

[X:600, Y:1888]

[X:1920, Y:1888]

[X:488, Y:278]

[X:988, Y:548]

[X:1440, Y:810]

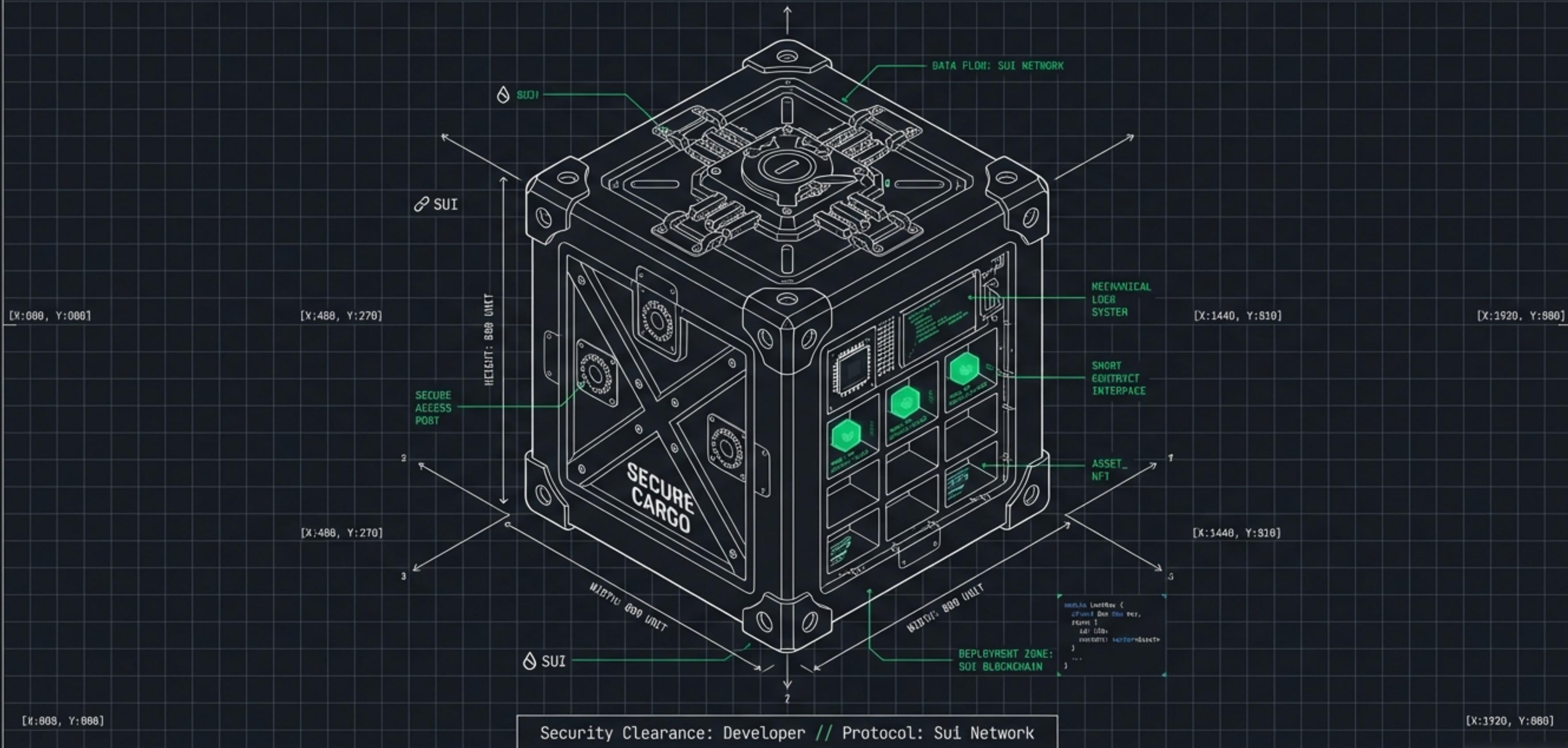
MISSION BRIEFING: DECENTRALISED LOOT BOX SYSTEM

