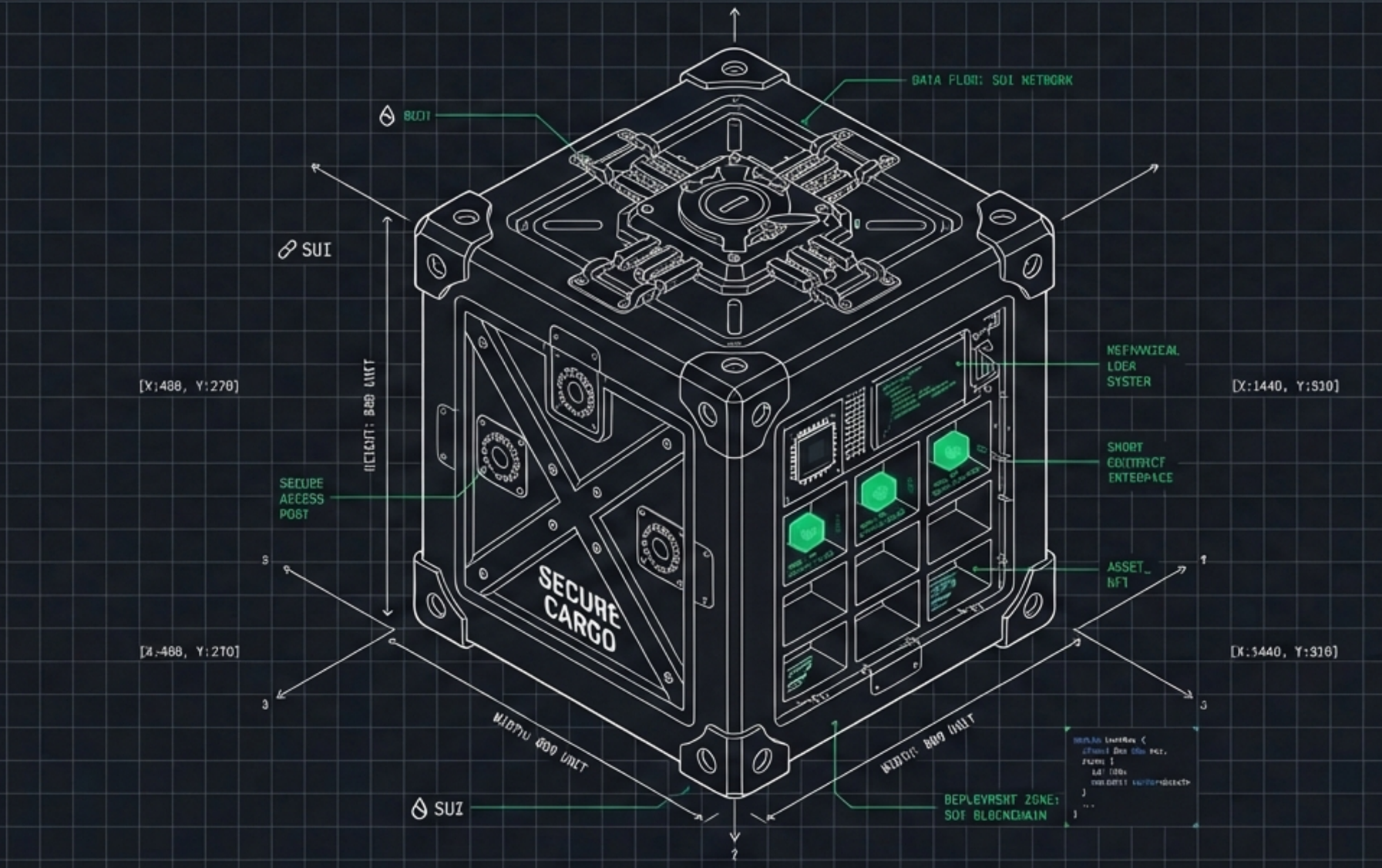
DECENTRALISED LOOT BOX SYSTEM

Architecture & Implementation Strategy | Sui Move Smart Contracts

Alkimi Hackathon

MoveFWD Phase II

Problem Statement #2



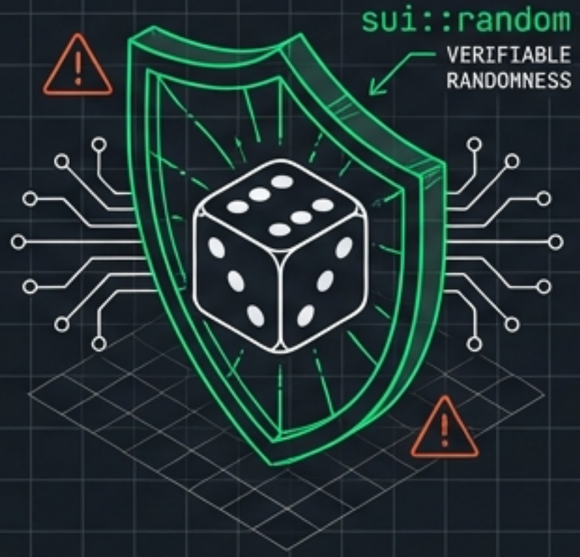
Security Clearance: Developer // Protocol: Sui Network

Space Grotesk
