👨‍🔬

ADR x: Endorsement Yield (Lazy Accumulator Model)

Author	@Omri Dagan
Status	Awaiting Review

Status

PROPOSED

Changelog

- 2025-05-22: Added lazy accumulator model for persistent reward accrual

Problem

Currently, endorsers must claim their rewards during a specific epoch window. If they miss it, the rewards are lost. This creates unnecessary pressure, poor UX, and misalignment with passive incentivization models.

Solution Summary

Introduce an accumulator-based accounting model (inspired by AMMs like Uniswap) where each endorser's rewards accumulate over time and can be claimed at any point. No epoch-based forfeiture.

This method tracks a global per-share reward accumulator and individual user snapshots of it. Claimable rewards are computed as:

```
claimable = (global_accumulator - user.last_seen_accumulator) × user.shares
```

Key Concepts

- **Accumulator**: A per-gauge variable representing total reward per share over time.
- **Shares**: Amount of voting power a user has allocated to the gauge.
- **Last Seen Accumulator (LSA)**: Stored per user to mark the accumulator state when they last claimed or joined.

Table Walkthrough: Multi-Epoch, Multi-User Scenario

LSA = last seen accumulator

Epoch	Event	Accumulator	Total Shares	U1 Shares	U1 LSA	U1 Claimable	U2 Shares
1	User1 endorses with 40 shares	0.0	40	40	0.0	(0 - 0) * 40 = 0 DYM	-
2	+100 DYM unlocked	0 + (100 / 40) = 2.5	40	40	0.0	(2.5 - 0) * 40 = 100 DYM	-
3	User2 endorses with 60 shares	2.5	40 + 60 = 100	40	0.0	100 DYM	60

4	+100 DYM unlocked	2.5 + (100 / 100) = 3.5	100	40	0.0	(3.5 - 0) * 40 = 140 DYM	60
4	User1 claims	3.5	100	40	3.5	✅ Claimed: 140 DYM (3.5 - 3.5) * 40 = 0 DYM	60
4	User2 un-endorses (auto-claim)	3.5	100 - 60 = 40	40	3.5	(3.5 - 3.5) * 40 = 0 DYM	-
5	+100 DYM	3.5 + (100 / 40) = 6.0	40	40	3.5	(6 - 3.5) * 40 = 100 DYM	-
6	User1 un-endorses (auto-claim)	6.0	40 - 40 = 0	-	-	✅ Claimed: 100 DYM	-
7	User2 re-endorses w 100 shares	6.0	0 + 100 = 100	-	-	-	100
8	+100 DYM unlocked	6 + (100 / 100) = 7.0	100	-	-	-	100

Final Totals

User	Total Claimed
User1	140 + 100 = 240 DYM
User2	60 + 100 = 160 DYM

Benefits

- No forfeited rewards
- Constant-time claiming (`O(1)`)
- No per-epoch user tracking
- Compatible with current gauge reward funding model

State Schema

```
message EndorserPosition {
  string address;
  uint64 shares;
  decimal last_seen_accumulator;
}

message EndorsementGauge {
  string id;
  decimal global_accumulator;
  uint64 total_shares;
  coins unlocked_rewards;
  coins distributed;
  ...
}
```

Logic Flow

Endorse

1. `shares = voting_power`
2. `last_seen_accumulator = gauge.accumulator`
3. `gauge.total_shares += shares`

Epoch End

1. `E = unlocked_rewards_per_epoch`
2. `accumulator += E / total_shares`
3. `gauge.unlocked_rewards += E`

Claim

1. `claimable = (accumulator - user.last_seen) * user.shares`
2. Transfer `claimable` to user
3. `user.last_seen = accumulator`

Un-endorse

1. Call `Claim()` first
2. `gauge.total_shares -= user.shares`
3. Delete user position

Edge Cases Handled

- Users joining just before epoch end don’t get that epoch’s rewards
- Users can miss epochs without penalty
- Vote → revoke → vote again: snapshots always protect past state
- Gauge might have multiple currencies as rewards. Use a list of LSA in user positions and endorsements.
- User’s shares might change in the background due to staking events, and therefore endorsement total shares also do.