Architecture & Implementation Strategy | Sui Move Smart Contracts

Alkimi
Hackathon

MoveFWD
Phase II

Problem
Statement #2



Space Grotesk OPERATIONAL OBJECTIVES

THE ENGINE



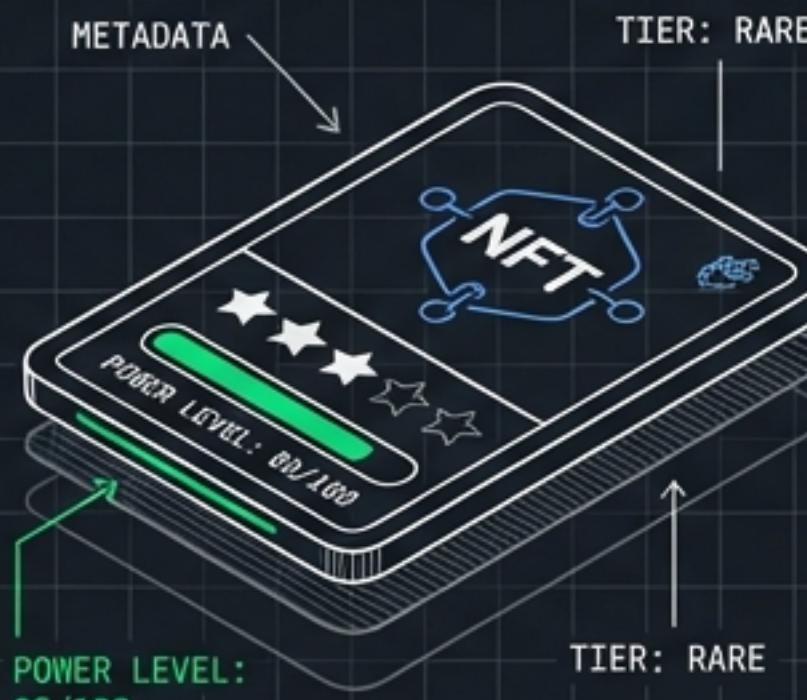
Implement a purchase system using fungible tokens (SUI).

THE CORE



Integrate verifiable, on-chain randomness (sui::random).

THE ASSET



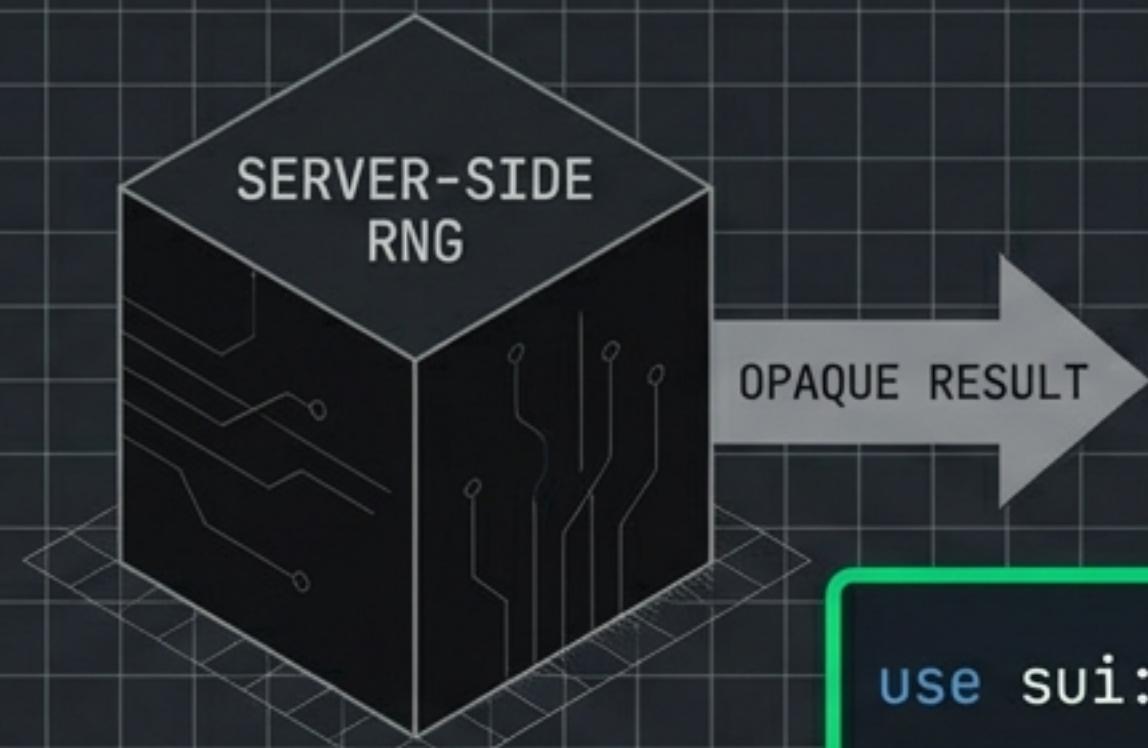
Mint NFTs with distinct rarity tiers and power levels.