OPERATIONAL OBJECTIVES

THE ENGINE

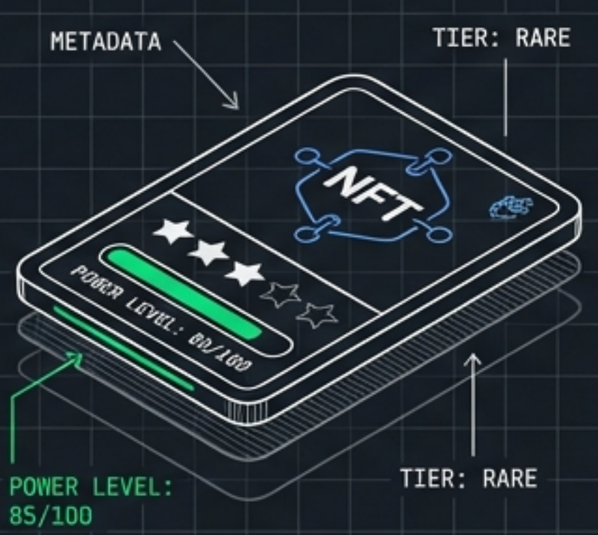
Implement a purchase system using fungible tokens (SUI).

THE CORE



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

THE ASSET



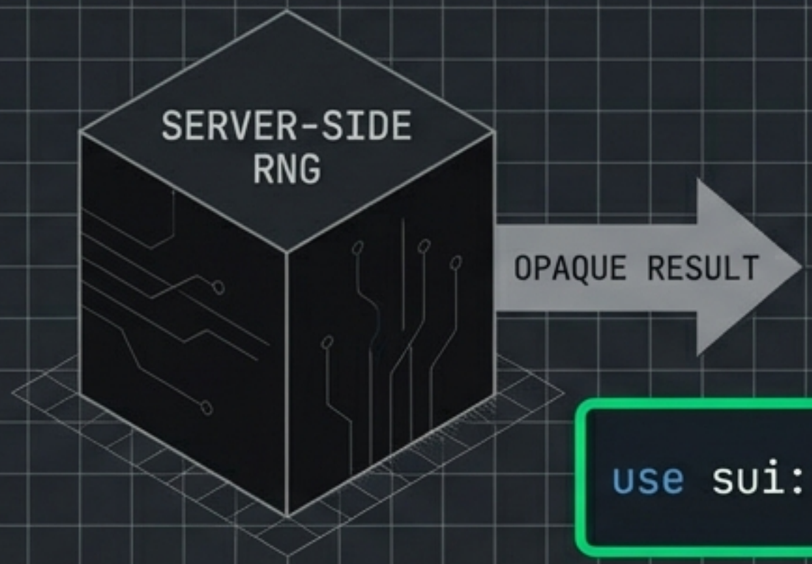
Mint NFTs with distinct rarity tiers and power levels.

