Overview

Raydium-Amm-v3 is an open-sourced concentrated liquidity automated market maker (CLAMM) program built for the Solana ecosystem.

Concentrated liquidity allow liquidity providers choose the price range they want to provide liquidity into. This improves capital efficiency by allowing to put more liquidity into a narrow price range, which makes amm-v3 more diverse: it can now have pools configured for pairs with different volatility.

Our open-source code can be find here: https://github.com/raydium-io/raydium-clmm

The Mathematics of AMM-V3

$$L = \sqrt{(xy)}$$

$$\sqrt{(P)} = \sqrt{(y/x)}$$

L is *the amount of liquidity*. Liquidity in a pool is the combination of token reserves . Here, L is actually the square root of `K`.

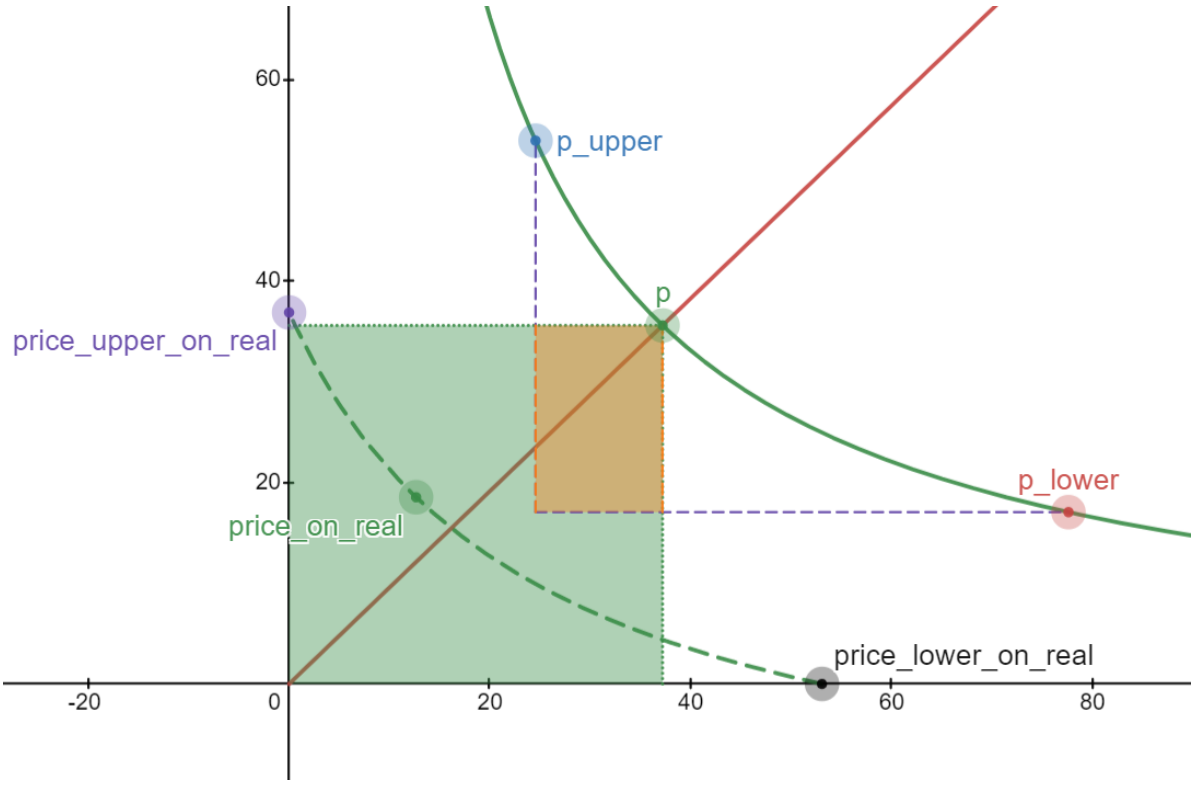
The reason why we use $\sqrt{P}$ instead of `P` is because square root calculation is not precise and causes rounding errors. Thus, it's easier to store the square root without calculating it in the contracts (we will not store *x* and *y* in the contracts).

