

# Lesson 7

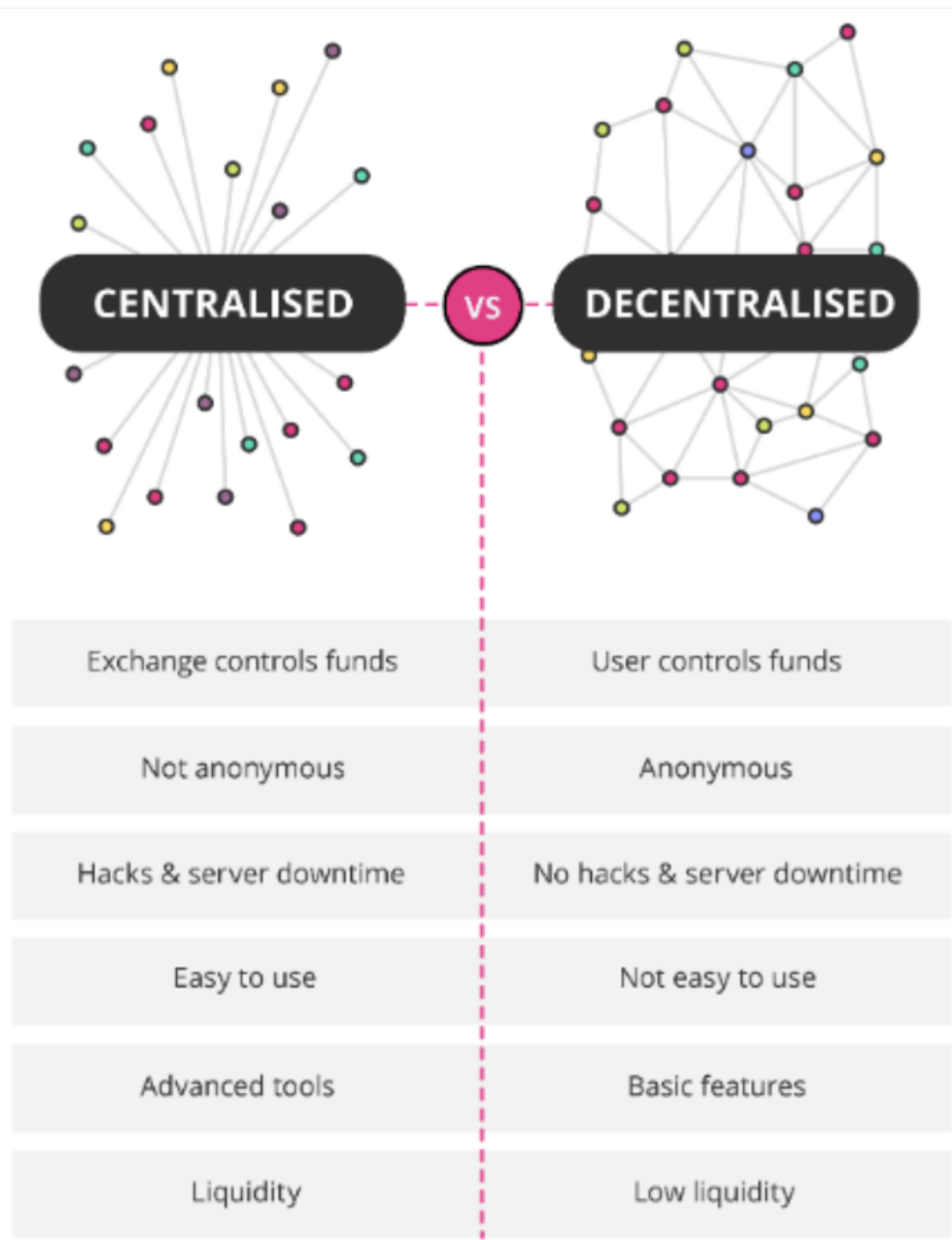
## DeFi Introduction

The main areas of DeFi include

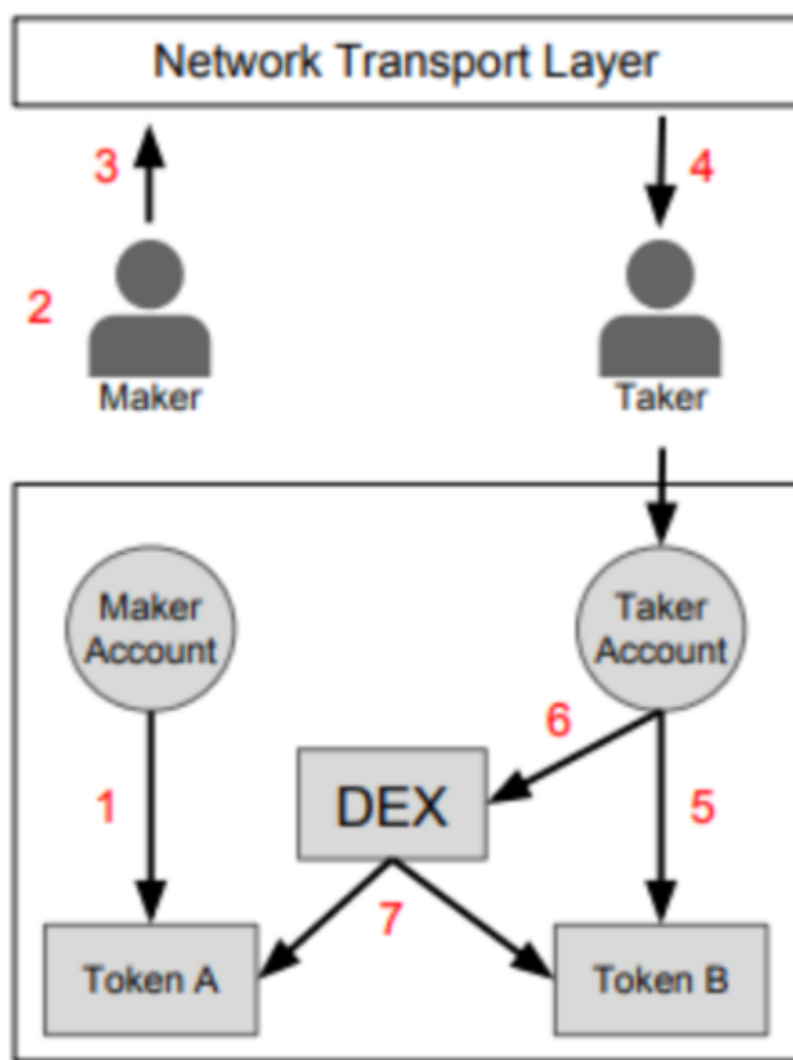
- Exchanges
- Asset management
- Stablecoins
- Lending / Borrowing
- Remittance

