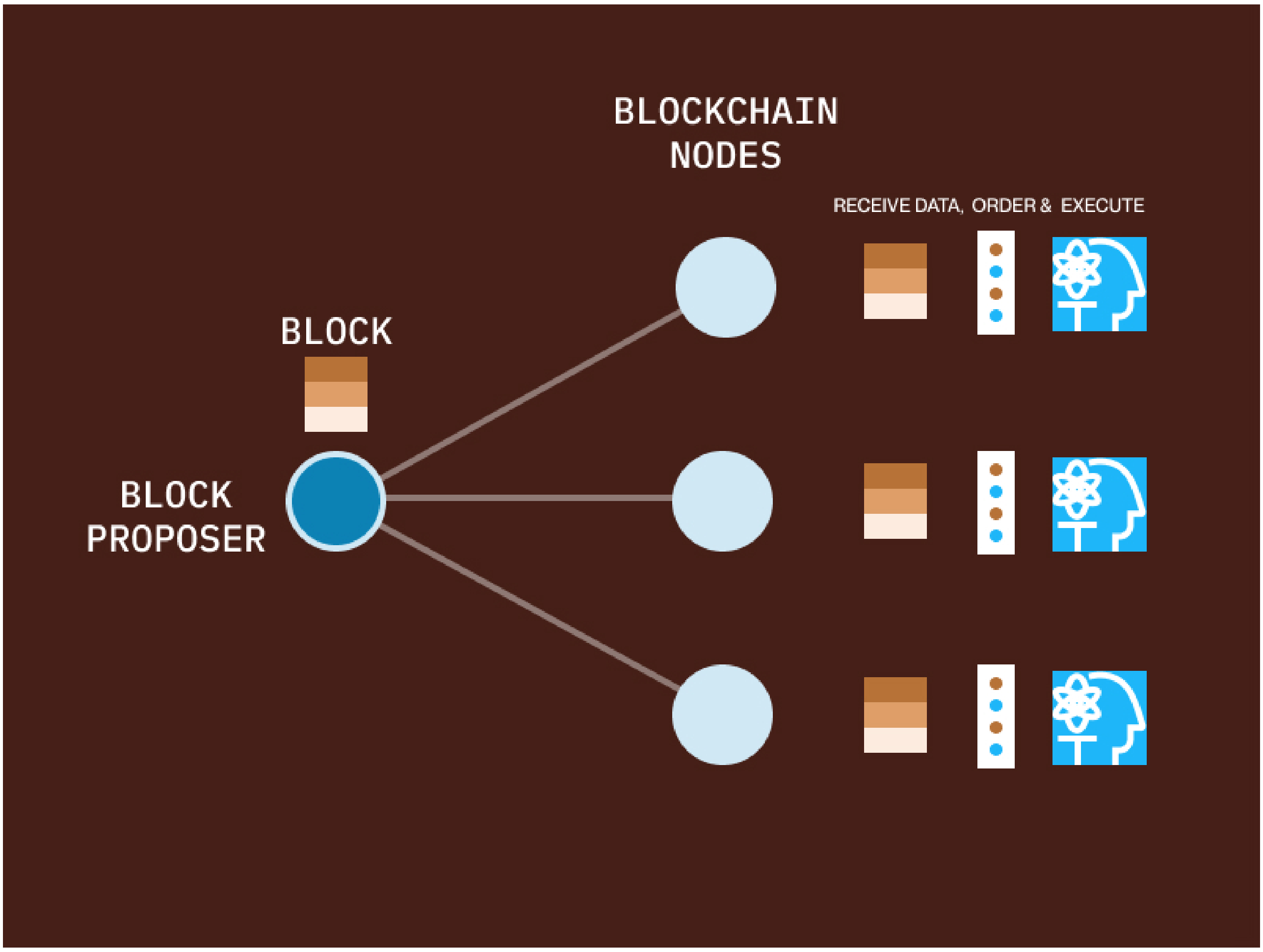
- Part I: Understanding the constraints of a sequencer network, and how it is different from a state machine replication (SMR) system
- Part II: HotShot Consensus
- Part III: Tiramisu Data Availability

