Based Espresso: ad-hoc shared sequencing for all L2s, from based rollups to validiums

Written by Ben Fisch, CEO and Co-Founder at Espresso Systems, published 8 March 2024

Many thanks to Justin Drake and Brendan Farmer for thoughtful discussions, helpful comments, and review of this article.

***TLDR:** Espresso is creating a marketplace for shared sequencing through which layer-2 chains sell blockspace to shared proposers, including the proposer for the Ethereum mainnet EVM itself, who create surplus value by satisfying user intents across multiple chains.*

Our mission has always been to bring renewed unity to Ethereum layer-2 chains. Over the last few years, rollups and validiums have enriched the greater Ethereum ecosystem with horizontal scalability and a diversity of execution environments, but at the expense of fragmentation—apps across rollups do not have the same shared liquidity and interoperability as apps on the Ethereum L1. The value of Ethereum is in its unity, not fragmentation—this is the promise of trust-minimized infrastructure and what differentiates Web3 from Web2.

Ultimately, end users want secure, convenient, and low-cost means to pay one another, buy and sell NFTs, provide DeFi liquidity, swap assets, etc. Users will seek out the optimal way to satisfy their intents even if it means interacting across multiple rollups and the Ethereum L1 at once. Wallets will naturally support this, abstracting the details away from users. Satisfying user intents will be most efficient across applications deployed to rollups with the greatest unity, not only with each other, but also with smart contracts on the Ethereum L1.

Today, rollups assemble blocks of transactions independently through siloed*sequencers* and post these blocks asynchronously to the Ethereum L1. Regaining interoperability between rollups appears to be somewhat of a contradiction at first glance—would it not turn them into one rollup, thereby erasing their scalability advantages and sovereignty?

Shared sequencing has emerged as a path through which rollup users may recover, in practice, the experience of being on one Ethereum chain, while retaining the sovereignty, independence, and performance (including low cost and low latency) of individual rollups. Similar to the Ethereum L1, rollups are updated in discrete blocks of transactions. A group of rollups are said to share a sequencer for a time slot if a shared proposer (in this time slot) proposes the next blocks for all these rollups simultaneously. This proposer can ensure any outcome it wants over the rollups. It can use this power to satisfy arbitrary user intents, including atomic execution of transactions across the rollups. Users may either trust the proposer to satisfy their intent, or may obtain an economically bonded *preconfirmation* as insurance. Shared sequencing is especially powerful among zk-rollups when combined with real-time proving, or proof aggregation as in Polygon's AggLayer design, as this allows for coordinated construction of rollup blocks that synchronously pass messages between each other (synchronous composability). Rollups may share a sequencer indefinitely, or ad-hoc, depending on the design.

A shared sequencing marketplace

Shared sequencing, in the most general sense, is a mechanism integrated among several chains (i.e., rollups, validiums, etc.) for doing two things:

(a) **Proposer Assignment:** Determining a set of proposers to propose the next blocks of all the chains

(b) **Finality Gadget:** Finalizing proposed blocks and providing confirmation of this finality to users

Today, centralized sequencers act as the proposers for their own rollups. In the original rollup design, also known as a "based rollup", the Ethereum L1 proposer is also the rollup proposer. In certain designs of *shared sequencing clusters*, groups of rollups share a fixed set of proposers and a leader-election protocol to rotate among them (e.g., randomly or round-robin). Yet another alternative is a *marketplace* through which rollups sell slots to agents who bid for the opportunity to be a shared proposer for one or more chains at once. Espresso Systems is building one such marketplace, which by virtue of involving the L1 proposer is also compatible with "based rollup" designs. We will revisit this later on.

At a minimum, the Ethereum L1 consensus provides the ultimate finality gadget for all rollups, but with relatively long confirmation latency (~15 minutes in the worst case, although in practice blocks may be considered confirmed by L1 consensus faster). In addition to acting as proposers, centralized sequencers also act as trusted finality gadgets for individual rollups that provide faster confirmations by signing the rollup state. This restricts the rollup's ability to reorg before the Ethereum L1 finalizes. However, unlike the Ethereum L1, these centralized finality gadgets are not shared across distinct rollups. A shared finality gadget is needed within a marketplace for proposer assignment to joint rollup blockspace.