From the above formula, we can deduce:

$$x = L/\sqrt{(P)}$$

$$y = L * \sqrt{(P)}$$

Core Formula



In the above picture, if the user's liquidity specified interval is $[p_{upper}, p_{lower}]$, using x and y to represent the two types of token quantity, which include virtual and actual added quantity, then it needs to satisfy `x * y = k`, the virtual is represented by $x_{virtual}$ and $y_{virtual}$, the actual amount added by the user is represented by Δx and Δy:

$$(\Delta x + x_{virtual}) * (\Delta y + y_{virtual}) = k$$

It is easy to see that the length of $x_{virtual}$ is actually the x at the `p_upper` point, and the length of $y_{virtual}$ is actually the y at the `p_lower` point.

$$x_{virtual} = L/\sqrt{p_{upper}}$$

$$y_{virtual} = L * \sqrt{p_{lower}}$$

So, it can be deduced that

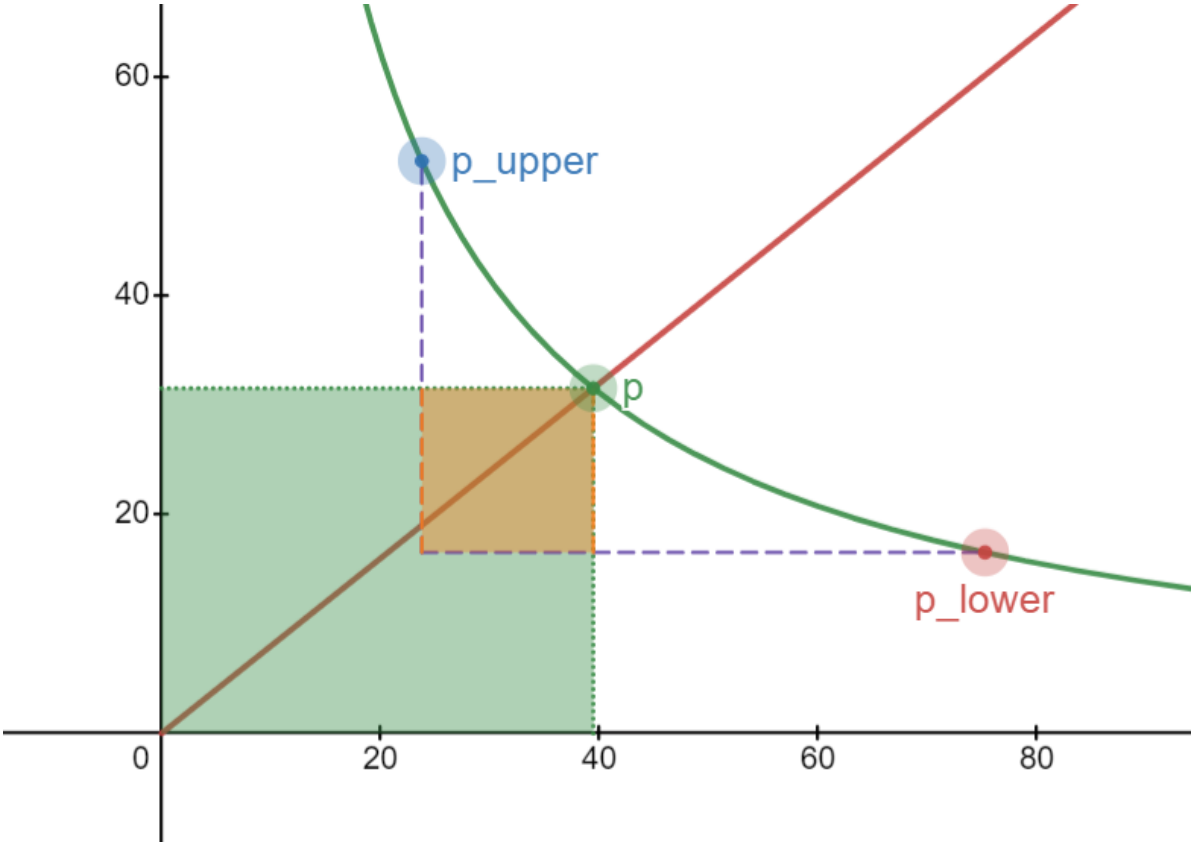
$$(\Delta x + L/\sqrt{p_{upper}}) * (\Delta y + L * \sqrt{p_{lower}}) = L^2$$

