TECHNICAL SPECIFICATION

Hydrozoa: Lightweight state channels for Cardano

WORKING DRAFT

2025-10-16

Hydrozoa: Lightweight state channels for Cardano

George Flerovsky
GeorgeFlerovsky (on Github)

Ilia Rodionov
euonymos (on Github)

Peter Dragos
dragospe (on Github)

Abstract

Hydrozoa is a dynamic, near-isomorphic multi-party state channel protocol. It allows a group of peers to transact with one another directly outside of the Cardano blockchain (on L2), significantly minimizing their activity on the Cardano blockchain (on L1). The peers enjoy high throughput, instant finality, and low costs while transacting with each other in this way. If the peers unanimously consent, they can add new peers, remove existing peers, deposit funds from L1, and withdraw funds from L2.

Contributors

Adam Dean
Crypto2099 (on Github)

Alexander Nemish
nau (on Github)

Pi Lanningham
Quantumplation (on Github)

Sebastian Nagel
ch1bo (on Github)

Branch: (None) @ c28633a • Commit date: 2025-10-16

Contents

Branch: (None) @ c28633a • Commit date: 2025-10-16

Overview

Hydrozoa is a dynamic, near-isomorphic multi-party state channel protocol:

Multi-party state channel. Hydrozoa allows a group of peers to transact with one another directly outside of the Cardano blockchain (on L2), significantly minimizing their activity on the Cardano blockchain (on L1). The peers enjoy high throughput, instant finality, and low costs while transacting with each other in this way.

Near-isomorphic. The peers' transactions on L2 follow the same ledger rules as Cardano L1 to the extent possible.[1]

Dynamic. If the peers unanimously agree, Hydrozoa allows them to add new peers or remove existing peers from the group,[2] deposit funds from L1, and withdraw funds from L2.

Hydrozoa evolved out of Cardano's Hydra Head protocol [10], dramatically simplifying the L1 smart contract architecture and increasing flexibility.

L1 protocol

The Hydrozoa L1 protocol has two distinct operating regimes. Every head is initialized by its peers in the **multisig** regime (§ 1), in which all funds and state are stored at a native script address controlled by the unanimous multi-signature of its peers. Deposits into the head are created by sending utxos to this address. They are either absorbed into the head's treasury or refunded to depositors by multi-signed transactions. Similarly, withdrawals are settled on L1 by multi-signed transactions that send utxos out of the head's treasury. Thus, the multisig regime's L1 script defers entirely to the L2 consensus protocol between the peers, which it perceives as a black box mechanism that produces multi-signed transactions to modify the L1 state.

Ideally, a head spends its entire life cycle in the multisig regime, right up to the final transaction that withdraws all remaining funds out of the head and de-initializes it. However, the multisig regime may stall if the peers stop multi-signing transactions before reaching finalization.

In this case, the head moves to the **rule-based** regime (§ 2), in which unabsorbed deposits are refunded and control over the head's treasury transfers to a suite of Plutus scripts.

This regime focuses on the peers' dispute about which L2 block is the latest among those confirmed by the L2 consensus protocol and compatible with the L1 treasury state. The Plutus scripts arbitrate the peers' dispute and manage withdrawals from the treasury based on the resolved L2 block.

[1] Unfortunately, full equivalence is impossible because the result of the L2 transactions ultimately needs to be settled on L1 by the peers, and Cardano's ledger rules do not authorize the peers to perform certain actions (§ 3.1).

[2] Adding and removing peers is not currently specified in this document, but will be specified and implemented in the final milestone of the Catalyst Fund 13 Hydrozoa project [4] (§ A.1).

L2 ledger state and blocks

The Hydrozoa L2 ledger state (§ 3) consists of a set of active utxos and has three types of transitions:

- L2 transactions spend utxos and may produce new utxos.
- L2 withdrawals spend utxos but do not produce any new utxos.
- L2 genesis events produce new utxos but do not spend any utxos.

Among these, transactions and withdrawals are manually initiated on L2 by peers, while genesis events are automatically inserted as L1 deposits are absorbed into the head's treasury.

For efficiency, L2 events are batched into blocks (§ 4) so that the peers can reach consensus on an entire block of L2 events at every round of the Hydrozoa L2 consensus protocol. Each block affirms a sequence of L2 ledger events, rejects a separate set of L2 ledger events, and absorbs a set of L1 deposits. There are three types of blocks:

- A minor block only affirms L2 transactions. It does not affirm L2 withdrawals and does not absorb any L1 deposits.
- A major block may withdraw some utxos and absorb some deposits.
- A final block is intended to be the last block confirmed by the peers for the head. It withdraws the entire active utxo set and does not absorb any L1 deposits.

L2 consensus protocol

Hydrozoa's L2 consensus protocol (§ 5):

- Broadcasts L2 transaction and withdrawal requests among the peers.
- Facilitates peer agreement on a growing common sequence of L2 ledger events grouped into blocks.
- Promptly settles the L1 effects of a confirmed block if it affirms any L2 withdrawals or absorbs any L1 deposits.
- Ensures that no funds get stranded as unabsorbed L1 deposits. If a deposit is determined not to be absorbed into the head, it can always be refunded to the depositor.
- Ensures that no funds get stranded in the head's treasury. If the peers stop responding to each other, they can always retrieve their funds on L1 according to the latest block matching the L1 treasury's major version.

Peers take round-robin turns to create L2 blocks—every i^{th} block must be created by the i^{th} peer.

Chapter 1

L1 multisig regime

Every Hydrozoa head uniquely corresponds to its peers' public key hashes. In the L1 multisig regime, the head's native script simply requires signatures from all these public key hashes:[1]

$$\begin{aligned}&\text{headNativeScript} :: \text{Set PubKeyHash} \longrightarrow \text{Timelock}\\&\text{headNativeScript} := \text{AllOf} \;.\; \text{map Signature}\end{aligned}$$

We could end this chapter here, as no other conditions are enforced by L1 scripts in the multisig regime. However, we will describe this regime's expected states and transitions, which the Hydrozoa L2 consensus protocol enforces while the peers engage with it (§ 5).[2]

1.1 Utxo state

In the multisig regime, a head's entire L1 state resides in utxos at its native script address.

$$\text{MultisigHeadState}^{L1} := \left\{ \begin{array}{lcl} \text{treasuryUtxo} & :: & (\text{OutputRef}, \text{Output}^{L1}) \\ \text{depositUtxos} & :: & \text{UtxoSet}^{L1} \\ \text{rolloutUtxos} & :: & \text{UtxoSet}^{L1} \end{array} \right\}$$

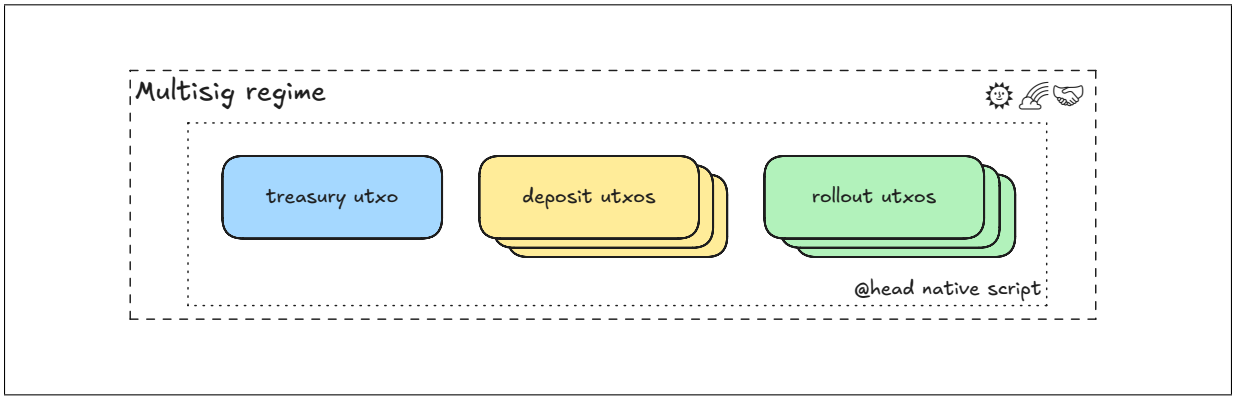

Figure 1.1: The L1 utxo state of the multisig regime.

[1] We use similar notation to the Cardano ledger library [7], simplifying for readability. In that library, native scripts are sometimes called "timelock" scripts.

[2] Of course, the peers can always manually override the L2 consensus protocol by unanimous consent.

Among these utxos, the head's treasury utxo is uniquely identified as the one holding the head's beacon token:

1. The policy ID corresponds to the head's native script (in minting policy form).
2. The first four bytes of the asset name are equal to the CIP-67 [1] prefix for class label 4937, which corresponds to Hydrozoa head beacon tokens.[3]
3. The last 28 bytes of the asset name are equal to the Blake2b-224 hash ($\mathcal{H}_{28}$) of the utxo nonce spent in the head's initialization transaction (§ 1.2).

No other utxo at the head's native script address should hold any tokens that meet the policy ID and CIP-67 prefix conditions above. In other words, there must only be one head at the address.[4]

The treasury utxo must have this datum:

$$\text{MultisigTreasuryDatum} := \left\{ \begin{array}{lcl} \text{utxosActive} & :: & \mathcal{RH}_{32}\ \text{UtxoSet}^{\text{L2}} \\ \text{versionMajor} & :: & \text{UInt} \\ \text{params} & :: & \mathcal{H}_{32}\ \text{ParamsConsensusL2} \end{array} \right\}$$

These fields are interpreted as follows:

active utxos. A 32-byte Merkle root hash ($\mathcal{RH}_{32}$) of the L2 ledger's active utxo set, as of the latest major L2 block settled on L1 (§ 1.5).

major version. The version number of the latest major L2 block settled on L1.

params. A 32-byte Blake2b-256 hash ($\mathcal{H}_{32}$) of the head's L2 consensus parameters (§ 4.1).

Non-treasury utxos at the head's native script address are recognized as follows:

- A rollout utxo holds a rollout beacon token of the head's minting policy, with CIP-67 prefix 7655.[5] It is a no-datum output of a settlement, finalization, or rollout transaction. It is transient—quickly spent in a subsequent rollout transaction.
- A deposit utxo has a datum with the `DepositDatum` type (§ 1.3).

1.2 Initialization

Before initializing a Hydrozoa head, the peers should agree on the L2 consensus parameters they want to use (§ 5). Initialization is achieved with a single transaction, multi-signed by all the peers, that:

- Mints the head's beacon token.
- Spends the utxo nonce reference by hash in the last 28 bytes of the beacon token's asset name.

[3] We chose 4937 because it spells "HYDR" (short for Hydra/Hydrozoa) on a phone dial pad.

[4] It may seem redundant to use a 28-byte unique nonce in the beacon token's asset name, given that only one head is allowed per native script address at any given time. However, the unique nonce helps distinguish between different heads that may exist at the same address across time.

[5] We chose 7655 because it spells "ROLL" (short for rollout) on a phone dial pad.

- Creates the treasury utxo at the head's native script address, placing the beacon token into it (with min-ADA) with this datum:

$$\text{initMultisigTreasuryDatum} :: \mathcal{H}_{32}\ \text{ParamsHead} \longrightarrow \text{MultisigTreasuryDatum}$$

$$\text{initMultisigTreasuryDatum initParams} := \left\{ \begin{array}{lcl} \text{utxosActive} & = & \mathcal{RH}_{32}\ \varnothing \\ \text{versionMajor} & = & 0 \\ \text{params} & = & \text{initParams} \end{array} \right\}$$

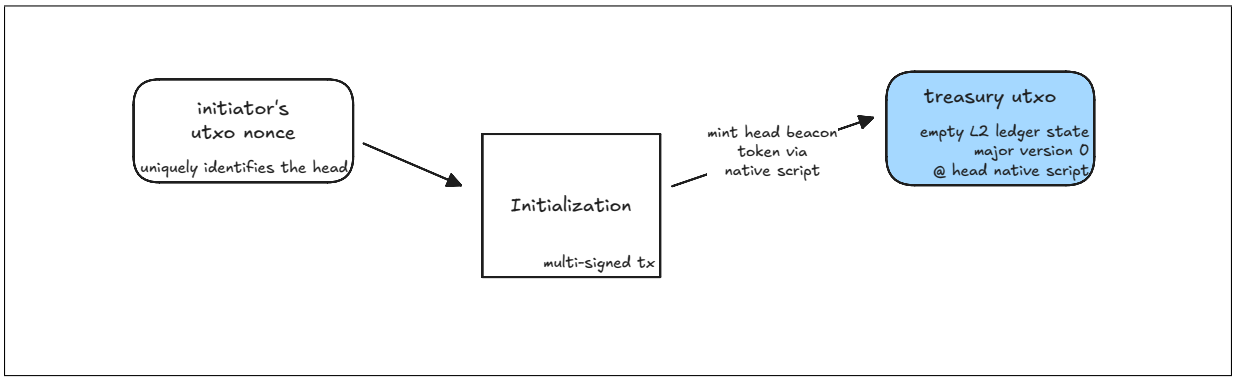

Figure 1.2: The Hydrozoa head's peers initialize its treasury utxo with an empty L2 ledger.

1.3 Deposit

Peers can deposit funds into a Hydrozoa head by sending utxos to its native script address, specifying their instructions in this datum type:

$$\text{DepositDatum} := \left\{ \begin{array}{lcl} \text{address} & :: & \text{Address}^{L2} \\ \text{datum} & :: & \text{Maybe Datum}^{L2} \\ \text{deadline} & :: & \text{PosixTime} \\ \text{refundAddress} & :: & \text{Address}^{L1} \\ \text{refundDatum} & :: & \text{Maybe Datum}^{L1} \end{array} \right\}$$

In this datum, the depositor instructs the peers to absorb the deposit into the treasury via a settlement transaction (§ 1.5) before the `deadline`, or else to refund it to the depositor (§ 1.4) if it will not be absorbed before the `deadline`.

The `address` and `datum` fields define the address and datum at which the utxo in the L2 ledger should be created if the deposit is absorbed into the head's treasury. The new utxo in the L2 ledger should contain the same funds as the deposit utxo.

The `refundAddress` and `refundDatum` fields define the address and datum to which the deposit's funds should be sent on L1 if the deposit is refunded. The funds in the deposit must be sufficient to pay for the refund transaction cost and create the refund utxo.

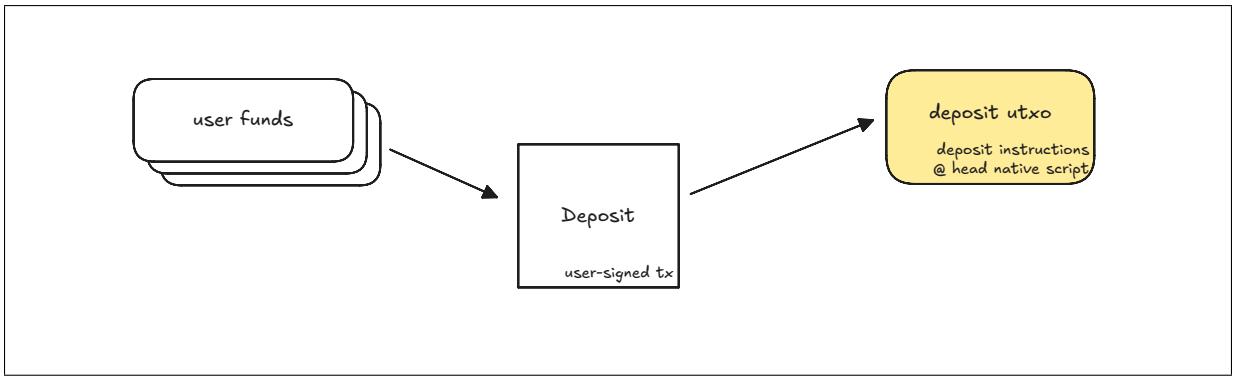

Figure 1.3: A user places some funds and instructions into a deposit utxo for a Hydrozoa Head.