Part I: Sequencers versus State Machine Replication systems

A Byzantine fault tolerant state machine replication (BFT SMR) system requires a set of nodes to agree on a sequence of transactions and then execute these transactions such that the system functions even if some of the nodes are Byzantine faulty. This typically falls into satisfying three separate requirements:

- Consensus: ensuring an agreement on the ordering of these transactions among all non-faulty nodes
- Data broadcast: ensuring that the transaction data is broadcast to every non-faulty node
- Execution: ensuring that every non-faulty node executes the transactions and updates the state machine

Each of these are separate requirements and from a performance standpoint, they are not necessarily at odds with one another. The overall performance of an SMR system is thus dictated by the slowest of them.



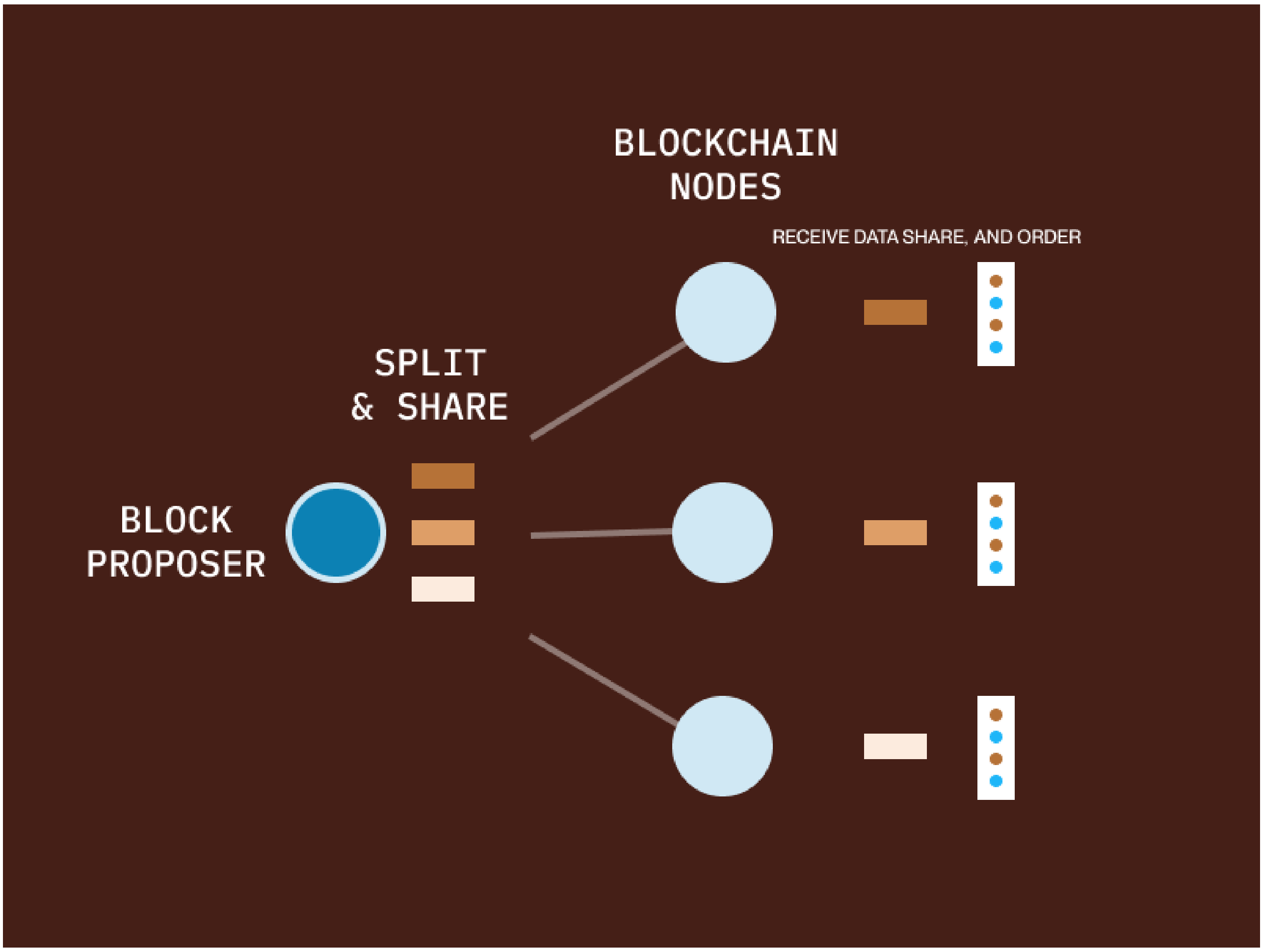
Traditional SMR system (Blockchain). All nodes receive, order and execute all transactions.

