

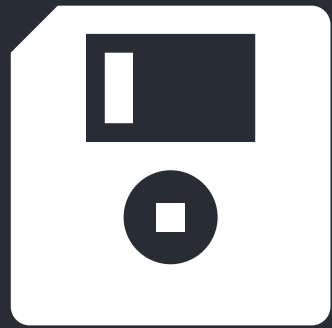
เริ่มต้น

Solana Dev

ฉบับเร็ว

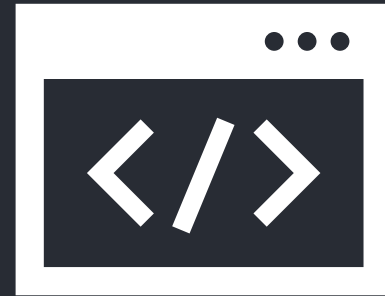


ใน Solana มีสิ่งที่เราต้องรู้จักอยู่ 2 อย่าง
ด้วยกันนั่นก็คือ



Account

+



Program

อันนี้มันไม่มีอะไรมากหรอก
มันก็คือตัว Smart Contract
นั่นแหละ 555 เป็นตัวใหญ่
คอยคุม Accounts อีกที



Program

ปกติใน Solana เราจะเขียนโปรแกรม
ด้วย ภาษา Rust + Anchor Framework



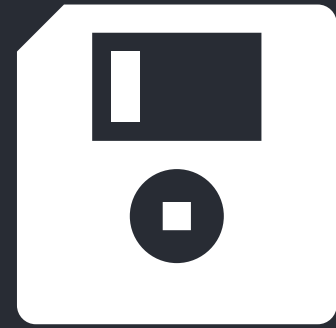
+





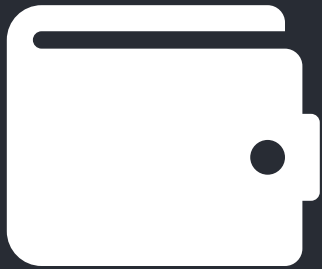
ถ้าไม่ออกก็ให้นักถึงพวกเกมแนวๆ Turn-Based ที่ตัวเรา (Program) สามารถควบคุม Units (Accounts) ในเกมได้ทั้งหมด

ส่วนอีกตัว อันนี้สำคัญมาก
ต้องเข้าใจ นั่นก็คือ Account
นั่นเอง ตัวนี้จะเป็นตัวที่เก็บ
ได้ทั้งคำสั่งและข้อมูลของตัว
Program

