The following is a brief description of issues we run into when building certain systems inBalancer, which are ultimately related to interface compatability, data availability, and an example of an use case for capsules.

Liquidity Mining 101

Many DeFi protocols run so-called 'liquidity mining' programs, with which they incentivize participation by granting tokens to depositors. Individual token allocation is typically a fairly involved process, considering not just the duration and scope of one's participation, but also which products were used, boosts for participants that met a certain criteria, etc. Balance had weekly whitelists that'd be updated, along with complex parameters such as the [warpFactor

](https://forum.balancer.fi/t/modifying-wrapfactor-applying-a-0-7-factor-to-soft-pegged-pairs/108) which attempted to prevent gaming the system, and the [govFactor

](https://forum.balancer.fi/t/proposal-govfactor/589) which aimed to increase participation in protocol decisions.

Since tracking this on-chain proved unfeasible (due to both gas costs and data unavailability), Python scripts were developed that would be fed data from subgraphs and configuration files (for whitelists, etc.) and output the weekly token allocation for each account. Anyone could re-run the scripts and verify that the computation had been done correctly.

Claiming Tokens

Transferring tokens to multiple thousand users on a weeky basis is prohibitely expensive however. A better approach would be to upload the allocation list to the blockchain and have each user pay for the gas of their claim, but the size of the data would still scale linearly with the number of users, and both submission (calldata) and storage (either contract storage or code) are still fairly pricey. The solution was to stora all claims in a merkle tree

, upload the root, and use a [special MerkleClaim

contract](https://github.com/balancer/balancer-v2-monorepo/blob/tob-audit-linear/pkg/distributors/contracts/MerkleRedeem.sol) to withdraw.

The entire process is then as follows:

- users participate in the protocol
- usage data is collected
- weekly data plus voted-on configuration is fed into a script to compute allocation
- a merkle tree is created with the resulting allocation, the tokens and tree root submitted on chain
- each user claims their weekly share by providing a merkle proof (a sibling path) to the contract, which verifies tree inclusion and transfers the associated tokens

Integration Woes

After some time, other projects such as Yearn started building automated systems that participated in different DeFi protocols, collected earnings and distributed them among users. In the case of Balancer, they needed to collect both swap fees and the liquidity mining incentives.

It was there they run into an issue: Balancer didn't have a simple claim()

interface, but rather claim(amount, siblingPath)

, which required a lot of extra information the Yearn contract did not have

.

This could presumably have been solved by having some third party make the claim amount and path available to the contract by e.g. preseeding it and keeping it on contract storage. This introduces other problems however: someone

must now push this data (on a weekly basis and for each Yearn contract!), and the system won't be able to be able to function until this happens.

Ultimately, Yearn never built their integration due to these issues, and the Balancer merkle liquidity mining system was scrapped in favor a complete rewrite, which tracked allocations on-chain and allowed for simpler claims (eventually leeading to multiple integrations being successfully completed).

Aztec Capsules and Oracles

An interesting thing to consider in the scenario described above is that the Yearn end user ultimately did not care or know about the Balancer merkle proofs. Had the system been able to magically produce the one sibling path that resulted in the sibling path check being completed, this would have been a complete non-issue. Moreover systems did exist which knew what the correct sibling path was for a given account: they just were not available on-chain.

I think this is a good example to consider and learn from when thinking about oracles that provide third party data deep-down in the callstack, which is the current use case for capsules.