A shared decentralized sequencer, on the other hand, does not need to satisfy all of these requirements. In a typical workflow, users submit transactions to the rollup which are eventually queued in the mempool. The sequencer network retrieves these transactions and outputs a finalized ordered list of transactions. Subsequent to this ordering, the rollups themselves are responsible for executing transactions (and they also submit state updates to be recorded on layer 1). Thus, the sequencer network is not responsible for executing, or even validating, transactions. Consequently, while ensuring transaction data availability, to a rollup, for example, is a requirement, the transaction data need not be broadcast to the nodes of the sequencer network themselves. In summary, we have the following requirements for a sequencer network:

- Consensus: ensuring an agreement on the ordering of these transactions among all non-faulty nodes
- Data availability: ensuring availability of data to those who need it (e.g., rollups)
- Execution: no execution needed

Thus, while a decentralized sequencer network can be implemented using an SMR protocol, we have the opportunity to develop a simpler and more efficient system. An ideal design for an SMR system may or may not be ideal for a sequencer network. In particular, while each of these three could have been bottlenecks in an SMR system, the relaxed data availability requirement and non-requirement of execution makes consensus an important bottleneck.

The Espresso Sequencer modularly separates data availability (DA) and consensus to enable better throughput and latency. The consensus layer is only responsible for agreeing upon a short commitment to the data block, whereas the availability of the full data is delegated to a DA layer running in parallel. In the next two sections, we present the design of HotShot, our solution to the consensus problem, and Tiramisu, our solution to the relaxed data availability problem.



Espresso Sequencer: In the sequencer, nodes only need to order transactions and only receive a small share of the data along with a guarantee that the rest of the data is available.

You can read more details about our sequencer design and HotShot in our academic paperhere.

Part II: HotShot Consensus

HotShot consensus is a core part of the Espresso Sequencer network. At a high level, the network accepts transactions from different rollups, notarizes and sequences them using the consensus nodes, and provides this ordering to the rollup for execution. At first glance, one may ask if such a network consensus achieves any goal. In particular, if rollups execute these transactions in the first place, they may order it themselves. In fact, that is precisely what happens today — rollups use their own (centralized) sequencers to order transactions. A decentralized consensus network provides two key advantages compared to this centralized world. First, decentralization can provide censorship resistance and prevent monopolization, e.g., through competition of block production. Second, decentralized consensus provides finality (or a notarization) on the order of transactions. Such a promise of non-equivocation is not possible with a centralized entity unless it is fully trusted.

Well, if we cannot entirely trust a centralized entity, how can we be ensured that a large fraction of consensus nodes in HotShot are not corrupt enough to equivocate or be bribed enough to censor transactions? Also, from an efficiency standpoint, centralized entities can be fast and provide instant finality (if trusted) — can a decentralized sequencer network achieve the same performance? We expand on these requirements in the following security and efficiency desiderata for the HotShot consensus.