1.4 Refund

A refund is a transaction, multi-signed by all peers, that sends a deposit utxo's funds to the deposit's `refundAddress` and `refundDatum`. While the peers follow the L2 consensus protocol, they will automatically multi-sign and submit refund transactions for all deposits not absorbed before their deadline.[6]

However, a depositor should obtain the other peers' signatures for a post-dated refund transaction before submitting the deposit transaction to Cardano. This transaction's validity interval should start at the deposit's `deadline`. Doing so ensures that the depositor can still retrieve the deposited funds if they have not been absorbed by the deadline, even if the other peers stop responding.

The L2 consensus protocol (§ 5.5) allows peers to obtain multi-signed post-dated refund transactions before posting their deposits on L1. It also allows them to immediately refund any deposits that its rules disqualify from being absorbed into the treasury.

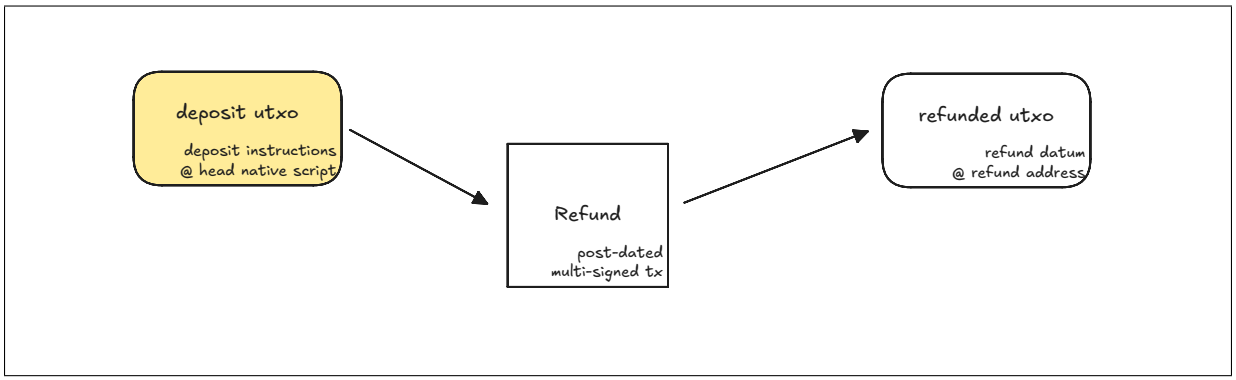

Figure 1.4: The peers refund the deposit if it isn't absorbed into the treasury before the user's deadline.

1.5 Settlement

A settlement is a transaction, multi-signed by all peers, that absorbs deposits into a Hydrozoa head's treasury and pays out withdrawals out of the treasury based on a major L2 block (§ 4).[7]

[6] To minimize uncertainty from L1 rollbacks, the L2 consensus protocol will not attempt to absorb a deposit unless sufficient time remains before its deadline. This safety margin is controlled by an offchain consensus parameter (§ 5.2.1).

[7] A block is "major" if it confirms new deposits or withdrawals that were not confirmed in the preceding blocks.

It also updates the `versionMajor` and `utxosActive` fields of the `MultisigTreasuryDatum` to the corresponding fields in the major L2 block while keeping the `params` field unchanged.

The absorbed deposit utxos are spent inputs in the settlement transaction. This means that Cardano's transaction size limit constrains the number of deposits settled per major L2 block.

On the other hand, the number of withdrawals per major L2 block is unconstrained. The withdrawn utxos of the major L2 block are paid out via outputs of the settlement and rollup transactions, in the ascending order of their output references in the L2 ledger.

The assignment of L2 withdrawn utxos to L1 outputs of the settlement and rollout transactions is handled by Hydrozoa's deterministic rollout algorithm (§ B). The settlement transaction greedily pays out as many withdrawals as it can, using the transaction size remaining after fitting the deposit inputs and the fixed transaction overhead. It outputs the remaining withdrawals' aggregate funds as a single rollout utxo, sent to the head's native script address without a datum. Rollout transactions iteratively pay out more utxos from the rollout utxo, until all withdrawals are settled (§ 1.7).

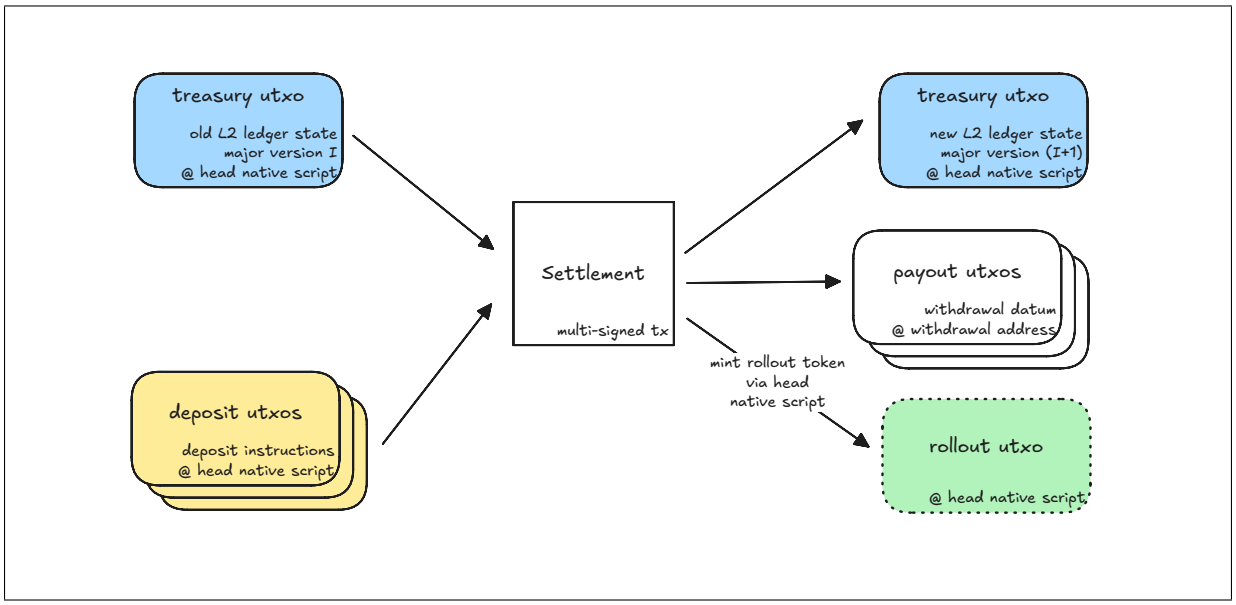

Figure 1.5: The peers absorb some deposits and pay out some withdrawals from the head's treasury. A rollout utxo can be produced to defer payout of some withdrawals for later.

1.6 Finalization

The finalization transaction is similar to the settlement transaction, except that:

- It burns the head's beacon token.
- It spends the head's treasury utxo without outputting a new treasury utxo.
- It does not absorb any deposits.
- It withdraws all utxos in the final L2 block's ledger, including those in the active utxo set, using rollout transactions if necessary.

After the finalization transaction deals with the treasury and immediate/post-dated refund transactions deal with the unabsorbed deposits, no trace of the Hydrozoa head should be left in the L1 ledger. Any other utxos that remain at the head's native script address—assumed to be unrelated to the head—can be manually spent by the peers.

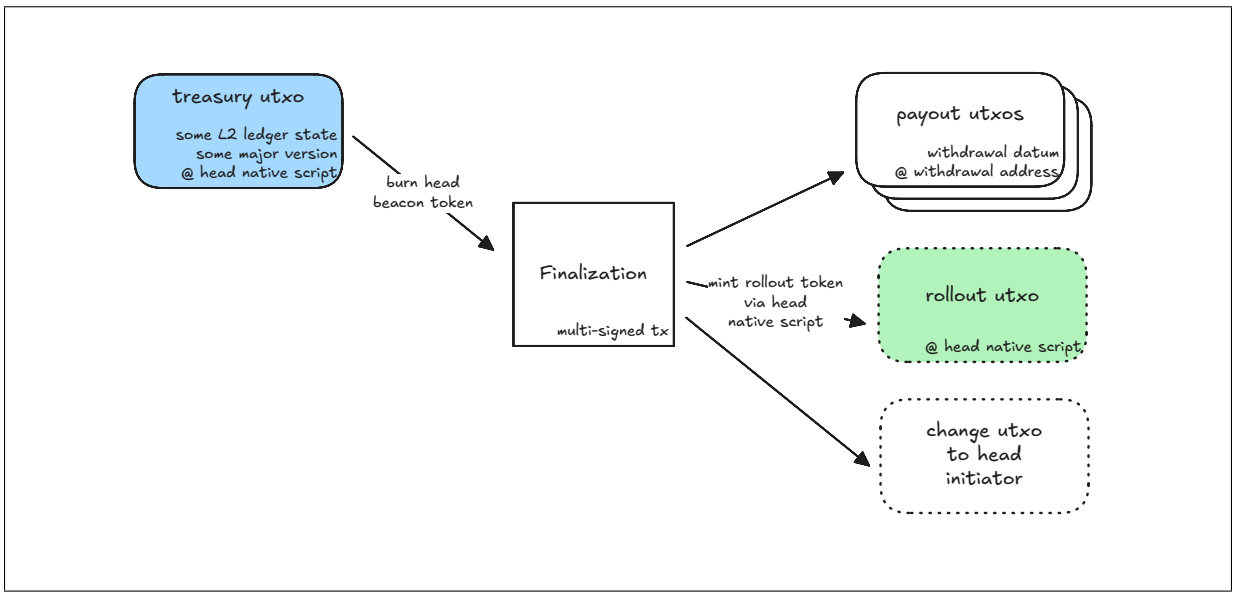

Figure 1.6: The peers pay out all remaining funds from the head's treasury. A rollout utxo can be produced to defer the payout of some withdrawals for later.

1.7 Rollout

The rollout utxo is spent in a rollout transaction, paying out some more withdrawals, and outputting a new rollout utxo with the aggregate funds of the rest of the withdrawals, if any remain. Further rollout transactions spend the shrinking rollout utxo, until all the withdrawals have been paid out (§ B.3).

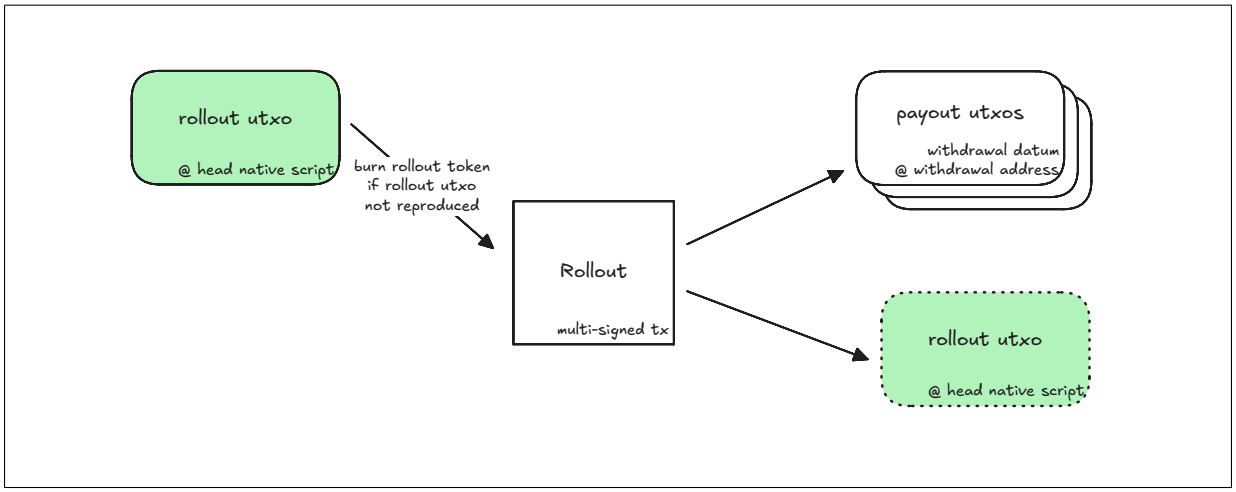

Figure 1.7: The peers pay out some withdrawals out of a rollout utxo, possibly leaving some for later payout.

Chapter 2

L1 rule-based regime

If the peers stop responding to each other in the L2 consensus protocol, the L1 rule-based regime ensures they can still withdraw their funds from the Hydrozoa head's treasury. It is less cost-efficient and more complex than the multisig regime but serves as a fallback when L2 consensus breaks down.

The fallback from the multisig regime to the rule-based regime pins the L1 treasury's major version, disqualifying any L2 blocks with lesser or greater major versions.[1] The rule-based regime then proceeds via the following suite of Plutus scripts:

- The dispute resolution scripts (§ 2.3) arbitrate the peers' dispute about the latest confirmed L2 block matching the L1 treasury's major version.
- The treasury spending validator (§ 2.4) awaits resolution and then manages withdrawals from the head's treasury based on the resolved L2 block.

The rule-based regime does not manage unabsorbed deposits left over from the multisig regime. Depositors can retrieve them with post-dated refund transactions (§ 1.4) from the multisig regime.

2.1 Utxo state

In the rule-based regime, the head's L1 state resides in utxos at two different UPLC-script addresses:

$$\mathsf{RuleBasedHeadState}^{\mathsf{L1}} := \left\{ \begin{array}{lcl} \mathsf{treasuryUtxo} & :: & (\mathsf{OutputRef}, \mathsf{Output}^{\mathsf{L1}}) \\ \mathsf{voteUtxos} & :: & \mathsf{UtxoSet}^{\mathsf{L1}} \end{array} \right\}$$

[1] Blocks with lesser major versions are disqualified because they assume a total treasury balance of funds that may no longer exist due to deposits and withdrawals adding/removing funds from the treasury. Blocks with greater major versions are disqualified because the fallback to the rule-based regime precludes their settlement transactions from being confirmed on L1 (utxo contention over the multisig treasury).

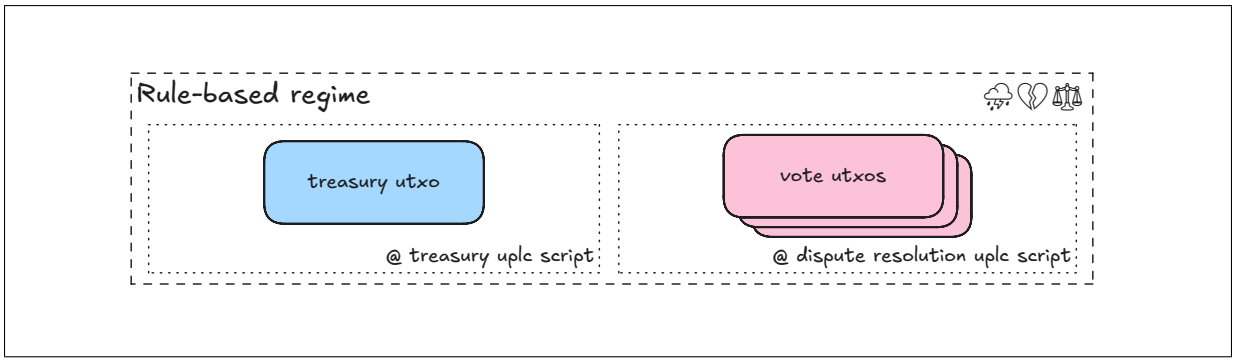

Figure 2.1: The L1 utxo state of the rule-based regime.

2.1.1 Treasury state

In the rule-based regime, the head's treasury utxo is at the head's Plutus-based treasury address (§ 2.4). It holds the head's beacon token and all treasury funds from the multisig regime (§ 1.1).