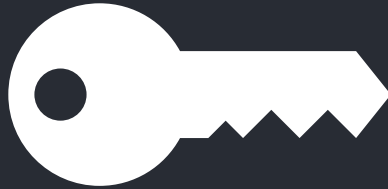


Account

ไม่ว่าจะเป็นอะไร มันเก็บได้หมด



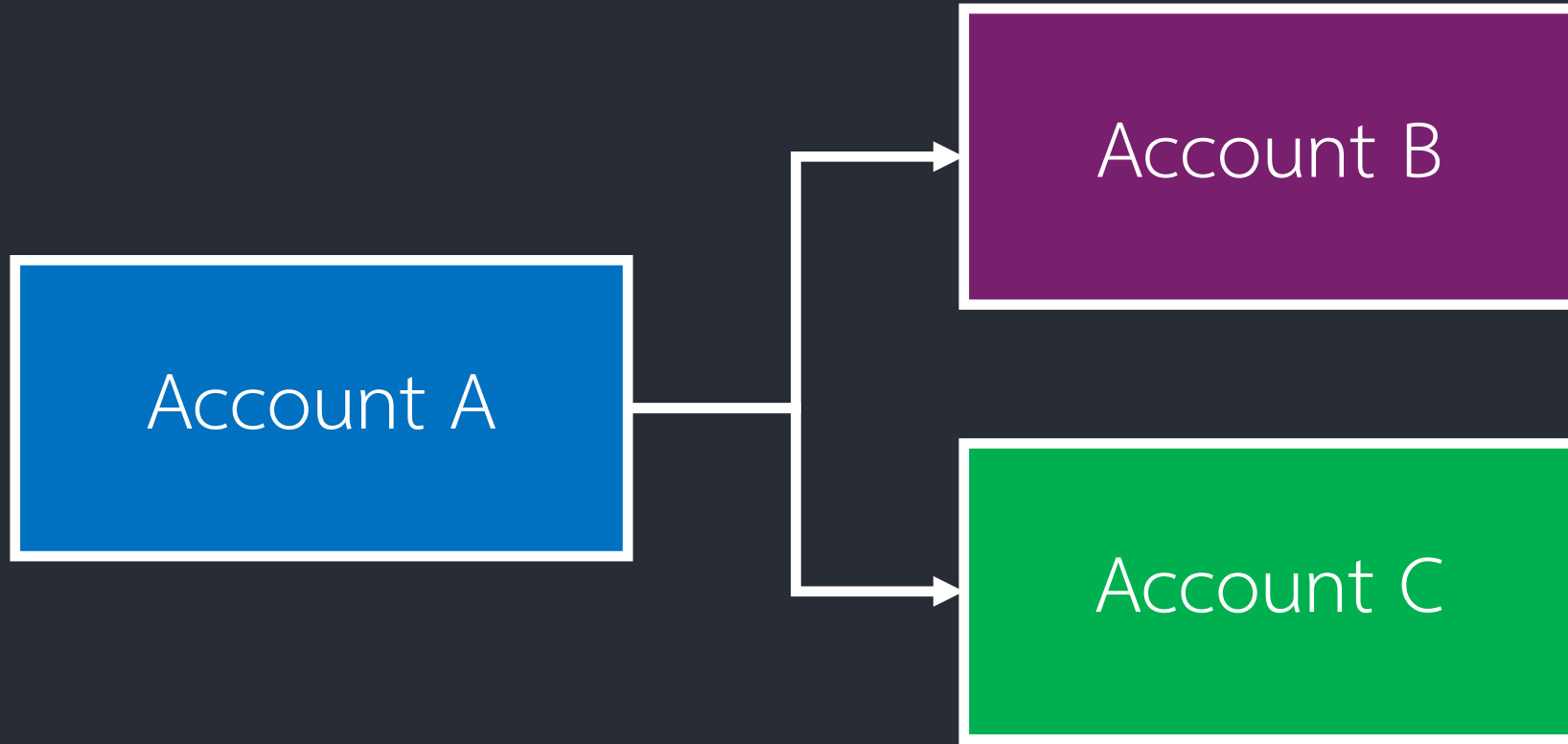
Balance



Pubkey



Any Data



ที่นี้ในตัว Account เอง มันก็สามารถที่จะ
เก็บ Account ซ้อนกันได้เรื่อยๆเหมือนกัน

ในขณะเดียวกัน Account มันก็มีอยู่ 2 ประเภทเหมือนกัน
ก็คือ Account ที่ Executable และ Non-Executable