THE CHALLENGE

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

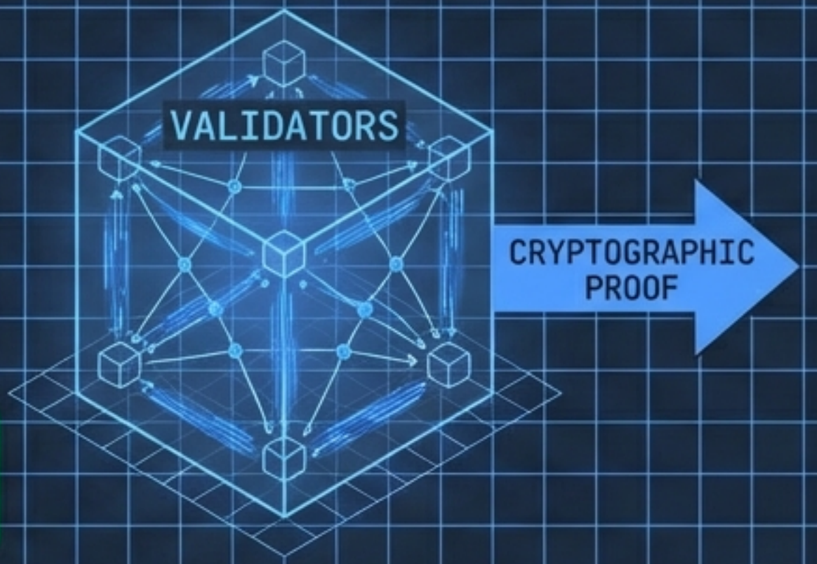
THE CORE CHALLENGE: VERIFIABLE RANDOMNESS