In the formula, `p_upper` and `p_lower` are known variables that are set by the user as the market making price range.

Calculating Liquidity

Price Range Calculation

`p_lower < p < p_upper`



In this scenario, the orange region is the actual liquidity that we need to add, and x and y are the quantities of the two assets at the current price, respectively。

```
delta_x = x - x_virtual
delta_y = y - y_virtual
```

We can calculate the required Δx and Δy based on the liquidity L and the price range.

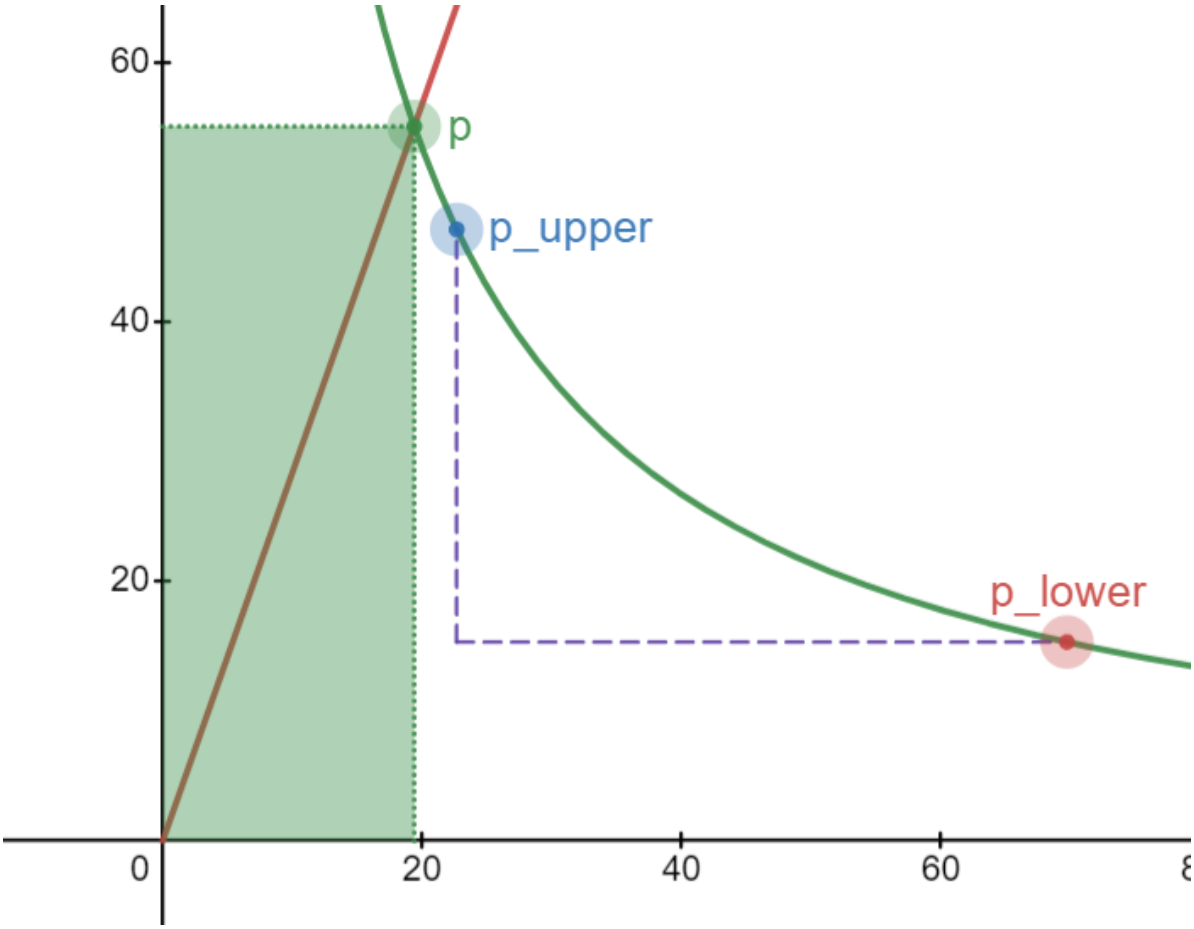
```
delta_x = L / √p - L / √p_upper = L * (√p_upper - √p) / (√p * √p_upper)
delta_y = L * √p  - L * √p_lower = L * (√p - √p_lower)
```