# Decentralised Exchanges

Decentralised Exchanges are a protocol to provide asset exchange without the platform holding the users assets



## Early Exchanges - 0x Protocol



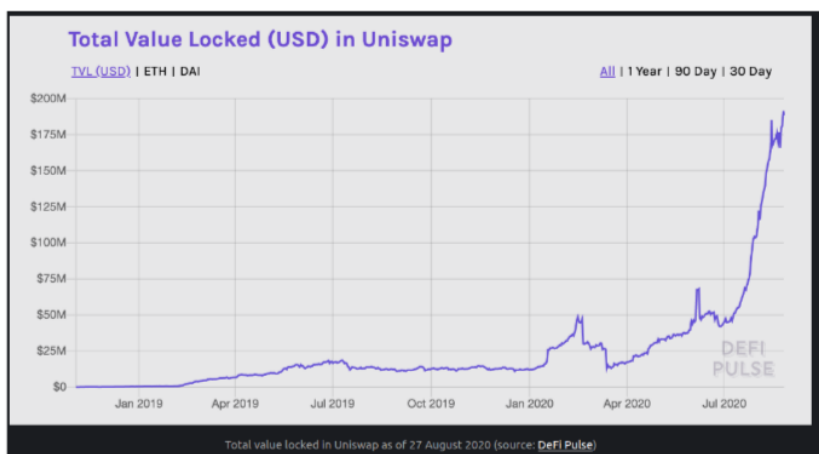
1. Maker approves the decentralized exchange (DEX) contract to access their balance of Token A.
2. Maker creates an order to exchange Token A for Token B, specifying a desired exchange rate, expiration time (beyond which the order cannot be filled), and signs the order with their private key.
3. Maker broadcasts the order over any arbitrary communication medium.
4. Taker intercepts the order and decides that they would like to fill it.
5. Taker approves the DEX contract to access their balance of Token B.

6. Taker submits the makers signed order to the DEX contract. 7. The DEX contract authenticates makers signature, verifies that the order has not expired, verifies that the order has not already been filled, then transfers tokens between the two parties at the specified exchange rate.

The first ideas came from Vitalik, Nick Johnson and Martin Koppelman in 2016 in a [Reddit post](#)

It was followed by an implementation from Hayden Adams and launched in Nov 2018

- Launched in 2018, Uniswap is a DEX featuring an AMM
- Solves the problem of illiquid assets since anyone can set up a liquidity pool



- Truly Decentralised
- Allows swap between any ERC20 pairs
- The code is robust

V2 Launched May 2020 allowing direct token swaps - halving gas fees

It solved many of the problems of the initial exchanges such as lack of incentives to provide liquidity for rarely traded assets.

It relies on a smart contract acting as an automatic market maker (AMM)

### Automatic Market Makers

#### Incentivising Users

- Users deposit funds into a liquidity pool, for example ETH and USDT
- This pool ( a token pair ) allows users to exchange (or maybe lend or borrow) tokens

- Interacting with the exchange incurs fees
- These fees are paid to the liquidity providers

They are characterised as constant function market makers.

From [Constant Function Market Makers](#)

The term “constant function” refers to the fact that any trade must change the reserves in such a way that the product of those reserves remains unchanged (i.e. equal to a constant).

The screenshot displays a swap interface on a decentralized exchange. At the top, there are navigation tabs: 'Swap' (selected), 'Pool', 'Vote', and 'Charts'. The main swap area is titled 'Swap' and includes a settings gear icon. It shows two input fields: the top one for 'ETH' with a value of '1' and a balance of '0 ETH' (~\$ 2,847.34); the bottom one for 'USDT' with a value of '2845.56' and a balance of '0 USDT' (~\$ 2,850.73, 0.119% difference). A double-headed arrow indicates the swap direction. Below the inputs, a share icon and the rate '1 USDT = 0.0003514 ETH' are shown. A large grey button at the bottom displays the error message 'Insufficient ETH balance'.



