Allocation

An allocation happens when a user (then called a *distributor*) decides to allocate the rewards of some of their staked MATIC to another user (then called a *recipient*).

This is mostly used as an accounting mechanism and the distributor is never obligated to ever distribute the rewards accrued or to even keep enough in its wallet for an eventual payout.

`allocate`

allocate only changes the allocation mappings and not the user shares. It takes as its arguments `distributor` , `recipient` and `amount` , the amount being in MATIC.

Variable changes

For an allocated amount a, to be allocated by distributor d to recipient r:

`allocations`

- $\alpha'_d(r) = \alpha_d(r) + a$ - the new amount allocated is the old amount plus the new amount
- $\frac{1}{\pi'_d(r)} = \frac{\alpha_d(r) \cdot \frac{1}{\pi_d(r)} + a \cdot \frac{1}{P}}{\alpha_d(r) + a}$- the new average price is the weighted average of the old price and current price.

▼ *Overflows and order of operations*

$$\underline{\pi'_d(r)} = \text{ceil}\left[\frac{\alpha_d(r)\underline{\pi_d(r)}}{\overline{\pi_d(r)}}\right] + \text{ceil}\left[\frac{a\underline{P}}{\overline{P}}\right]$$

$$\overline{\pi'_d} = \alpha_d(r) + a$$

For the numerator, we need to use `mulDiv` to calculate each term to avoid overflows. We take the ceiling here to make sure we are generally underestimating the share price we save for allocations (this is so that generally, slightly more money is locked for distributors for strict allocations).

`totalAllocated`

- $\alpha'_d = \alpha_d + a$ - add `amount` to the total allocated.
- $\frac{1}{\pi'_d} = \frac{\alpha_d \frac{1}{\pi_d} + a \frac{1}{P}}{\alpha_d + a}$ - update the priceAllocated to the correct average price including the new allocation.

▼ *Overflows and order of operations*

$$\underline{\pi'_d} = \frac{\alpha_d \underline{\pi_d}}{\overline{\pi_d}} + \frac{a\underline{P}}{\overline{P}}$$
$$\overline{\pi'_d} = \alpha_d + a$$

For the numerator, we need to use `mulDiv` to calculate each term to avoid overflows.. Solidity will round down each of the first terms leading to the price stored in `totalAllocated` to generally be low. This is so that the amount allocated will generally be overestimated and the balance withdrawable underestimated

`recipients` and `distributors`

- Add any new recipients to the distributor⇒[recipient] mapping
- Add any new distributors to the recipient⇒[distributor] mapping

▼ *Calculations*

- The recipient essentially is long TruMATIC via a contract for difference (CFD) with strike price $\pi_d(r)$ and MATIC notional $\alpha_d(r)$.
- When an allocation or deallocation occurs, it is equivalent to entering a new CFD. Rather than storing every single CFD entered, we instead store the aggregate amount and average strike prices via the above. It's exactly the same as calculating the average price of several clips of FX / equity trading.
- In the case of deallocation, we essentially settle any profit from the CFD by distributing before the deallocation takes place.

To do the calculation, we want allocating twice to be equivalent to allocating once with a different allocation amount and different allocation price:

$$(\alpha_0, P_0) + (\alpha_1, P_1) \equiv (\alpha, P)$$

The way to do this is to calculate the PnL for each allocation relative to a current share price Π. Set $\alpha = \alpha_0 + \alpha_1$.

$$\begin{aligned}
\frac{\alpha_0}{P_0}\Pi - \alpha_0 + \frac{\alpha_1}{P_1}\Pi - \alpha_1 &= \frac{\alpha}{P}\Pi - \alpha \\
&= \frac{(\alpha_0 + \alpha_1)}{P}\Pi - (\alpha_0 + \alpha_1) \\
\frac{\alpha_0}{P_0} + \frac{\alpha_1}{P_1} &= \frac{(\alpha_0 + \alpha_1)}{P}, \text{ cancelling and rearranging} \\
\frac{1}{P} &= \frac{\alpha_0 \frac{1}{P_0} + \alpha_1 \frac{1}{P_1}}{\alpha_0 + \alpha_1}
\end{aligned}$$

`deallocate`

deallocate only changes the allocation mappings. It takes as its arguments `distributor` , `recipient` and `amount` , the amount being in MATIC.

Variable changes

For a deallocated amount a, to be allocated by distributor d to recipient r:

`allocations`

- $\alpha'_d(r) = \alpha_d(r) - a$ - the new amount allocated is the old amount less the deallocation amount

`totalAllocated`

- $\alpha'_d = \alpha_d - a$ - add `amount` to the total allocated.
- $\frac{1}{\pi'_d} = \frac{\alpha_d \frac{1}{\pi_d} - a \frac{1}{P}}{\alpha_d - a}$ for strict allocations. For loose ones, replace P with $\pi_d(r)$, the old (and new) individual allocation price.

▼ *Overflows and order of operations*

$$\underline{\pi'_d} = \frac{\alpha_d(r)\underline{\pi_d}}{\overline{\pi_d}} - \frac{a\underline{P}}{\overline{P}}$$

$$\overline{\pi'_d} = \alpha_d - a$$

For the numerator, we need to use `mulDiv` to calculate each term. Solidity will round down the first terms leading to the price stored in `totalAllocated` to generally be low. This is so that the amount allocated will generally be overestimated and the balance withdrawable underestimated

`recipients` and `distributors`

- Remove any recipients whose allocation has gone to zero from the distributor⇒[recipient] mapping

- Remove any distributors whose allocation has gone to zero from the recipient⇒[distributor] mapping

▼ *Calculations*

The calculations here are identical to those for straight allocation, just replacing the amount a with $-a$.

Checks before variable changes

Several checks need to be made before the change in variables:

- Two div/0 errors need to be handled correctly:
    - $\alpha_d(r) - a = 0$ - i.e. the deallocation completely removes recipient from distributors recipient list.
    - $\alpha_d - a = 0$ - i.e. the deallocation takes the distributor's total allocation to zero. (Note this is a subcase of the first case).
- $\alpha_d(r) - a > 0$ - the amount in this deallocation request never takes the balance negative.
- $\pi_d(r) = P$. If this is not the case, then the distributor is trying to deallocate after share price increase, i.e. there are rewards which have not yet been distributed.