Lesson 8 - Token Program

Intro to DeFi

Areas within DeFi

- Exchanges
- Asset Management
- Stablecoins
- Bridges
- Lending / Borrowing
- Remittance

Class Question

Which DeFi products have you used ?

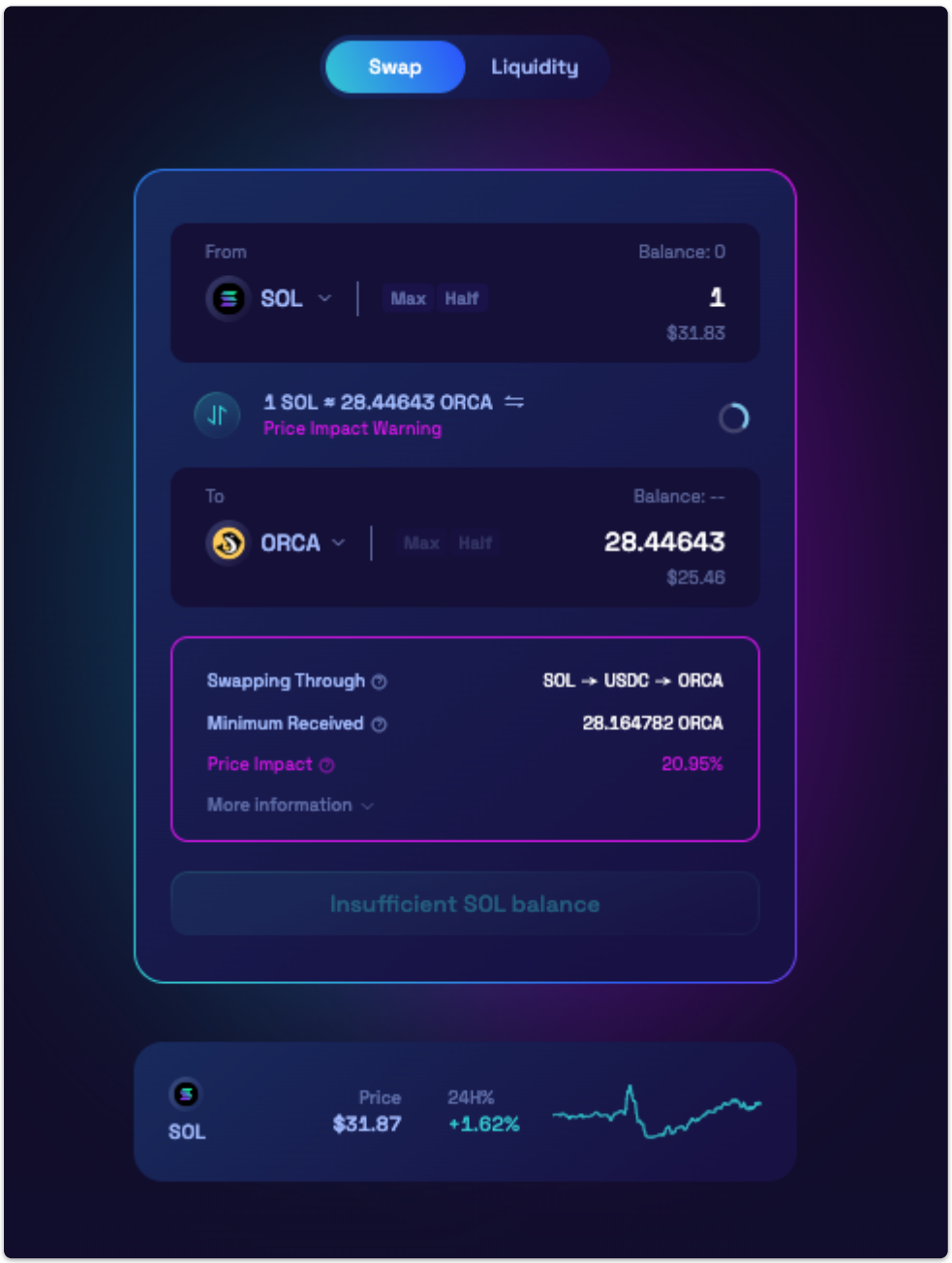
Digression about Decentralised Exchanges

LIQUIDITY POOLS

Users deposit funds into liquidity pools and get LP tokens in return
The depositors can later exchange their LP tokens for their original funds (+ fees accrued)
Other users can use the liquidity pools to exchange between the tokens in the pool and will pay a fee to do so.
The program controlling this is an automatic market maker AMM, and relies upon a constant function to calculate the relative price of the tokens.