THE CHALLENGE

In Web3 gaming, the critical friction point is trust. You must prove the 'luck' is not manipulated by the server.

THE CORE CHALLENGE: VERIFIABLE RANDOMNESS

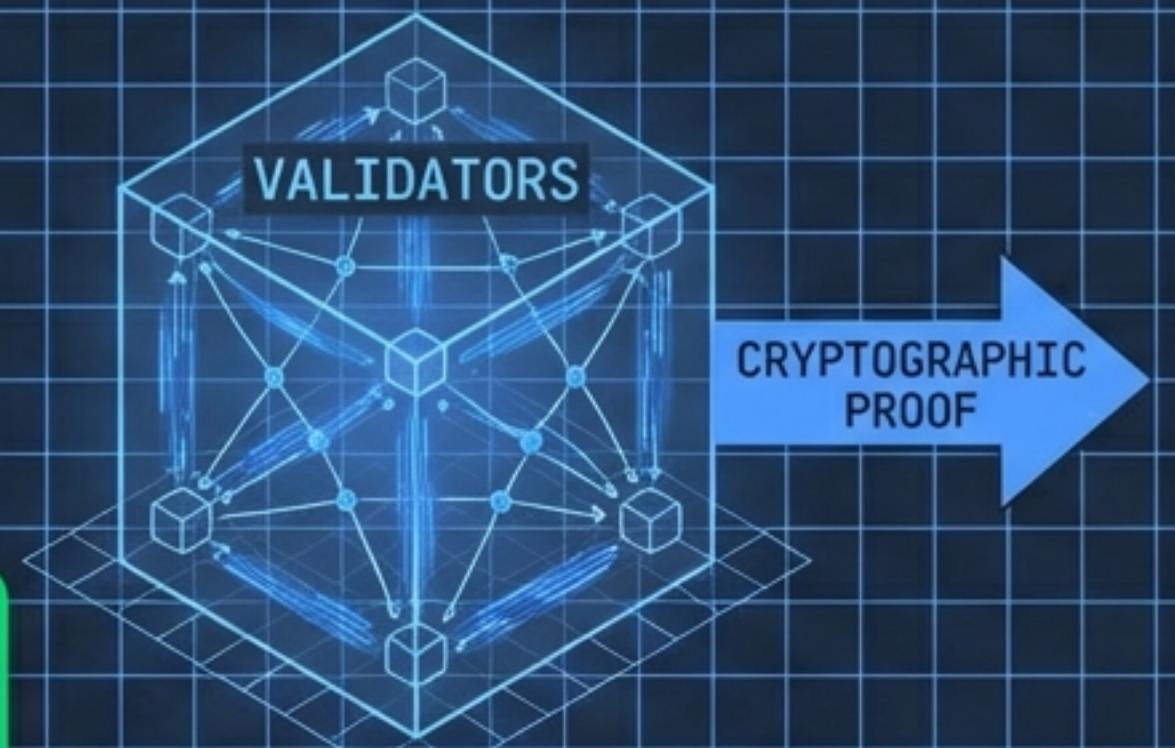
THE OLD WAY



UNVERIFIABLE / CENTRALIZED