The treasury utxo must have the following datum type, which defines its unresolved and resolved states in the rule-based regime:

$$
\begin{aligned}
\text{RuleBasedTreasuryDatum} &:= \text{Resolved ResolvedDatum} \\
&\quad \mid \text{Unresolved UnresolvedDatum} \\
\text{ResolvedDatum} &:= \left\{ \begin{array}{lcl} \text{headMp} & :: & \text{PolicyId} \\ \text{utxosActive} & :: & \mathcal{RH}_{32}\ \text{UtxoSet}^{L2} \\ \text{version} & :: & (\text{UInt}, \text{UInt}) \\ \text{params} & :: & \mathcal{H}_{32}\ \text{ParamsConsensusL2} \end{array} \right\} \\
\text{UnresolvedDatum} &:= \left\{ \begin{array}{lcl} \text{headMp} & :: & \text{PolicyId} \\ \text{disputeId} & :: & \text{AssetName} \\ \text{peers} & :: & [\text{VerificationKey}] \\ \text{peersN} & :: & \text{UInt} \\ \text{deadlineVoting} & :: & \text{PosixTime} \\ \text{versionMajor} & :: & \text{UInt} \\ \text{params} & :: & \mathcal{H}_{32}\ \text{ParamsConsensusL2} \end{array} \right\}
\end{aligned}
$$

The resolved treasury's datum is the same as the multisig treasury's datum, with these exceptions:

head MP. The policy ID of the head's native script minting policy.

version. The full version of the resolved L2 block, which includes both the major and minor version numbers. It replaces the `versionMajor` field of the multisig treasury.

The unresolved treasury's datum fields are interpreted as follows:

head MP. The policy ID of the head's native script minting policy.

dispute ID. The unique identifier used as the asset name of the dispute tokens (see below).

peers. The peers' public keys.

n peers. The number of peers in the head.

deadline voting. A POSIX time before which votes can be cast in the dispute.

major version. The treasury's major version, pinned by the fallback to the rule-based regime.

params. A 32-byte Blake2b-256 hash ($\mathcal{H}_{32}$) of the head's L2 consensus parameters (§ 4.1).

2.1.2 Dispute state

Besides the treasury utxo, the head's state in the rule-based regime also includes the vote utxos for the dispute. Each of these utxos is at the head's Plutus-based dispute address and holds one or more of the dispute's fungible tokens:

- The policy ID corresponds to the head's native script (in minting policy form) (§ 1).
- The first four bytes of the asset name are equal to the CIP-67 [1] prefix for class label 3477, which corresponds to Hydrozoa head dispute tokens.[2]
- The last 28 bytes of the asset name are equal to the Blake2b-224 hash ($\mathcal{H}_{28}$) of the multisig treasury utxo spent in the head's fallback to the rule-based regime (§ 2.2).

Each vote utxo has this datum type:

$$\text{VoteDatum} := \left\{ \begin{array}{lll} \text{key} & :: & \text{UInt} \\ \text{link} & :: & \text{UInt} \\ \text{peer} & :: & \text{Maybe PubKeyHash} \\ \text{voteStatus} & :: & \text{VoteStatus} \end{array} \right\}$$

$$\text{VoteStatus} := \text{NoVote} \mid \text{Vote VoteDetails}$$

$$\text{VoteDetails} := \left\{ \begin{array}{lll} \text{utxosActive} & :: & \mathcal{RH}_{32}\ \text{UtxoSet}^{L2} \\ \text{versionMinor} & :: & \text{UInt} \end{array} \right\}$$

These fields are interpreted as follows:

key. An integer that uniquely identifies the vote utxo.

link. An integer that uniquely references another vote utxo by its `key`.

peer. The public-key hash of the peer (if any) who can cast a vote in this utxo. It is only empty for the default vote utxo (with `key` 0), which contains a vote that is automatically cast at the fallback to the rule-based regime (§ 2.2).

vote status. Either no vote or a vote for a minor L2 block.

utxos active. A 32-byte Merkle root hash ($\mathcal{RH}_{32}$) of the L2 ledger's active utxo set, as of the block voted by the peer.

minor version. The minor version of the minor L2 block voted by the peer. This block's major version must match the major version that the treasury had when it fell back to the rule-based regime.

Collectively, the vote utxos constitute a linked list data structure on L1.

[2] We chose 3477 because it spells "DISP" (short for dispute) on a phone dial pad.

2.2 Fallback to rule-based regime

The L2 consensus protocol pairs every multisig treasury utxo it creates with a post-dated fallback the treasury to the rule-based regime (§ 5).

While L2 consensus holds, the peers store these post-dated transactions without submitting them. However, if the multisig utxo is not spent by the time its corresponding post-dated transaction becomes valid, then L2 consensus is assumed to have stalled. In that case, every peer should trigger the fallback to the rule-based regime by submitting the transaction to Cardano.

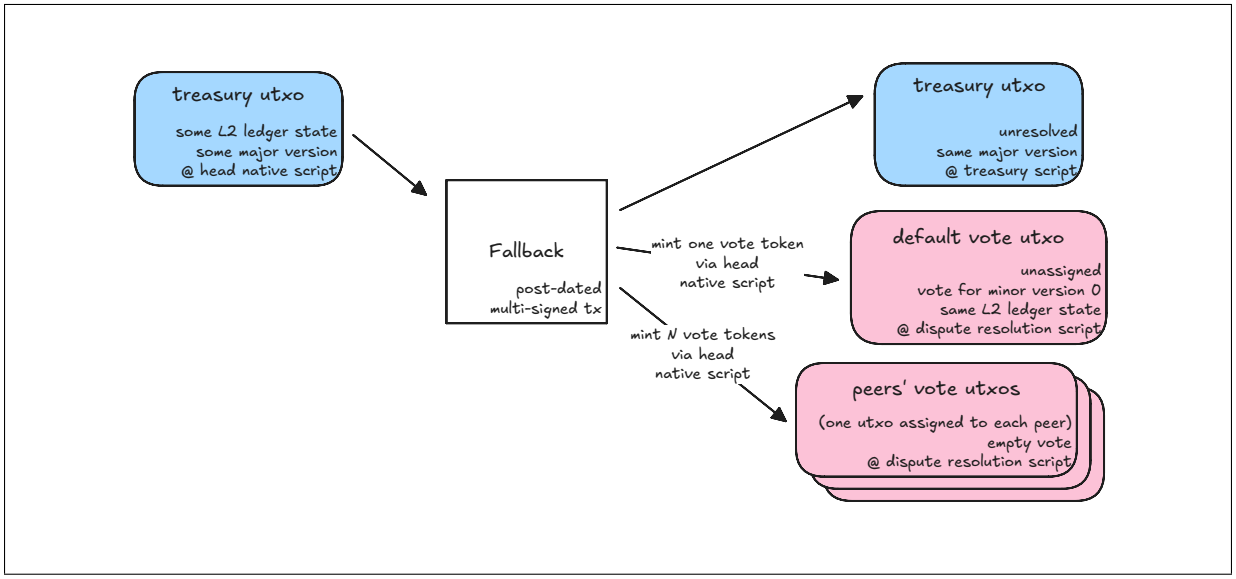

Figure 2.2: The peers arrange for the head to fall back to the rule-based regime after a timeout, in case they fail to agree on any new settlement transactions to keep the multisig regime alive.

The transaction body is as follows:

1. Spend the multisig treasury utxo (`multisigTreasury`). Let `mt` be its datum.
2. Let `headMp` be the head's native script minting policy.
3. Let `deadlineVoting` be the voting deadline for this dispute, as defined by the L2 consensus for this fallback to the rule-based regime.
4. Let `disputeId` be the CIP-67 prefix for 3477, concatenated with the Blake2b-224 hash ($\mathcal{H}_{28}$) of the `multisigTreasury` output reference.
5. Let `peers` be a list of the peers' public keys and `peersN` be its length.[3]
6. Mint (`peersN` + 1) dispute tokens of (`headMp`, `disputeId`).

[3]The indices of the `peers` list (and all other lists in this specification) start from 1.

7. Create the rule-based treasury utxo (`ruleBasedTreasury`), containing the funds and beacon token from `multisigTreasury` and this datum:

$$\text{FallbackTreasuryDatum} := \text{Unresolved} \left\{ \begin{array}{lcl} \text{headMp} & = & \text{headMp} \\ \text{disputeId} & = & \text{disputeId} \\ \text{peers} & = & \text{peers} \\ \text{peersN} & = & \text{peersN} \\ \text{deadlineVoting} & = & \text{deadlineVoting} \\ \text{versionMajor} & = & \text{mt.majorVersion} \\ \text{params} & = & \text{mt.params} \end{array} \right\}$$

8. Create the default vote utxo, containing a vote for the major L2 block's active utxo set:

$$\text{defaultVoteDatum} := \left\{ \begin{array}{lcl} \text{key} & = & 0 \\ \text{link} & = & 0 < \text{peersN} \ ?\ 1 : 0 \\ \text{peer} & = & \text{Nothing} \\ \text{voteStatus} & = & \text{Vote} \left\{ \begin{array}{lcl} \text{utxosActive} & = & \text{mt.utxosActive} \\ \text{versionMinor} & = & 0 \end{array} \right\} \end{array} \right\}$$

9. For i iterated from 1 to `peersN`, create a vote utxo with one dispute token and this datum:

$$\text{initVoteDatum } i := \left\{ \begin{array}{lcl} \text{key} & = & i \\ \text{link} & = & i < \text{peersN} \ ?\ (i+1) : 0 \\ \text{peer} & = & \text{Just } (\mathcal{H}_{32}\ \text{peers}[i]) \\ \text{voteStatus} & = & \text{NoVote} \end{array} \right\}$$

2.3 Dispute resolution

During the voting period of a dispute, each peer gets one opportunity to cast their vote on the latest minor L2 block confirmed by the L2 consensus protocol before it stalled. When all votes are cast or the voting period ends, the dispute's state can be resolved to a single vote utxo containing the latest block among the votes.

Voting is handled by the Plutus-based spending validator of the dispute address. We describe its redeemers in the rest of this section.

2.3.1 Vote

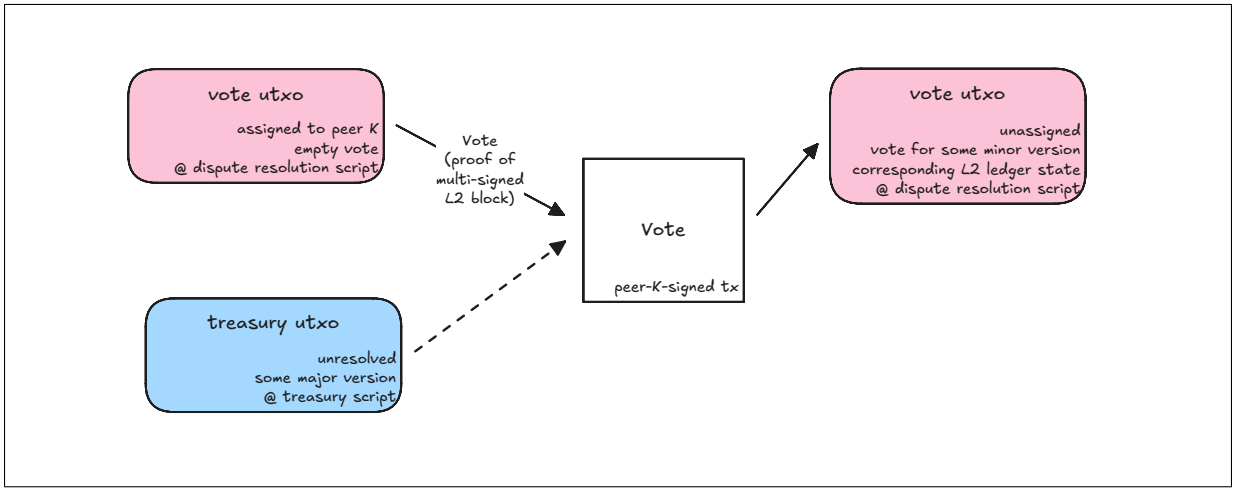

Figure 2.3: A peer casts its vote for the latest confirmed L2 block version from the multisig regime.

Replace the input's no-vote status with an abstention or a valid vote for a block. Conditions:

- Verify the vote inputs:
  1. Let `voteOutref` be the output reference of the input being spent.
  2. Let `voteInput` be the input being spent. There must not be any other spent input matching `voteOutref` on transaction hash.
  3. The `voteStatus` field of `voteInput` must be `NoVote`.
  4. The transaction must be signed by the `peer` field of `voteInput`, if it is non-empty.
  5. Let (`headMp`, `disputeId`) be the minting policy and asset name of the only non-ADA tokens in `voteInput`.
- Verify the treasury reference input:
  6. Let `treasury` be the only reference input matching `voteOutref` on tx hash.
  7. A head beacon token of `headMp` and CIP-67 prefix `4937` must be in `treasury`.
  8. `headMp` and `disputeId` must match the corresponding fields of the `Unresolved` datum in `treasury`.
  9. The transaction's time-validity upper bound must not exceed the `deadlineVoting` field of `treasury`.
- Verify the redeemer:

10. Let `voteRedeemer` be an *optional* redeemer argument with this type:[4]

$$\text{MinorBlockL1Effect} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader}^{L2} \\ \text{multisig} & :: & [\text{Signature}] \end{array} \right\}$$

$$\text{BlockHeader}^{L2} := \left\{ \begin{array}{lll} \text{blockNum} & :: & \text{UInt} \\ \text{blockType} & :: & \text{BlockType}^{L2} \\ \text{timeCreation} & :: & \text{PosixTime} \\ \text{versionMajor} & :: & \text{UInt} \\ \text{versionMinor} & :: & \text{UInt} \\ \text{utxosActive} & :: & \mathcal{RH}_{32}\ \text{UtxoSet}^{L2} \end{array} \right\}$$

11. If `voteRedeemer` is provided, then both of these must hold:

    (a) The `multisig` field of `voteRedeemer` must have signatures of the `blockHeader` field of `voteRedeemer` for all the public keys in the `peers` field of `treasury`.

    (b) The `versionMajor` field must match between `treasury` and `voteRedeemer`.

- Verify the vote output:

12. Let `voteOutput` be an output with the same number of (`headMp`, `disputeId`) tokens and same address as `voteInput`. It must not hold any other non-ADA tokens.

13. If `voteRedeemer` is provided, the `voteStatus` field of `voteOutput` must be a `Vote` matching `voteRedeemer` on the `utxosActive` and `versionMinor` fields.

14. All other fields of `voteInput` and `voteOutput` must match.

2.3.2 Tally

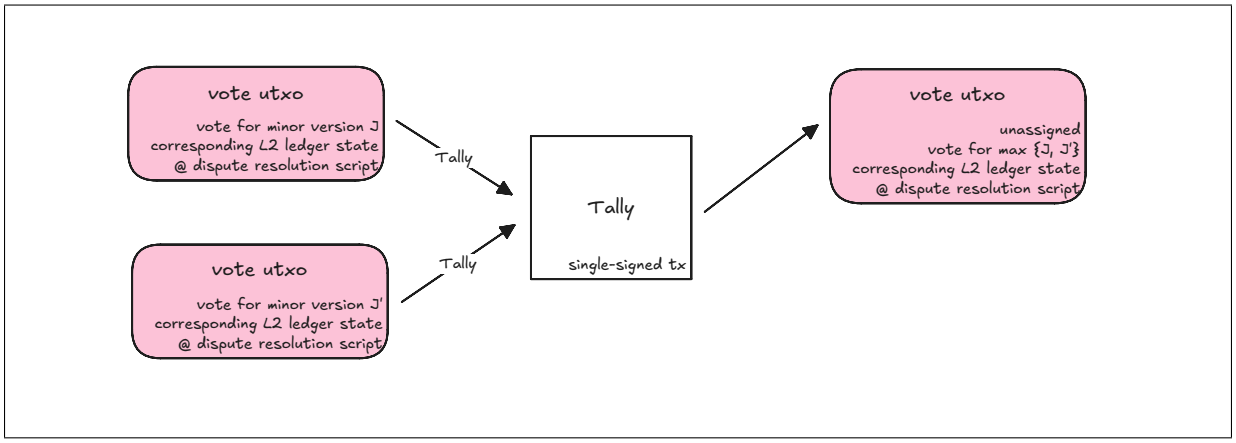

Figure 2.4: Collapse two votes into one, keeping the latest confirmed L2 block version among them.

Collapse two votes into one, keeping the latest confirmed L2 block version among them. Conditions:

- Verify the vote inputs:

[4] These types are defined in § 4 and repeated here.