Non-Executable

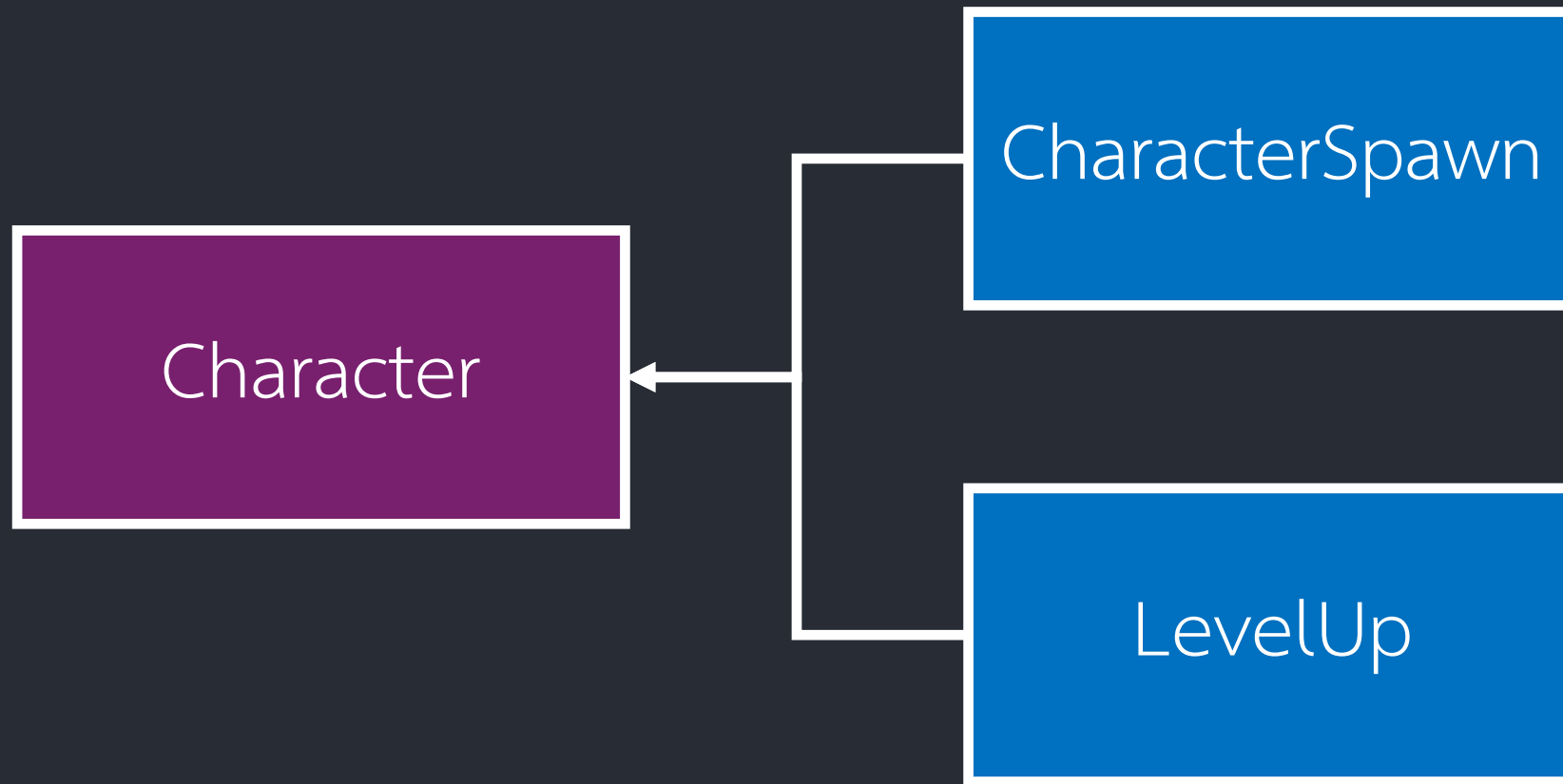
เอาไว้เก็บข้อมูล



Executable

เอาไว้เก็บคำสั่งการทำงาน
ย่อยของ Program

ลองยกตัวอย่างง่ายๆ สมมติผมออกแบบ Program ง่ายๆ
อาจจะระบบเกม RPG แบบง่ายๆแบบนี้



เอาไว้เก็บข้อมูล

Character

Non-Executable

เก็บคำสั่งการ Spawn

CharacterSpawn

Executable

เก็บคำสั่งการ Up Level

LevelUp

Executable

เรามาแจกแจงกันดีกว่าว่า Account ไหนเป็นประเภทไหนบ้าง



ถ้าเอามาเขียนเป็น Code มันจะเป็นแบบนี้

ปล. อย่าพึ่งไปใส่ใจพวก PDA หรือ Macros นะ เอา
ง่ายๆให้เห็นภาพก่อนก็พอ

ตัวอย่าง Executable Account (Instruction)

```
#[derive(Accounts)]
pub struct CharacterSpawn<'info> {
    #[account(mut)]
    pub player: Signer<'info>,
    #[account(
        init,
        payer = player,
        seeds = [b"character", player.key().as_ref()],
        space = 8 + Character::INIT_SPACE,
        bump
    )]
    pub character: Account<'info, Character>,
    pub system_program: Program<'info, System>,
}
```



จะเข้าใช้งานได้ต้องเป็น player คนนี้จริงๆ

```
#[derive(Accounts)]
pub struct CharacterSpawn<'info> {
    #[account(mut)]
    pub player: Signer<'info>,
    #[account(
        init,
        payer = player,
        seeds = [b"character", player.key().as_ref()],
        space = 8 + Character::INIT_SPACE,
        bump
    )]
    pub character: Account<'info, Character>,
    pub system_program: Program<'info, System>,
}
```

PDA

เข้าถึง character ได้



ตัวอย่าง Executable Account (Instruction)

```
#[derive(Accounts)]
pub struct LevelUp<'info> {
    pub player: Signer<'info>,
    #[account(
        mut,
        seeds = [b"character", player.key().as_ref()],
        bump = character.bump
    )]
    pub character: Account<'info, Character>,
}
```