In addition to providing a marketplace for shared proposers, Espresso also provides a decentralized, economically-secured finality gadget, called HotShot, that rollups can share for much faster confirmations than the Ethereum L1 provides. This is operated as a BFT (i.e., proof-of-stake) protocol. L1 validators that have staked collateral for participating in Ethereum's

consensus can also participate in HotShot, with no additional capital requirements, through restaking contracts. Ultimately, if all L1 validators participate, then the finality gadget will have the same economic security as Ethereum itself.

An additional advantage of a decentralized, shared finality gadget is faster, low-cost bridging. As soon as two rollups are finalized using HotShot, it is safe to accept bridge transactions from one rollup to the other. The reason is that all rollup transactions are finalized by HotShot, and thus, the state will never change post-HotShot finalization. The bridging can either happen through zero-knowledge proofs or through a liquidity provider that simply receives funds on one rollup and sends funds on the other. The finality gadget reduces the liquidity provider's risk, not only by signaling the moment the risk of reversal is low, but also by ensuring that any subsequent reversal of a deposit on the origin chain will also erase the liquidity provider's release of funds on the destination chain.

Proposer assignment

The simplest example of Proposer Assignment imposes a single proposer, or perhaps a rotating proposer set, on all chains. Alternatively, in what we may call *ad-hoc shared sequencing*, proposer assignment functions as more of a marketplace in which chains sell blockspace by the slot to sophisticated parties who bid for the right to propose bundles of rollup blocks. In ad-hoc shared sequencing, the outcome in any slot may be a partitioning of rollups among multiple proposers. A rollup's governance may also designate a party—or decentralized protocol, as in distributed validator technology (DVT)—to bid on its own blockspace, which functions as a reserve price: the rollup's blockspace is only sold to an outside shared proposer for a bid that exceeds this price. This allows rollups to ensure they are capturing their fair share of sequencing revenue. Espresso's mechanism for proposer assignment is a type of combinatorial auction, which we will discuss more later on in this post.

Preconfirmations

In any proposer assignment mechanism, a proposer that holds the right to build the next blocks for a bundle of multiple rollups simultaneously has a lock on their joint state—it can act as a liquidity provider, or make promises to satisfy arbitrary user intents, including atomic execution of several transactions across different rollups, even conditioned on the joint state outcome. These are known as *preconfirmations.* Some concrete examples of user intents include:

- Capturing an arbitrage opportunity between DeFi exchanges on two different chains (including Ethereum L1)
- Finding the cheapest way to execute an asset swap, which may involve multiple DeFi exchanges or liquidity providers across multiple chains
- Using an application (e.g., buying an NFT) on one chain with funds from another and not paying to bridge/swap assets unless the purchase succeeds

A user's willingness to trust a preconfirmation may be based on trust in the proposer and/or economic incentive alignment (e.g., the proposer may be slashed for violating the promise). Preconfirmations can also be designed to function as an insurance policy—a user pays the proposer a tip for the ability to collect damages from escrowed collateral up to a certain amount should the promise be violated. The proposer absorbs the risk that it will be unable to fulfill the promise and prices the risk accordingly. With rollups that support advanced designs such as Polygon's AggLayer, a shared proposer may also coordinate multiple rounds of message passing between rollups that occur within the span of a single block. The surplus value created by a shared proposer, which may come from transactions that wouldn't exist otherwise, tips users pay for preconfirmations, or even arbitrage exercised by the proposer itself, broadly encompasses cross-domain MEV.

Based sequencing

The original rollup design, in fact, allowed the rollup smart contracts to be updated by the Ethereum L1 block without the involvement of external sequencers, which de facto gives the Ethereum L1 block proposer shared sequencing rights over all rollups. External sequencers were introduced as a way to achieve higher throughput (via enhanced data compression) and fast transaction preconfirmations (based on trust in the sequencer, which rollups are now looking to decentralize). Rollups with external sequencers still inherit Ethereum L1 liveness by allowing the smart contract to force transactions through when the sequencer is offline for too long.