If we also know the price range and given either Δx or Δy , we can calculate the liquidity L.

```
L = delta_x * (√p * √p_upper) / (√p_upper - √p)
L = delta_y / √(p - p_lower)
```

Price Out Range Calculation

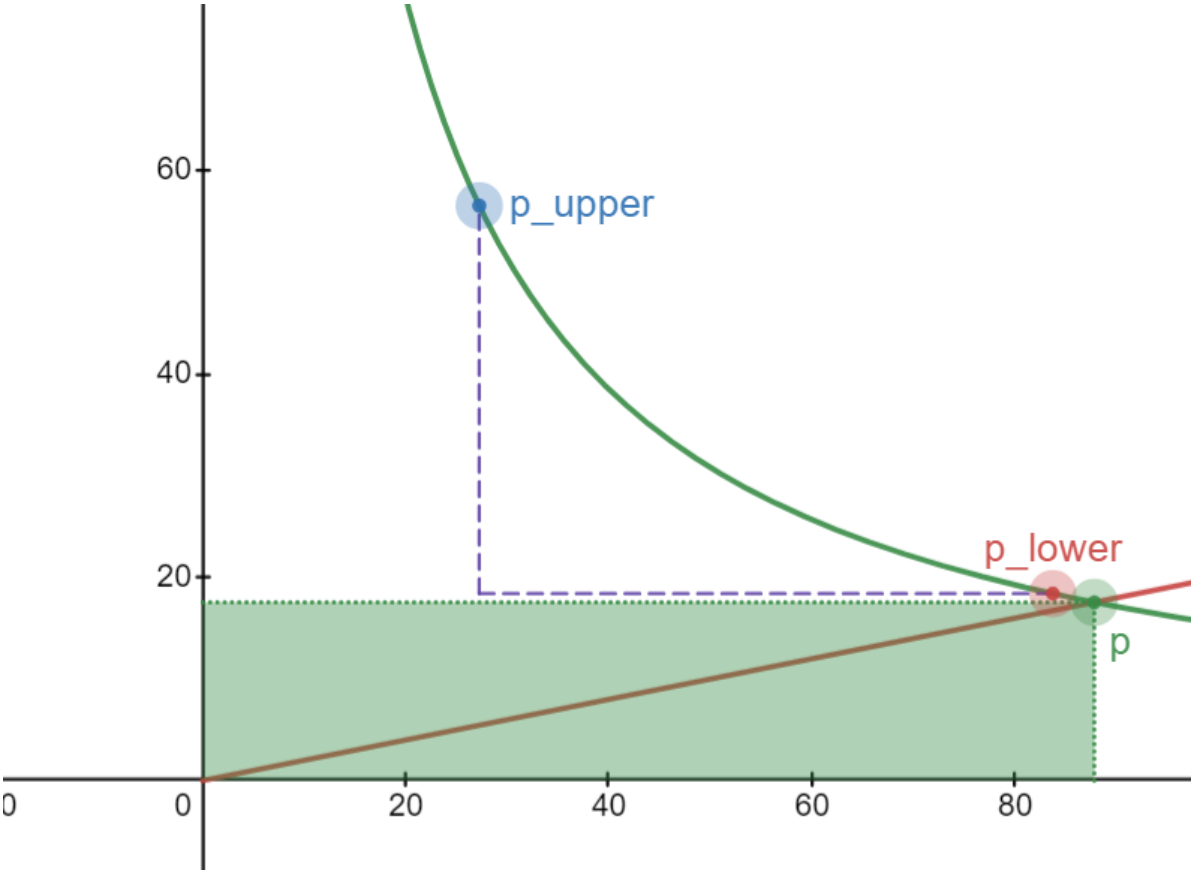
p > p_upper



So, in this case, the amount of asset X is 0, and all the liquidity becomes asset Y, whose quantity is the distance between the `p_upper` and `p_lower` on the y-axis, which is the purple dotted line part in the figure. Based on delta y, we can calculate L.

$$L = \Delta y / \sqrt{p_{upper} - p_{lower}}$$

p < p_lower



In this case, the amount of asset Y is 0, and the liquidity is fully transformed into asset X, whose quantity is the distance along the x-axis from `p_upper` to `p_lower`. We calculate `L` based on `delta x`.

