ackee | blockchain security

School of Solana

LECTURE 3

Solana Programming Model Introduction

Ondrej Rehacek | Josef Gattermayer, Ph.D.

About this and next lectures

ackee | blockchain security

About this lecture

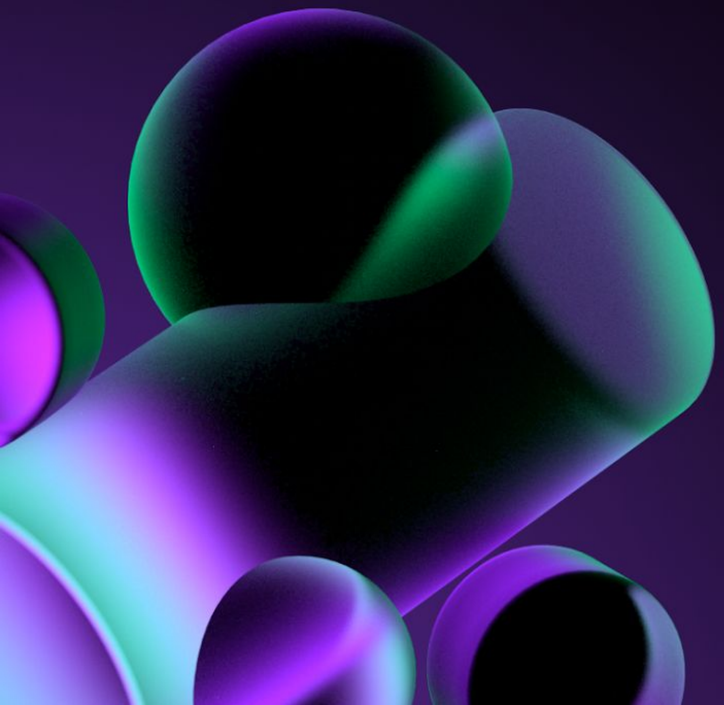
- **Introduction to Solana Programming model**
  - Overview
  - Accounts
  - Transactions
- **Hands on Example**
  - Setting up a new Anchor project
  - Your first Solana Program in Anchor (Calculator) + writing tests

ackee | blockchain security

About next lectures

- **More Solana Programming model**
  - PDAs (Program-derived accounts)
- **Solana Program frontend**
- **Building out our project**
  - Your first PROPER Solana Program (Twitter Clone)
  - Turnstile testing

Core Concepts



ackee | blockchain security

Overview

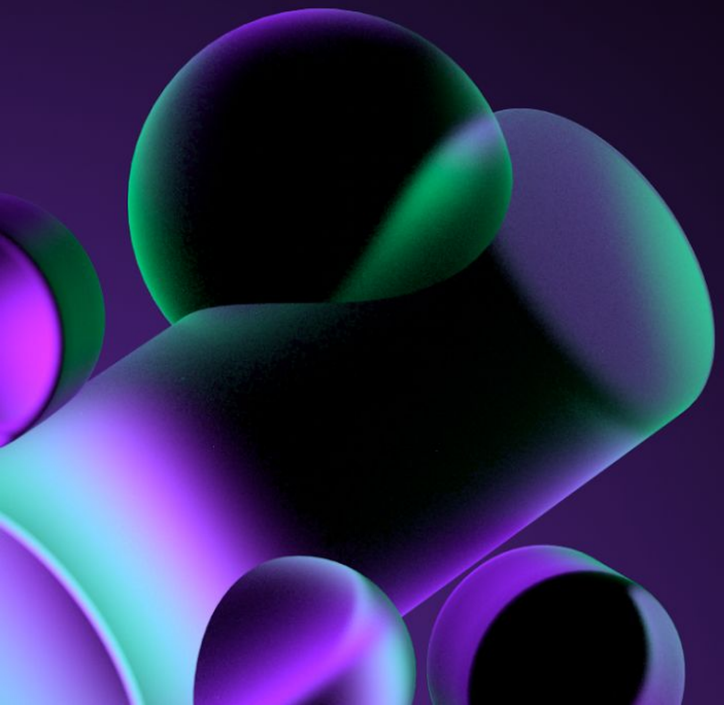
- An app interacts with a Solana cluster by sending it transactions with one or more instructions.
- The Solana runtime passes those instructions to programs deployed by app developers beforehand.
- Instructions are executed sequentially and atomically for each transaction.
- If any instruction is invalid, all account changes in the transaction are discarded

ackee | blockchain security

Program

- Piece of code that runs by Solana blockchain
- Programs are stateless. Meaning you can't store any data in them.
- We use Accounts to store both program's code and "data" - We can imagine them as "files"