## Add Liquidity

[Clear All](#)UNI ETH

### Select Pair

ETH ▼UNI ▼

0.3% fee tier

1% select

Edit

### Deposit Amounts

ETH

Balance: 0 ETH (Max)

~\$ 2,847.22

UNI

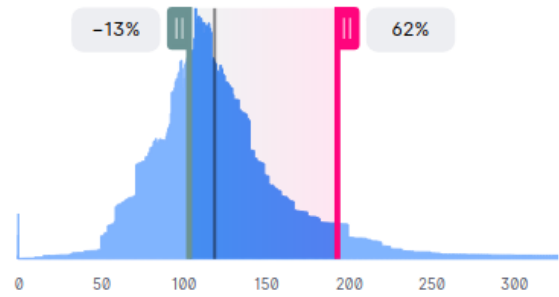
Balance: 0 UNI (Max)

~\$ 862.921

### Set Price Range



Current Price: 118.94 UNI per ETH



Min Price

-

103.94

+

UNI per ETH

Max Price

-

192.82

+

UNI per ETH

Full Range

Insufficient UNI balance

## LP Tokens

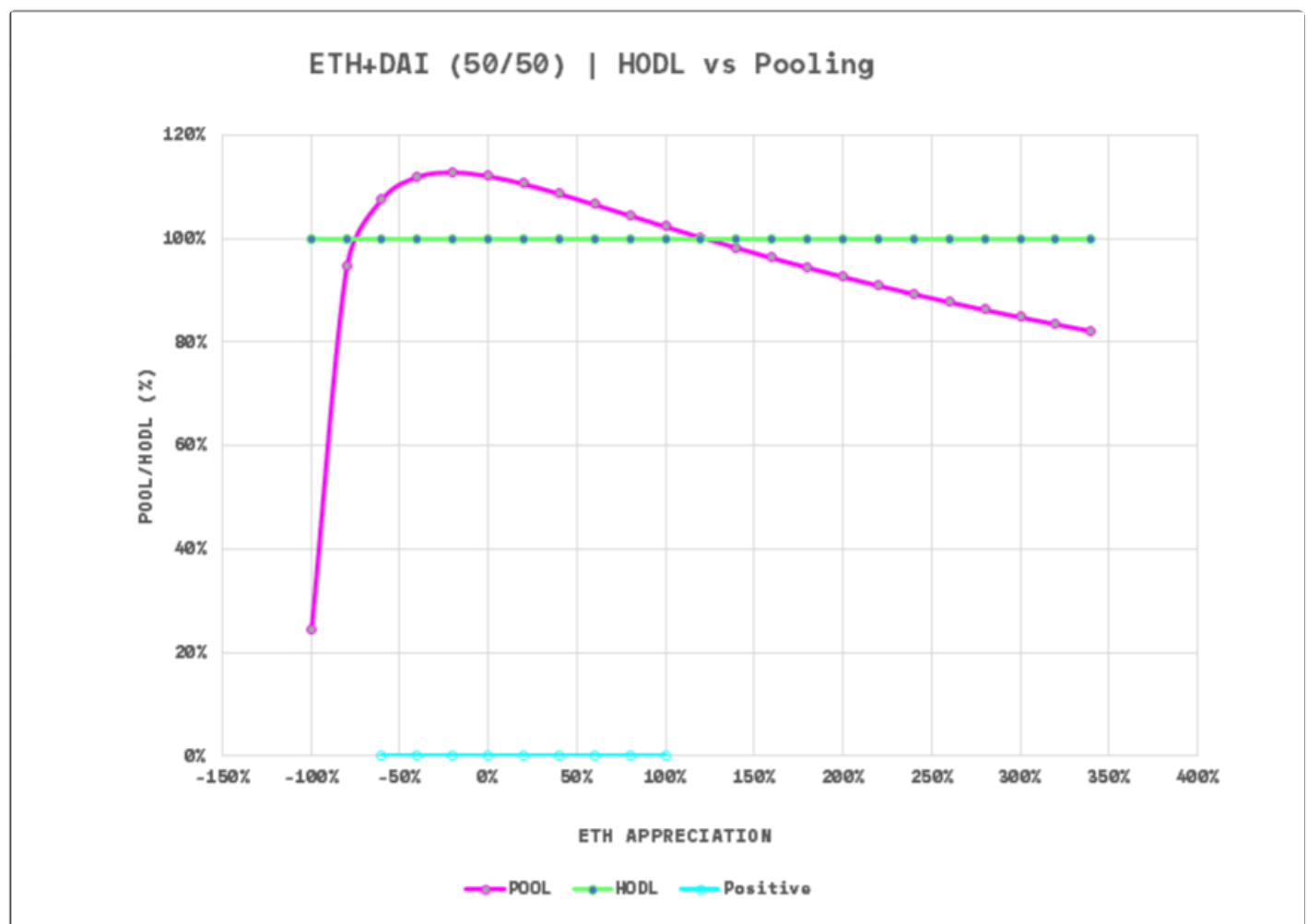
Typically the liquidity provider receives LP tokens when they add liquidity, say ETH and USDT

Later they can take liquidity by providing LP tokens to the contract and will receive back ETH and USDT.

Ideally they will make a profit

## Risks associated with AMMs

- Slippage
- Impermanent loss



While liquidity providers can use stablecoins, yields, and rewards to help lessen the impact of impermanent loss



they can also reduce this by using liquidity pools that use ratios other than 50/50.

Note that on Uniswap v3 LP positions are given by NFTs as opposed to the ERC-20 tokens used in Uniswap V1 and V2.