1. Let `tallyRedeemer` be a redeemer argument with this type:

$$\text{TallyRedeemer} := \left\{ \begin{array}{lll} \text{continuingOutref} & :: & \text{OutputRef} \\ \text{removedOutref} & :: & \text{OutputRef} \end{array} \right\}$$

2. Let `continuingInput` and `removedInput` be spent inputs with the respective output references of `tallyRedeemer`.
3. `continuingInput` and `removedInput` must match on address.
4. `continuingInput` and `removedInput` must have non-ADA tokens of only one asset class, which must match between them and satisfy both of the following:
   (a) The minting policy (`headMp`) must be a native script.
   (b) The asset name (`disputeId`) must have the CIP-67 prefix `3477`.
5. The `key` field of `removedInput` must be greater than the `key` field and equal to the `link` field of `continuingInput`.
6. There must be no other spent inputs from the same address as `continuingInput` or holding any tokens of `headMp`.

- Verify the treasury reference input:

7. If the `voteStatus` of either `continuingInput` or `removedInput` is `NoVote`, all of the following must be satisfied:
   (a) Let `treasury` be a reference input holding the head beacon token of `headMp` and CIP-67 prefix `4937`.
   (b) `headMp` and `disputeId` must match the corresponding fields of the `Unresolved` datum in `treasury`.
   (c) The `deadlineVoting` field of `treasury` must not exceed the transaction's time-validity lower bound.

- Verify the vote output:

8. Let `continuingOutput` be an output with the same address and the sum of all tokens (including ADA) in `continuingInput` and `removedInput`.
9. The `voteStatus` field of `continuingOutput` must match the highest `voteStatus` of `continuingInput` and `removedInput`, totally ordered as follows:

$$\begin{aligned} &\forall a, b \in \text{VoteDetails} : \\ &\quad a.\text{minorVersion} \leq b.\text{minorVersion} \implies \\ &\qquad \text{NoVote} < \text{Vote } a \leq \text{Vote } b \end{aligned}$$

10. The `link` field of `removedInput` and `continuingOutput` must match.
11. All other fields of `continuingInput` and `continuingOutput` must match.

Branch: (None) @ c28633a • Commit date: 2025-10-16

2.3.3 Resolve

The sole remaining vote utxo can be spent when the treasury utxo is spent. Conditions:

1. Let `voteInput` be the input being spent.
2. Let (`headMp`, `disputeId`) be the minting policy and asset name of the only non-ADA tokens in `voteInput`.
3. Let `treasury` be a spent input that holds a head beacon token of `headMp` and CIP-67 prefix `4937`.
4. `headMp` and `disputeId` must match the corresponding fields of the `Unresolved` datum in `treasury`.

2.4 Treasury

The treasury spending validator awaits the dispute's resolution and adopts the active utxo set of the resolved L2 block. Peers can then withdraw funds from the treasury by creating utxos on L1 and showing Merkle proofs of equivalent utxos being removed from the treasury's active utxo set.

We describe the treasury spending validator's redeemers in the rest of this section.

2.4.1 Resolve

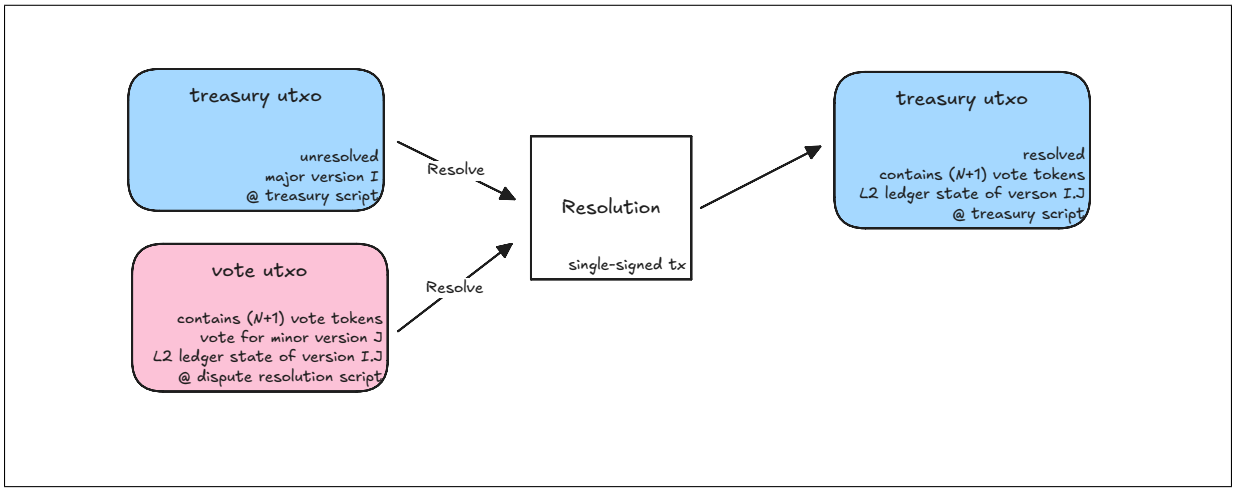

Figure 2.5: Resolve the peers' dispute by applying the latest confirmed L2 block to the treasury utxo. Funds will now be withdrawable based on that block's L2 ledger state.

Resolve the treasury's state based on the sole remaining vote utxo. Conditions:

1. Let `treasuryInput` be the input being spent.
2. Let `headMp`, `disputeId`, `peersN`, and `versionMajor` be the corresponding fields of `treasuryInput`.
3. Let `voteInput` be a spent input with (`peersN` + 1) tokens of (`headMp`, `disputeId`) and no other non-ADA tokens.

4. Let `treasuryOutput` be an output with the same address as `treasuryInput` and the sum of all tokens (including ADA) in `treasuryInput` and `voteInput`.
5. Let `voteStatus` be the corresponding field of `voteInput`. `voteStatus` must be `Vote`.
6. Let `versionMinor` be the corresponding field in `voteStatus`.
7. The `version` field of `treasuryOutput` must match (`versionMajor`, `versionMinor`).
8. `voteStatus` and `treasuryOutput` must match on `utxosActive`.
9. `treasuryInput` and `treasuryOutput` must match on all other fields.

2.4.2 Withdraw

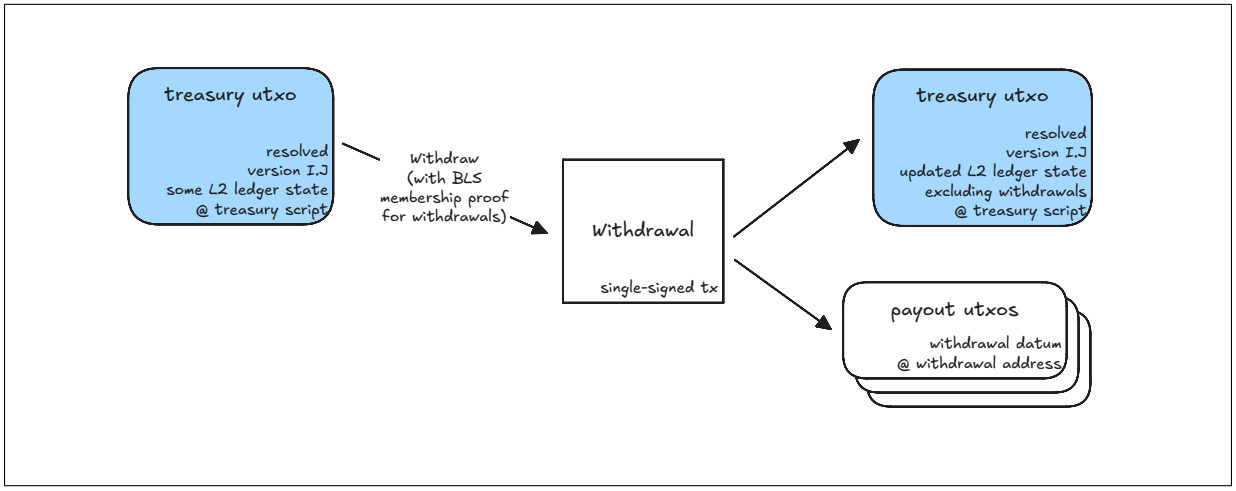

Figure 2.6: Withdraw funds from the treasury via membership proof of utxos in the resolved L2 ledger state. Update the resolved L2 ledger state to exclude those utxos.

Withdraw treasury funds and update its active utxo set accordingly. Conditions:

1. Let `treasuryInput` be the input being spent.
2. Let `headMp` be the corresponding field in `treasuryInput`.
3. Let `headId` be the asset name of the only `headMp` token in `treasuryInput` with CIP-67 prefix `4937`.
4. Let `withdrawals` be a list of triples provided as a redeemer argument. Each triple contains an output reference, an L2 output, and a Merkle inclusion proof. The proof shows a path from some Merkle root hash to the L2 output at the output reference.

$$\left[\left(\text{OutputRef},\ \text{Output}^{\text{L2}},\ \text{MerkleProof}\right)\right]$$

5. Let `treasuryOutput` be the first transaction output. It must hold the (`headMp`, `headId`) beacon token.
6. Let `outputsRemaining` be a mutable variable initialized to the transaction's outputs, excluding `treasuryOutput`.
7. Let `valueTotal` be a mutable variable initialized to the value of `treasuryOutput`.

8. Let `rootHash` be a mutable variable initialized to the `utxosActive` of `treasuryInput`.
9. For each triple `(outref, outputL2, proof)` in `withdrawals`:
   (a) Let `outputL1` be the first element in `outputsRemaining`, which must exist.
   (b) `outputL1` and `outputL2` must match on the `value`, `datum`, and `script` fields.
   (c) `outputL1` and `outputL2` must match on the `addr` field by payment credential and staking reference.
   (d) Update `outputsRemaining` by excluding `outputL1`.
   (e) Update `valueTotal` by adding the value of `outputL1`.
   (f) Update `rootHash` by applying the Merkle library's `delete` function with the triple:

$$\text{delete} :: \mathcal{RH}_{32}\ \text{UtxoSet}^{L2} \to (\text{OutputRef}, \text{Output}^{L2}, \text{Proof}) \to \mathcal{RH}_{32}\ \text{UtxoSet}^{L2}$$

10. `valueTotal` must match the value of `treasuryInput`.
11. `rootHash` must match the `utxosActive` of `treasuryOutput`.

2.4.3 Deinit

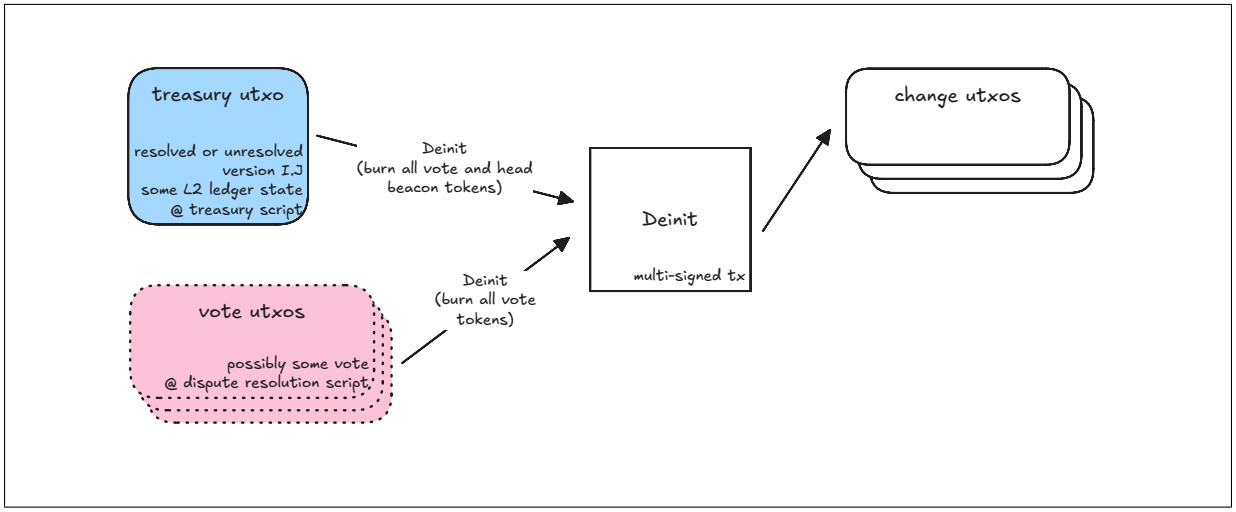

Figure 2.7: The peers deinitialize the head's treasury and clean-up any remnants of head state.

De-initialize the head, removing all traces of it. Conditions:[5]

1. Let `treasury` be the input being spent.
2. Let `headMp` be the corresponding field in `treasury`.
3. All `headMp` tokens in `treasury` with CIP-67 prefixes of 3477 and 4937 must be burned.

[5]This redeemer does not require the treasury's active utxo set to be empty, but it implicitly requires the transaction to be multi-signed by all peers to burn the `headMp` tokens. Thus, the peers can use this redeemer to override the treasury's spending validator with their multi-signature.

Chapter 3

L2 ledger

The L2 ledger state has this type:

$$\text{LedgerState}^{\text{L2}} := \{\ \text{utxosActive} \quad :: \quad \text{UtxoSet}^{\text{L2}}\ \}$$

These fields are interpreted as follows:

utxos active. The set of utxos (§ 3.2) that can be spent in transactions or withdrawn. These utxos are created by L2 transactions and genesis events.

3.1 Ledger rules

Hydrozoa's L2 ledger rules are the same as Cardano's ledger rules,[1] but with the following additions:

No staking or governance actions. Hydrozoa's consensus protocol is not based on Ouroboros proof-of-stake, and its governance protocol is not based on Cardano's hard-fork-combinator update mechanism. Furthermore, Hydrozoa's L1 scripts cannot authorize arbitrary staking or governance actions on behalf of users. For this reason, staking and governance actions cannot be used in L2 transactions.[2]

No pre-Conway features. Hydrozoa does not need to maintain backward compatibility with pre-Conway eras. L2 utxos cannot use bootstrap addresses or Shelley addresses with Plutus versions older than Plutus V3.

Public key hash credentials, native scripts, and Plutus scripts at or above V3 are allowed.

No datum hashes in outputs. Hydrozoa requires all utxo datums to be inline, which avoids the need to index the datum-hash to datum map from transactions' datum witnesses.

No minting or burning of tokens. Hydrozoa's L1 scripts are not authorized to mint or burn arbitrary user tokens onchain, so enforcing such events is impossible when settling the L2 state on L1. Thus, L2 transactions cannot mint or burn tokens.[3]

[1] This specification is pinned to this version of the Cardano ledger rules: [7].

[2] In future work (§ A.3.1), Hydrozoa may implement support on L2 for observer scripts (CIP-112 [2]) as a better alternative for dApps that currently rely on withdraw-zero staking validators for performance optimization.

[3] In future work (§ A.2), Hydrozoa may implement support for minting and burning on L2.

No collateral on L2. Phase-2 scripts (e.g. Plutus) can potentially have arbitrary resource costs (up to the Cardano's ExUnits limits for txs), but block producers do not receive normal fees from transaction that fail phase-2 validation. To cover the resource costs incurred for such failing phase-2 validations and to prevent denial-of-service attacks using such transactions, Cardano requires collateral to be provided in all transactions containing phase-2 scripts.

However, in Hydrozoa's unanimous consensus protocol, any peer can deny service simply by stopping its responses to L2 messages. Furthermore, a peer can disconnect from the source of too many transactions failing phase-2 validation, mitigating the incurred resource costs. Therefore, Hydrozoa does not need collateral inputs, fees, and outputs on L2.

POSIX and time-validity intervals. Hydrozoa uses POSIX for time-validity interval bounds on L2. An L2 transaction or withdrawal can be included in a block's valid events if the block's creation time falls within the transaction or withdrawal's time-validity interval.