However, this original rollup design, which has started to regain popularity under the name *based rollups*, has an overlooked key advantage: better L1 composability. The randomly elected proposer of the Ethereum L1 block, who has a lock on all rollups *and* the L1 state for a single slot, can issue preconfirmations not only over joint rollup states but also involving the L1 state itself. Moreover, in a future where validity proofs (i.e., SNARKs) reach ever higher throughputs and lower latency, the L1 proposer could leverage *real-time* (i.e., subsecond) *proving* to make function calls between L1 and L2 applications within the span of a single L1 block, including moving assets in and out of rollups.

Design principles of based sequencing

The scope of based rollups, or *based sequencing*, has evolved beyond the original concept to encompass alternative designs that achieve a broader set of desiderata. *Based Espresso* is one such design. The ideal properties that we strive to

achieve are:

- **L1 liveness & neutrality:** a rollup inherits liveness and credibly neutrality from the Ethereum L1, without excessive delays due to failure of infrastructure external to the L1.
- **L1 composability:** the proposer of a block for the Ethereum mainnet EVM (L1) should have the ability to update rollup states synchronously with the L1 and achieve any atomic, conditional execution between the rollups and the L1 that it desires. Ultimately with real-time proving this proposer should be able to facilitate synchronous composability between applications on rollups and the L1.
- **Fast (pre)confirmations:** users of rollups should obtain fast confirmations of rollup transactions with little to no delay after they are included in a block. Confirmations of rollup transactions should ideally carry negligible risk of reversal, at least conditioned on the finality of the Ethereum L1 state they build off of. Users should also be able to express intents encompassing atomic dependencies between transactions on rollups and Ethereum mainnet EVM (L1) execution, obtaining pre-confirmations of intent satisfaction from collateralized proposers (or low-cost insurance coverage for intent violation).
- **Revenue (MEV) redistribution:** rollups should receive a share of joint sequencing MEV, proportional to their marginal contribution (exceeding the amount they would have captured running their own sequencer in isolation).

L1 composability vs. fast finality

Every rollup has a smart contract on the L1 that tracks the state of the rollup, but may lag behind. A rollup smart contact is updated by calling it inside an L1 block. Today, rollup contracts are typically only called once per Ethereum block, or even less frequently. However, if L1 block building is coordinated with L2 block building (i.e., when the L1 proposer is also the rollup sequencer) then it is possible for a rollup contract to be called multiple times within a single L1 block. Additionally, with sufficiently low-latency validity proofs (i.e., SNARKs), the proposer constructing the L1 block and rollup block simultaneously could pass verifiable messages between the rollup and L1 in order to enable synchronous interactions between contracts on the L1 and rollup. While the latency of validity proofs are too high for this to work today, the technology has been improving at a rapid pace, especially with the development of specialized proving hardware.

There is a fundamental tradeoff between L1 composability and fast confirmations of rollup transactions. L1 composability mandates that the L1 proposer has the ability to take actions in a newly published Ethereum block that immediately affects the state of a rollup. The Ethereum L1 has slow finality. Blocks are published every 12 seconds but are not finalized until after 15 minutes, when the full weight of Ethereum's economic security prevents *reorgs* (i.e., any reorg at that point would cause 1/3 of Ethereum's stake to be slashed). The result of a transaction on a rollup cannot be final any earlier than the most recent Ethereum L1 state it depends on.

For this reason, some rollups today (such as Arbitrum and Polygon) choose to build off a finalized L1 block instead of the tip of the chain—any action on the L1, such as bridge deposits into the rollup, cannot be recognized in the rollup until it has been finalized on the L1. This enables the rollup sequencer (which may be a centralized party, or a decentralized BFT protocol) to finalize rollup (L2) transactions and give instantaneous preconfirmations to users, because they are not dependent on any uncertain L1 state. These are called "preconfirmations" because a malicious sequencer (whether single party or protocol) could ultimately decide to publish a conflicting block of rollup transactions. However, a user is certain over the outcome if it trusts the sequencer not to equivocate (e.g., based on an honest majority assumption and/or economic incentive alignment). Once the sequencer publishes a block on the L1 it is more limited in its ability to equivocate unless the L1 reorgs, which is still a risk for ~15 minutes after publication.