THE OLD WAY



UNVERIFIABLE / CENTRALIZED

THE SUI WAY

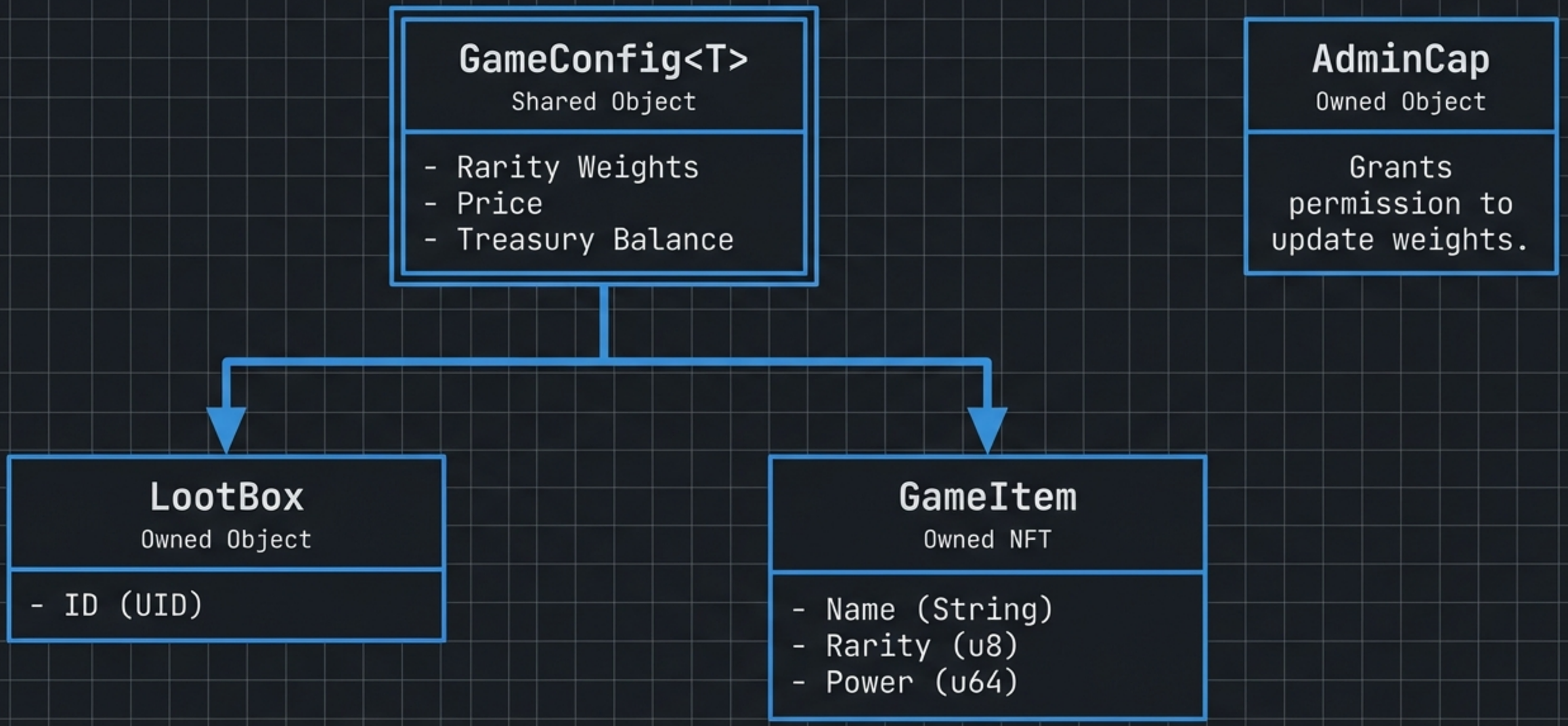



```
use sui::random::Random;
```

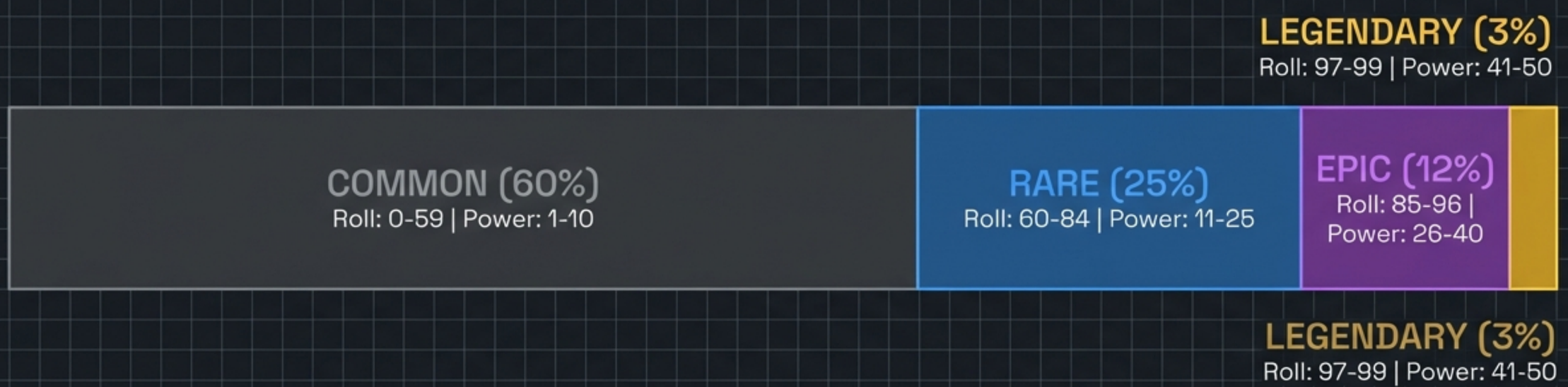
SECURE / DISTRIBUTED BEACON

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

THE OBJECT MODEL (THE NOUNS)