## Composability

A multitude of DeFi applications ("Money LEGOs") can be connected to create new financial products.

See [Monolith article](#)

The applications on the Ethereum network can run interchangeably, and they all support ETH and other ERC-20 tokens. They can be used in endless combinations, with no third party intermediary controlling any element of the network activity. Composability is a core basis of DeFi, and it's what's helped the ecosystem grow so quickly.

### Risks associated with composability

There are the usual risks associated with a protocol or smart contracts, but in addition there is a risk when combining products.

The contracts for one application may be secure until combined with those of another application. For example, on 12th March 2020, a crash in the price of ETH wreaked havoc for DeFi protocols as holders rushed to exit their positions. A gas price spike caused a lag across price oracles, leading to widespread liquidations in protocols like Maker.

### Compound

Compound III is an EVM compatible protocol that enables supplying of crypto assets as collateral in order to borrow the *base asset*. Accounts can also earn interest by supplying the base asset to the protocol.

The initial deployment of Compound III is on Ethereum and the base asset is USDC.



### Yield Farming

Yield Farming at its simplest is a means of earning rewards for depositing tokens

Users are rewarded for providing liquidity





Different strategies are used by investors to maximise their rewards from the many DeFi projects

Compound and yearn.finance introduced this area to DeFi

June 2020 BAT token

USD

Native

Assets ▾	Market size ▾	Total borrowed ▾	Deposit APY ▾	Borrow APR	
				Variable ▾	Stable ▾
 Basic Attention Token...	1.85M	1.47M	110.63 % 30D 1.51 % Avg.	193.70 % 30D 4.23 % Avg.	199.70 %
 WBTC Coin (WBTC)	143.78	127.03	24.17 % 30D 0.76 % Avg.	28.87 % 30D 1.36 % Avg.	38.04 %
 sUSD	332.92K	281.76K	14.03 % 30D 6.60 % Avg.	16.58 % 30D 8.07 % Avg.	—
 Binance USD (BUSD)	254.55K	216.03K	14.57 % 30D 5.38 % Avg.	17.17 % 30D 6.78 % Avg.	—

An innovative financial product

Does a risk free loan with no collateral required, of virtually any value , with an extremely low fee (say 0.09 %) seem to good to be true ?