Accounts



ackee | blockchain security

Accounts

- There are 3 kinds of accounts on Solana:
  - Data accounts store data.
  - Program accounts store executable programs.
  - Native accounts that indicate native programs on Solana such as `System`, `Stake`, `BPF Loader`, ...

Data Account
lamports: 10
owner: [Program]
executable: False
rent_epoch: 0
data: Vec<u8>

Program Account
lamports: 10
owner: BPF Loader
executable: True
rent_epoch: 0
data: executable byte code

Native Account
lamports: 10
owner: NativeLoader
executable: True
rent_epoch: 0
data: executable byte code

ackee | blockchain security

Accounts

```python
class Counter:
  def __init__(self, value):
    self.value = value

  def increment(self):
    self.value += 1

  def decrement(self):
    self.value -= 1
```

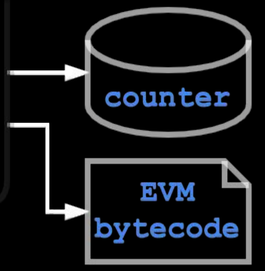
Smart Contract Account

balance: 10

storage hash: [xyz]

code hash: [abc]

nonce: [nonce]



Program Account

lamports: 10

owner: BPF Loader

executable: True

rent_epoch: 0

data: executable byte code

Data Account

lamports: 10

owner: Program Account

executable: False

rent_epoch: 0

data: counter = [value]

ackee | blockchain security

Accounts

- Accounts can only be owned by programs.
  - Owner has full autonomy over the owned accounts.
  - It is up to the program's creator to limit this autonomy and up to the users of the program to verify the program's creator has really done so

System Program: transfer

```
// ...
if from.signer_key().is_none() {
    ic_msg!(
        invoke_context,
        "Transfer: `from` account {} must sign",
        from.unsigned_key()
    );
    return Err(InstructionError::MissingRequiredSignature);
}

transfer_verified(from, to, lamports, invoke_context)
// ...
```

ackee | blockchain security

Accounts

- Used to identify an account.
- 256-bit long.
- Usually a public key of `ed25519` keypair.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- A program owner of this account.
- The only account that can write to the data.
- Only the owner of an account may assign a new owner.
- It can only be changed if the data is zero.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- The number of lamports owned by this account.
- Only the owner of an account may subtract its lamports.
- 1 ◎ = 1_000_000_000 lamports.
- Runtime asserts total_lamports_before == total_lamports_after.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- The raw data byte array stored by this account.
- Up to 10 MB of mutable storage.
- Can only be written by the owner account.
- Can not be resized (currently).