While L1 composability is at odds with instantaneous preconfirmations of L2 state, there is perhaps a happy middle ground between these two extremes. The exploration of this middle ground began with a seminal post by Justin Drake on *based preconfirmations*. First, shallow (single block) reorgs of the Ethereum L1 are infrequent, and deeper reorgs are rare. While this risk may still be too great to depend on for extremely high value transactions, by viewing preconfirmations as a form of insurance, the parties (or protocols) issuing preconfirmations can in general absorb this risk and price it appropriately for users. We will call these *conditional preconfirmations* to reflect that they carry risk exogenous to the party/protocol providing them and to distinguish them from preconfirmations that can only be violated by malicious actions of the provider. Users may ultimately be exposed to the same risk when trusting either flavor of preconfirmation (i.e., when preconfirmations act as an insurance policy), but pay a higher price in the former to cover the provider's risk.

Second, many rollup transactions (such as peer-to-peer payments or asset transfers) may not be dependent on parts of the L2 state that were affected by non-finalized changes in the L1 state. It is in fact possible to enable fast preconfirmations for these transactions, without any risk to the provider, while retaining L1 composability. The original based sequencing design, however, fails to achieve this—transactions proposed solely through the rollup smart contract in an L1 block are always at risk of reversal until the L1 block reaches finality, regardless of preconfirmations provided by the L1 proposer. We will see how the Based Espresso design achieves this nice property later on in this post.

BFT vs. single-party preconfirmations

We have distinguished between types of preconfirmations based on the risk they pose to the provider. Another distinction is between preconfirmations backed by a single provider versus a committee or decentralized protocol. A proposer for a bundle

of rollups can offer sophisticated preconfirmations with the lowest possible latency, but preconfirmations backed by a decentralized protocol will likely have significantly greater economic security (staked collateral) and also require greater coordination to violate. It is easier to decentralize simple preconfirmations that demand little sophistication from participants. Shared stateless finality gadgets are one such example. They only promise non-equivocation of published rollup blocks and do not require protocol participants to compute rollup states. The proposer for a bundle of rollups may also be controlled by a committee via threshold cryptography (as in DVT). The committee may provide preconfirmations for arbitrary intent satisfaction, but this requires greater sophistication of all committee participants (e.g., executing for these rollups, participating in proposer selection auctions, etc).

The Based Espresso design

The Based Espresso design achieves the ideal desiderata for based sequencing described in the previous section. Compared to vanilla based sequencing, Based Espresso offers rollups stronger preconfirmation security (finality) and a share of sequencing revenue. In fact, rather than simply granting sequencing rights to the L1 proposer for free, with Based Espresso, rollups are participating in a marketplace where they can dynamically sell these rights. This is based on two separate ideas, one that leverages a BFT finality gadget in which L1 validators can participate through restaking, and one that uses a proposer selection mechanism similar to execution tickets.

This version of based sequencing offers two flavors of preconfirmations: proposer and attester. The proposer preconfirmation comes first (subsecond latency) backed by collateral of an individual proposer for a shared sequencer slot, while the attester preconfirmation comes second (~1-2 second latency) is backed by the collateral of participating attesters. In addition to having higher collateral at stake, violating an attester preconfirmation would require collusion of attesters (sufficiently many to control a third of all stake collateral). Ethereum L1 validators can participate as an attester by restaking their ETH (e.g., through EigenLayer).

