Chapter 5: Deep Dive into OpenBook V2 Program

```
                            +-----------------------+
                            |       RootNode        |
                            |-----------------------|
                            |  tag: 0               |
                            |  padding: [0, 0, 0]   |
                            |  prefix_len: 2        |
                            |  key: 10              |
                            |  children: [0, 1]     |
                            |  child_earliest_expiry|
                            | [1844674407370..]     |
                            |  reserved: [0,..]     |
                            +-----------------------+
                                       |
                       +---------------+---------------+
                       |                               |
           +-----------v-----------+      +------------v------------+
           |       InnerNode       |      |        InnerNode        |
           |  tag: 1               |      |  tag: 1                 |
           |  padding: [0, 0, 0]   |      |  padding: [0, 0, 0]     |
           |-----------------------|      |-------------------------|
           |  prefix_len: 3        |      |  prefix_len: 4          |
           |  key: 5               |      |  key: 6                 |
           |  children: [2, 2]     |      |  children: [4, 5]       |
           |  child_earliest_expiry|      |  child_earliest_expiry  |
           | [184467440737095, ..] |      | [184467440737095, ..]   |
           |  reserved: [0, ...]   |      | reserved: [0, ...]      |
           +--------+--------------+      +-------------+-----------+
                    |                                   |
         +----------v--------+             +------------v------------+
         |                   |             |        LeafNode         |
+--------v--------+ +--------v--------+    |-------------------------|
|    LeafNode     | |    InnerNode    |    |  tag: 2                 |
|  tag: 2         | |  tag: 1         |    |  padding: [0, 0, 0]     |
|-----------------| |-----------------|    |  owner_slot: 1          |
|  owner_slot: 2  | |  prefix_len: 5  |    |  time_in_force: 2       |
| time_in_force: 3| |  key: 7         |    |  key: 7                 |
|  key: 15        | |  children: [5]  |    |  owner: [0x21..],       |
|  owner: [0x21..]| +--------+--------+    |  quantity: 22           |
|  quantity: 50   |          |             |  timestamp: 12345       |
|  timestamp: 6789|          |             |  peg_limit: 0           |
+-----------------+          |             |  client_order_id: 0     |
                     +-------v--------+    +-------------------------+
                     |    FreeNode    |
                     |----------------|
                     |  tag: 3        |
                     |  padding: [0,] |
                     |  next: 0       |
                     |  reserved: [0,]|
                     +----------------+
```

Introduction

Welcome to the fifth chapter of our series exploring the world of Web3 in Rust, with a particular focus on the Solana blockchain. In the previous chapters, we explored the internals of OpenBook v1. This chapter will take an in-depth look at each component of the OpenBook V2 Central Limit Order Book (CLOB), examining their roles, interactions, and implementation details within the OpenBook v2 DEX. We'll also highlight the key differences and similarities between OpenBook v1 and v2.

Let's start with an introduction to OpenBook V2.

1. OpenBook V2

Like OpenBook V1, OpenBook V2 is a DEX program built on the **Solana blockchain**, aimed at enabling efficient and transparent trading of digital assets. Leveraging Solana's high throughput and low transaction costs, OpenBook V2 facilitates peer-to-peer trading through its decentralized **order book** mechanism. This chapter explores the architecture, operation, and key components of OpenBook V2, highlighting its role in **the DeFi** ecosystem.

Like OpenBook V1, OpenBook V2 operates as a decentralized order book where buyers and sellers can interact directly, eliminating the need for intermediaries like traditional exchanges. This setup ensures that trades are executed peer-to-peer using programs, aka **smart contracts** in the solana world, deployed on the Solana blockchain. By leveraging Solana's infrastructure, OpenBook V2 achieves near-instant transaction finality and supports a high volume of trades per second.

2. Architecture Overview

```
+-------------------------------------------------+
|              OpenBook V2 Components             |
+-------------------------------------------------+
|           +---------------------------+         |
|           |        Order Book         |         |
|           +---------------------------+         |
|           |       Event Heap          |         |
|           +---------------------------+         |
|           |         Market           |         |
|           +--------------------------+         |
|           |  Open Orders Indexer     |         |
|           +--------------------------+         |
|           |  Open Orders Account     |         |
|           +--------------------------+         |
|           |          Oracle          |         |
|           +--------------------------+         |
+------------------------------------------------+
```

The architecture of OpenBook V2 consists of several key components/modules:

- **Orderbook**: Manages the actual order matching and bookkeeping operations, including bid and ask sides.
- **Event Heap**: Stores pending events such as order fills, ensuring sequential processing.
- **Market**: Stores market-specific parameters and handles order validation and execution logic.
- **Open Orders Account**: Tracks owner's account, their orders, and positions, among other components.
- **Open Orders Indexer**: This component is designed to index open orders in the DEX, ensuring efficient access and management of orders. Each Open Order account must be created under at least one indexer account.
- **Oracle**: Oracles provide market data and are essential for accurate pricing and other market operations. The oracle module is extensive, with configurations and implementations for different types of oracles such as Pyth, SwitchboardV1, SwitchboardV2, and RaydiumCLMM.