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- Accounts are marked as executable during a successful program deployment process.
- Executable accounts are fully immutable once they are marked as final.
- Owned by the bpf loader program.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- Accounts are held in validator memory and pay ”rent” to stay there.
- Charged every epoch (~2 days) and are determined by account size.
- Accounts with sufficient balance to cover 2 years of rent are exempt from fees.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- Flag indicating if an account has signed a transaction.
- It is not actually stored in the account.
- It's just runtime metadata.

```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

Accounts

- Flag indicating if an account is writable.
- It is not actually stored in the account.
- It's just runtime metadata.

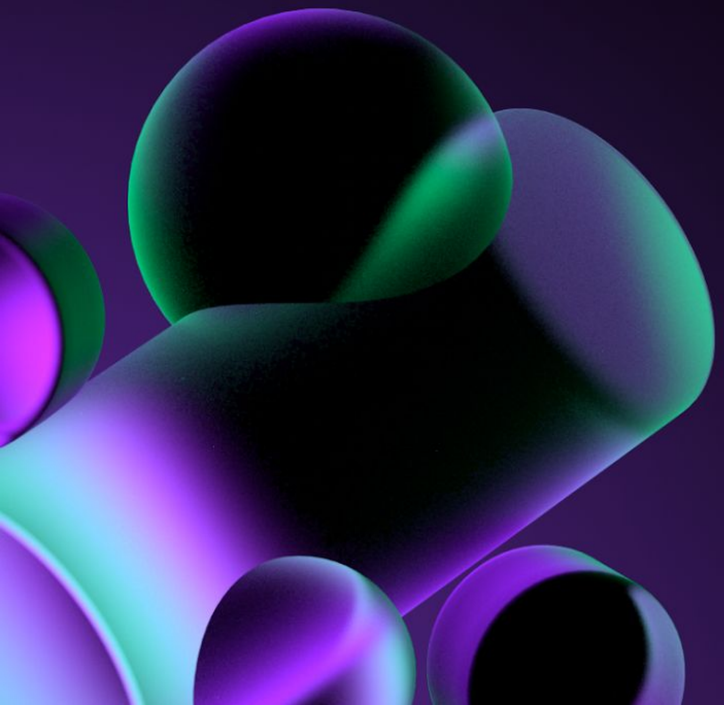
```rust
pub struct AccountInfo<'a> {
    pub key: &'a Pubkey,
    pub is_signer: bool,
    pub is_writable: bool,
    pub lamports: Rc<RefCell<&'a mut u64>>,
    pub data: Rc<RefCell<&'a mut [u8]>>,
    pub owner: &'a Pubkey,
    pub executable: bool,
    pub rent_epoch: Epoch,
}
```

ackee | blockchain security

7 Commandments of Solana Accounts

1. Each account has an unique address and an owner (some program).
1. Owner has full autonomy over the owned accounts.
1. Only a data account's owner can modify its data and debit lamports.
1. Program accounts don't store state.
1. Accounts must pay rent to stay alive (otherwise they will be deleted at the end of the transaction).
1. Anyone is allowed to credit lamports to a data account.
1. The owner of an account may assign a new owner if the account's data is zeroed out.

Transactions



ackee | blockchain security

Transactions

- The basic operational unit on Solana is an **instruction**:
  - The `program_id` of the intended program.
  - An array of all `accounts` it intends to read from or write to.
  - An `instruction_data` byte array that is specific to the intended program.

System Program: transfer instruction