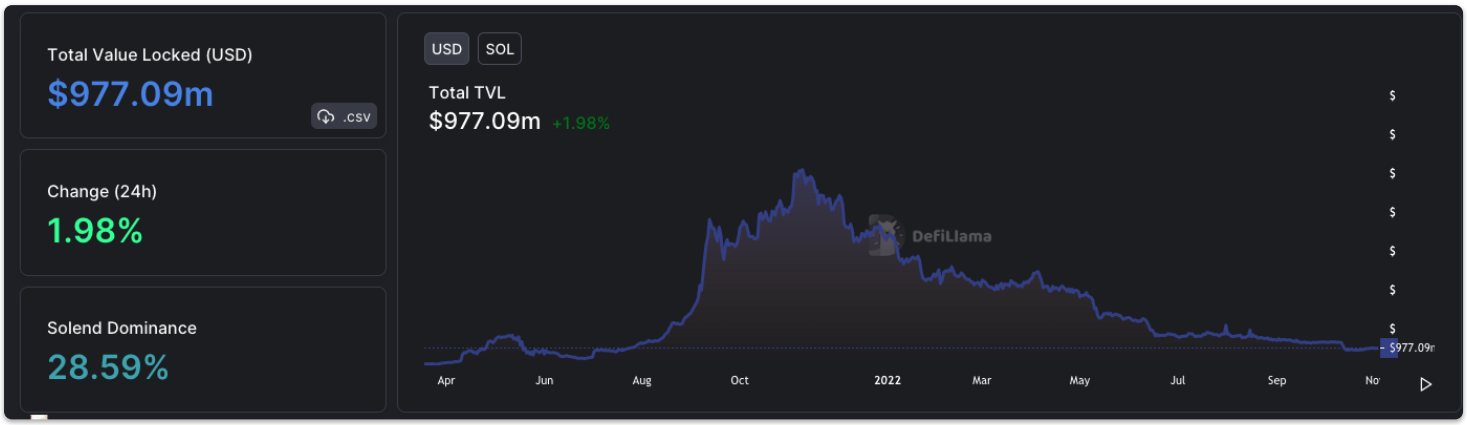
You may also receive governance tokens for contributing to a product.