$$L = \Delta x * (\sqrt{p_{upper}} * \sqrt{p_{lower}} / \sqrt{p_{upper}} - \sqrt{p_{lower}})$$

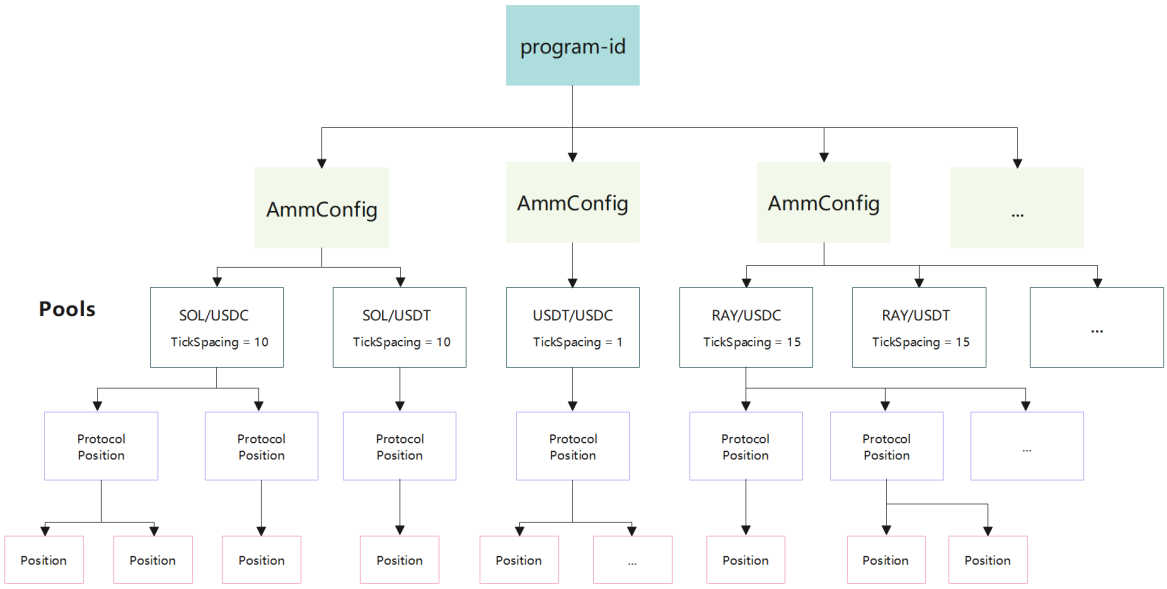
Program Overview

Parameters

Program ID

Cluster	Address
Mainnet-Beta	CAMMCzo5YL8w4VFF8KVHrK22GGUsp5VTaW7grrKgrWqK
Devnet	devi51mZmdwUJGU9hjN27vEz64Gps7uUefqxg27EAtH
Devnet for Immunefi	proKtffCScMcwkFkPHFcuHawN7mWxRkhyh8PGxkTwYx

Account Architecture



AmmConfig

`AmmConfig` is a PDA account created by an administrator, which cannot be duplicated. It stores the value of tickspacing and fee-related parameters. Each `AmmConfig` can be referenced by multiple pools, meaning that these pools have the same fee rates.