Imagine that line 2 in this contract increases the account balance by 5

### Processing a transaction

Initial account balance = 5

line 1  
line 2  
line 3  
line 4  
line 5

Process flow

Final account balance = 10

### Transactions are atomic

Initial account balance = 5

line 1  
line 2  
line 3  
**line 4**  
line 5

Process flow

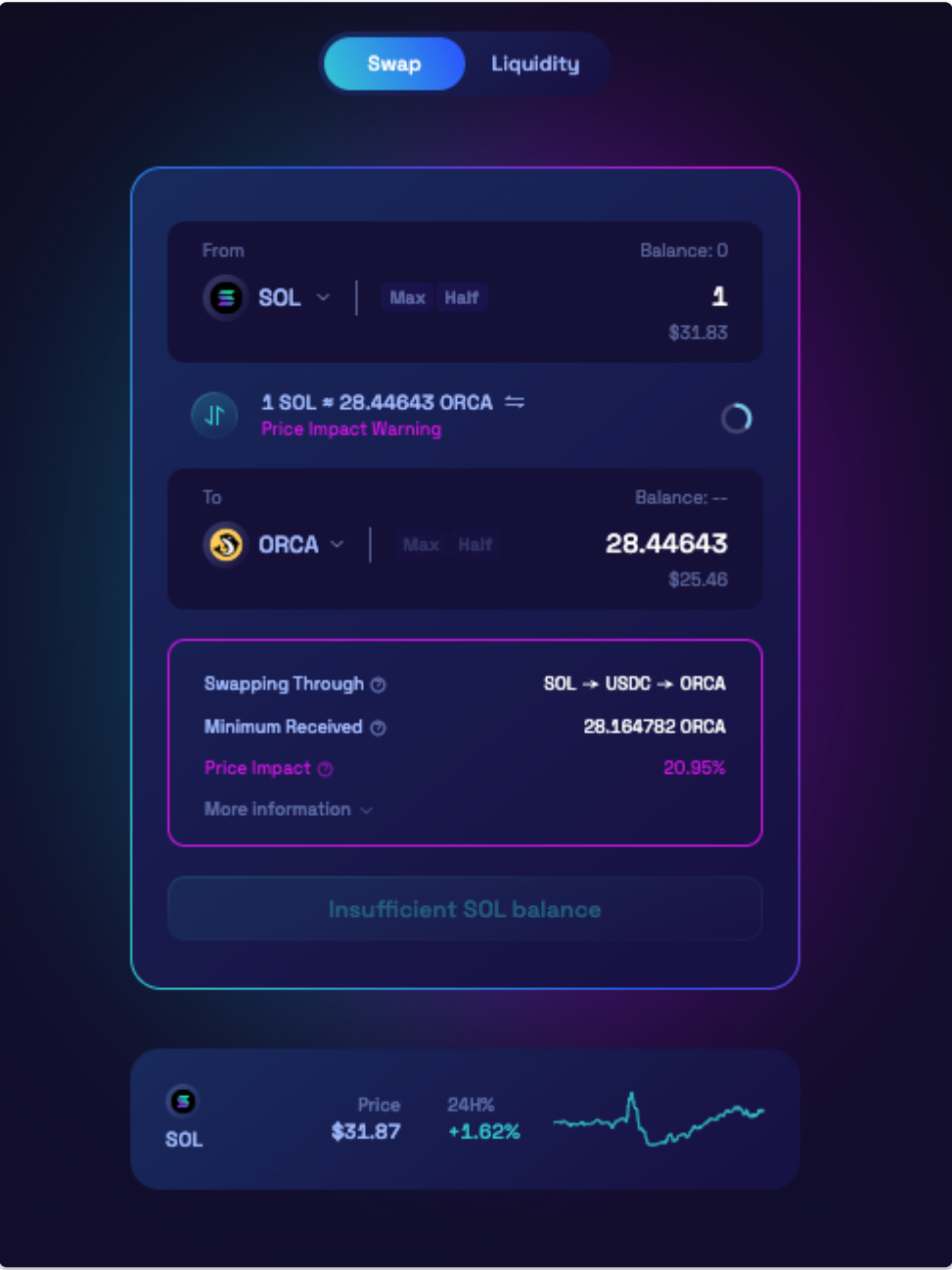
EVM reverts transaction

Final account balance = 5



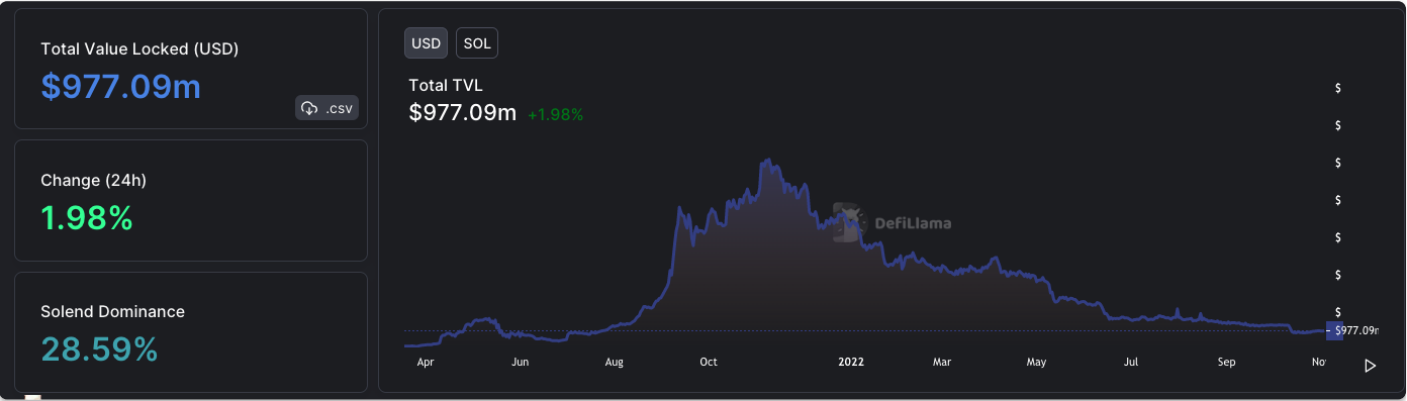
# DeFi on Solana

## Example from Raydium



# DeFi on Solana

## From [DeFiLlama](#)



Name	Category	Chains	1d Change	7d Change	1m Change	TVL	Mcap/TVL
1  Solend	Lending		+0.76%	+2.38%	-1.49%	\$279.4m	0.08
2  Marinade Finance	Liquid Staking		+1.48%	+2.39%	-9.27%	\$245.78m	0.05
3  Lido	Liquid Staking		+2.15%	+11.94%	-1.01%	\$143.88m	7.79
4  Raydium	Dexes		+0.78%	+0.79%	-26.65%	\$120.69m	0.6
5  Serum	Dexes		+12.81%	+4.35%	+49.05%	\$116.9m	2.43
6  Tulip Protocol	Yield		-0.50%	+1.45%	-27.68%	\$82.4m	0.07
7  Orca	Dexes		+0.13%	-7.24%	-17.48%	\$68.7m	0.31
8  Quarry	Yield		+0.54%	+5.34%	-36.88%	\$51.52m	
9  Saber	Dexes		+0.23%	+3.57%	-12.89%	\$48.56m	0.05
10  Francium	Yield		+1.19%	+2.01%	-35.49%	\$39.93m	
11  Swim Protocol	Cross Chain		+0.03%	-0.19%	-0.58%	\$37.06m	
12  Parrot Protocol	CDP		+0.21%	-0.38%		\$32.3m	
13  Atrix	Dexes		+0.58%	-1.54%	-81.05%	\$30.54m	
14  Cega	Options		+134%	+51.13%		\$30.44m	



# #### NFTs

**~970M**

market cap

**\$1.5**

average mint cost

(stats as of 12/29/21)

**5.7M+**

NFTs

WHY SOLANA

– Get started with the best reference implementations. Ecosystem projects provide resources to launch your NFT on Solana in **record speed**.

## On-chain, always

From auctions to perpetual royalties coded right into the NFT, Solana supports a fully decentralized on-chain experience for artists and collectors.

LEARN MORE

## NFT Standard

Focus on the artwork, not writing a new smart contract. The Solana NFT standard and minting program offers extreme customizability, with ecosystem-wide support.

LEARN MORE

## Permanent storage

Configure the best web3 storage option for your project, whether permanent, decentralized storage provided by ARweave, or other popular standards like IPFS.

LEARN MORE

We will look at NFT projects such as [Metaplex](#) in later lessons.