3.2 Utxo set

Conceptually, a utxo set is a set of transaction outputs tagged with unique references to their origins. In practice, we equivalently represent a utxo set as a map from output reference to output:

$$\begin{aligned}\mathsf{UtxoSet}^{\mathsf{L2}} &:= \mathsf{Map\ OutputRef\ Output}^{\mathsf{L2}} \\ &:= \left\{(k_i :: \mathsf{OutputRef}, v_i :: \mathsf{Output}^{\mathsf{L2}}) \;\middle|\; \forall i \neq j.\ k_i \neq k_j\right\}\end{aligned}$$

An output reference is a tuple that uniquely identifies an output by a hash of the transaction that created it and its index among that transaction's outputs:

$$\mathsf{OutputRef} := \left\{ \begin{array}{lcl} \mathsf{id} & :: & \mathsf{TxId} \\ \mathsf{index} & :: & \mathsf{UInt} \end{array} \right\}$$

An output is a tuple describing a bundle of tokens, data, and a script that have been placed at an address in the ledger:

$$\mathsf{Output}^{\mathsf{L2}} := \left\{ \begin{array}{lcl} \mathsf{addr} & :: & \mathsf{Address}^{\mathsf{L2}} \\ \mathsf{value} & :: & \mathsf{Value} \\ \mathsf{datum} & :: & \mathsf{Datum}^{\mathsf{L2}} \\ \mathsf{script} & :: & \mathsf{Maybe\ Script}^{\mathsf{L2}} \end{array} \right\}$$

$$\mathsf{Address}^{\mathsf{L2}} := \left\{ \begin{array}{l} \mathsf{Network}^{\mathsf{L2}} \\ \mathsf{PaymentCredential} \\ \mathsf{StakeReference} \end{array} \right\}$$

$$\mathsf{Datum}^{\mathsf{L2}} := \mathsf{NoDatum} \mid \mathsf{Datum\ Data}$$

$$\begin{aligned}\mathsf{Script}^{\mathsf{L2}} &:= \mathsf{TimelockScript\ Timelock} \\ &\quad \mid \mathsf{PlutusScript\ PlutusVersion}^{\mathsf{L2}}\ \mathsf{PlutusBinary}\end{aligned}$$

$$\mathsf{PlutusVersion}^{\mathsf{L2}} := \mathsf{PlutusV3}$$

3.3 Transaction

An L2 transaction spends one or more utxos in `utxosActive`, replacing them with zero or more new utxos. The rules for Hydrozoa's L2 transactions are the same as Cardano's ledger rules for L1 transactions but with the additions listed in § 3.1.

Conveniently, we can avoid forking Cardano's ledger rules library to implement these additional rules in Hydrozoa. Instead, we express them as requirements for specific transaction fields to be omitted or restricted to a narrower selection of values.

The hydrozoa node software should reject (with an error message) L2 transactions that violate these requirements, while peers should ignore `ReqTx` messages (§ 5.3.1) that broadcast such transactions.

In the following type definition, we use these notational conventions:

- Some fields are already optional in Cardano transactions, with suitable defaults provided when omitted. We indicate these by prefixing their field types with a question mark (?).
- We indicate fields that must be omitted in Hydrozoa L2 transactions by prefixing their field names with an empty-set symbol (∅).

$$\mathsf{Tx}^{\mathsf{L2}} := \left\{ \begin{array}{lll} \mathsf{body} & :: & \mathsf{TxBody}^{\mathsf{L2}} \\ \mathsf{wits} & :: & \mathsf{TxWits}^{\mathsf{L2}} \\ \mathsf{is\_valid} & :: & \mathsf{Bool} \\ \mathsf{auxiliary\_data} & :: & ?\ \mathsf{TxMetadata} \end{array} \right\}$$

$$\mathsf{TxBody}^{\mathsf{L2}} := \left\{ \begin{array}{lll} \mathsf{spend\_inputs} & :: & \mathsf{Set\ OutputRef} \\ \varnothing\ \mathsf{collateral\_inputs} & :: & ?\ \mathsf{Set\ OutputRef} \\ \mathsf{reference\_inputs} & :: & ?\ \mathsf{Set\ OutputRef} \\ \mathsf{outputs} & :: & [\mathsf{Output}^{\mathsf{L2}}] \\ \varnothing\ \mathsf{collateral\_return} & :: & ?\ \mathsf{Output}^{\mathsf{L2}} \\ \varnothing\ \mathsf{total\_collateral} & :: & ?\ \mathsf{Coin} \\ \varnothing\ \mathsf{certificates} & :: & ?\ [\mathsf{Set\ Certificate}] \\ \varnothing\ \mathsf{withdrawals} & :: & ?\ \mathsf{Map\ RewardAccount\ Coin} \\ \mathsf{fee} & :: & \mathsf{Coin} \\ \mathsf{validity\_interval} & :: & ?\ \mathsf{ValidityInterval} \\ \mathsf{required\_signer\_hashes} & :: & ?\ [\mathsf{VKeyCredential}] \\ \varnothing\ \mathsf{mint} & :: & ?\ \mathsf{Value} \\ \mathsf{script\_integrity\_hash} & :: & ?\ \mathsf{ScriptIntegrityHash} \\ \mathsf{auxiliary\_data\_hash} & :: & ?\ \mathsf{AuxiliaryDataHash} \\ \mathsf{network\_id} & :: & ?\ \mathsf{Network}^{\mathsf{L2}} \\ \varnothing\ \mathsf{voting\_procedures} & :: & ?\ \mathsf{VotingProcedures} \\ \varnothing\ \mathsf{proposal\_procedures} & :: & ?\ \mathsf{Set\ ProposalProcedure} \\ \varnothing\ \mathsf{current\_treasury\_value} & :: & ?\ \mathsf{Coin} \\ \varnothing\ \mathsf{treasury\_donation} & :: & ?\ \mathsf{Coin} \end{array} \right\}$$

$$\mathsf{TxWits}^{\mathsf{L2}} := \left\{ \begin{array}{lll} \mathsf{addr\_tx\_wits} & :: & ?\ \mathsf{Set}\ (\mathsf{VKey}, \mathsf{Signature}, \mathsf{VKeyHash}) \\ \varnothing\ \mathsf{boot\_addr\_tx\_wits} & :: & ?\ \mathsf{Set\ BootstrapWitness} \\ \mathsf{script\_tx\_wits} & :: & ?\ \mathsf{Map\ ScriptHash\ Script}^{\mathsf{L2}} \\ \varnothing\ \mathsf{data\_tx\_wits} & :: & ?\ \mathsf{TxDats} \\ \mathsf{redeemer\_tx\_wits} & :: & ?\ \mathsf{Redeemers} \end{array} \right\}$$

Cardano's ledger rules do not allow omitting the `collateral_inputs`, `collateral_return`, and `total_collateral` fields in transactions containing Plutus scripts. Thus, we *deactivate* the following Cardano ledger rules when validating L2 transactions:

- Minimum collateral amount
- Non-empty collateral inputs (if Plutus scripts are used)
- Collateral balancing:

$$\sum \mathsf{lovelace}(\mathsf{collateral\_inputs}) = \mathsf{lovelace}(\mathsf{collateral\_return}) + \mathsf{total\_collateral}$$

3.4 Withdrawal

A withdrawal is an L2 ledger event that spends one or more utxos in `utxosActive` but does not output any utxos. The L2 ledger rules for withdrawals are similar to transactions but with the following modifications:

No outputs. Withdrawals cannot create utxos in `utxosActive`.

No fees. Withdrawals do not pay transaction fees.

No metadata. Withdrawals cannot contain transaction metadata.

We can express the rules for L2 withdrawals ($\mathtt{Tx}^{\mathtt{L2W}}$) as further restrictions on the fields of L2 transactions ($\mathtt{Tx}^{\mathtt{L2}}$):[4]

$$\mathsf{Tx}^{\mathsf{L2W}} := \left\{ \begin{array}{lll} \mathsf{body} & :: & \mathsf{TxBody}^{\mathsf{L2W}} \\ \mathsf{wits} & :: & \mathsf{TxWits}^{\mathsf{L2}} \\ \mathsf{is\_valid} & :: & \mathsf{Bool} \\ \varnothing\ \mathsf{auxiliary\_data} & :: & ?\ \mathsf{TxMetadata} \end{array} \right\} \subseteq \mathsf{Tx}^{\mathsf{L2}}$$

[4] Here, "omitting" the outputs and fee fields means setting them to an empty list and zero, respectively.

$$\mathsf{TxBody}^{\mathsf{L2W}} := \left\{ \begin{array}{lcl} \mathsf{spend\_inputs} & :: & \mathsf{Set\ OutputRef} \\ \varnothing\ \mathsf{collateral\_inputs} & :: & ?\ \mathsf{Set\ OutputRef} \\ \mathsf{reference\_inputs} & :: & ?\ \mathsf{Set\ OutputRef} \\ \varnothing\ \mathsf{outputs} & :: & [\mathsf{Output}^{\mathsf{L2}}] \\ \varnothing\ \mathsf{collateral\_return} & :: & ?\ \mathsf{Output}^{\mathsf{L2}} \\ \varnothing\ \mathsf{total\_collateral} & :: & ?\ \mathsf{Coin} \\ \varnothing\ \mathsf{certificates} & :: & ?\ [\mathsf{Set\ Certificate}] \\ \varnothing\ \mathsf{withdrawals} & :: & ?\ \mathsf{Map\ RewardAccountCoin} \\ \varnothing\ \mathsf{fee} & :: & \mathsf{Coin} \\ \mathsf{validity\_interval} & :: & ?\ \mathsf{ValidityInterval} \\ \mathsf{required\_signer\_hashes} & :: & ?\ [\mathsf{VKeyCredential}] \\ \varnothing\ \mathsf{mint} & :: & ?\ \mathsf{Value} \\ \mathsf{script\_integrity\_hash} & :: & ?\ \mathsf{ScriptIntegrityHash} \\ \mathsf{auxiliary\_data\_hash} & :: & ?\ \mathsf{AuxiliaryDataHash} \\ \mathsf{network\_id} & :: & ?\ \mathsf{Network}^{\mathsf{L2}} \\ \varnothing\ \mathsf{voting\_procedures} & :: & ?\ \mathsf{VotingProcedures} \\ \varnothing\ \mathsf{proposal\_procedures} & :: & ?\ \mathsf{Set\ ProposalProcedure} \\ \varnothing\ \mathsf{current\_treasury\_value} & :: & ?\ \mathsf{Coin} \\ \varnothing\ \mathsf{treasury\_donation} & :: & ?\ \mathsf{Coin} \end{array} \right\}$$

Cardano's ledger rules do not allow the `fee` to be zero. Thus, we *deactivate* the following Cardano ledger rules when validating L2 withdrawals, in addition to the rules deactivated for L2 transactions:

- Minimum fee amount
- Transaction balancing:

$$\sum \mathsf{value}(\mathsf{inputs}) + \mathsf{mint} = \sum \mathsf{value}(\mathsf{outputs}) + \mathsf{fee}$$

3.5 Genesis

L2 genesis is a ledger event automatically inserted into the ledger's history immediately following the affirmed events of a confirmed major L2 block. It adds new utxos to the `utxosActive` set without spending any utxos:

$$\mathsf{Genesis}^{\mathsf{L2}} := \left\{\ \mathsf{utxosAdded} \quad :: \quad \mathsf{UtxoSet}^{\mathsf{L2}}\ \right\}$$

These L2 utxos uniquely correspond to the L1 `absorbedDeposits` of the major block (§ 4):

output reference:

id. The Blake2b-256 hash ($\mathcal{H}_{32}$) of `absorbedDeposits`. Incidentally, this is also the genesis event's unique ID (`TxId`).

index. The corresponding L1 deposit's positional index in `depositsAbsorbed`.

output:

addr. The `address` field in the corresponding L1 deposit's datum.

value. The value of the corresponding L1 deposit.

datum. The `datum` field in the corresponding L1 deposit's datum.

script. Empty.

Chapter 4

L2 blocks

It would be inefficient for a head's peers to reach a consensus on each L2 ledger event individually. Instead, they reach a consensus on an entire block of L2 ledger events at every Hydrozoa L2 consensus protocol round.

All L2 blocks have this type:

$$\text{Block}^{\text{L2}} := \left\{ \begin{array}{lll} \text{header} & :: & \text{BlockHeader}^{\text{L2}} \\ \text{body} & :: & \text{BlockBody}^{\text{L2}} \end{array} \right\}$$

$$\text{BlockHeader}^{\text{L2}} := \left\{ \begin{array}{lll} \text{blockNum} & :: & \text{UInt} \\ \text{blockType} & :: & \text{BlockType}^{\text{L2}} \\ \text{timeCreation} & :: & \text{PosixTime} \\ \text{versionMajor} & :: & \text{UInt} \\ \text{versionMinor} & :: & \text{UInt} \\ \text{utxosActive} & :: & \mathcal{RH}_{32}\ \text{UtxoSet}^{\text{L2}} \end{array} \right\}$$

$$\text{BlockType}^{\text{L2}} := \text{Minor} \mid \text{Major} \mid \text{Final}$$

$$\text{BlockBody}^{\text{L2}} := \left\{ \begin{array}{lll} \text{eventsValid} & :: & \text{Sequence}\ (\text{EventType}^{\text{L2}}, \text{TxId}) \\ \text{eventsInvalid} & :: & \text{Map TxId EventType}^{\text{L2}} \\ \text{depositsAbsorbed} & :: & \text{Sequence OutputRef} \end{array} \right\}$$

$$\text{EventType}^{\text{L2}} := \text{Transaction} \mid \text{Withdrawal} \mid \text{Genesis}$$

Each block affirms a sequence of L2 ledger events, rejects a separate set of L2 ledger events, and absorbs a set of L1 deposits. The block's header contains a Merkle root hash of the active utxo set that results from applying the affirmed L2 ledger events to the latest confirmed L2 ledger state.

Depending on the types of L2 ledger events affirmed by the block and the peers' intent to continue operating the head, there are three kinds of L2 blocks in Hydrozoa:

Minor block. A block that neither affirms any L2 withdrawals nor absorbs any L1 deposits. It does not affect the head's L1 utxo state in the multisig regime and only affects the L2 ledger state's active utxo set. They may affect the L1 utxo state in the rule-based regime.

Major block. A block that affirms some L2 withdrawals or absorbs some L1 deposits. It implicitly affirms an L2 genesis event corresponding to its absorbed L1 deposits, appended to the end of its affirmed L2 event sequence. As soon as the peers confirm it, it immediately affects the

head's L1 utxo state via the L1 settlement and rollout transactions that mirror the utxos it adds and withdraws from the L2 ledger.

Final block. A block with which the peers finalize the head, resulting in an empty L2 ledger. It implicitly affirms withdrawals for the active utxo set remaining after its affirmed L2 ledger events sequence. It does not absorb any L1 deposits.

Blocks are numbered consecutively by `blockNum`, and they are versioned by `versionMajor` and `versionMinor` numbers:

- The notional initial block has major and minor versions set to zero.[1]
- A minor block keeps its predecessor's major version and increments the minor version.
- A major block increments the major version and resets the minor version to zero.
- The final block increments the major version and resets the minor version to zero.

A block's the `timeCreation` field indicates the time it was created.

4.1 Head state

Every peer requires access to an up-to-date source of this information:

$$\text{HeadState} := \left\{ \begin{array}{lll} \text{blocksConfirmedL2} & :: & \text{Sequence Block}^{L2} \\ \text{eventsL2} & :: & \text{Table Event}^{L2} \\ \text{finalizing} & :: & \text{Bool} \\ \text{params} & :: & \text{ParamsConsensusL2} \\ \text{peers} & :: & [(\text{PubKeyHash}, \text{VerificationKey})] \\ \text{stateL1} & :: & \text{MultisigHeadState}^{L1} \\ \text{stateL2} & :: & \text{LedgerState}^{L2} \\ \text{timeCurrent} & :: & \text{PosixTime} \end{array} \right\}$$

$$\text{ParamsConsensusL2} := \left\{ \begin{array}{lll} \text{blockLatencyTolerance} & :: & \text{UDiffTime} \\ \text{depositMarginMaturity} & :: & \text{UDiffTime} \\ \text{depositMarginExpiry} & :: & \text{UDiffTime} \\ \text{multisigRegimeKeepAlive} & :: & \text{UDiffTime} \\ \text{multisigRegimeTimeout} & :: & \text{UDiffTime} \end{array} \right\}$$

$$\text{Event}^{L2} := \left\{ \begin{array}{lll} \text{timeReceived} & :: & \text{PosixTime} \\ \text{eventId} & :: & \text{TxId} \\ \text{eventType} & :: & \text{EventType}^{L2} \\ \text{event} & :: & \text{Tx}^{L2} \\ \text{blockNum} & :: & \text{Maybe UInt} \end{array} \right\}$$

These fields are interpreted as follows:

blocks confirmed L2. The sequence of all L2 blocks confirmed by the L2 consensus protocol.[2]

[1] The initial block is never actually created or explicitly confirmed by the peers. It is the same for all hydrozoa heads; its only effect is initializing the empty L2 ledger state. We only care about it because its major and minor versions are zero.