Ethereum L1 proposers may purchase the right to act as the shared proposer for L2 blocks as well. The rollup still requires transactions to receive attester preconfirmations before being included. Unlike vanilla based rollups, this ensures that honestly preconfirmed rollup transactions are final conditioned on the last Ethereum block they depend on. In other words, they are guaranteed to be executed unless this past Ethereum block is reorged or a threshold of the attesters are malicious and willing to have their collateral slashed. In this way the attester preconfirmation plays a similar role to external sequencers in today's rollups. This enables proposers to give fast preconfirmations for pure L2 transactions unaffected by the most recent Ethereum block without risk. In contrast, in vanilla based sequencing, new rollup transactions included by an L1 proposer in some block B_t that builds on block B_{t-1} are not conditionally final based on B_{t-1}. They will be abandoned in a reorg that drops B_{t}! A preconfirmation from the L1 proposer for the transactions included in B_t cannot immediately be given risk-free.

On the other hand, a caveat of requiring L2 transactions to receive attester preconfirmations is that the attester BFT finality gadget could stall liveness if too many attesters go offline (e.g., more than a third) even while the Ethereum L1 is still live. However, the original design of based sequencing with based preconfirmations by Justin Drake also suffers a similar problem when too many L1 validators that have opted into participating as preconfirmation providers, a.k.a. *preconfers*, are offline (e.g., if none of the next 32 proposers include preconfers). In that design, preconfirmed transactions are given priority in the rollup, so the L2 has to stall when preconfers are offline. A mitigation in both scenarios is to have a timeout period after which sequencing falls back to (pure based sequencing on) the Ethereum L1 if there is no progress on the L2. This is a choice that rollups can make.

In Based Espresso, as in vanilla based sequencing, when the L1 proposer for block B_t also acquires the right to propose L2 blocks in the same time slot, then it has the option to interleave L1 and L2 transactions. Combined with low-latency proving, it could even facilitate synchronous composability between L1 contracts and L2 contracts. However, preconfirmations for these synchronous interactions are higher risk due to the dependency on recent unfinalized L1 transactions. Thus, it may prefer to *first* sequence rollup transactions that are conditionally final on block B_{t-1}. These transactions are guaranteed to execute on the rollup even in the event that B_t is reorged. Users may pay tips to enjoy this stronger finality, i.e. being conditioned on B_{t-1} rather than B_t, particularly transactions on the L2 that do not depend on any L1 activity in block B_t. Even without synchronous interactions between the L2 and L1 blocks, the shared L1 proposer can always enforce atomic execution conditions between L1 and L2 transactions.

Another significant distinction is that while L1 proposers may acquire the right to propose for rollups and give preconfirmations, in the event that an L1 proposer does not exercise this right for their slot, another party acquires it instead and has the same ability to provide preconfirmations (albeit for L2 transactions only). This is particularly beneficial in the case that initially only a small fraction of L1 proposers participate in rollup sequencing and preconfirmations, and thus L1-to-L2 synchronous interactions are less frequent.

Outside of blocks that have synchronous interaction between L1 and L2, which may not be every block for most rollups, transactions can be recorded and made available through the attester BFT protocol and settled to the rollup contract periodically (instead of every Ethereum block). Given that this attester BFT protocol has higher throughput and cheaper data availability than the Ethereum L1, this helps with data compression for rollups. This is also particularly advantageous for validiums.

Construction details

There are two key components that are introduced on top of the Ethereum L1.

- **HotShot finality gadget:** HotShot is a state machine replication (SMR) protocol that achieves safety and liveness under partial synchrony, so long as not more than 1/3 of the participating stake is corrupt. Any safety violation will result in 1/3 of the stake being slashed (and also requires coordination of a set of nodes controlling 1/3 of the stake). It optimistically reaches finality on block proposals within a few seconds. We call participants *attesters*. The state tracks a chain of blocks for each unique domain identifier (aka namespace). It is also synchronized with a proposer selection mechanism (below) so that in any given state it can only append blocks to a namespace signed by the proposer(s) assigned to that namespace. The slots of this protocol, called HotShot slots, are synchronized with (but may be more frequent than) the Ethereum L1. HotShot enables participation from restaked L1 validators (e.g., via EigenLayer).