The `AmmConfig` account has already been created:

Cluster	index	tickspacing	feeRate	Address
Mainnet-Beta	0	10	1bps	4BLNHtVe942GSs4teSZqGX24xwKNkqU7bGgNn3iUiUpw
	1	60	25bps	E64NGkDLLCdQ2yFNPcavaKptrEgmiQaNykUuLC1Qgwyp
	2	10	5bps	HfERMT5DRA6C1TAqecrJQFpmkf3wsWTMncqnj3RDg5aw
	3	100	100bps	A1BBtTYJd4i3xU8D6Tc2FzU6ZN4oXZWXKZnCxwbHXr8x
Devnet	0	10	1bps	CQYbhr6amxUER4p5SC44C63R4qw4NFc9Z4Db9vF4tZwG
	1	60	25bps	B9H7TR8PSjJT7nuW2tuPkFC63z7drtMZ4LoCtD7PrCN1
	2	10	5bps	GVSwm4smQBYcgAJU7qjFHLQBHTc4AdB3F2HbZp6KqKof
	3	100	100bps	GjLEiquek1Nc2YjcBhufUGFRkaqW1JhaGjsdFd8mys38
Devnet for Immunefi	0	10	1bps	4m7dQubqQYGJvYDjUxeGq1gouo378HW9Zr9p6cPSuF1V
	1	60	25bps	3pM6bcrmAUDgmhNwYnXs3Mg6CpZC8wkWwvdpJXrR1Mbk
	2	10	5bps	BZ9TSquyD5bNRwjBgMgX6r6wTLgEgqmAjFbHvDL4YP2D
	3	120	100bps	Amz56QcJMUcoLHwg2Uz7jHj5JeaKuws7TfvDNadvinfS

Position

A position represents a user-provided liquidity position and is a range interval, with the lower limit commonly referred to as the tick_lower and the upper limit referred to as the tick_upper.

Protocol Position

"Positions" with the same liquidity range are consolidated into a "Protocol position".

Protocol Concepts

Tick

In V3, the entire price range is demarcated by evenly distributed discrete ticks. Each tick has an index and corresponds to a certain price:

$$p(i) = 1.0001^i$$

Where `p(i)` is the price at tick `i`. Taking powers of 1.0001 has a desirable property: the difference between two adjacent ticks is 0.01% or *1 basis point*.

V3 stores $\sqrt{P}$, not *P*. Thus, the formula is in fact:

$$\sqrt{p_i} = \sqrt{1.0001^i} = 1.0001^{i/2}$$

Ticks are integers that can be positive and negative and, of course, they're not infinite. V3 stores $\sqrt{P}$` *P*as a fixed point Q64.64 number, which is a rational number that uses 64 bits for the integer part and 64 bits for the fractional part. Thus, prices (equal to the square of $\sqrt{P}$) are within the range: $[2^{-128}, 2^{128}]$ And ticks are within the range: [−887272,887272].

But in our setup, the range of the tick is [-307200,307200].

Tick Spacing

The range of selectable ticks is very large and the difference in price between adjacent ticks is very small, which is necessary for the price of some cryptocurrencies, such as stablecoin pools. But for pools with high prices and large price fluctuations, such as Bitcoin, it is not necessary, so the concept of `tickspacing` was introduced in v3, which represents the spacing of the ticks and ticks between two tickspacing can be ignored and not participate in calculation.

This has several benefits:

1. Save compute cost and rent constraints

2. Granularity of user definable price ranges

The smaller your tick-spacing, the more granular the price users can deposit their liquidity in. For more stable pools, a more granular tick-spacing would let users define a tighter range to maximize their leverage.