HotShot Security Desiderata

- **Ethereum-level security.** As a baseline, HotShot should be at least as secure as the Ethereum network itself. In the absence of such a requirement, end users would obtain two different levels of guarantees — first, there would be a finality by HotShot and then Ethereum would provide a separate finality guarantee several minutes later. If the users need to wait for these several minutes for the eventual guarantee, that would be a missed opportunity for using a consensus network. We intend to achieve this goal by relying on the same set of nodes as Ethereum (e.g., through EigenLayer). In effect, Ethereum validators would subscribe to additional slashing rules relevant to HotShot. Thus, if any malicious behavior (e.g. double signing HotShot blocks) leads to a safety violation in HotShot, the validators would lose their stake. Consequently, HotShot provides an economic guarantee for its finality akin to Ethereum's. We also note that our ability to slash implies that the HotShot consensus protocol is accountable.

- **Bribery resistance.** Finality by Ethereum nodes provides security for accountable actions. What about other forms of behaviors that one cannot slash? e.g., censorship or a liveness violation. In a decentralized consensus network, an adversary can easily bribe participants in positions of power, e.g., leaders, to commit specific actions that are not desirable. For instance, an adversary can create a smart contract that bribes nodes to censor transactions. On the one hand, we want that an adversary should not be able to cause permanent performance concerns with a small bribing budget. Thus, this rules out small committee-based solutions such as Algorand. The reason is that an adversary could bribe one committee to permanently bias a beacon, or prevent data availability. On the other hand, observe that an adversary with an unlimited bribing budget can be too strong to be tackled. Thus, we focus on providing bribery resistance from an adversary with a large budget sufficient to bribe even a linear number (e.g., one-third) of the nodes. Our solution ensures that such a bribing adversary can only cause temporary loss of progress. This means, eventually the network will continue to make progress once an adversary runs out of money. More importantly this strongly disincentivizes bribery in the first place, as the potential effect of it is limited.

HotShot Efficiency Desiderata

Our goal is to approach the efficiency of centralized sequencers.

- Low-latency high-throughput protocol. From a performance standpoint, our goal is to complement Ethereum. Thus, while Ethereum achieves dynamic availability to support liveness even under sporadic participation, HotShot aims to obtain fast finality with low latency and high-throughput.

- Scalability to a large number of nodes. Since a security requirement is to use the same set of nodes as Ethereum, our goal is to be able to scale to tens of thousands of nodes.

Note, that while our security requirements are very strong, and are designed to prevent the worst case, e.g. a permanent break of the system, it suffices if the efficiency requirements are only achieved in the normal aka optimistic case, i.e., when the network is up and running and there is no active attack. This is motivated by both theory and practice, as attacks and network outages are usually temporary events and we can set incentives, e.g. slashing criteria and bribery resistance, to ensure that the adversarial gains are not enough to sustain a long term active attack.

Our solution: Responsive, communication-efficient consensus adapted to a PoS setting using a CDN at the network layer

Our solution aims to achieve all of the goals described earlier. HotShot, a variant of HotStuff, runs a simple consensus protocol among nodes in a view-by-view manner where a designated leader proposes the next block of transactions in each view. Each of the nodes verify whether the proposal is valid based on consensus invariants and then send a vote only to the next leader. Transactions in a block are committed when any two blocks from consecutive views and extending them receive a quorum of votes. The protocol obtains the following exciting features:

- Optimistic linear communication. Our consensus protocol only requires linear communication complexity in the optimistic case. This is important to our setting for a few reasons. First, to engage with the Ethereum validator set, we aim to scale to tens of thousands of nodes (In a quadratic protocol with 10k nodes, sending even just one signature per round requires 3.2GB of communication, vs. 320kb in a linear protocol). Second, our need to obtain finality with low latency implies we should not rely on P2P communication in the optimistic case (since in a P2P network, the latency is proportional to the network diameter). This requirement makes communication complexity a vital metric to optimize. Finally, given our relaxed data availability requirement, our DA solution (described in Part III) also incurs a much lower communication on the critical path compared to a typical SMR protocol. Thus, the consensus protocol incurs a larger share of the overall system communication; reducing communication complexity is an important step to improve the overall system.