The `orderbook/book` module in OpenBook V2 serves as the core component responsible for matching buy and sell orders. It maintains separate records for **bids (buy orders)** and **asks (sell orders)**, organized by price, asks from lowest to highest and bids from highest to lowest price. This structure ensures fair execution of trades based on the best available prices, enhancing market efficiency and **liquidity**.

```
pub struct Orderbook<'a> {
    pub bids: RefMut<'a, BookSide>,
    pub asks: RefMut<'a, BookSide>,
}
```

Reference: openbook-dex/openbook-v2

The `EventHeap` object is designed to handle various types of events that occur within the OpenBook V2 DEX.

```rust
pub struct EventHeap {
    pub header: EventHeapHeader,
    pub nodes: [EventNode; MAX_NUM_EVENTS as usize],
    pub reserved: [u8; 64],
}
```

Reference: openbook-dex/openbook-v2

These events include fill and out events:

```rust
pub struct OutEvent {
    pub event_type: u8,
    pub side: u8,
    pub owner_slot: u8,
    padding0: [u8; 5],
    pub timestamp: u64,
    pub seq_num: u64,
    pub owner: Pubkey,
    pub quantity: i64,
    padding1: [u8; 80],
}

pub struct FillEvent {
    pub event_type: u8,
    pub taker_side: u8,
    pub maker_out: u8,
    pub maker_slot: u8,
    pub padding: [u8; 4],
    pub timestamp: u64,
    pub market_seq_num: u64,
    pub maker: Pubkey,
    pub maker_timestamp: u64,
    pub taker: Pubkey,
    pub taker_client_order_id: u64,
    pub price: i64,
    pub peg_limit: i64,
    pub quantity: i64,
    pub maker_client_order_id: u64,
    pub reserved: [u8; 8],
}
```

Reference: openbook-dex/openbook-v2

By maintaining a dedicated heap for events, and like its v1 predecessor, OpenBook V2 ensures that all events are processed in a timely and orderly fashion, which is critical for maintaining market integrity and responsiveness.

The **`Market`** object is a fundamental part of the OpenBook V2 DEX, responsible for storing and managing market-specific parameters. These parameters include the number of decimals for base and quote tokens, lot sizes, fee rates, and other crucial settings.

```
pub struct Market {
    /// PDA bump
    pub bump: u8,

    /// Number of decimals used for the base token.
    ///
    /// Used to convert the oracle's price into a native/native price.
    pub base_decimals: u8,
    pub quote_decimals: u8,

    pub padding1: [u8; 5],

    // Pda for signing vault txs
    pub market_authority: Pubkey,

    /// No expiry = 0. Market will expire and no trading allowed after time_expiry
    pub time_expiry: i64,

    /// Admin who can collect fees from the market
    pub collect_fee_admin: Pubkey,
    /// Admin who must sign off on all order creations
    pub open_orders_admin: NonZeroPubkeyOption,
    /// Admin who must sign off on all event consumptions
    pub consume_events_admin: NonZeroPubkeyOption,
    /// Admin who can set market expired, prune orders and close the market
    pub close_market_admin: NonZeroPubkeyOption,

    /// Name. Trailing zero bytes are ignored.
    pub name: [u8; 16],

    /// Address of the BookSide account for bids
    pub bids: Pubkey,
    /// Address of the BookSide account for asks
    pub asks: Pubkey,
    /// Address of the EventHeap account
    pub event_heap: Pubkey,

    /// Oracles account address
    pub oracle_a: NonZeroPubkeyOption,
    pub oracle_b: NonZeroPubkeyOption,
    /// Oracle configuration
    pub oracle_config: OracleConfig,
    /// Number of quote native in a quote lot. Must be a power of 10.
    ///
    /// Primarily useful for increasing the tick size on the market: A lot price
    /// of 1 becomes a native price of quote_lot_size/base_lot_size becomes a
    /// ui price of quote_lot_size*base_decimals/base_lot_size/quote_decimals.
    pub quote_lot_size: i64,

    /// Number of base native in a base lot. Must be a power of 10.
    ///
    /// Example: If base decimals for the underlying asset is 6, base lot size
    /// is 100 and and base position lots is 10_000 then base position native is
    /// 1_000_000 and base position ui is 1.
    pub base_lot_size: i64,

    /// Total number of orders seen
    pub seq_num: u64,

    /// Timestamp in seconds that the market was registered at.
    pub registration_time: i64,

    /// Fees
    ///
    /// Fee (in 10^-6) when matching maker orders.
    /// maker_fee < 0 it means some of the taker_fees goes to the maker
    /// maker_fee > 0, it means no taker_fee to the maker, and maker fee goes to the referr
    pub maker_fee: i64,
    /// Fee (in 10^-6) for taker orders, always >= 0.
    pub taker_fee: i64,

    /// Total fees accrued in native quote
    pub fees_accrued: u128,
    /// Total fees settled in native quote
    pub fees_to_referrers: u128,

    /// Referrer rebates to be distributed
    pub referrer_rebates_accrued: u64,

    /// Fees generated and available to withdraw via sweep_fees
    pub fees_available: u64,

    /// Cumulative maker volume (same as taker volume) in quote native units
    pub maker_volume: u128,

    /// Cumulative taker volume in quote native units due to place take orders
    pub taker_volume_wo_oo: u128,
    pub base_mint: Pubkey,
    pub quote_mint: Pubkey,

    pub market_base_vault: Pubkey,
    pub base_deposit_total: u64,

    pub market_quote_vault: Pubkey,
    pub quote_deposit_total: u64,

    pub reserved: [u8; 128],
}
```