```
use sui::random::Random;
```

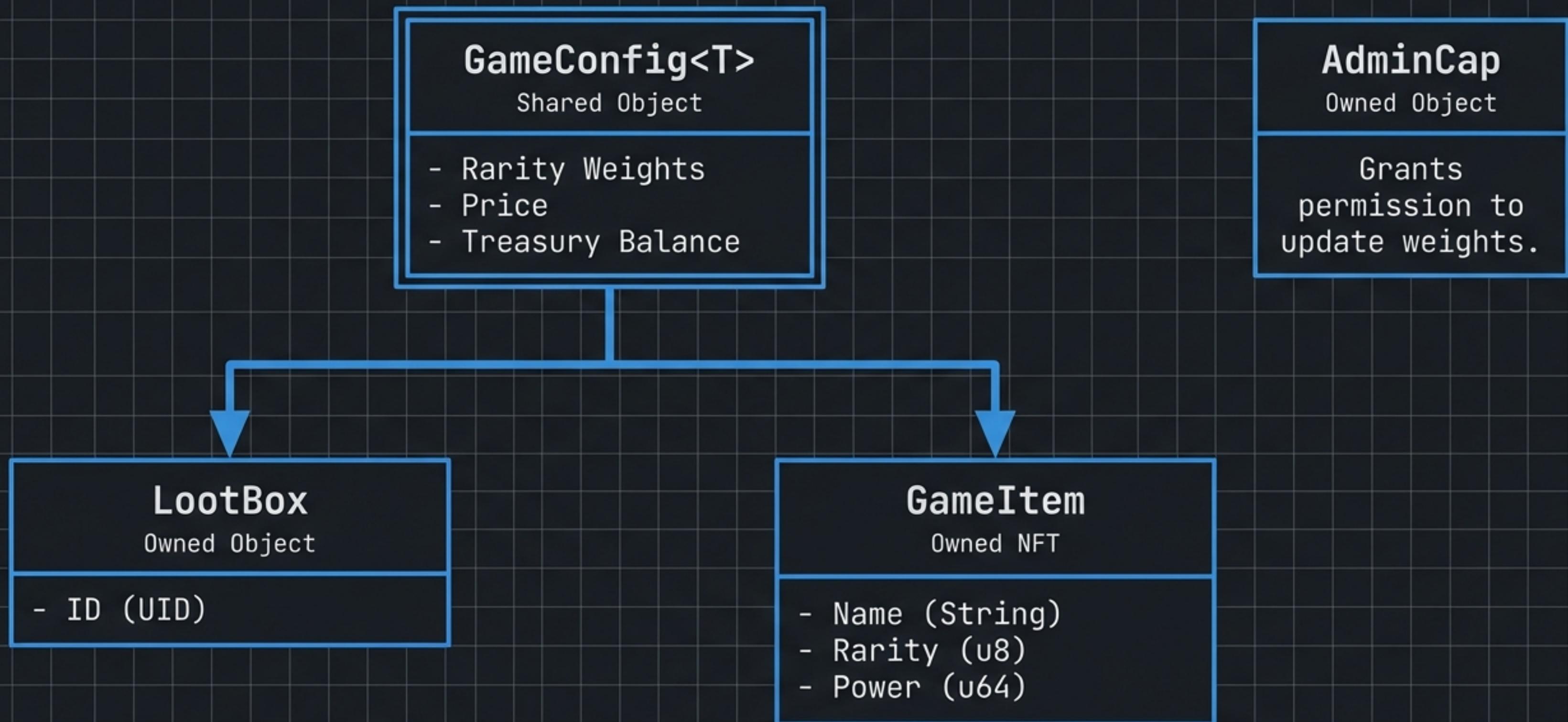
THE SUI WAY



SECURE / DISTRIBUTED BEACON

INSIGHT: Randomness is generated by validators, not the developer. To ensure fairness, the result must be unpredictable and unbiased.