EXAMPLE FROM RAYDIUM



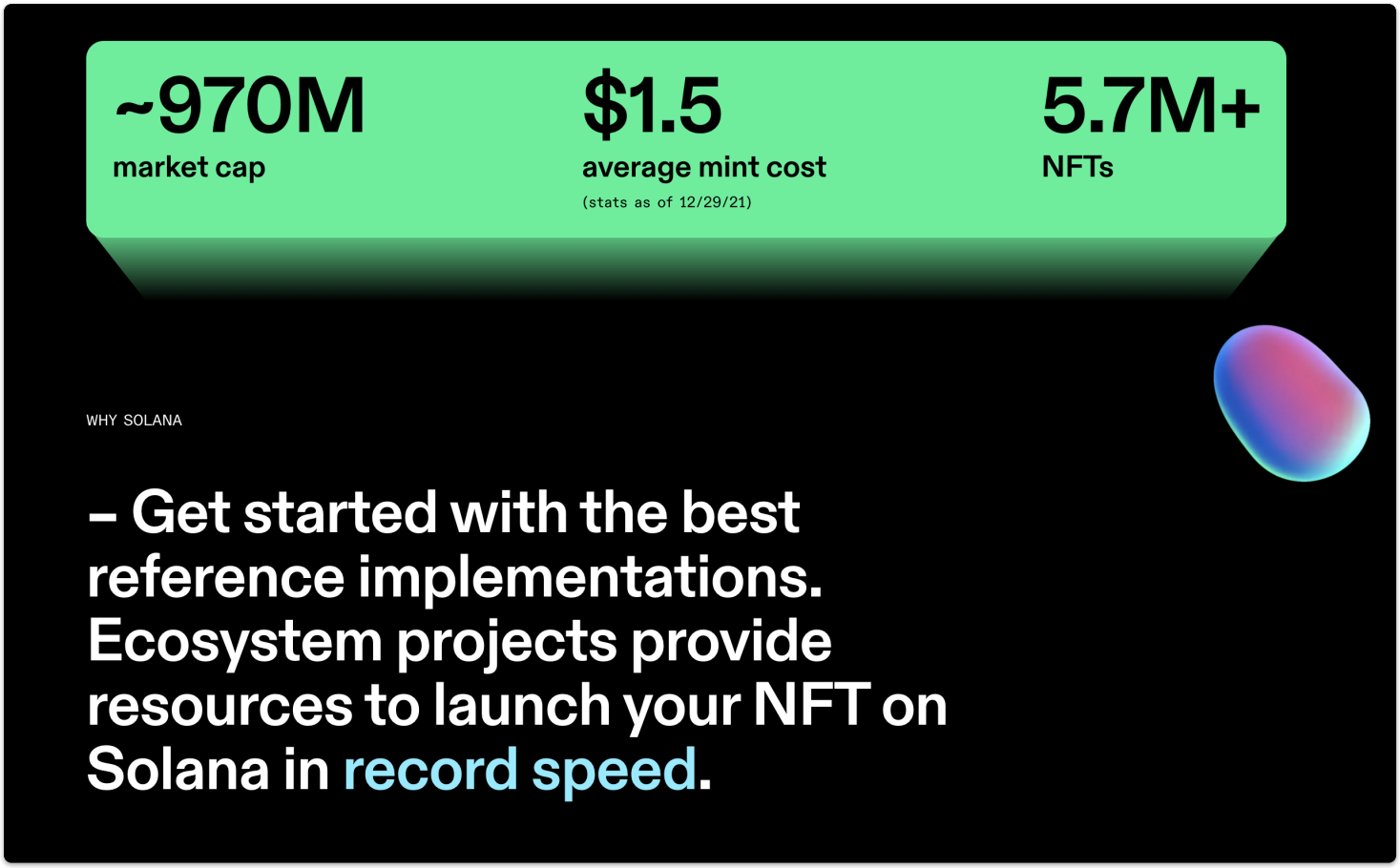
DeFi on Solana

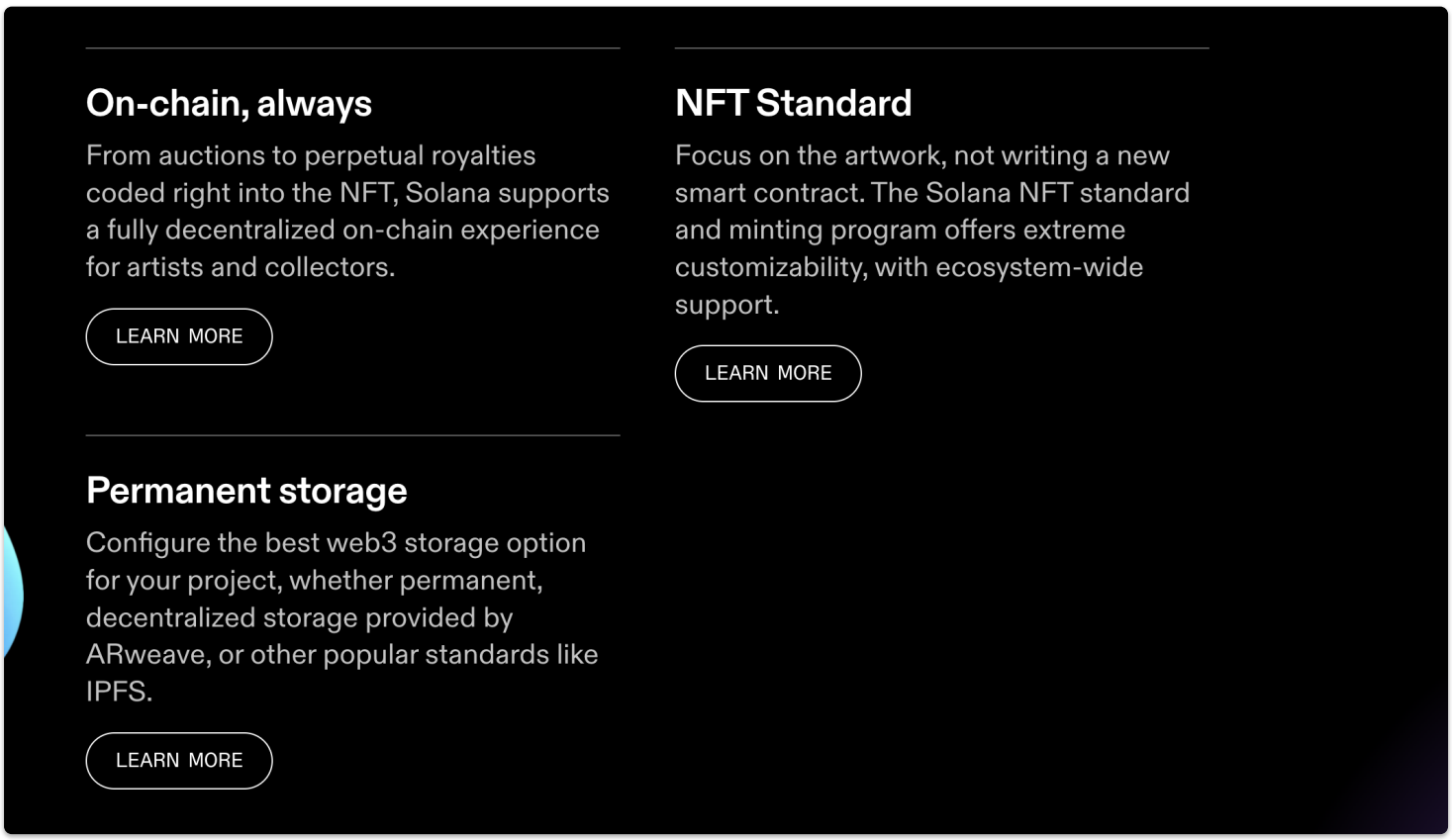
From DeFiLlama



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





We will look at NFT projects such as Metaplex in later lessons.

Token Program

See Docs

The token program is part of the solana program library which is 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