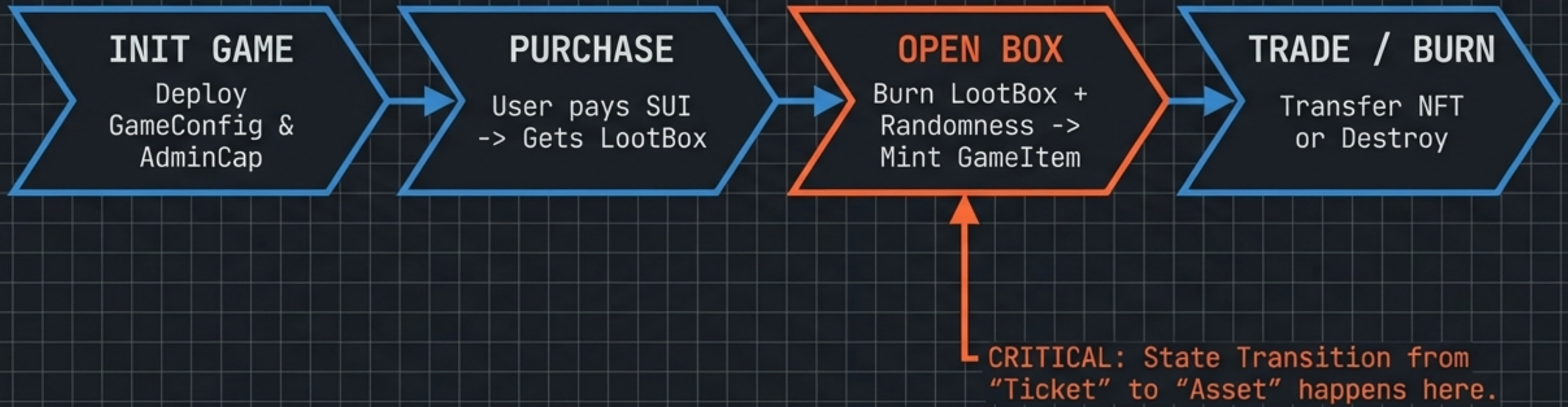


THE ECONOMY OF RARITY



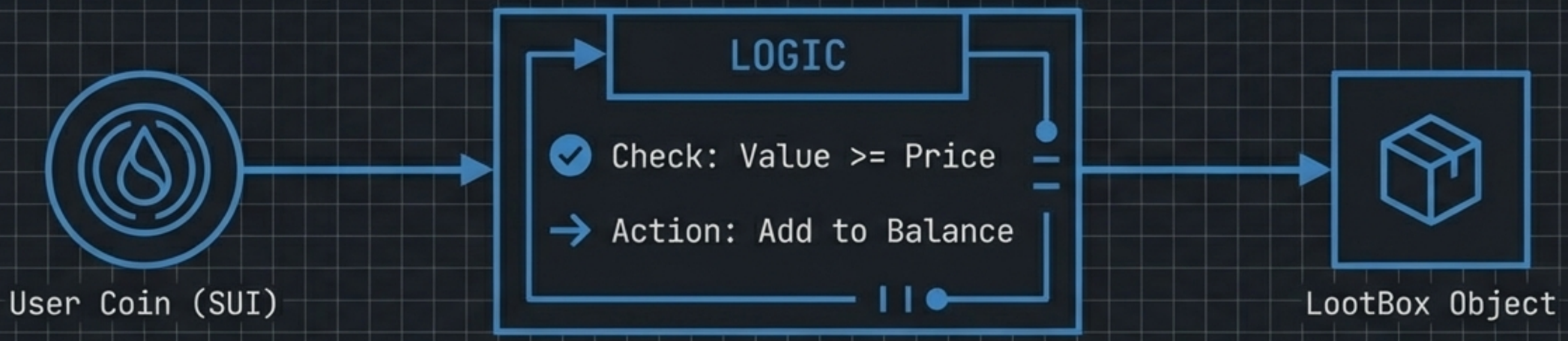
Total Weight = 100. The system generates a u8 between 0-99 to determine the tier.