THE OBJECT MODEL (THE NOUNS)



THE ECONOMY OF RARITY

COMMON (60%)
Roll: 0-59 | Power: 1-10

RARE (25%)
Roll: 60-84 | Power: 11-25

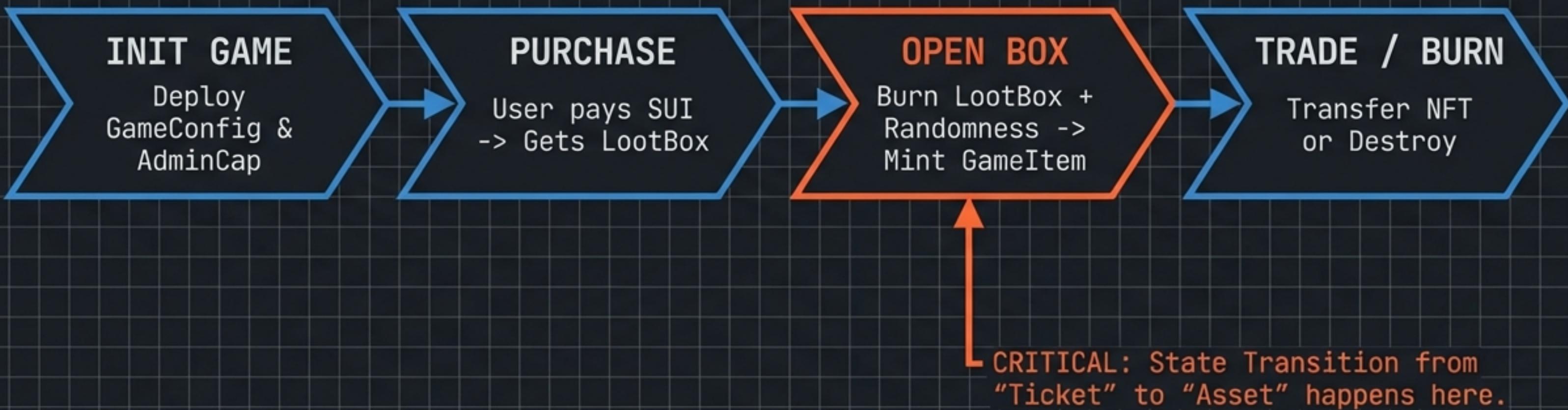
LEGENDARY (3%)
Roll: 97-99 | Power: 41-50

EPIC (12%)
Roll: 85-96 |
Power: 26-40

LEGENDARY (3%)
Roll: 97-99 | Power: 41-50

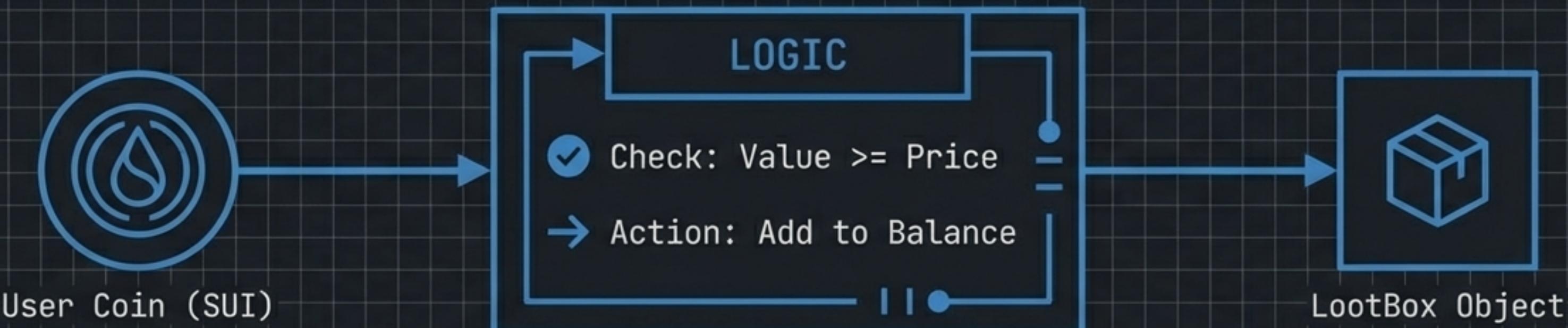
Total Weight = 100. The system generates a u8 between 0-99 to determine the tier.

SYSTEM LIFECYCLE OVERVIEW



Phase 1: The Purchase Logic

```
fun purchase_loot_box<T>(...)
```



Architecture Note: Ensure the LootBox object is purely a wrapper. It contains no value until opened.