```
// ...
let ix = Instruction::new_with_bincode(
    // program_id:
    system_program::id(),
    // instruction_data:
    &SystemInstruction::Transfer { 50 }
    // accounts:
    vec![
        AccountMeta { pubkey: from_pubkey, is_signer: true, is_writable: true },
        AccountMeta { pubkey: to_pubkey, is_signer: false, is_writable: true },
    ]
);
// ...
```

ackee | blockchain security

Transactions

- One or more instructions can be bundled into a **transaction**:
  - An array of all `accounts` it intends to read from or write to.
  - One or more `instructions`.
  - A recent `blockhash`.
  - One or more `signatures`.

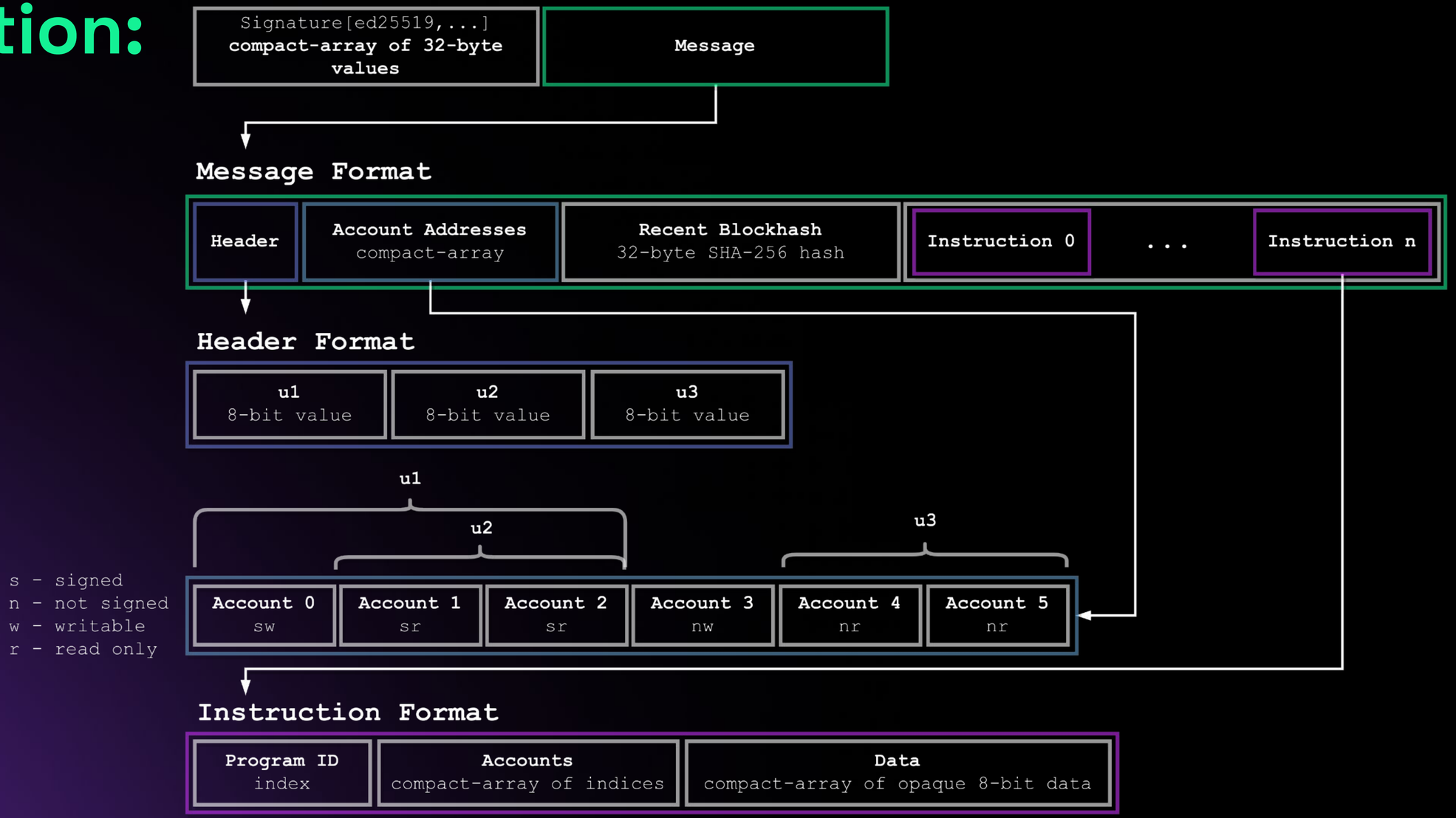
- Writable signer account that is serialized first will be the fee payer.

System Program: transfer transaction