- Optimistic responsiveness. Our protocol is optimistically responsive and commits at the network's speed without waiting for any pessimistic network delays. Thus, when the network is behaving well, it allows us to commit data blocks much faster and, in turn, helps improve the throughput too. This also motivates optimizing the average case, when the network is fully operational, as the protocol can take advantage of it. In contrast, without responsiveness, one must set parameters conservatively (like in Ethereum), which slows the protocol execution even during optimistic conditions. One such example of such an optimization is the CDN described below.

- Use of a CDN at the network layer. Obtaining low latency requires parties to communicate with each other faster than relying on a P2P network. However, if we have tens of thousands of consensus nodes, directly connecting to all of them and maintaining these connections is not efficient for all consensus nodes. We leverage a CDN at the network layer akin to what's used in Web2 architectures. A CDN is powerful hardware with high bandwidth connections that enable efficient information routing at low latency. Such an infrastructure allows us to scale much better even with a large number of nodes both from the standpoint of computation and communication. For communication, data can be disseminated to all consensus nodes faster. From a computation standpoint, computationally-heavy tasks such as signature aggregation can be offloaded to this hardware. Moreover, the optimistic responsiveness property of our protocol synergizes with this modification at the networking layer to achieve even better performance.

- Adapting to a PoS setting. HotShot adapts HotStuff from a permissioned setting to a proof-of-stake setting where the set of consensus nodes can change over time. Thus, we designate a set of views into epochs, and the set of nodes in an epoch remains fixed. As time progresses and the set of nodes changes, the protocol is reconfigured to a new set of consensus nodes. HotShot relies on the underlying L1 layer to track the stake of consensus nodes.

- Using Naor-Keider view synchronization. In HotStuff, liveness is guaranteed only if the consensus nodes are in the same view for a sufficient amount of time. HotStuff achieves this using a pacemaker module that suffers from quadratic communication complexity and non-optimal latency. In HotShot, we instead use the Naor-Keidar view synchronizer which achieves a linear communication complexity and constant latency in the optimistic case.

You can read more details about our sequencer design and HotShot in our academic paperhere.

Part III: Tiramisu: The Three-Layered Espresso DA

Data availability (DA) is a crucial security property of blockchain consensus (aka state machine replication) systems. It ensures that all honest users can gain access to the state of the system, even if there exists an adversary that tries to prevent this. Having access to the state is important for users to be able to create new transactions and utilize their assets within the system.

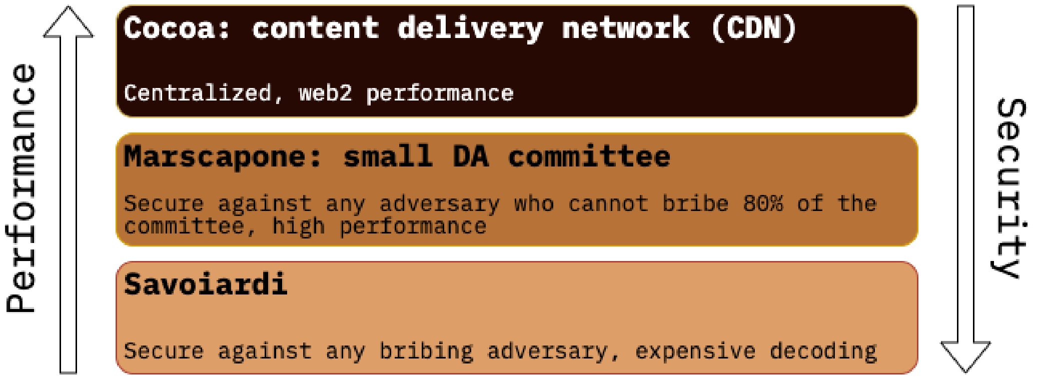
DA is an important property for all blockchain systems, but it has received significant attention in rollups. A rollup prover proves that an ordered (or sequenced) list of transactions leads to a certain state update. This proof usually is performed relative to short commitments, e.g. Merkle hashes, of the state. However, this proof by itself is insufficient as it does not guarantee data availability, i.e., users knowing the rollup's state. Many rollups rely on a layer 1 blockchain, e.g. Ethereum, to provide data availability. However, space on layer 1 blockchains is particularly scarce and expensive. Using the layer 1 chain for data availability creates additional congestion and results in higher fees. Additionally, as we will discuss in this post, Ethereum's consensus is not optimally designed to support fast and cheap data availability, mainly due to the fixed block time.