Phase 2: Secure Randomness Protocol

GOLDEN RULES



MUST BE AN ENTRY FUNCTION

`open_loot_box` cannot be public. Call directly to prevent composition attacks.



INSTANTIATION

`RandomGenerator` must be created INSIDE the function.

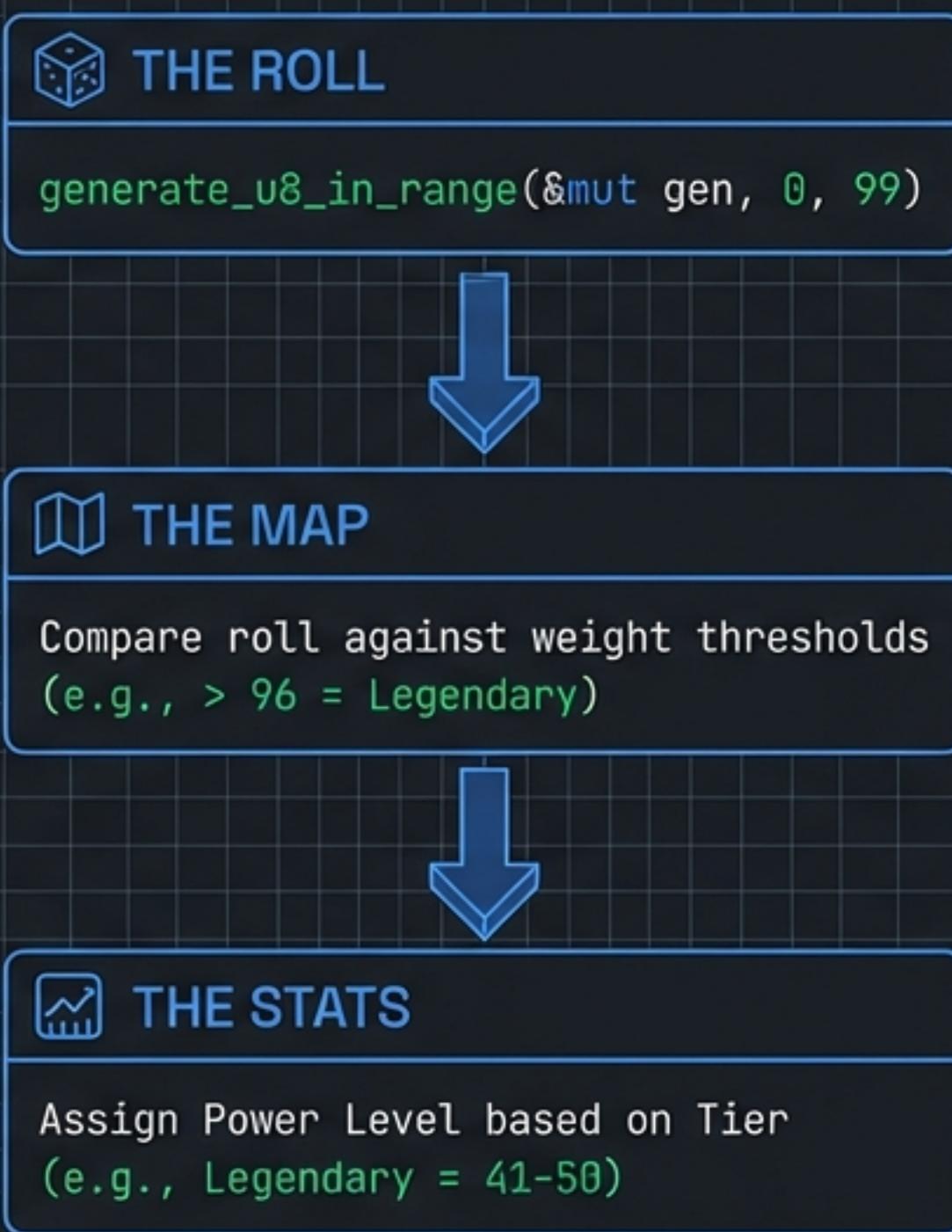


PROHIBITED

NEVER pass `RandomGenerator` as a function argument.

Objective: Prevent 'Reroll Attacks' where malicious contracts revert the transaction if the outcome is unfavourable.