[2] In practice, it may be acceptable to remove blocks when they are settled on L1 with sufficient finality.

events L2. The table of all L2 events witnessed by the peer but not rejected by any confirmed block.[3] The events must be unique on `eventId` and sorted in the ascending order of `timeReceived`:

time received. The POSIX time at which the peer received the event.

event ID. The transaction ID corresponding to the event's effect on the L2 active utxo set.

event type. The L2 event is a transaction, withdrawal, or genesis event.

event. The transaction representing the event's effect on the L2 active utxo set.

block number. The block number of the block (if any) that includes the event, regardless of whether the block is confirmed.

finalizing. A boolean indicator of whether the peers want to finalize the head immediately. Its initial value is `False`, and it becomes `True` when a peer expresses a wish to close the head.

peers. The peers' public key hashes and verification keys, which must be unique and sorted in ascending order of public key hash. Each peer's integer ID in the head is assigned according to the position of the key's public key hash in this list.

params. The head's parameters:

block latency tolerance. During validation, a new block's creation time must not deviate from the peer's current time by more than this non-negative time duration.

deposit margin maturity. After an L1 deposit is created on L1,[4] the peers must wait for this non-negative time duration before attempting to absorb it into the head's treasury.

deposit margin expiry. If no more than this non-negative time duration is left before an L1 deposit's deadline, the peers must not attempt to absorb it into the head's treasury.

multisig regime keep-alive. If this non-negative time duration has passed since the latest confirmed major block's creation, the next block must be major.

multisig regime timeout. If this non-negative time duration has passed since the latest confirmed major block's creation, the head can fall back to the rule-based regime with this block's major version.

state L1. The peer's view of the head's L1 utxo state in the multisig regime (§ 1.1).[5]

state L2. The head's L2 ledger state (§ 3) as of the latest confirmed block.

time current. The peer's current POSIX time, synchronized via the NTP protocol [9].

[3] It may be acceptable to keep only a small buffer of these events corresponding to several recently confirmed blocks.

[4] An L1 deposit's creation time corresponds to the slot number of the block that includes the transaction that outputs it. While this may be ill-defined at first due to chain forks, we expect it to stabilize by the time L1 deposit matures enough to be absorbed—that is the whole point of maturation!

[5] The head's L1 state is only needed to create/validate major and final blocks, which cannot be done when the head is in the L1 rule-based regime.

4.2 Block creation

Given a peer's head state, create a new block as follows:

1. Initialize the variables and arguments (immutable by default):
   (a) Let `block` be a mutable variable initialized to an empty $\texttt{Block}^{\texttt{L2}}$.
   (b) Let `previousBlock` be the latest block in `blocksConfirmedL2`.
   (c) Let `previousMajorBlock` be the latest major block in `blocksConfirmedL2`.
   (d) Let `utxosActive` be a mutable variable initialized to `stateL2.utxosActive`.
   (e) Let `utxosAdded` be a mutable variable initialized to an empty $\texttt{UtxoSet}^{\texttt{L2}}$.
   (f) Let `utxosWithdrawn` be a mutable variable initialized to an empty $\texttt{UtxoSet}^{\texttt{L2}}$.
2. Set `block.timeCreation` to `timeCurrent`.
3. For each non-genesis L2 event `x` in `eventsL2`, where `x.blockNum` is empty:[6]
   (a) If `x.eventType` is `Transaction`, apply `x.event` to `utxosActive` using Hydrozoa's ledger rules for L2 transactions ($\texttt{Tx}^{\texttt{L2}}$). If it is valid:
      i. Append (`x.eventId`, `x.eventType`) to `block.eventsValid`.
      ii. Update `utxosActive` to the result of this transition.
   (b) If `x.eventType` is `Withdrawal`, apply `x.event` to `utxosActive` using Hydrozoa's ledger rules for L2 withdrawals ($\texttt{Tx}^{\texttt{L2W}}$). If it is valid:
      i. Append (`x.eventId`, `x.eventType`) to `block.eventsValid`.
      ii. Update `utxosActive` to the result of this transition.
      iii. Insert the spent inputs of `x.event` into `utxosWithdrawn`.
   (c) Otherwise, insert (`x.eventId`, `x.eventType`) into `block.eventsInvalid`.
4. If `finalizing` is `False`, for each deposit `d` in `stateL1.depositUtxos`:
   (a) Skip to the next deposit if this fails to hold:

$$(\text{block.timeCreation} \geq \text{d}'.\text{timeCreation} + \text{depositMarginMaturity}) \wedge$$
$$(\text{block.timeCreation} < \text{d}'.\text{deadline} - \text{depositMarginExpiry})$$

   (b) Otherwise:
      i. Insert `d` into `block.depositsAbsorbed`.

[6] The non-genesis filter is just a self-consistency check—the L2 consensus protocol does not provide any way for a genesis event to be appended to eventsL2 with an empty blockNum. A genesis event can only be appended by a valid block.

ii. Insert this utxo into `utxosAdded`:[7]

$$\left\{ \begin{array}{lcl} \text{outputRef} & := & \left\{ \begin{array}{lcl} \text{id} & := & \mathcal{RH}_{32}\ \text{block.depositsAbsorbed} \\ \text{index} & := & \text{elemIndex d block.depositsAbsorbed} \end{array} \right\} \\ \text{output} & := & \left\{ \begin{array}{lcl} \text{addr} & := & \text{d.address} \\ \text{value} & := & \text{d.value} \\ \text{datum} & := & \text{d.datum} \\ \text{script} & := & \varnothing \end{array} \right\} \end{array} \right\}$$

5. If `finalizing` is `True`:
   - Move all utxos from `utxosActive` to `utxosWithdrawn`.
6. Set `block.blockType` according to the first among these conditions to hold:
   (a) `Final` if `finalizing` is `True`.
   (b) `Major` if `utxosAdded` is non-empty, `utxosWithdrawn` is non-empty, or:

   $$\text{block.timeCreation} \geq \text{previousMajorBlock.timeCreation} + \text{multisigRegimeKeepAlive}$$

   (c) `Minor`, otherwise.
7. Set the rest of the block header:
   (a) Set `block.blockNum` to (`previousBlock.blockNum` + 1).
   (b) Set `block.utxosActive` to the Merkle root hash of `utxosActive`.
   (c) If `block.blockType` is `Major` or `Final`, both of these:
      i. Set `block.versionMajor` to (`previousBlock.versionMajor` + 1).
      ii. Set `block.versionMinor` to zero.
   (d) If `block.blockType` is `Minor`, both of these hold:
      i. Set `block.versionMajor` to `previousBlock.versionMajor`.
      ii. Set `block.versionMinor` to (`previousBlock.versionMinor` + 1).
8. Return `block`, `utxosActive`, `utxosAdded`, and `utxosWithdrawn`.

[7] Actually, the output reference of each L2 utxo in utxosAdded can only be constructed after the iteration through stateL1.depositUtxos is complete. In practice, this may require a second pass through utxosAdded to fill in the blanks.

4.3 Block validation

From a peer's perspective, a new block's validity has three states:

Valid. The block is valid and should be confirmed by the peer.

Not yet known. The peer has detected an error that could be resolved if the peer waits and tries again—e.g., the peer may soon receive an L2 event that was included in the block.

Invalid. The peer has detected an error that cannot be resolved by waiting.

Given a peer's head state, validate a new block as follows:[8]

1. Initialize the variables and arguments (immutable by default):
   (a) Let `block` be the new block being validated.
   (b) Let `previousBlock` be the latest block in `blocksConfirmedL2`.
   (c) Let `previousMajorBlock` be the latest major block in `blocksConfirmedL2`.
   (d) Let `utxosActive` be a mutable variable initialized to `stateL2.utxosActive`.
   (e) Let `utxosAdded` be a mutable variable initialized to an empty $\texttt{UtxoSet}^{\texttt{L2}}$.
   (f) Let `utxosWithdrawn` be a mutable variable initialized to an empty $\texttt{UtxoSet}^{\texttt{L2}}$.
2. Return `Invalid` if this fails to hold:

$$\text{block.timeCreation} \in [\text{timeCurrent} \pm \text{blockLatencyTolerance})$$

3. For each event `x` in `eventsValid`:
   (a) If `x.eventId` is not in `eventsL2`, return `NotYetKnown`.
   (b) If `block.blockType` is `Minor` and `x.eventType` is `Withdrawal`, return `Invalid`.
   (c) If `x.eventType` is `Transaction`, apply `x.event` to `utxosActive` using Hydrozoa's ledger rules for L2 transactions ($\texttt{Tx}^{\texttt{L2}}$):
      i. If it is invalid, return `Invalid`.
      ii. Update `utxosActive` to the result of this transition.
   (d) If `x.eventType` is `Withdrawal`, apply `x.event` to `utxosActive` using Hydrozoa's ledger rules for L2 withdrawals ($\texttt{Tx}^{\texttt{L2W}}$):
      i. If it is invalid, return `Invalid`.
      ii. Update `utxosActive` to the result of this transition.
      iii. Insert the spent inputs of `x.event` into `utxosWithdrawn`.
   (e) If `x.eventType` is `Genesis`, return `Invalid`.
4. For each event `y` in `eventsInvalid`:

[8] In the block validation procedure, "return" statements immediately break execution.

(a) If `y.eventId` is not in `eventsL2`, return `NotYetKnown`.

(b) If `y.eventType` is `Transaction`, apply `y.event` to `utxosActive` using Hydrozoa's ledger rules for L2 transactions ($\texttt{Tx}^{\texttt{L2}}$). If it is incorrectly marked as valid, return `Invalid`.

(c) If `y.eventType` is `Withdrawal`, apply `y.event` to `utxosActive` using Hydrozoa's ledger rules for L2 withdrawals ($\texttt{Tx}^{\texttt{L2W}}$). If it is incorrectly marked as valid, return `Invalid`.

5. If `finalizing` is `False`, for each deposit `d'` in `depositsAbsorbed`:

   (a) If `d'` is not in `stateL1.depositUtxos`, return `NotYetKnown`.

   (b) Let `d` be the corresponding deposit in `stateL1.depositUtxos`.

   (c) Return `Invalid` if this fails to hold:

$$(\text{block.timeCreation} \geq \text{d.timeCreation} + \text{depositMarginMaturity}) \wedge$$
$$(\text{block.timeCreation} < \text{d.deadline} - \text{depositMarginExpiry})$$

   (d) Insert this utxo into `utxosAdded`:

$$\left\{ \begin{array}{lcl} \text{outputRef} & := & \left\{ \begin{array}{lcl} \text{id} & := & \mathcal{RH}_{32}\ \text{block.depositsAbsorbed} \\ \text{index} & := & \text{elemIndex d block.depositsAbsorbed} \end{array} \right\} \\ \text{output} & := & \left\{ \begin{array}{lcl} \text{addr} & := & \text{d.address} \\ \text{value} & := & \text{d.value} \\ \text{datum} & := & \text{d.datum} \\ \text{script} & := & \varnothing \end{array} \right\} \end{array} \right\}$$

6. If `finalizing` is `True`:

   (a) If `depositsAbsorbed` is non-empty, return `Invalid`.

   (b) Move all utxos from `utxosActive` to `utxosWithdrawn`.

7. Return `Invalid` if `block.blockType` is not set according to the first among these to hold:

   (a) `Final` if `finalizing` is `True`.

   (b) `Major` if `utxosAdded` is non-empty, `utxosWithdrawn` is non-empty, or:

$$\text{block.timeCreation} \geq \text{previousMajorBlock.timeCreation} + \text{multisigRegimeKeepAlive}$$

   (c) `Minor`, otherwise.

8. Return `Invalid` if any of these fails to hold:

   (a) `block.blockNum` matches (`previousBlock.blockNum` + 1).

   (b) `block.utxosActive` matches the Merkle root hash of `utxosActive`.

   (c) If `block.blockType` is `Major` or `Final`, both of these hold:

      i. `block.versionMajor` matches (`previousBlock.versionMajor` + 1).

      ii. `block.versionMinor` is zero.

(d) If `block.blockType` is `Minor`, both of these hold:

i. `block.versionMajor` matches `previousBlock.versionMajor`.

ii. `block.versionMinor` matches (`previousBlock.versionMinor` + 1).

9. Return `Valid`, along with `utxosActive`, `utxosAdded`, and `utxosWithdrawn`.

4.4 Block effects

A confirmed block's L2 effect is immediate, and it is handled by a common procedure for all block types.

On the other hand, a confirmed block's L1 effect varies by its block type, and a peer cannot directly apply it to its head state. Instead, it must be executed by submitting L1 transactions and waiting for the L1 Ouroboros consensus protocol to confirm the transactions.

4.4.1 Any block – L2 effect

Let `utxosActive`, `utxosAdded`, and `utxosWithdrawn` be the sets produced by the block creation or validation procedure for a given block (`block`) and head state. Upon confirmation of `block` by the peers, it immediately affects the head state as follows:

1. Update `stateL2.utxosActive` to `utxosActive`.
2. For each L2 event in `block.eventsValid`, set the `blockNum` of the corresponding event in `eventsL2` to `block.blockNum`.
3. For each L2 event in `block.eventsInvalid`, remove the corresponding event in `eventsL2`.
4. If `utxosAdded` is non-empty, then append this genesis event to `eventsL2`:

$$\text{Genesis}^{L2}\ \{\ \text{utxosAdded} := \text{utxosAdded}\ \}$$

4.4.2 Minor block – L1 effect

A confirmed minor block's L1 effect is the block's header, multi-signed by all peers:

$$\text{MinorBlockL1Effect} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader}^{L2} \\ \text{multisig} & :: & [(\text{VerificationKey}, \text{Signature})] \end{array} \right\}$$

This effect is manually triggered if a peer uses it as a redeemer to vote during the rule-based regime's dispute (§ 2.3). If the dispute resolution process resolves with this minor block, then the rule-based treasury will allow funds to be withdrawn according to this minor block's active utxo set.

4.4.3 Major block – L1 effect

A confirmed major block's L1 effect is a package of L1 transactions, multi-signed by all peers:

$$\mathsf{MajorBlockL1Effect} := \left\{ \begin{array}{lll} \mathsf{settlement} & :: & \mathsf{Tx}^{\mathsf{L1}} \\ \mathsf{rollouts} & :: & [\mathsf{Tx}^{\mathsf{L1}}] \\ \mathsf{postDatedFallback} & :: & \mathsf{Tx}^{\mathsf{L1}} \end{array} \right\}$$

These L1 transactions are as follows:

settlement. The L1 settlement transaction (§ 1.5) that absorbs the block's `absorbedDeposits` and pays out as many utxos corresponding to the block's `utxosWithdrawn` as fit within Cardano's L1 transaction constraints.

rollouts. The L1 rollout transactions (§ 1.7) that pay out utxos corresponding to the rest of the block's `utxosWithdrawn`.

post-dated fallback. An L1 transaction that transitions the multisig treasury to the rule-based regime with the block's major version (§ 2.2). The transaction's time-validity interval starts at the major block's creation time, shifted forward by the `multisigRegimeTimeout` parameter.

All peers must immediately submit the settlement and rollouts to Cardano L1.