# Tokens on Solana

## Token Program

See [Docs](#)

The token program is part of the [solana program library](#) which is a collection of on-chain programs targeting the [Sealevel parallel runtime](#) , covering a number of areas such as

- tokens
- governance
- name service
- token swaps
- lending

The Solana token program is heavily used, its program id is : [TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA](#)

The program is used to create tokens with functionality similar to the ERC20 and ERC721 (NFT) standards.

## Comparison with Ethereum

Ethereum doesn't have any built in support for tokens, they rely on

- A standard to be followed - for example ERC20

- Libraries to be created by third parties to provide implementations

If you want to create a token on Ethereum, you will need to write and deploy a contract.

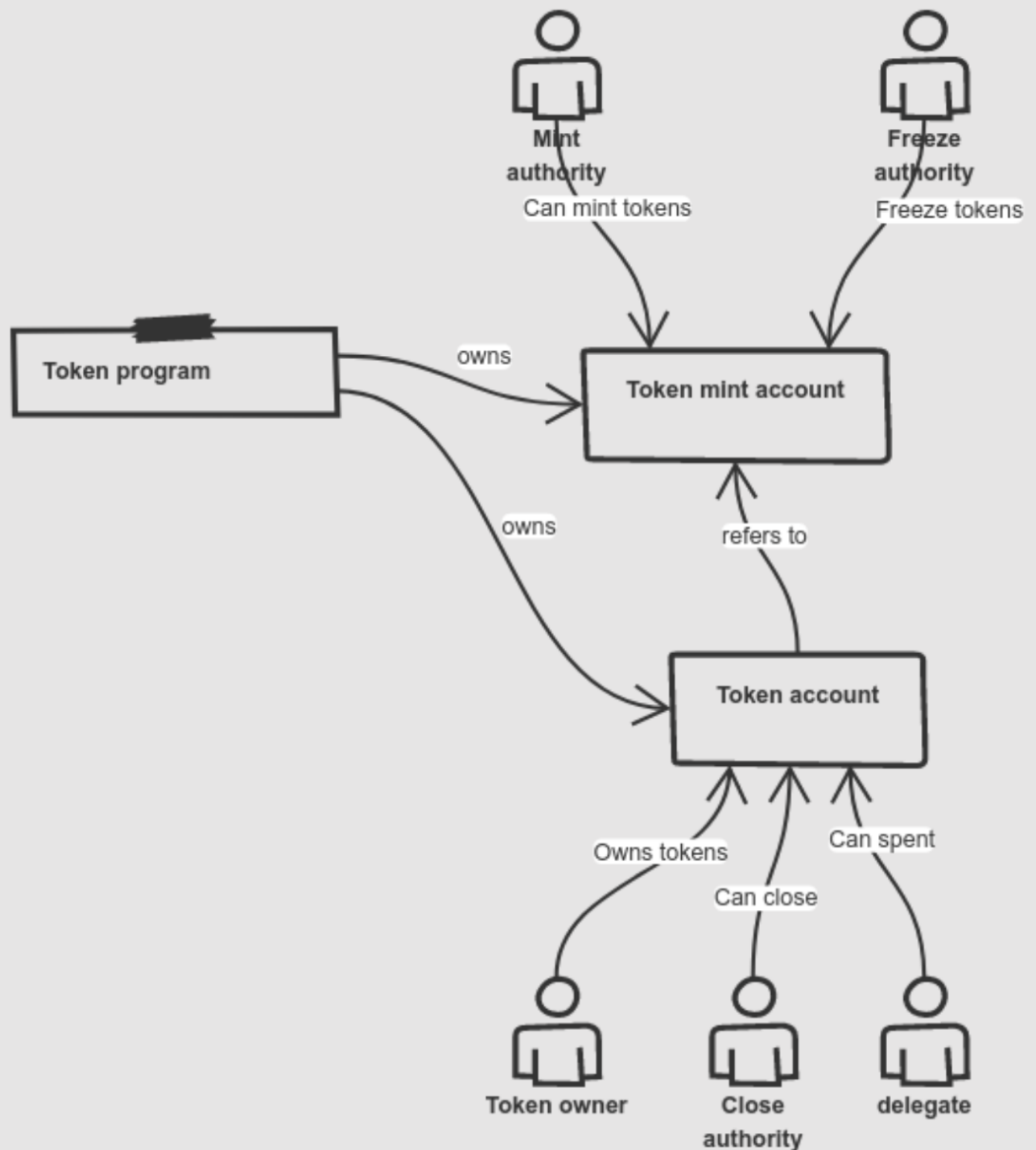
---

## Understanding the Accounts involved

See these articles

[article 1](#)

[article 2](#)



## The process of creating a token

When we create a token we need to create a token mint account which contains details about the token and who can do administrative functions.

This doesn't hold user's balances, those are held in the token account.

Any user who desires to hold any given token, needs a token account for the corresponding token.

The overall process is

1. Create token
2. Mint tokens to an account
3. Transfer tokens to other users

Behind the scenes these are running various functions.