Phase 3: Minting Logic & Stats



Item Lifecycle Management



TRANSFER

```
transfer::public_transfer {  
    Item ID,  
    Rarity,  
    Owner  
}
```

Enables secondary market and trading economy.



BURN

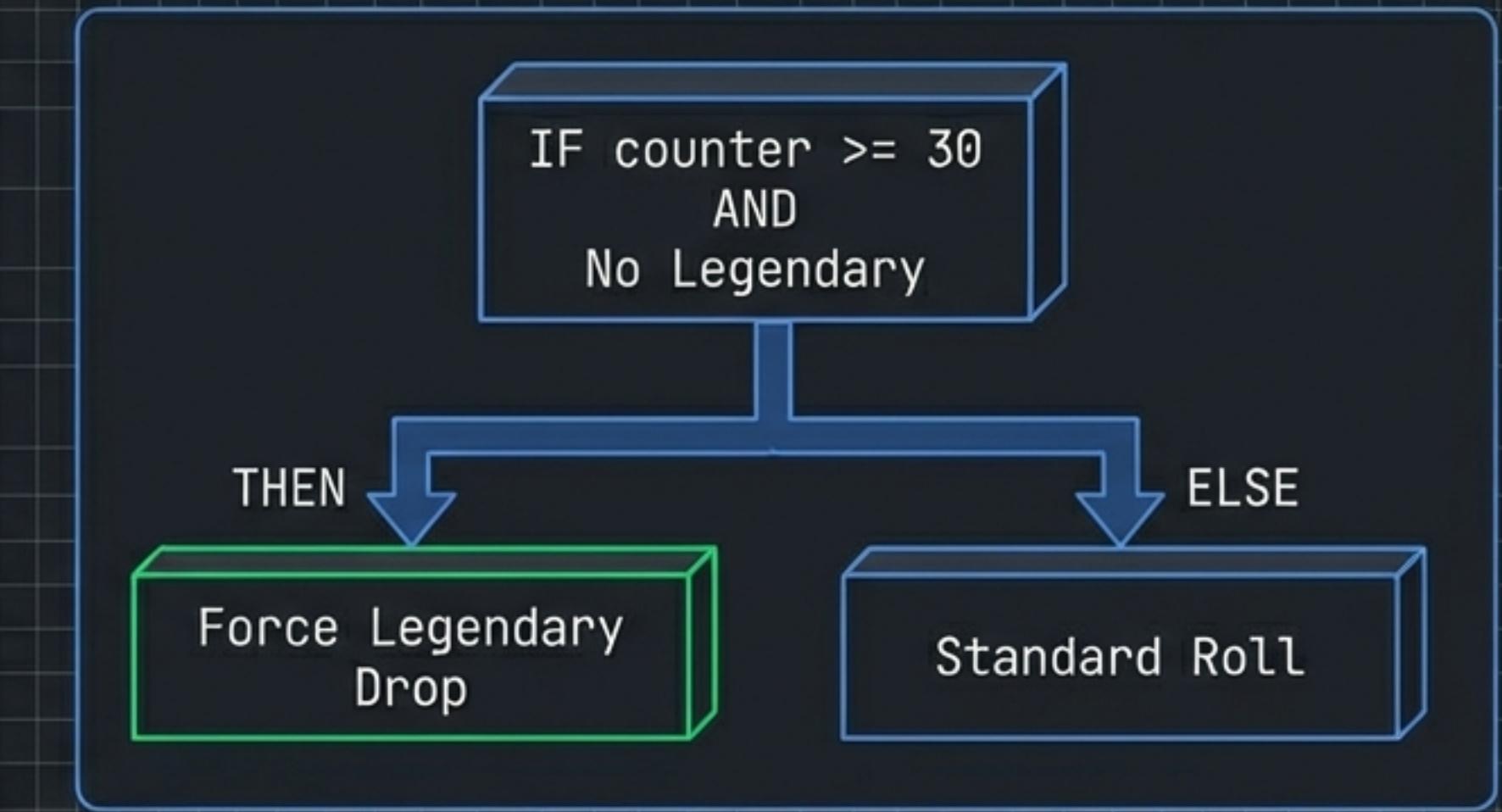
```
burn_item {  
    Item ID,  
    Rarity,  
    Owner  
}
```

Allows owners to destroy unwanted items (Deflationary).