- **Combinatorial proposer selection:** This component runs a *combinatorial lottery* to assign proposers to namespaces for each HotShot slot. A HotShot slot is for a duration of time and may accept multiple sequential block proposals during a single slot. A proposer may be assigned to more than one namespace at a time, which we call a *bundle*. Participants in the selection mechanism may purchase lottery tickets both for individual namespaces and namespace bundles. A combinatorial lottery is the lottery equivalent of a combinatorial auction, where participants would bid on bundles; or equivalently, it can be viewed as a pay-as-you-bid combinatorial auction with proportional allocation. This lottery for proposer assignment is run at least 32 L1 slots in advance of the slot. Finally, the L1 proposer for an Ethereum slot (when it becomes known 32 slots in advance) will be given the option to purchase the HotShot proposer rights for any namespace bundle from the proposer to whom it was assigned (i.e., at the winning bid/ticket price). In essence, the L1 proposer holds an "ROFR" (right of first refusal) for the sale of any namespace bundle proposing rights within its slot.

Next we describe how these components come together. First let us distinguish two types of rollup contract calls:

- $L1 \rightarrow L2$ transactions: function call depends on and may write to state of Ethereum L1. These include bridge deposits. This will not immediately write to the L2 state but will update a queue that is eventually consumed by an $L2\rightarrow L2$ transactions.

- $L2 \rightarrow L2/L1$: updates the rollup state *st*, has no immediate side effect on L1 state, but may depend on queue of $L1 \rightarrow L2$ transactions, and may update a queue of L1 transactions (e.g., withdrawals) that is eventually consumed by an L1 transactions.

1. Tracking the state of Ethereum L1 in HotShot. While the HotShot state does not execute rollup transactions, there is a minimal state and execution engine within the protocol. The HotShot state will track the state of Ethereum as follows. Each HotShot state points to the tip of an Ethereum L1 fork, which we can call the . Every HotShot block proposal has the opportunity to update the to point to a new Ethereum block if either (a) the block is at the tip of a chain that extends the old , or (b) the block is at the tip of a fork that is “heavier”, i.e. preferred according to the fork choice rule of Ethereum. Note that the HotShot state is not guaranteed to track the true latest state of Ethereum at any given time, nor is this equivalent to having all the HotShot validators to vote on their opinion of the Ethereum state. However, any honest HotShot proposer that knows the latest state will be able to successfully update HotShot to reflect this. This is the key property that we want. In essence, HotShot acts as a trusted client for the Ethereum L1.

2. Implicit DAG in HotShot. The chain of HotShot blocks induces a DAG structure over L2 states indexed by L1 forks. Every round of HotShot may update pointing to a fork of Ethereum L1 and also adds L2 blocks on the respective rollup namespaces. The of any L2 block is the of the *previous* HotShot block. All successive L2 blocks that share the samewill be interpreted by rollups as being executed after the block and before the next L1 block that follows . Each L2 block is an $L2 \rightarrow L2$ contract call to the rollup contract, and thus can be retroactively inserted before the next L1 block without having any immediate impact on L1 state. L2 blocks may still queue operations (e.g. withdrawals) that will eventually have effect on L1 when the contract is updated in an L1 block. L2 blocks may also consume queued transactions from the L1 (e.g., deposits). The L1 block also cannot change the state of the rollup contract without going through HotShot, but can queue $L1 \rightarrow L2$ transactions (like bridge deposits). Queued transactions from L2 blocks will be settled in the L1 block that finally updates the L2 state of the rollup contract. Thus, every HotShot block is interpreted by the rollups' derivation pipelines as being executed on a particular fork of Ethereum, and is by default only final conditioned on the fork finalizing on Ethereum. However, an individual transaction included in a HotShot block may elect to be conditionally final based on an earlier Ethereum block that precedes . Concretely, if a transaction sets its reference point to some Ethereum block *id*, then rollups will execute this transaction on any fork of Ethereum that includes*id.* There is a strong reason to do so if the transaction does not depend on any $L1 \rightarrow L2$ transactions appearing between blocks *id* and because it will have stronger finality (block *id* is less likely to reorg). We provide more details on the contract next.

3. Proposer for L1 pointer. Every HotShot slot has an assigned proposer to update the tracked Ethereum state, just as any other namespace. The L1 proposer always has the ability to acquire the proposer rights for this as well. Note that while an update may be proposed simultaneously with rollup blocks in the same HotShot slot, as detailed in item 2 above the rollup blocks will be applied first (i.e., occurring before the new L1 block not after).