For example when minting the tokens

We first call `InitializeMint`

This takes some parameters that are stored in a struct

```
pub struct Mint {  
    /// Optional authority used to mint new  
    tokens. The mint authority may only be  
    provided during mint creation. If no mint  
    authority is present then the mint has a  
    fixed supply and no further tokens may be  
    minted.  
    pub mint_authority: COption<Pubkey>,  
}
```

language-none

```
/// Total supply of tokens.
pub supply: u64,

/// Number of base 10 digits to the right of
the decimal place.
pub decimals: u8,

/// Is `true` if this structure has been
initialized
pub is_initialized: bool,

/// Optional authority to freeze token
accounts.
pub freeze_authority: COption<Pubkey>,

}
```

This then calls `InitializeAccount` which sets up the account struct

```
pub struct Account {
  /// The mint associated with this account
  pub mint: Pubkey,
  /// The owner of this account.
  pub owner: Pubkey,
  /// The amount of tokens this account holds.
  pub amount: u64,
  /// If `delegate` is `Some` then
```

language-none

```
`delegated_amount` represents
/// the amount authorized by the delegate
pub delegate: COption<Pubkey>,
/// The account's state
pub state: AccountState,
/// If is_native.is_some, this is a native
token, and the value logs the rent-exempt
reserve. An
/// Account is required to be rent-exempt,
so the value is used by the Processor to
ensure that
/// wrapped SOL accounts do not drop below
this threshold.
pub is_native: COption<u64>,
/// The amount delegated
pub delegated_amount: u64,
/// Optional authority to close the account.
pub close_authority: COption<Pubkey>,

}
```

Next the `MintTo` instruction is called, taking

- Public key of the mint
- Address of the token account to mint to
- The mint authority
- Amount to mint
- Signing accounts if `authority` is a multisig
- SPL Token program account

This will mint tokens to the destination account

---



## Transfer

To transfer tokens we invoke the function `process_transfer` this transfers a certain amount of token from a source account to a destination account:

We pass in the source and destination accounts and the amount.

The program will check that

1. Neither source account nor destination account is frozen
2. The source account's mint and destination account's mint are the same
3. The transferred `amount` is no more than source account's token amount

Note the source and destination can be the same.

## BURN

Burn is the opposite of Mint and removes tokens, from the supply and the given account.

## Approve

This allows transfer of a certain amount by a delegate.

- Only one delegate is possible per account / token.
- A new approval will override the previous one.

## Revoke

Removes the approval

### Freeze / Thaw Account

This will freeze / unfreeze the account preventing / allowing transfers / mints to it.

---

## Token-2022 Program

Note that extensions have been added to the token program to produce the Token-2022

Mint extensions currently include:

- confidential transfers
- transfer fees
- closing mint
- interest-bearing tokens
- non-transferable tokens

Account extensions currently include:

- memo required on incoming transfers
  - immutable ownership
  - default account state
-

## Associated Token Account Program

Why do we need an associated token account program ?

- A user may own arbitrarily many token accounts belonging to the same mint which makes it difficult for other users to know which account they should send tokens to and introduces friction into many other aspects of token management.

This program introduces a way to *deterministically* derive a token account key from a user's main System account address and a token mint address, allowing the user to create a main token account for each token they own. We call these accounts *Associated Token Accounts*.

- In addition, it allows a user to send tokens to another user even if the beneficiary does not yet have a token account for that mint. Unlike a system transfer, for a token transfer to succeed the recipient must have a token account with the compatible mint already, and somebody needs to fund that token account. If the recipient must fund it first, it makes things like airdrop campaigns difficult and just generally increases the friction of token transfers. The Associated Token Account program allows the sender to create the associated token account for the receiver, so the token transfer just works.