If the post-dated fallback becomes valid, all peers must immediately submit it.

4.4.4 Final block – L1 effect

A confirmed final block's L1 effect is a package of L1 transactions, multi-signed by all peers:

$$\mathsf{FinalBlockL1Effect} := \left\{ \begin{array}{lll} \mathsf{finalization} & :: & \mathsf{Tx}^{\mathsf{L1}} \\ \mathsf{rollouts} & :: & [\mathsf{Tx}^{\mathsf{L1}}] \end{array} \right\}$$

These L1 transactions are as follows:

finalization. The L1 finalization transaction (§ 1.6) that pays out as many utxos corresponding to the block's `utxosWithdrawn` as fit within Cardano's L1 transaction constraints. It does not absorb any L1 deposits.

rollouts. The L1 rollout transactions (§ 1.7) that pay out utxos corresponding to the rest of the block's `utxosWithdrawn`.

All peers must immediately submit the finalization and rollouts to Cardano L1.

Chapter 5

L2 consensus protocol

L2 consensus between a Hydrozoa head's peers is achieved via a peer-to-peer protocol that:

- Broadcasts L2 transaction and withdrawal requests among the peers.
- Facilitates peer agreement on a growing common sequence of L2 ledger events grouped into blocks.
- Promptly settles the L1 effects of a confirmed block if it affirms any L2 withdrawals or absorbs any L1 deposits.
- Ensures that no funds get stranded as unabsorbed L1 deposits. If a deposit is determined not to be absorbed into the head, it can always be refunded to the depositor.
- Ensures that no funds get stranded in the head's treasury. If the peers stop responding to each other, they can always retrieve their funds on L1 according to the latest block matching the L1 treasury's major version.

Peers take round-robin turns to create L2 blocks—every i^{th} block must be created by the i^{th} peer.

While participating in the L2 consensus protocol, each peer locally maintains this head state:[1]

$$\text{HeadState} := \left\{ \begin{array}{lcl} \text{blocksConfirmedL2} & :: & \text{Sequence Block}^{\text{L2}} \\ \text{eventsL2} & :: & \text{Table Event}^{\text{L2}} \\ \text{finalizing} & :: & \text{Bool} \\ \text{params} & :: & \text{ParamsConsensusL2} \\ \text{peers} & :: & [(\text{PubKeyHash}, \text{VerificationKey})] \\ \text{stateL1} & :: & \text{MultisigHeadState}^{\text{L1}} \\ \text{stateL2} & :: & \text{LedgerState}^{\text{L2}} \\ \text{timeCurrent} & :: & \text{PosixTime} \end{array} \right\}$$

[1] This type is defined in § 4.1 and repeated here.

5.1 Assumptions

The L2 consensus protocol relies on these assumptions:

1. The peers have pre-established pairwise communication channels with each other.
2. Every network message received from a peer is checked for authentication. Implementing this specification requires finding a suitable method to authenticate the communication channels or individual messages.
3. Every peer is promptly and correctly notified about all relevant L1 transactions, and L1 consensus finality and liveness properties always hold.
4. The lifecycle of a Hydrozoa head never crosses any L1 protocol update boundaries.[2]

5.2 Setup

During setup, the initiator proposes the L2 consensus parameters for the head, collects the peers' verification keys, and drafts the L1 initialization transaction for the head. The other peers respond to the initiator's requests, broadcasting their responses to all peers.

When all the peers agree on this information, they submit the multi-signed initialization transaction.

5.2.1 Parameters (`ReqParams` / `AckParams`)

To start the setup process, the initiator must broadcast a request for the other peers to adopt the initiator's proposed L2 consensus parameters:[3]

$$\mathsf{ReqParams} := \{\ \mathsf{params} :: \mathsf{ParamsConsensusL2}\ \}$$

$$\mathsf{ParamsConsensusL2} := \left\{ \begin{array}{lcl} \mathsf{blockLatencyTolerance} & :: & \mathsf{UDiffTime} \\ \mathsf{depositMarginMaturity} & :: & \mathsf{UDiffTime} \\ \mathsf{depositMarginExpiry} & :: & \mathsf{UDiffTime} \\ \mathsf{multisigRegimeKeepAlive} & :: & \mathsf{UDiffTime} \\ \mathsf{multisigRegimeTimeout} & :: & \mathsf{UDiffTime} \end{array} \right\}$$

On receiving a `ReqParams`, if the receiving peer accepts the proposed parameters, the peer must update the `params` in its head state to the proposed parameters and broadcast an acknowledgment that contains the peer's verification key:

$$\mathsf{AckParams} := \{\ \mathsf{vkey} :: \mathsf{VerificationKey}\ \}$$

The initiator must do the same immediately after sending the `ReqParams` request.

On receiving a `AckParams`, the receiving peer must update the `peers` list in its head state to include the received verification key and its Blake2b-256 ($\mathcal{H}_{32}$) hash. This list must have unique verification keys and be sorted in ascending order of public key hash. Furthermore, it must always include the peer's own verification key and public key hash.

[2] These boundaries are often announced in advance, before the hard-fork combinator event, so Hydrozoa peers should be capable of avoiding them by finalizing existing heads and delaying the initialization of new heads.

[3] The L2 consensus parameters are defined in § 4.1 and repeated here.

5.2.2 Initialization (`ReqInit` / `AckInit`)

On receiving all parameter acknowledgments from the peers, the initiator must draft an L1 initialization transaction (§ 1.2) for the head, using the peers' public key hashes, and broadcast it to the peers:

$$\text{ReqInit} := \{\ \text{initTx} :: \text{Tx}^{\text{L1}}\ \}$$

On receiving a `ReqInit`, the receiving peer must verify that the initialization transaction meets the conditions in § 1.2 and uses a native script (§ 1) parametrized by all the peers' public key hashes, corresponding to the `peers` list its head state. If it is valid, the peer must broadcast a signature of the initialization transaction:

$$\text{AckInit} := \left\{ \begin{array}{lll} \text{initTxId} & :: & \text{TxId} \\ \text{signature} & :: & \text{Signature} \end{array} \right\}$$

The initiator must do the same immediately after sending the `ReqInit` request.

On receiving an `AckInit`, the receiving peer must verify that `signature` is a valid signature of the initialization transaction by the sender's verification key. If so, the peer must attach the signature to the initialization transaction.

Each peer must submit the initialization transaction when all required signatures are attached.

5.3 Ledger requests

Any peer can broadcast a request to the other peers, asking them to witness a new L2 transaction or withdrawal. The peers do not explicitly acknowledge the request, but a future block must affirm the L2 transaction or withdrawal if it is valid relative to the block creator's head state.

5.3.1 Transaction (`ReqTx`)

To submit a new L2 transaction to the head, the submitting peer must broadcast this request:

$$\text{ReqTx} := \{\ \text{tx} :: \text{Tx}^{\text{L2}}\ \}$$

On receiving a `ReqTx` at time `timeReceived`, if the transaction body hash is valid, the receiving peer must append the following L2 event to the `eventsL2` table in in its head state (§ 4.1):

$$\left\{ \begin{array}{lll} \text{timeReceived} & := & \text{timeReceived} \\ \text{eventId} & := & \text{tx.txId} \\ \text{eventType} & := & \text{Tranasction} \\ \text{event} & := & \text{tx} \\ \text{blockNum} & := & \varnothing \end{array} \right\}$$

The submitter must do the same immediately after sending the request.

No acknowledgment is required. Peers other than the next block creator should not attempt to validate this L2 transaction at this time.

5.3.2 Withdrawal (`ReqWithdrawal`)

To submit a new L2 withdrawal to the head, the submitting peer must broadcast this request:

$$\text{ReqWithdrawal} := \{\ \text{withdrawal} :: \text{Tx}^{\text{L2W}}\ \}$$

On receiving a `ReqWithdrawal` at time `timeReceived`, if the transaction body hash is valid, the receiving peer must append the following L2 event to the `eventsL2` table in its head state (§ 4.1):

$$\left\{\begin{array}{lcl} \text{timeReceived} & := & \text{timeReceived} \\ \text{eventId} & := & \text{w.txId} \\ \text{eventType} & := & \text{Withdrawal} \\ \text{event} & := & \text{w} \\ \text{blockNum} & := & \varnothing \end{array}\right\}$$

The submitter must do the same immediately after sending the request.

No acknowledgment is required. Peers other than the next block creator should not attempt to validate this L2 withdrawal at this time.

5.4 Consensus on blocks

If there are unconfirmed L2 events and no new block awaits the peers' confirmation, the next block must be created and sent to the peers for confirmation. The i^{th} peer has the exclusive right to create every i^{th} block:

$$\text{rightfulBlockCreator peersN blockNum} := \text{mod blockNum peersN}$$

To create a new block, apply the block creation procedure (§ 4.2) to the block creator's head state. Depending on the new block's type, the block creator must broadcast it in either a `ReqMinor`, `ReqMajor`, or `ReqFinal` request to the other peers.

Each recipient must validate the new block relative to the recipient's head state. If the new block is valid, the recipient must broadcast a corresponding acknowledgment (`AckMinor`, `AckMajor1`, or `AckFinal1`) to the other peers.

The block creator must broadcast a corresponding acknowledgment of the new block to the other peers immediately after broadcasting the new block.

When the new block is major or final, acknowledgment proceeds in two rounds.[4] When a peer has received the first acknowledgment (`AckMajor1` or `AckFinal1`) from all peers, the peer must broadcast a corresponding second acknowledgment (`AckMajor2` or `AckFinal2`).

When a peer receives all acknowledgments for a block, the peer must consider the block and all its affirmed events to be confirmed. The peer must update its head state as described in § 4.4.

[4] Major and final blocks require two acknowledgment rounds to ensure block confirmation because they have multiple effects requiring the peers' signatures. The first round collects signatures for all effects except for the L1 settlement/finalization transaction. The second round collects signatures for the L1 settlement/finalization transaction. Doing so prevents any peer from withholding their signature for an undesirable block effect after obtaining all signatures for a desirable block effect. Since none of the other effects can happen without the L1 settlement transaction, requiring it last ensures all peers sign all effects.

Both the `AckMinor` and `AckMajor2` responses have a `nextBlockFinal` boolean flag, with which each peer can indicate whether the peer wishes the next block (after the currently pending block) to be the head's final block. If this flag is `True` in any of the acknowledgments of a confirmed block, all peers must set the `finalizing` variable to `True` in the head state, and the next block creator must create a final block.

5.4.1 Minor block (`ReqMinor` / `AckMinor`)

A block creator's broadcast of a new minor block contains the block:

$$\text{ReqMinor} := \{\ \text{block} :: \text{Block}\ \}$$

A peer's acknowledgment of a minor block contains the block header signed by the peer:

$$\text{AckMinor} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader} \\ \text{signature} & :: & \text{Signature} \\ \text{nextBlockFinal} & :: & \text{Bool} \end{array} \right\}$$

5.4.2 Major block (`ReqMajor` / `AckMajor1` / `AckMajor2`)

A block creator's broadcast of a new major block contains the block;

$$\text{ReqMajor} := \{\ \text{block} :: \text{Block}\ \}$$

A peer's first acknowledgment of a major block identifies the block by its header and contains the recipient's signatures for the block's rollout and post-dated fallback transactions:

$$\text{AckMajor1} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader} \\ \text{rollouts} & :: & [\text{Signature}] \\ \text{postDatedFallback} & :: & \text{Signature} \end{array} \right\}$$

A peer's second acknowledgment of a major block identifies the block by its header and contains the recipient's signature for the block's settlement transaction:

$$\text{AckMajor2} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader} \\ \text{settlement} & :: & \text{Signature} \\ \text{nextBlockFinal} & :: & \text{Bool} \end{array} \right\}$$

5.4.3 Final block (`ReqFinal` / `AckFinal1` / `AckFinal2`)

A block creator's broadcast of a new major block contains the block;

$$\text{ReqFinal} := \{\ \text{block} :: \text{Block}\ \}$$

A peer's first acknowledgment of a major block identifies the block by its header and contains the recipient's signatures for the block's rollout transactions:

$$\text{AckFinal1} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader} \\ \text{rollouts} & :: & [\text{Signature}] \end{array} \right\}$$

A peer's second acknowledgment of a major block identifies the block by its header and contains the recipient's signature for the block's finalization transaction:

$$\text{AckFinal2} := \left\{ \begin{array}{lll} \text{blockHeader} & :: & \text{BlockHeader} \\ \text{finalization} & :: & \text{Signature} \end{array} \right\}$$

5.5 Consensus on refunds

To ensure that deposited funds never get stranded, every depositor must obtain a multi-signed post-dated refund transaction from the head's peers before sending the L1 deposit utxo to the head's native script address.

Similarly, the peers may sometimes determine that an L1 deposit will not be absorbed into the head—e.g., insufficient time before the deposit deadline, disruption in L2 block consensus. In those cases, the peers should attempt to multi-sign and submit an immediate refund transaction for the deposit as a courtesy to the depositor.

5.5.1 Post-dated refund (`ReqRefundLater` / `AckRefundLater`)

A request for a post-dated refund of an L1 deposit must contain the unsigned transaction that will create the L1 deposit:

$$\text{ReqRefundLater} := \left\{ \begin{array}{lll} \text{depositTx} & :: & \text{Tx}^{\text{L1}} \\ \text{index} & :: & \text{UInt} \end{array} \right\}$$

On receiving a `ReqRefundLater`, the receiving peer must:

1. Verify that the deposit transaction creates a valid L1 deposit (§ 1.3) at the specified index.
2. Construct an L1 refund transaction that spends the L1 deposit and sends its funds to the `refundAddress` and `refundDatum`, with time validity interval starting at the `deadline`.
3. Broadcast the peer's signature for the post-dated refund transaction:

$$\text{AckRefundLater} := \left\{ \begin{array}{lll} \text{deposit} & :: & \text{OutputReference} \\ \text{signature} & :: & \text{Signature} \end{array} \right\}$$

On receiving all peer signatures for the post-dated refund, the deposit caches it and submits the deposit transaction.

5.5.2 Immediate refund (`ReqRefundNow` / `AckRefundNow`)

A request for an immediate refund must contain the list of L1 deposit utxos to be refunded:

$$\text{ReqRefundNow} := \{\ \text{deposits} :: [\text{OutputReference}]\ \}$$

On receiving a `ReqRefundNow`, the receiving peer must:

1. Verify that each of the L1 deposit utxos exists on L1 and that no more than `depositMarginExpiry` time duration is left until the deposit's `deadline`.

2. For each L1 deposit, construct an L1 refund transaction that spends it and send a corresponding output to its `refundAddress` and `refundDatum`. This transaction's time-validity interval is unbounded.

3. Broadcast the peer's signatures for the immediate refund transactions:

$$\text{AckRefundNow} := \left[\left\{\begin{array}{lll} \text{deposit} & :: & \text{OutputReference} \\ \text{signature} & :: & \text{Signature} \end{array}\right\}\right]$$

On receiving all the peers' signatures for the immediate refund, every peer must submit it.

Appendix A

Future work

In this appendix, we describe some interesting further developments of the Hydrozoa protocol beyond the current specification. Of these, only adding/removing peers (§ A.1) is formally in scope for the Catalyst Fund 13 Hydrozoa project [4], but we will look for opportunities to pursue this future work where we can.

A.1 Add/remove peers

The Cardano community identified dynamic peer membership as a highly desirable feature for running Hydrozoa heads. Indeed, the ability to add or remove a peer from a head by unanimous consent would significantly boost the flexibility of operating them. It will likely improve the chances that peers stay in the multisig regime, as any peer wanting to exit could do so in an orderly fashion with minimal disruption, rather than resorting to finalization or dispute resolution.

In terms of the L1 protocol currently described in this specification (§§ 1 and 2), changing the peers list in the multisig regime[1] can be as simple as a single multi-signed transaction that:

1. Spends the head's multisig treasury utxo.
2. Burns the head's beacon token.
3. Mints a new head beacon token with a minting policy corresponding to the new peer list.
4. Sends the multisig treasury utxo input's funds and the newly minted head beacon token to the head native script address corresponding to the new peer list. The datum is the same as in the treasury input.