```
// ...
let tx = Transaction::new_signed_with_payer(
    // instructions:
    &[ix],
    // payer:
    Some(&from_pubkey),
    // signing_keypairs
    &[&from_keypair],
    // recent_blockhas
    recent_blockhash,
);
// ...
```

ackee | blockchain security

Transaction:



ackee | blockchain security

Transactions

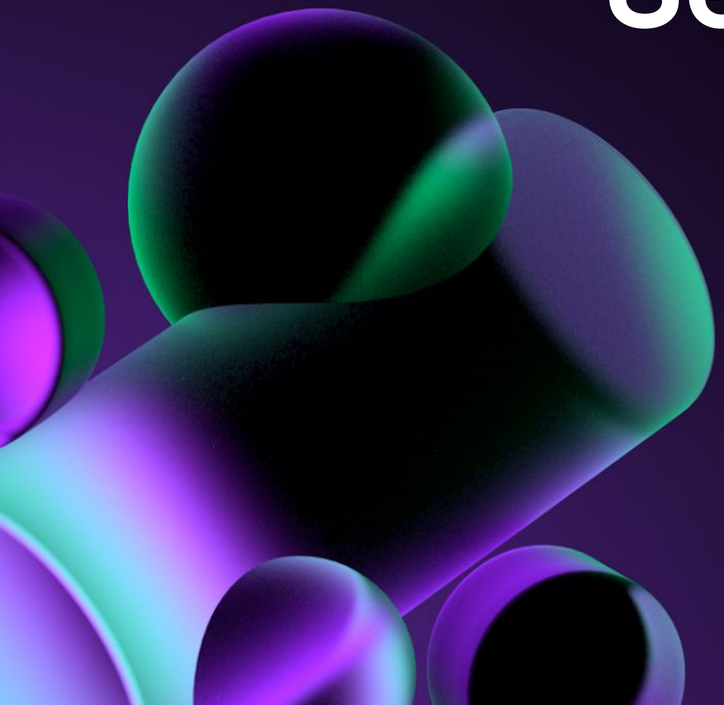
- Instructions in one transaction are processed in order and atomically.
- If any part of an instruction fails, the entire transaction fails.
- Transaction are limited to 1232 bytes.
- To prevent a program from abusing computation resources each instruction in a transaction is given a compute budget.