To solve this, we introduce Tiramisu, a highly efficient DA solution that is made up of three separate tiers of data availability. It provides super efficient, web2 level data availability in the optimistic case while giving strong Ethereum-level guarantees in any pessimistic case, e.g. when there are network outages or attacks. It is focused not just on providing data availability as a fallback but also on efficient access to data, even under malicious behavior. Tiramisu is fully integrated with HotShot consensus and is run by HotShot nodes, so blocks available through Tiramisu DA benefit from HotShot's optimistic responsiveness. Rollups using the Espresso Sequencer can additionally choose to use other DA availability systems to provide a different set of tradeoffs.

Savoiardi: a bribery-resilient DA (akin to Danksharding)

Savoiardi, the base layer of Tiramisu, is a bribery resilient DA similar toEthereum's danksharding proposal. The core idea of Savoiardi is to encode payload data via an erasure code, and then split the encoded data among all hotshot nodes. The erasure code allows recovery of the full payload from any sufficiently large subset of nodes (e.g. 1/4 of all nodes). An adversary trying to undermine Savoiardi would need to bribe a super-majority of all nodes—in other words, completely break the network—in order to produce a malicious DA certificate for unavailable data. The protocol is communication efficient because a payload of size L requires only O(L) total communication among all nodes—that's only a constant multiple of payload size, independent of the number of nodes! By contrast, a conventional DA solution would broadcast a full copy of the size-L payload to all N nodes for a total of L*N network communication. Unlike danksharding or Celestia's DA solution, Savoiardi does not require data availability sampling (DAS). Instead it relies on a technique called verifiable information dispersal (VID). Examples here and here. VID directly splits the data amongst the HotShot proof of stake nodes in a way that recoverability is ensured. This is more efficient than DAS as it limits unnecessary redundancies. Additionally, Danksharding uses 2-dimensional erasure codes, in order to ensure that subsets of the data can be recovered more efficiently (It still requires ~256 nodes to recover even just one transaction). The downside of this approach is that in the worst case 75% of the nodes need to be honest and provide the data in order to recover an entire block. Our design, with the same parameters, only requires 1/4th of the nodes to be honest. In order to achieve efficient recovery, we add additional layers of security, such that there will be honest nodes who can provide fine grained access to any subset of the data.

Savoiardi is communication efficient and provides the highest (Ethereum-like) level of security. It is thus the base layer for the Tiramisu DA. However, no single node has all the payload data, and recovery of the full payload from individual shares requires communication among nodes and computation to decode the erasure code. Additionally, every user gets a different share of the data and the block producer is responsible for broadcasting all the individual shares to everyone in the network. For these reasons, Savoiardi alone does not provide a good level of user experience. Furthermore, an attacker could potentially trigger the slower and complicated decoding recovery process by withholding data. To remedy this, we add two additional layers to our Tiramisu. The middle layer, Mascarpone, guarantees efficient recovery of data, unless an entire committee is bribed. Finally, the top layer, cocoa, is an optional CDN that is untrusted but can massively accelerate data dispersal and retrieval.