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

THE ROLL

```
generate_u8_in_range(&mut gen, 0, 99)
```

THE MAP

Compare roll against weight thresholds
(e.g., > 96 = Legendary)

THE STATS

Assign Power Level based on Tier
(e.g., Legendary = 41-50)

EVENT EMISSION

```
emit LootBoxOpened {
  Item ID,
  Rarity,
} Owner
```

Item Lifecycle Management

TRANSFER

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

Enables secondary market and trading economy.

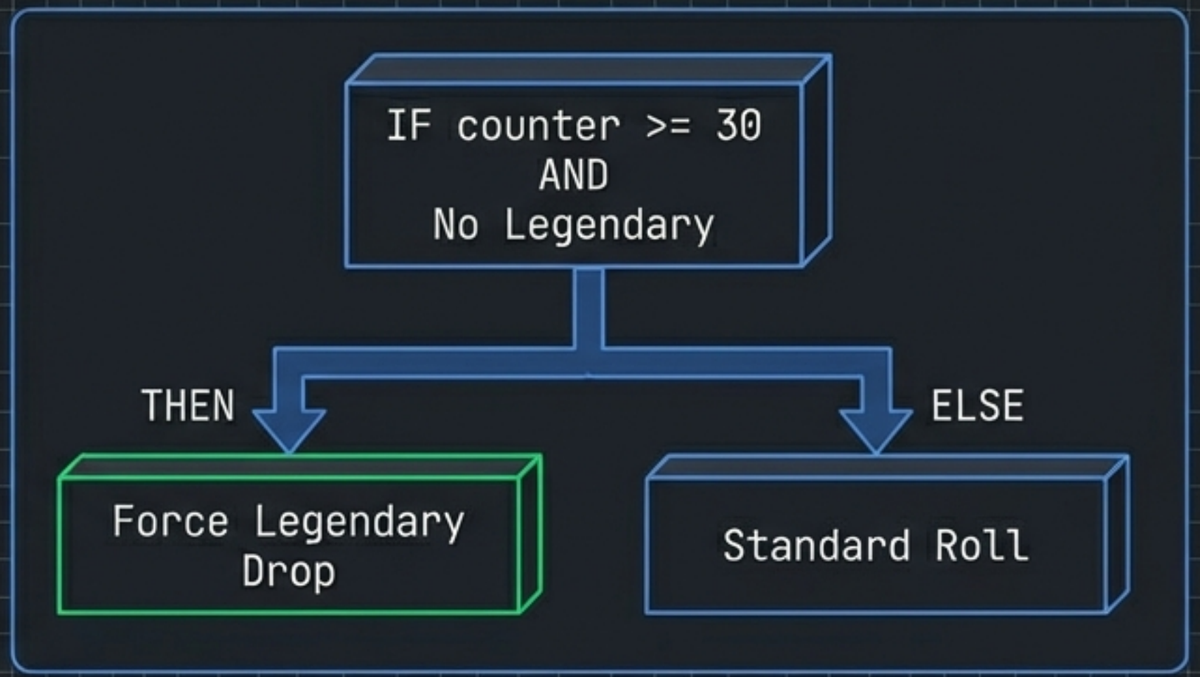
BURN

```
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.

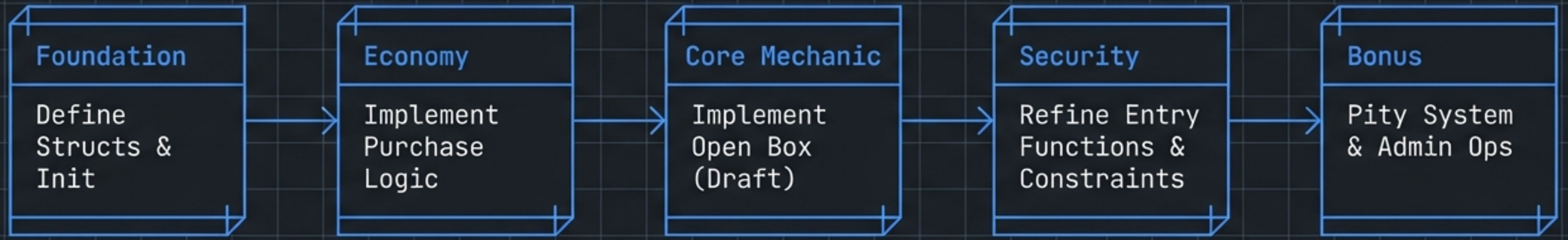
Testing Strategy