The main challenge of implementing this feature involves adapting the L2 protocol (§ 5) to negotiate this transition seamlessly. Pragmatically, we have decided to postpone this until the current specification is implemented. However, we are committed to implementing this feature in the final milestone of the Hydrozoa Catalyst project [4].

[1] Changing the peers list in the rule-based regime is infeasible because their inability to reach a timely consensus is the reason why the head is in this regime.

A.2 Minting on L2

Many dApps on Cardano rely on custom tokens to authenticate their internal state,[2] but Hydrozoa currently precludes minting and burning on L2. Supporting this feature opens up a range of complex issues that have not yet yielded satisfactory solutions in discussions [8].

Fortunately, the Midgard team is exploring a promising approach to this problem. The Hydrozoa team is in close contact with the Midgard team and will likely adapt Midgard's solution to L2 minting when it is more fully developed.

A.3 Observer scripts

Cardano CIP-112 [2], which is expected to arrive on Cardano mainnet in an upcoming hardfork, introduces observer scripts to Cardano's ledger rules. Observer scripts could revolutionize dApps on a scale similar to the reference inputs/scripts of the Vasil hardfork.

A.3.1 Alternative to withdraw-zero trick

Many dApps on Cardano rely on withdraw-zero staking validators to optimize performance,[3] but this specification currently precludes staking actions on L2. However, observer scripts allow dApp developers to implement the same functionality without using withdraw-zero staking validators, reducing overhead and complexity.

Instead of conjuring a complex workaround to enable staking actions on L2, Hydrozoa will wait for CIP-112 to be implemented and merged to Cardano mainnet.

A.3.2 Eliminate post-dated transactions

Currently, Hydrozoa relies on multi-signed post-dated transactions for two features:

- Refunding a deposit after its deadline (§ 2.2).
- Fallback to the rule-based regime after the multisig regime timeout (§ 2.2).

These transactions must be multi-signed because they spend utxos from the head's multisig native address. However, they are only intended as a fallback when the peers fail to reach a timely consensus on the preferred alternatives. For this reason, the L2 consensus protocol requires them to be post-dated; in this way, they can be multi-signed ahead of time while the peers can still reach a timely consensus.

Still, managing these post-dated contingencies incurs a significant communication and storage overhead for Hydrozoa heads. Luckily, observer scripts will allow Hydrozoa to replace the post-dated transactions by adding a `RequireObserver` clause to the head's native script. With such a clause, the native script can selectively forward its validation to a Plutus-based observer script.

[2] For example, see the Cardano Swaps peer-to-peer DeFi protocol [3].

[3] For example, Hydrozoa itself uses a withdraw-zero staking validator for performance optimization when tallying votes in the rule-based regime (§ 2.3).

With observer scripts, we can redefine the head's native script as follows:

$$\begin{aligned}
&\mathsf{headNativeScript} :: \mathsf{Set\ PubKeyHash} \to \mathsf{Timelock} \\
&\mathsf{headNativeScript\ keys} := \mathsf{AnyOf} \left[\begin{array}{l} \mathsf{AllOf\ (map\ Signature\ keys)} \\ \mathsf{RequireObserver\ (headObserverScriptHash\ keys)} \end{array} \right] \\
&\mathsf{headObserverScriptHash} :: \mathsf{Set\ PubKeyHash} \to \mathcal{H}_{32}\ \mathsf{PlutusScript}
\end{aligned}$$

This new observer script can classify its spent input as the treasury utxo or a deposit utxo. Then, it can decide whether to allow the rule-based fallback or refund by comparing the transaction's time validity lower bound to the datum. In other words, it allows a rule-based exit from a state managed by a native script.

A.3.3 Fragmented treasury

Cardano's ledger rules limit the serialized size of a utxo's value field based on the Alonzo-era `MaxValSize` protocol parameter. In other words, a Hydrozoa head that stores its entire treasury in a single utxo is limited in the variety of asset classes it can hold. For example, a million ADA will fit, but a million different NFTs might not fit in a single utxo.

Given the somewhat transient nature of state channels and the small numbers, we do not expect this constraint to affect most typical Hydrozoa use cases. However, if this becomes a problem, we could explore how to fragment a Hydrozoa head's treasury across multiple utxos. Observer scripts are helpful for fragmented treasuries because they facilitate an orderly fallback of treasury fragments to the rule-based regime.

A.4 Hydrozoa network and L2 interoperability

From the recent article "Cardano Layer 2 Interoperability: Midgard and Hydrozoa" [5]:

> Midgard and Hydrozoa are Layer 2 (L2) solutions at the forefront of Cardano's next scalability era. Interoperability between them (and other L2s) mitigates their relative weaknesses while reaping the full benefits of their strengths. Midgard provides massive throughput and a non-multisig consensus protocol ideal for many smart-contract-based dApps to co-exist on a single ledger with vast userbases, but it suffers from delayed settlement for withdrawals to L1. A Hydrozoa head provides instant finality and low cost for its peers' transactions and withdrawals, but only its peers benefit from the protocol's full security guarantees.
> An eUTXO-enhanced hashed timelock contract (HTLC) could work well as a universal mechanism of L2-L2 transfers of funds and information. An HTLC transfer from Midgard to a Hydrozoa head can eliminate the Midgard settlement delay for any peer of the Hydrozoa head. HTLC transfers within a network of Hydrozoa heads liberate the flow of funds and information within the network, with Midgard, and with other L2s that support HTLCs (incl. Bitcoin Lightning).
> Standardizing L2-L2 transfers, L2 state queries, and wallet integration can mitigate the UX fragmentation problem as Cardano pursues complementary L2 solutions.

Appendix B

Deterministic rollout algorithm

The Hydrozoa protocol strives to generate L1 transactions deterministically whenever they require consensus from all peers (i.e., in the multisig regime). This determinism allows the L2 consensus protocol and block validation procedure to focus on the essential contents of blocks without getting distracted in verifying the minutiae of the blocks' effects expressed as L1 transactions.

This determinism is easy to accomplish for a refund (§ 1.4) and the fallback to the rule-based regime (§ 2.2) because each of those transactions has a bounded number of inputs/outputs that are guaranteed to fit within Cardano's transaction size limit. On the other hand, settlement of a major block on L1 (§ 1.5) may require a large number of L2 withdrawals to be paid out on L1, such that one or more rollout transactions may be needed to handle its overflow of L1 outputs. Finalization (§ 1.6) is also unconstrained in the number of L2 withdrawals it can pay out.

This appendix describes the deterministic algorithm that assigns withdrawn L2 utxos to the outputs of a settlement/finalization transaction and its rollout transactions.

B.1 Settlement transaction

Suppose we have a treasury utxo with sufficient funds, and we need to settle a major L2 block on L1 such that all these conditions are satisfied:

1. Absorb a given set of L1 deposit utxos (`depositsAbsorbed`) in the ascending order of output reference. The number of deposits in this set is constrained such that the deposits (as spent inputs) fit well within Cardano's transaction size limit.
2. Pay out L1 utxos equivalent to a given set of L2 withdrawn utxos (`utxosWithdrawn`) in the ascending order of output reference. The number of withdrawals in this set is unconstrained.
3. Spend the treasury utxo and reproduce it with the total deposited funds added and the total withdrawn funds removed.

The deterministic algorithm's goal for the settlement transaction is to absorb `depositsAbsorbed` and to pay out as many L2 withdrawn utxos in `utxosWithdrawn` as possible, deferring payout for the rest to subsequent rollout transactions. It greedily achieves this goal by following these steps:

1. Let `txFixedPart` be a partial transaction draft that meets conditions (1) and (3) above and has signatures from all peers.

2. For i ranging from 1 to (size utxosWithdrawn), define a candidate transaction as the variation of `txFixedPart` that meets all these additional conditions:

   (a) Let `utxosWithdrawnPrefix` be the first i L2 utxos in `utxosWithdrawn`.

   (b) Produce L1 utxos equivalent to `utxosWithdrawnPrefix`. Equivalence means that each L2 utxo has a corresponding L1 utxo that contains the same funds and datum at the same address.

   (c) If i is less than (size utxosWithdrawn), produce an L1 rollout utxo that contains an empty datum, a newly-minted rollout beacon token (§ 1.1) and the rest of the funds that need to be paid out (value (utxosWithdrawn \ utxosWithdrawnPrefix)).

3. Select the largest-i candidate that fits within Cardano's transaction size limit.

If the settlement transaction produces an L1 rollout utxo, then the deterministic algorithm meets the rest of condition (2) above via one or more rollout transactions (§ B.3).

B.2 Finalization transaction

Suppose we have a treasury utxo with sufficient funds, and we need to finalize the head on L1 such that all these conditions are satisfied:

1. Absorb no L1 deposit utxos (depositsAbsorbed $\coloneqq \varnothing$).
2. Pay out L1 utxos equivalent to a given set of L2 withdrawn utxos (`utxosWithdrawn`) in the ascending order of output reference. The number of withdrawals in this set is unconstrained.
3. Spend the treasury utxo but do not reproduce it.

The deterministic algorithm's goal for the finalization transaction is to pay out as many L2 withdrawn utxos in `utxosWithdrawn` as possible, deferring payout for the rest to subsequent rollout transactions. It greedily achieves this goal by following these steps:

1. Let `txFixedPart` be a partial transaction draft that meets conditions (1) and (3) above and has signatures from all peers.
2. For i ranging from 1 to (size utxosWithdrawn), define a candidate transaction as the variation of `txFixedPart` that meets all these additional conditions:

   (a) Let `utxosWithdrawnPrefix` be the first i L2 utxos in `utxosWithdrawn`.

   (b) Produce L1 utxos equivalent to `utxosWithdrawnPrefix`. Equivalence means that each L2 utxo has a corresponding L1 utxo that contains the same funds and datum at the same address.

   (c) If i is less than (size utxosWithdrawn), produce an L1 rollout utxo that contains an empty datum, a newly-minted rollout beacon token (§ 1.1), and the rest of the funds that need to be paid out (value (utxosWithdrawn \ utxosWithdrawnPrefix)).

3. Select the largest-i candidate that fits within Cardano's transaction size limit.

If the finalization transaction produces an L1 rollout utxo, then the deterministic algorithm meets the rest of condition (2) above via one or more rollout transactions (§ B.3).

B.3 Rollout transactions

Suppose we have an L1 rollout utxo with sufficient funds, and we need to construct one or more rollout transactions that collectively meet all these conditions:

1. Pay out L1 utxos equivalent to a given set of L2 withdrawn utxos (`utxosWithdrawn`) in the ascending order of output reference. The number of withdrawals in this set is unconstrained.
2. Spend the L1 rollout utxo.
3. Burn the rollout beacon token in the L1 rollout utxo.

The deterministic algorithm's goal for each rollout transaction is to pay out as many L2 withdrawn utxos in `utxosWithdrawn` as possible, deferring payout for the rest to subsequent rollout transactions. It greedily achieves this goal by following these steps:

1. Let `txFixedPart` be a partial transaction draft that meets condition (2) above and has signatures from all peers.
2. For i ranging from 1 to (size utxosWithdrawn), define a candidate transaction as the variation of `txFixedPart` that meets all these additional conditions:
   (a) Let `utxosWithdrawnPrefix` be the first i L2 utxos in `utxosWithdrawn`.
   (b) Produce L1 utxos equivalent to `utxosWithdrawnPrefix`. Equivalence means that each L2 utxo has a corresponding L1 utxo that contains the same funds and datum at the same address.
   (c) If i is less than (size utxosWithdrawn), produce an L1 rollout utxo that contains an empty datum, the rollout beacon token (§ 1.1), and the rest of the funds that need to be paid out (value (utxosWithdrawn \ utxosWithdrawnPrefix)).
   (d) Otherwise, burn the rollout beacon token.
3. Select the largest-i candidate that fits within Cardano's transaction size limit.

The deterministic algorithm recursively generates a sequence of rollout transactions until all three of the above conditions are met.

Appendix C

Architecture

C.1 Codebase

Hydrozoa uses a single mono-repository called `cardano-hydrozoa/hydrozoa` [6]. Tentatively, we will organize the codebase into the following packages:

Spec. The source code for this specification.

Multisig. Onchain scripts, transaction building code, and blockchain queries for the L1 multisig regime (§ 1).

Plutus. Onchain scripts, transaction building code, and blockchain queries for the L1 rule-based regime (§ 2).

Ledger. L2 ledger state and transitions (§ 3).

Blocks. L2 block creation, validation, and effects (§ 4).

Head state. L2 head state, storage, and queries. (§ 4.1).

Consensus. L2 consensus protocol messages, handlers, API (§ 5).

Node. Hydrozoa node software and interface (§ C.3).

Infra. Development environment, CI/CD, deployment scripts, and configurations.

C.2 Onchain script interdependencies

Hydrozoa's L1 protocol has the following onchain script interdependencies:

Native script A native script parametrized on the list of peers (§ 1).

Native minting policy. A minting policy controlled by the native script (§ 1.1).

Treasury script. A Plutus-based spending validator parametrized on the native script (§ 2.4).

Tallying script. A Plutus-based staking validator parametrized on the native script (§ 2.3).

Voting script. A Plutus-based spending validator parametrized on the native script and the tallying script (§ 2.3).

C.3 Processes and components

In order to operate a Hydrozoa head, each peer will need to run the following processes/software:

Hydrozoa UI. A web-based or terminal-based interface for a user to initialize, fund, and interact with a Hydrozoa head.

Hydrozoa node. An executable that manages the peer's L2 state for the head and facilitates the peer's participation in the L2 consensus protocol. It provides a backend API for the Hydrozoa UI's interactions with the head.

User wallet. A wallet that holds the user's cryptographic keys for manual transaction signing. User wallet signatures are needed in the head's initialization and deposit transactions, which bring funds into the head.

Cardano node (L1). A Cardano node synchronized to the chain tip of the target L1 network (e.g., Cardano mainnet). It is the underlying source of events about the head's L1 state, and the Hydrozoa node submits all L1 transactions to it.

Blockchain indexer (L1). A blockchain indexer (e.g., Blockfrost, Ogmios/Kupo) that indexes the events from the Cardano node. It responds to the Hydrozoa node's queries about the head's L1 state.

Bibliography

[1] Alessandro Konrad and Thomas Vellekoop. 2022. CIP-67: Asset Name Label Registry. https://github.com/cardano-foundation/CIPs/tree/master/CIP-0067

[2] Philip DiSarro. 2024. CIP-112: Observe Script Type. https://github.com/cardano-foundation/CIPs/tree/master/CIP-0112

[3] fallen-icarus. 2025. Cardano Swaps P2P DeFi Protocol. https://github.com/fallen-icarus/cardano-swaps

[4] George Flerovsky. 2024. Catalyst Milestones for the Hydrozoa Fund 13 Project. https://milestones.projectcatalyst.io/projects/1200059

[5] George Flerovsky and Philip Disarro. 2024. Cardano Layer 2 Interoperability: Midgard and Hydrozoa. https://github.com/GeorgeFlerovsky/interop-midgard-hydrozoa

[6] George Flerovsky and Ilia Rodionov. 2025. Hydrozoa Repository (Github). https://github.com/cardano-hydrozoa/hydrozoa

[7] IntersectMBO. 2025. Cardano Ledger v1.17.4.0: Conway-era, Plomin Hardfork. https://github.com/IntersectMBO/cardano-ledger/tree/cardano-ledger-conway-1.17.4.0

[8] IOG. 2025. Exclude Phantom Tokens from Snapshots (Github Issue #358). Cardano Scaling. https://github.com/cardano-scaling/hydra/issues/358

[9] D. Mills, J. Martin, J. Burbank, and W. Kasch. 2010. Network Time Protocol (4.2.8). https://www.ntp.org/documentation/4.2.8-series/

[10] Sebastian Nagel, Sasha Bogicevic, Franco Testagrossa, and Daniel Firth. 2024. Hydra HeadV1 Specification: Coordinated Head Protocol. (2024). https://hydra.family/head-protocol/docs/dev/specification