4. Rollup smart contract on the L1. The rollup smart contract on the L1 tracks the state of the rollup, but may lag behind. A rollup smart contract is updated by calling it inside an L1 block. Today, rollup contracts are typically only called once per Ethereum block or even less frequently. However, if L1 block building is coordinated with L2 block building, then it is

possible that a rollup contract may be called multiple times within a single L1 block, interacting synchronously with other contracts on the L1. Today, most rollup smart contracts have a designated sequencer, a party that must sign off on any update to the contract (with the exception of bridge transactions).

To integrate as a based rollup through Espresso, rollups would replace this with the following logic.

HotShot light-client. The rollup tracks a state of HotShot (it is a light client for HotShot). HotShot forms a continuous chain (it never forks).

Calling the rollup contract. The rollup contract stores the rollup state *st*, a *queue* of L1 contract calls, and pointers to the last HotShot block *h* used to update this state. When it receives bridge transactions from contract call inside L1 block, they are added to the *queue* but this doesn't immediately change *st*, which can only be changed by providing proof of the correct execution of L2 blocks finalized by HotShot from the last time the contract was updated on the present branch. In more detail, if the contract is called to update the rollup state in an L1 block with identifier *id,* with current state *st* and pointer to *h* where it was updated last, the new state *st'* with pointers to *h'* should be the result of executing all finalized HotShot blocks between *h* to *h'* along the unique fork that *id* extends. (In the case of an optimistic rollups the rollup contract receives*st'* which can be challenged through fraud proofs, and in a zk-rollup it directly receives a proof of correct derivation). Importantly, it is the order in which L2 blocks and L1 references appear in HotShot that determines the relative ordering of L1/L2 transactions. The L1 block in which the L2 state update is ultimately recorded has no impact on relative ordering, and the recorded state in the L1 contract may lag behind the true state of the L2.

L1/L2 synchronous interactions. In the description so far, HotShot only interleaves entire L1 blocks with entire L2 blocks. We add one more feature to enable synchronous interaction between L1 and L2, when the L1 proposer is simultaneously the HotShot proposer, which is for HotShot to record *partial L1 states*. Essentially, this allows for building a branch off some block that interleaves L1 and L2 transactions before a full L1 block is complete, and which will be abandoned if the L1 block containing these transactions is never proposed on the L1. These transactions have much weaker finality until the L1 block containing them is proposed. The L1 proposer can easily reorg this temporary branch recorded in HotShot by never proposing. The main purpose of this feature is to enable synchronous composability between L1 and L2 without breaking anything else. We can't simply allow every L1 block to include L2 transactions without HotShot's involvement as that would break the ability of HotShot to provide finality on L2 blocks between L1 blocks as described above. Instead, when the L1 proposer building off of is also the HotShot proposer, it can first get HotShot to accept as the current Ethereum chain head, and then gets HotShot to record each successive $L1\rightarrow L2$ and $L2\rightarrow L2/L1$ transaction in order. The rollup derivation pipeline will execute these transactions in the order they appear. After HotShot finalizes each $L2 \rightarrow L2/L1$ txn the L1 proposer can record the state change in the L1 contract. This enables passing messages from L1 to L2 and from L2 back to L1 respectively (e.g., deposits and withdrawals) in the same block.

Other notes:

- HotShot slots will be synchronized with Ethereum 12 second slot times.
- The L1 proposer gets the ROFR to update the L1 reference in HotShot for free.
- The combinatorial auction takes place before the L1 proposer is known, such that everyone is incentivized to participate in the auction.
- Once the L1 proposer is known they'll need to get HotShot to register that they've excercised the ROFR. This can be done through creating a namespace to track the L1 ROFR. The proposer of this namespace is the L1 proposer executing their ROFR for 32 blocks in the future (32 blocks is how far in advance the L1 proposer is known).
- HotShot will support multiple simultaneous proposers in the same slot (e.g. one proposer per namespace), similar to DAG-BFT protocols.