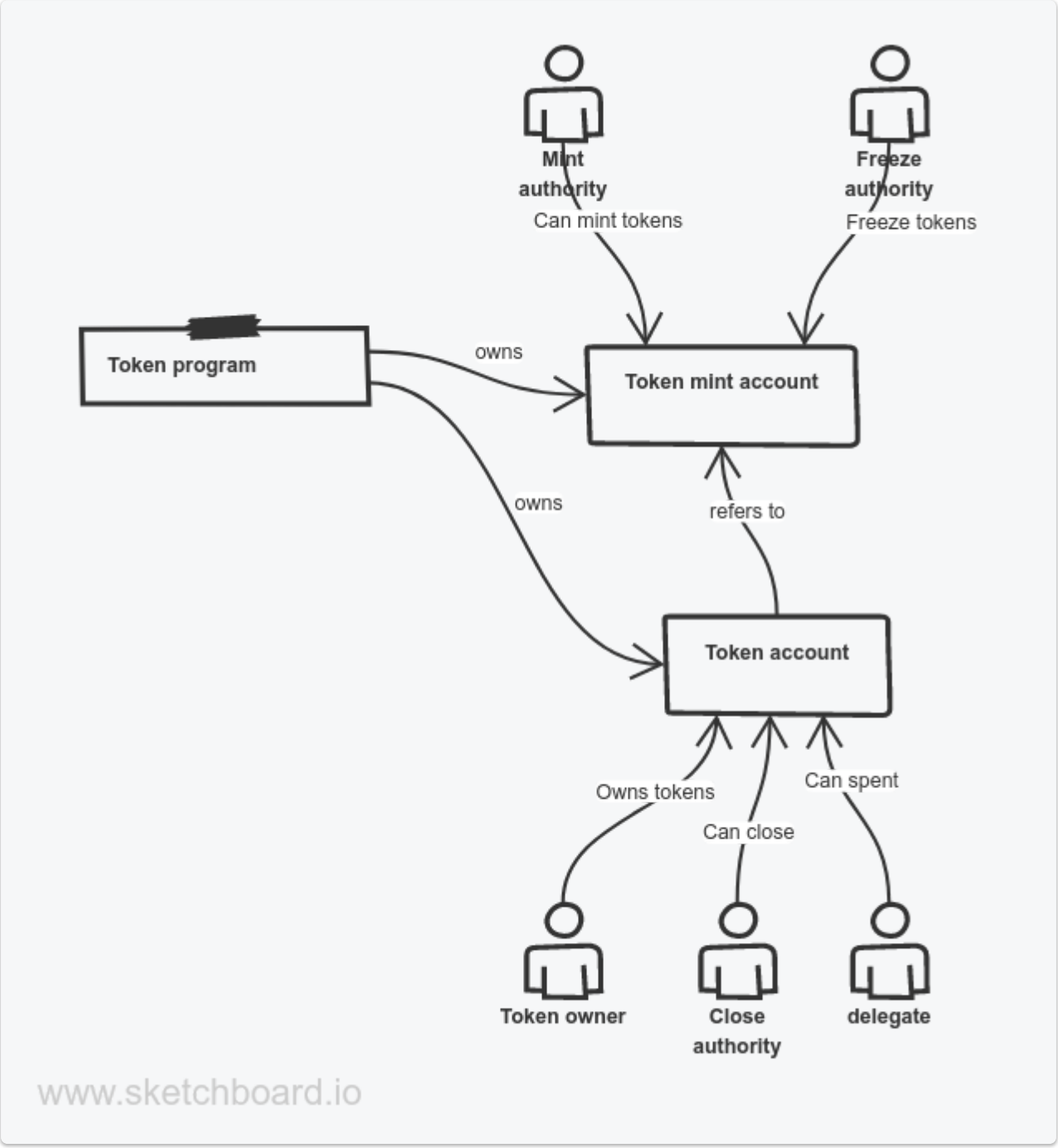
If you want to create a token on Ethereum, you will need to write and deploy a contract.

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 prov
pub mint_authority: COption<Pubkey>,

/// 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 `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-exemp
/// Account is required to be rent-exempt, so the value is used by the Processor to
/// 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 overrride 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