Cocoa: content delivery network (CDN)

Centralized, web2 performance

Marscapone: small DA committee

Secure against any adversary who cannot bribe 80% of the committee, high performance

Savoiardi

Secure against any bribing adversary, expensive decoding

Mascarpone: small DA committee with fast validity condition

To improve the usability and security of the Savoiardi DA, the HotShot consensus elects a small DA committee. This committee consists of less than 200 nodes, and an adversary controlling less than 33% of the HotShot stake has a negligible probability of controlling more than 1/2 of the committee. This committee directly receives the data from the CDN, or directly from a block proposer, and produces a certificate that the block's data is available. We can generate smaller, more efficient certificates if a higher percentage of the nodes is honest using the GearBox approach. The idea is that we can adaptively use signatures from a much smaller committee as long as we get a sufficient amount of signatures from that committee, i.e., if everyone signs, then a committee of just 25 nodes is sufficient.

Mascarpone not only guarantees that the data is available but also that it is efficiently recoverable. Any honest node in the committee can directly provide access to the data or any subset of the data. The main downside of Mascarpone is that a malicious node could potentially bribe a majority of the committee to provide a certificate of data availability without the data actually being available. Bribing a small committee is significantly cheaper than bribing a supermajority of the entire consensus, which could consist of tens of thousands of nodes. But this is where the Savoiardi base layer provides the bribery resistance, as it involves all consensus nodes instead of a small committee.

Cocoa: accelerating the data dissemination through a CDN

We can massively accelerate the Tiramisu DA to provide web2 level performance by using a content delivery network. Our CDN implementation uses the open-source NGINX load balancer and is designed for quickly multicasting the same data, e.g. a block’s content, to many different nodes. It can massively accelerate the data dissemination, and in our Americano testnet we show a data dissemination of around 2.5MB/s to 100 nodes (the size of the Mascarpone committee which can then further disseminate the data). The CDN can also help with efficient recovery of subsets of the data, such as single transactions. Importantly, the CDN is not trusted for security and thus doesn’t present a single point of failure. Tiramisu works perfectly fine without the Cocoa CDN. The CDN is only helpful for accelerating the DA and can easily be replaced or removed. It is not baked into the protocol but is just sprinkled on top like cocoa powder on a Tiramisu.

Like the timeless Italian dessert, Tiramisu is greater than the sum of its individual layers.

Adding a HotShot of Espresso for Responsiveness

Note that all three data availability solutions run in parallel. Although this means that they can support each other (for example, nodes of the Mascarpone committees can assist with Savioadori sharding), it also means that the system is bottlenecked by the slowest component. Savadori is only a constant factor more efficient than danksharding DA, so it is reasonable to ask why users should prefer Espresso DA to Ethereum DA after danksharding has been activated. This is where the HotShot consensus comes in. Unlike the Ethereum consensus or Tendermint, HotShot is responsive. Informally speaking, this means that there is no block time, but new blocks can appear as soon as they have a sufficient quorum of signatures. Only in the worst case, e.g. under a DoS attack, HotShot blocks take as long as Ethereum blocks (12s per block). Under optimistic conditions, however, HotShot will be much faster. Further, HotShot’s finality only takes 3 network round-trips (or 2 with HotStuff2), vs. 2 entire epochs (~12 minutes) for Ethereum. The logic is that Ethereum's parameters are set conservative to handle attacks and bad network states, vs HotShot which is fully adaptive to varying conditions. Thus, under good network conditions and when the CDN is fully operational, the Espresso DA can have significantly improved latency and throughput as compared to non-responsive designs such as Ethereum or Tendermint, even after full danksharding is implemented.

Compatibility with Other DAs

Tiramisu is the default data availability system for Hotshot and is required to be run by HotShot nodes. There are many other data availability solutions, such as Celestia, Eigen DA or Ethereum itself. Tiramisu is fully compatible with any of them as data availability, itself, is additive. Rollups can post the block data to Tiramisu and another DA and the availability guarantee is the stronger of the two. In fact this other DA, could be viewed as an additional layer, on top of Tiramisu.

You can read more details about Tiramisu, our sequencer design, and HotShot in our academic paperhere.