Bonus Challenge: The Pity System



Bad Luck Protection



Storage Strategy

Use Dynamic Fields on GameConfig to map user addresses to open-counts. Keeps the main struct clean.

Testing Strategy

- [PASS] ✓ test_init_game (Config defaults)
- [PASS] ✓ test_purchase (Payment flow)
- [PASS] ✓ test_open (Minting correctness)
- [PASS] ✓ test_pi

MOCKING RANDOMNESS:

MOCKING RANDOMNESS: Use
`'sui::random::new_generator_for_testing'` to simulate
outcomes in deterministic environments.

Implementation Roadmap (48 Hours)



Pro-Tip:

Start with 'init_game' to get the basic setup working first.

Resources & Tooling

CLI Cheat Sheet

```
$ sui move build  
$ sui move test  
$ sui client publish --gas-budget  
1000000000
```

Key Access Points

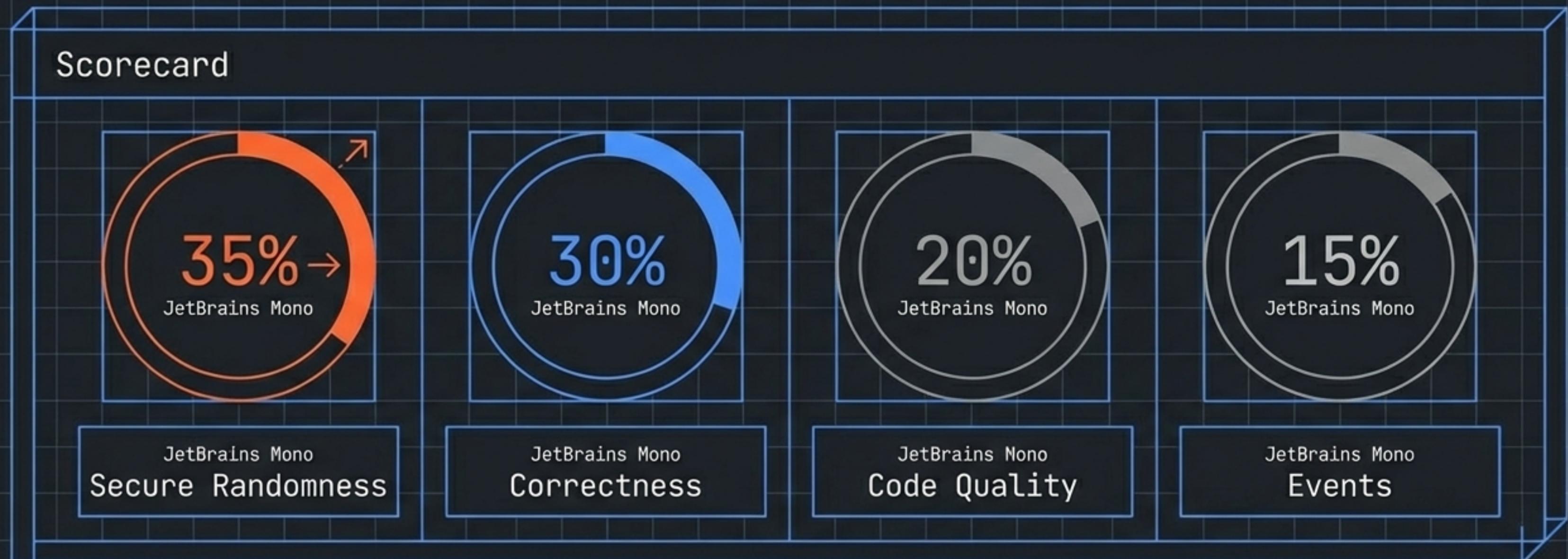
Documentation: [sui::random Module](#)

Faucet: !faucet <ADDRESS> (Discord)

Network: faucet.testnet.sui.io

Remember to use Testnet for validation.

Mission Success Criteria



Good luck. Build something amazing.