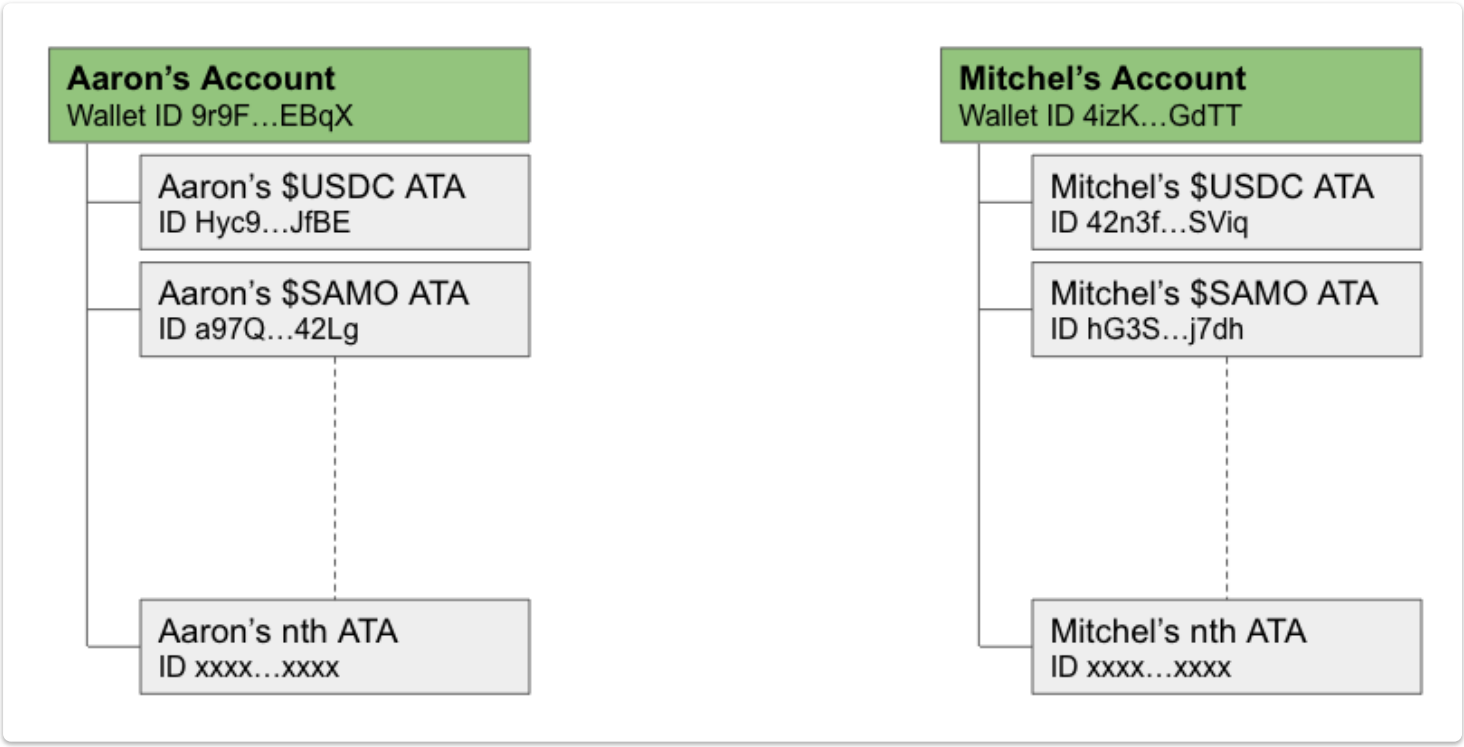
Associated Token Account Program

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). 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/new-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:
47hsLFxWRCg8azaZZPSnQR8DNTRsGyPNfUK7jqyzgt7wf9eag3nSnewqoZrVZHKm8zt3B6gzxhr91gdQ5q
YrsRG4
```

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:
42Sa5eK9dMEQyvD9GMHuKxXf55WLZ7tfjabUKDhNoZRAxj9MsnN7omriWMEHXLea3aYpjZ862qocRLVikv
kHkyfy
```

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`

Transfering 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 they 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>`

Rust - (more) pattern matching

See Docs

Ignoring values and matching literals

We can use `_` to show we are ignoring a value, or to show a default match, where we don't care about the actual value.

```
    let x = ...;

    match x {
        1 => println!("one"),
        3 => println!("three"),
        5 => println!("five"),
        _ => println!("some other value"),
    }
```

Ranges

Use `..=`
For example

```
let x = 5;
match x {
        1..=5 => println!("one through five"),
        _ => println!("something else"),
}
```

Variables

Be aware the match starts a new scope, so if you use a variable it may shadow an existing variable.

```
let x = Some(5);
let y = 10;
match x {
        Some(50) => println!("Got 50"),
        Some(y) => println!("Matched, y = {y}"),
        _ => println!("Default case, x = {:?}", x),
}
println!("at the end: x = {:?}, y = {y}", x);
```

What we get printed out is
`Matched, y = 5`.
`x = Some(5), y = 10`.

Destructuring

We can destructure structs and match on their constituent parts

For example

```
fn main() {
let p = Point { x: 0, y: 7 };
match p {
        Point { x, y: 0 } => println!("On the x axis at {}", x),
        Point { x: 0, y } => println!("On the y axis at {}", y),
        Point { x, y } => println!("On neither axis: ({}, {})", x, y), }
}
```

Adding further expressions

```
let num = Some(4);
match num {
    Some(x) if x % 2 == 0 => println!("The number {} is even", x),
    Some(x) => println!("The number {} is odd", x),
    None => (),
}
```

Next Week

We will continue with development of Solana programs, and introduce the Anchor framework.