จะเข้าใช้งานได้ต้องเป็น player คนนี้จริงๆ

```
#[derive(Accounts)]
pub struct LevelUp<'info> {
  pub player: Signer<'info>,
  #[account(
    mut,
    seeds = [b"character", player.key().as_ref()],
    bump = character.bump
  )]
  pub character: Account<'info, Character>,
}
```

PDA

เข้าถึง character ได้



ตัวอย่าง Non-Executable Account (State | Data)

```
#[account]
#[derive(InitSpace)]
pub struct Character {
    pub player: Pubkey,
    pub level: u8,
    pub exp: u32,
    pub exp_to_next_level: u32,
    pub bump: u8,
}
```

เก็บข้อมูลเฉยๆไม่มีอะไร





หวังว่าพอจะมองความเชื่อมโยงออกแล้วนะครับ ที่นี้มา
ลองดู Code ตัว Program กันดีกว่า ว่ามันเอา
Account มาใช้อย่างไร

```
use anchor_lang::prelude::*;

declare_id!("5Vw5JJUcxKn98ztWTuvX3AfPCYacwGQS6k1oVHerkjjq");

#[program]
pub mod take_my_sol {
    use super::*;

    pub fn character_spawn(ctx: Context<CharacterSpawn>) -> Result<()> {
        let character = &mut ctx.accounts.character;
        let player = &ctx.accounts.player;

        character.player = player.key();
        character.level = 1;
        character.exp = 0;
        character.exp_to_next_level = 100;
        character.bump = ctx.bumps.character;

        Ok(())
    }
}
```



```
use anchor_lang::prelude::*;
```

```
declare_id!("5Vw5JJUcxKn98ztWTuvX3AfPCYacwGQS6k1oVHerKjjq");
```

```
#[program]
```

```
pub mod take_my_sol {  
    use super::*;
```

```
pub fn character_spawn(ctx: Context<CharacterSpawn>) -> Result<()> {  
    let character = &mut ctx.accounts.character;  
    let player = &ctx.accounts.player;
```

```
    character.player = player.key();  
    character.level = 1;  
    character.exp = 0;  
    character.exp_to_next_level = 100;  
    character.bump = ctx.bumps.character;
```

```
    Ok(())
```

```
}  
}  
}
```

ดึงออกจาก Account มา
เปลี่ยนค่า

ProgramID ที่เป็นเจ้าของ
Accounts ทั้งหมด

ผ่าน Account เข้าไปเพื่อใช้งาน
ใน Program



ผ่าน Account เข้าไปเพื่อใช้งาน
ใน Program

```
#[program]
pub mod take_my_sol {
...
    pub fn level_up(ctx: Context<LevelUp>) -> Result<()> {
        let character = &mut ctx.accounts.character;

        if character.exp >= character.exp_to_next_level {
            character.level += 1;
            character.exp -= 0;
            character.exp_to_next_level *= 2;
        }

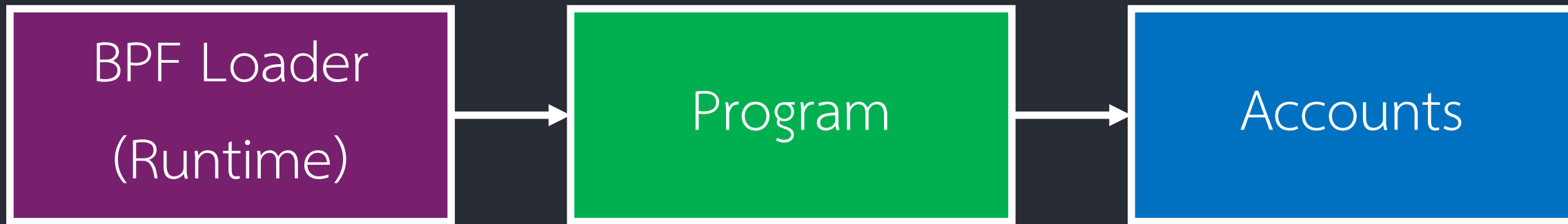
        Ok(())
    }
}
```

ดึงออกจาก Account มา
เปลี่ยนค่า





เพื่อส่งสัยว่า PDA คืออะไร มันก็คือเป็นการกำหนด ID
ให้กับ Account นั้นๆแล้วให้ Program ดูแลกันเองไป
เลย ไม่ต้องให้คนมากด Sign เพื่อเข้าถึงหรือ
เปลี่ยนแปลง โดยมันจะเป็น Seeds + ProgramID



สรุปลำดับการทำงานของ Solana