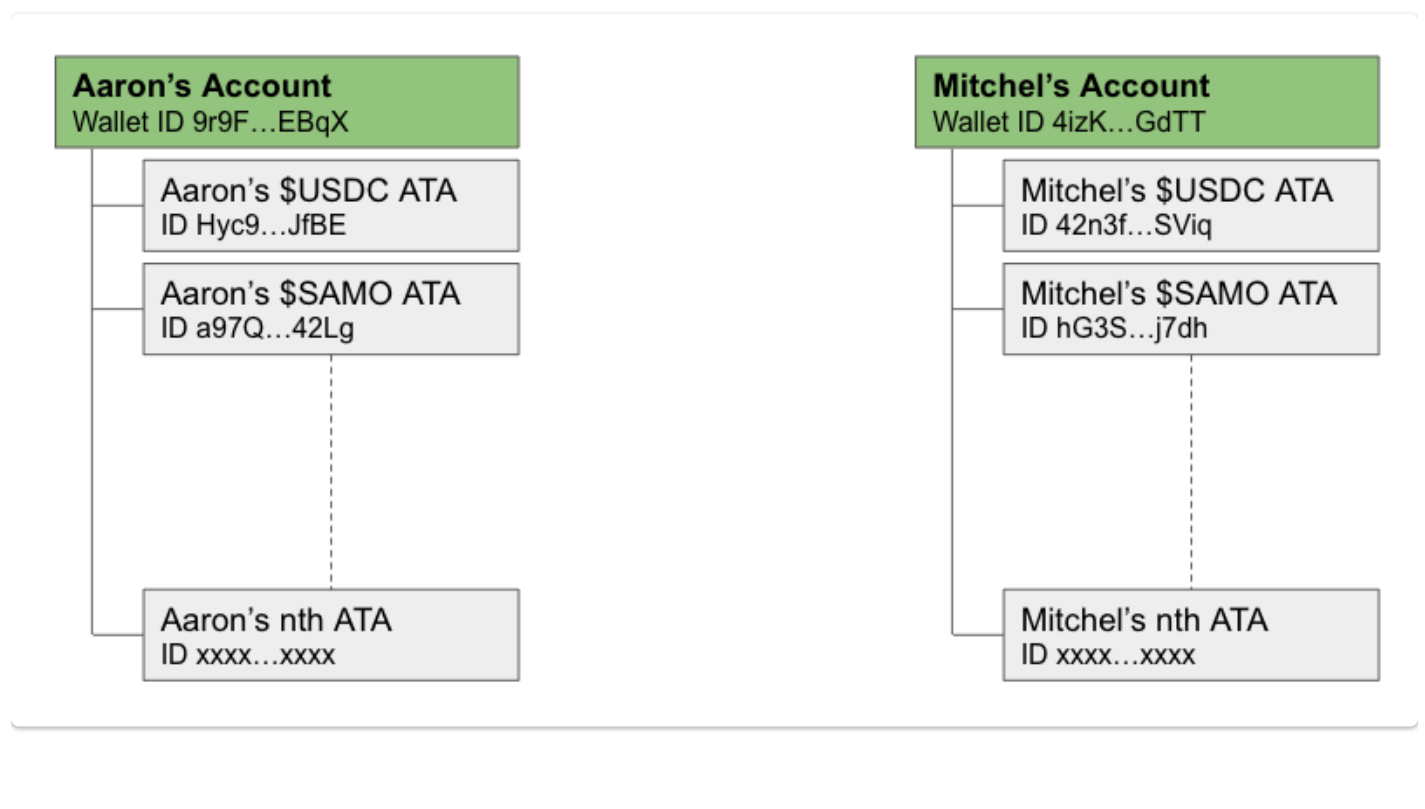


From [QuickNode Guide]

(<https://www.quicknode.com/guides/solana-development/how-to-transfer-spl-tokens-on-solana>)

"The Solana Token Program derives "a token account key from a user's main System account address and a token mint address, allowing the user to create a main token account for each token they own"

(Source: [spl.solana.com](https://spl.solana.com)). That account is referred to as an Associated Token Account or "ATA." Effectively an ATA is a unique account linked to a user and a specific token mint."



## Using the command line tools to create tokens

We can use the `spl-token-cli` tool to create tokens. There is detailed [documentation](#) available

### Setup

Make sure you have Rust and the Solana CLI installed then

Install the SPL token CLI

```
cargo install spl-token-cli
```

If you have a missing `libudev` dependency you can install on linux with

```
sudo apt install -y pkg-config libusb-1.0-0-dev libftdi1-dev  
sudo apt-get install libudev-dev
```

Check which network you are configured for

```
solana config get
```

You can set the required cluster with

```
solana config set --url  
https://api.devnet.solana.com
```

If you don't have a key pair, then generate one

```
mkdir ~/my-solana-wallet  
solana-keygen new --outfile ~/my-solana-
```



```
wallet/my-keypair.json
```

display the result with

```
solana-keygen pubkey ~/my-solana-wallet/my-  
keypair.json
```

verify your address

```
solana-keygen verify <PUBKEY> ~/my-solana-  
wallet/my-keypair.json
```

and set the keypair

```
solana config set --keypair ~/my-solana-  
wallet/my-keypair.json
```

Airdrop yourself some tokens

```
solana airdrop 1
```

---

## Creating a fungible token

```
spl-token create-token
```

You should see something like

```
Creating token  
AQoKYV7tYpTrFZN6P5oUufbQKAUr9mNYGe1TTJC9wajM  
Signature:  
47hsLFxWRCg8azaZZPSnQR8DNTRsGyPNfUK7jqyzgt7wf9e  
ag3nSnewqoZrVZHKm8zt3B6gzxhr91gdQ5qYrsRG4
```

Note that this doesn't have a supply yet

You can test this with

```
spl-token supply <Token ID>
```

Now create an account to hold the token

```
spl-token create-account <Token ID>`  
  
Creating account  
7UX2i7SucgLMQcfZ75s3VXmZZY4YRUyJN9X1RgfMoDUi  
Signature:  
42Sa5eK9dMEQyvD9GMHuKxXf55WLZ7tfjabUKDhNoZRAxj9  
MsnN7omriWMEHXLea3aYpjZ862qocRLVikvkHkyfy
```

Mint some tokens into that account

```
spl-token mint <Token ID> 100
```

You can check the token balance for an account with

```
spl-token balance <Token ID>
```

and the supply with

```
spl-token supply <Token ID>
```

If you want a summary of the tokens that you own, use

```
spl-token accounts
```

### Transferring tokens

See [Docs](#)

```
spl-token transfer <Token ID> <amount>  
<destination>
```

Note that the destination account must already be set up for that token

If the account is not already setup for that token you can use

```
spl-token transfer --fund-recipient <Token ID>  
<amount> <destination>
```

The alternative is for the receiver to first set up their account to receive that token

```
spl-token create-account <Token ID>
```

They can then receive tokens by the sender using

```
spl-token transfer <Token ID> <amount>  
<destination>
```

---

## Non Fungible Tokens

To create an NFT we

1. Create a token with zero decimal places

```
spl-token create-token --decimals 0
```

2. Setup the account as for a fungible token

```
1. spl-token create-account <Token ID>
```

3. Mint 1 token to that account

```
1. spl-token mint <Token ID> 1 <Account>
```

4. Disable future minting:

```
spl-token authorize <Token ID> mint --  
disable
```

You can check the token details with

```
$ spl-token account-info <Token ID>
```

```
spl-token supply <Token ID>
```