ackee | blockchain security

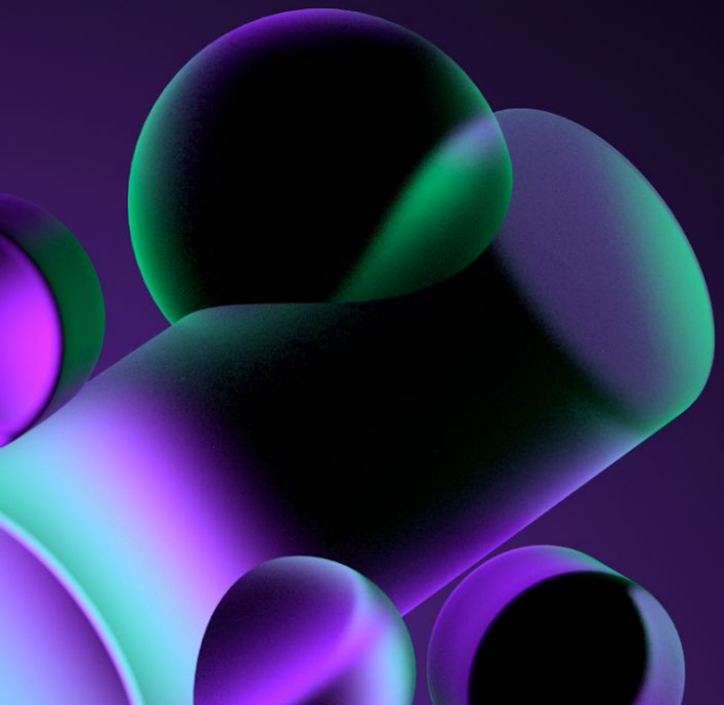
5 Commandments of Solana Transactions

1. All program inputs are potentially malicious.
    a. User composes an instruction/transaction.
    b. User provides all the accounts.
1. Check the signers.
1. Check the owner.
1. Beware of unexpected order of instructions within the transaction.
1. Think of compute budget.

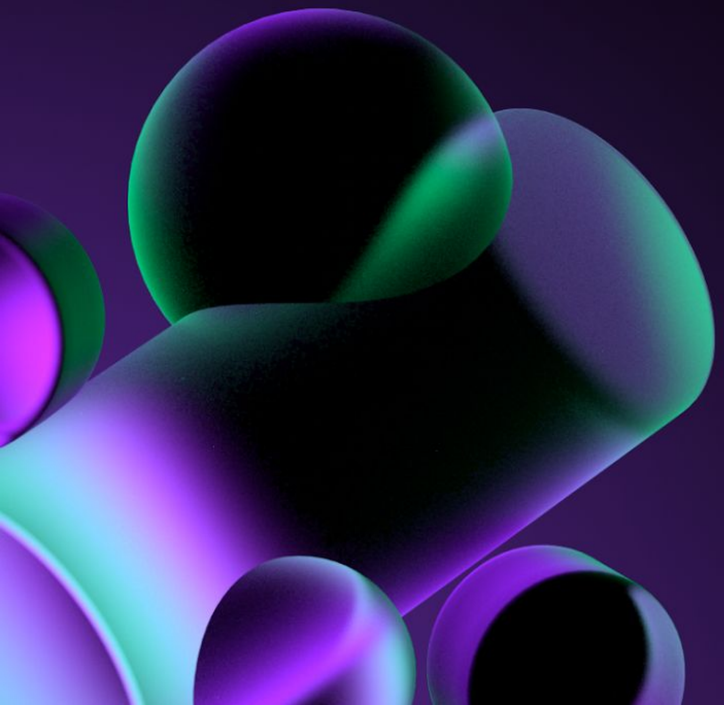
Set Up Anchor Project



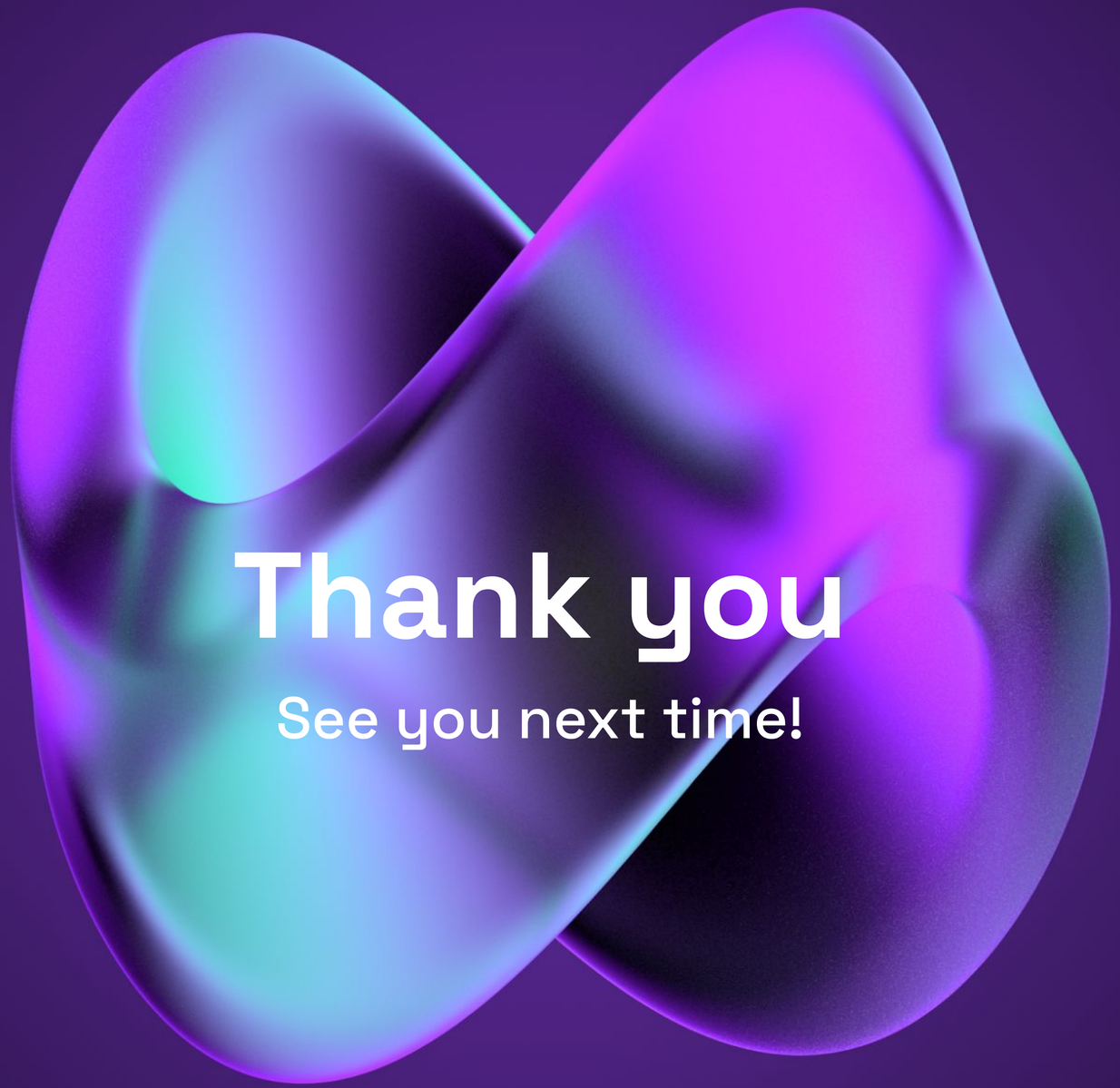
Our First Program



Task 3



ackee | blockchain security



Thank you

See you next time!