Next Steps

- Implement new accumulator logic inside `x/incentives`
- Add accumulator fields to `sponsorship.Endorsement`
- Migrate or adapt old per-epoch logic to accumulator tracking

---

Appendix: Parallels to AMMs

AMM (Uniswap)	Endorsement Model
LP token shares	Endorsement shares
Pool fees accrued	Gauge rewards unlocked
LP token value grows	Claimable balance grows
LP withdraws anytime	Endorser claims anytime

Appendix: Implementation Remarks

Types

`Endorsement` is stored per RA and is created automatically along with RA creation (hook in x/streamer). Endorsements are stored in x/sponsorship keeper using collections: `collections.Map[string, types.Endorsement]`

```
// Endorsement is an info about the endorsement made by users to the RollApp.
// It stores information about the gauge associated with the RollApp and
// the total shares endorsers hold.
//
// The number of shares if adjusted when someone endorses the RollApp or
// when voting power of some endorser changes.
message Endorsement {
  // RollappId is a rollapp associated with the given endorsement.
  string rollapp_id = 1;
  // RollappGaugeId is a rollapp gauge associated with the given rollapp.
  uint64 rollapp_gauge_id = 2;
  // TotalShares defines total shares issued to the endorsement.
  string total_shares = 3 [
    (gogoproto.nullable) = false,
    (cosmos_proto.scalar) = "cosmos.Dec",
    (gogoproto.customtype) = "cosmossdk.io/math.LegacyDec"
  ];
  // Accumulator is a variable representing total reward per share over time.
  // It is an array of coins since every currency should have its own
  // accumulator.
  repeated cosmos.base.v1beta1.DecCoin accumulator = 4 [
    (gogoproto.nullable) = false,
    (gogoproto.castrepeated) = "github.com/cosmos/cosmos-sdk/types.DecCoins"
  ];
  // TotalCoins is the total amount of coins that have been in the endorsement.
  repeated cosmos.base.v1beta1.Coin total_coins = 5 [
    (gogoproto.nullable) = false,
    (gogoproto.castrepeated) = "github.com/cosmos/cosmos-sdk/types.Coins"
  ];
  // DistributedCoins are coins that have been distributed already.
  repeated cosmos.base.v1beta1.Coin distributed_coins = 6 [
    (gogoproto.nullable) = false,
    (gogoproto.castrepeated) = "github.com/cosmos/cosmos-sdk/types.Coins"
  ];
}
```

`EndorserPosition` is stored per user. If the user casts a vote to RA1, then the respective position is created. If the user un-endorses this RA, then the position is removed. Positions are stored in x/sponsorship keeper using collections: `collections.Map[collections.Pair[sdk.AccAddress, string], types.EndorserPosition]`

```
// EndorserPosition is the position of a single endorser in a given rollapp
// endorsement.
message EndorserPosition {
  // Sharers is the number of shares the endorser holds.
  string shares = 1 [
    (gogoproto.nullable) = false,
    (cosmos_proto.scalar) = "cosmos.Dec",
    (gogoproto.customtype) = "cosmossdk.io/math.LegacyDec"
  ];
  // LastSeenAccumulator marks the accumulator state when the endorser last
  // claimed or endorsed.
  repeated cosmos.base.v1beta1.DecCoin last_seen_accumulator = 2 [
    (gogoproto.nullable) = false,
    (gogoproto.castrepeated) = "github.com/cosmos/cosmos-sdk/types.DecCoins"
  ];
}
```

Flow

1. We have RA1 (rollapp 1)
2. We have EG1 (endorsement gauge 1), EG2, EG3 pointing to RA1
3. We have **one single** `Endorsement` object associated with RA1. It holds `total_shares`, `global_accumulator`, `unlocked_rewards`, etc
4. On every epoch, every EG releases some portion of funds **and increases** `global_accumulator`, `unlocked_rewards` in `Endorsement`. We don't do any actual `bank.send` operations, just change the numbers.
5. User votes on RA1 and we create `EndorserPosition` **only for (user; RA1)**
6. User claims **only for RA1 (1 operation)**. And we send funds from x/incentives account

So endorsement gauges are dummies that just release funds. In that case, RA1 does not need to know the entire list of EG1, EG2, etc, so we can get rid of the index. And we don't need to store `EndorserPosition` for every endorsement gauge separately, only by user + RA.