```
[PASS]  ✅ test_init_game (Config defaults)
[PASS]  ✅ test_purchase (Payment flow)
[PASS]  ✅ test_open (Minting correctness)
[PASS]  ✅ test_pi
```

MOCKING RANDOMNESS:

MOCKING RANDOMNNESS: 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

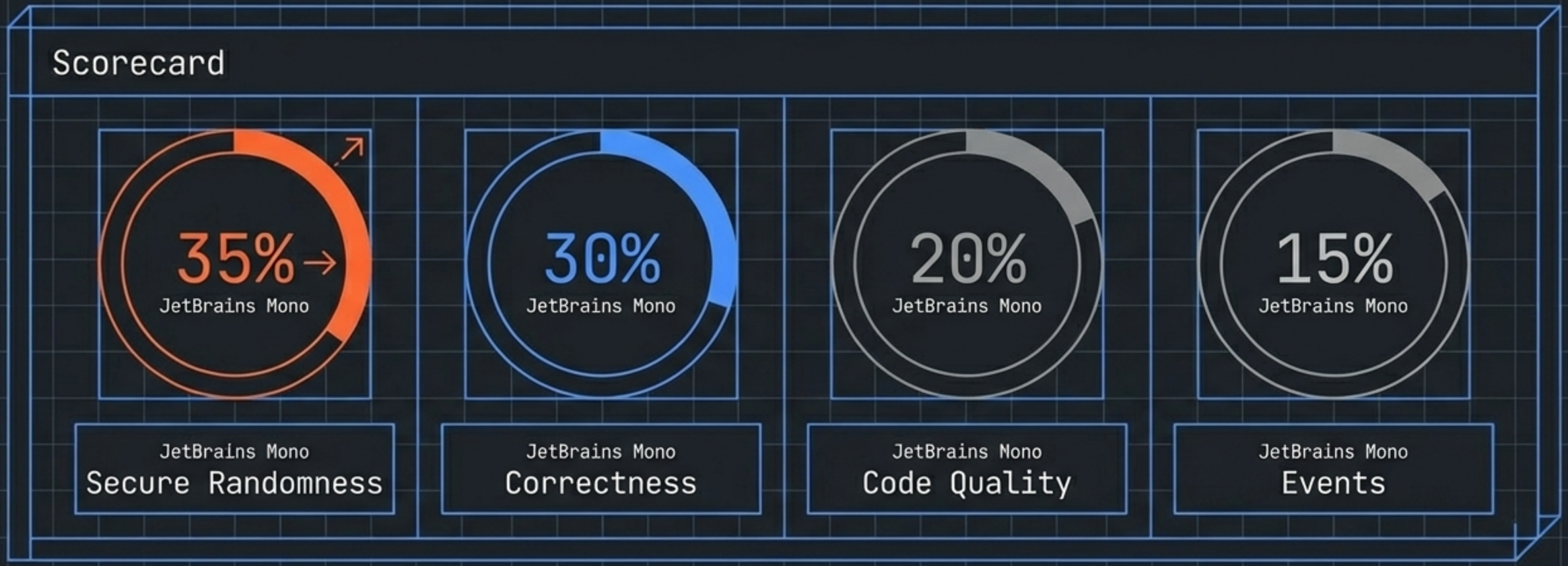
Documentation: `sui::random Module`

Faucet: `!faucet <ADDRESS>` (Discord)

Network: faucet.testnet.sui.io

Remember to use Testnet for validation.

Mission Success Criteria



Good luck. Build something amazing.