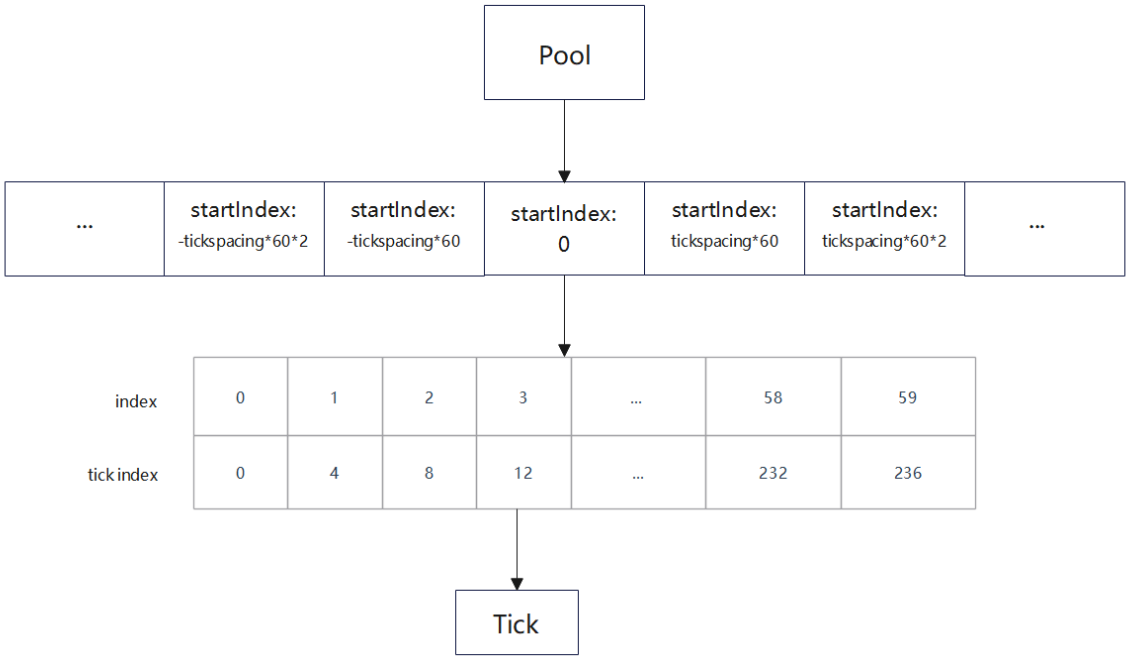
3. Maximum price movement per swap

The size of the tick-spacing defines the maximum price movement a single swap can move the price by for a pool.

swap operates by iterating through each ticks with initialized liquidity. The larger the gap between initialized ticks are, the more it can theoretically traverse the price range.

A low tick-spacing pool undergoing a massive price movement may require multiple swap instructions to complete the price movement. Therefore, more volatile pairs that often has large price swings should look at higher tick-spacing to mitigate this pain point for their pool users.

Tick Array



A tick array is a collection of ticks. A tick-array is keyed by the start-index of its hosted ticks and can hold 60 physical ticks in an array. It only hosts the tick objects for the initializable tick indices based on the pool's tick-spacing. The total range for a tick-array is therefore 60 * tick-spacing.

Usage in Instructions

When you interact with ticks on v3 instructions, often you will need to derive the correct tick-array so the program can get access to the designated tick object.

Open Position

A position opening up in a brand new tick/price area would need to initialize the tick-array prior to creating the position. This means the user who invoke that position would have to pay for the rent-exempt cost .

Modify Liquidity (increase / decrease liquidity)

Users of these instructions would need to pass in the tick-array that houses the tick-index that are passed in. The instruction would need access to these accounts to read the appropriate Tick object.

Swap

Swap users will have to provide the series of tick-arrays that the swap will traverse across.

The first tick-array in the sequence is the tick-array that houses the pool's current tick index. The remaining tick arrays are the next tick-arrays in the swap direction.