Reference: openbook-dex/openbook-v2

The Market struct also implements logic for validating and executing orders, ensuring that all trades adhere to the market's rules and regulations. The struct is designed to support various administrative roles, manage fees, handle oracles for price feeds, and maintain state information such as order counts and timestamps. Additionally, the Market struct includes fields for vault management, fee accrual and distribution, and oracle configuration to provide accurate pricing data.

The **OpenOrdersAccount** object in the OpenBook V2 program tracks individual associated tokens accounts. Each account contains information about the owner's open orders, balances, and trade history. This component is crucial for ensuring that users can manage their orders and assets effectively, and for the system to accurately track and settle trades.

```
pub struct OpenOrdersAccount {
    pub owner: Pubkey,
    pub market: Pubkey,

    pub name: [u8; 32],

    // Alternative authority/signer of transactions for a openbook account
    pub delegate: NonZeroPubkeyOption,

    pub account_num: u32,

    pub bump: u8,

    // Introducing a version as we are adding a new field bids_quote_lots
    pub version: u8,

    pub padding: [u8; 2],

    pub position: Position,
    pub open_orders: [OpenOrder; MAX_OPEN_ORDERS],
}
```

Reference: openbook-dex/openbook-v2

3. OpenBook V2 Central Limit Orderbook (CLOB)

```
+-------------------------------------------------+             +-----------------------------
|                Bids (Buy Orders)                |             |              Asks (Sell (
|             |-------------------|               |             |             |--------------
|             |  Order 1: $105    |               |             |             |  Order 1: $1
|             |-------------------|               |             |             |--------------
|             |  Order 2: $100    |               |             |             |  Order 2: $1
|             |-------------------|               |             |             |--------------
|                     ...                         |             |                   ...
+-------------------------------------------------+             +-----------------------------
```

Just like OpenBook V1, and as previously mentioned, the central limit orderbook in OpenBook V2 organizes buy (bids) and sell (asks) orders by price. This structure ensures fair execution of trades based on the best available prices. Each side of the orderbook, bids, and asks, is managed independently but interacts during order matching.

The orderbook maintains two main structures:

- **Bids**: Contains buy orders sorted from highest to lowest price.
- **Asks**: Contains sell orders sorted from lowest to highest price.

These structures use efficient data structures like critbit-trees to enable quick insertion, deletion, and lookup operations of orders, crucial for **high-frequency trading** environments. When a new order is placed, the DEX checks against the opposite side (bids for sell orders and asks for buy orders) to find matches based on price and order size.

The efficient matching engine within the orderbook ensures that trades are executed at the best possible prices. For instance, if a buy order is placed at a price higher than the current best sell price, the program will immediately match the buy order with the sell order, ensuring that both parties get the best possible deal. This matching process continues until all possible matches are made, ensuring optimal market liquidity.

Additionally, the orderbook handles various types of orders, including **limit orders**, and **market orders**.

```
ub enum PlaceOrderType {
    /// Take existing orders up to price, max_base_quantity and max_quote_quantity.
    /// If any base_quantity or quote_quantity remains, place an order on the book
    Limit = 0,
    /// Take existing orders up to price, max_base_quantity and max_quote_quantity.
    /// Never place an order on the book.
    ImmediateOrCancel = 1,

    /// Never take any existing orders, post the order on the book if possible.
    /// If existing orders can match with this order, do nothing.
    PostOnly = 2,

    /// Ignore price and take orders up to max_base_quantity and max_quote_quantity.
    /// Never place an order on the book.
    ///
    /// Equivalent to ImmediateOrCancel with price=i64::MAX.
    Market = 3,

    /// If existing orders match with this order, adjust the price to just barely
    /// not match. Always places an order on the book.
    PostOnlySlide = 4,

    /// Take existing orders up to price, max_base_quantity and max_quote_quantity.
    /// Abort if partially executed, never place an order on the book.
    FillOrKill = 5,
}
```

Reference: openbook-dex/openbook-v2

Each order type has specific rules for execution and prioritization, contributing to a dynamic and flexible trading environment. For instance, limit orders are executed only at the specified price or better, ensuring precise execution conditions. In contrast, market orders execute immediately at the best available price without price restrictions. The `PlaceOrderType` enum further outlines various order behaviors such as `ImmediateOrCancel`, which prioritizes immediate execution over order placement on the book, and `FillOrKill`, which mandates complete order execution or none at all, with no order placement. These distinctions allow traders to choose strategies that best suit their trading objectives and market conditions.

4. Order Matching and Execution

```
          +-------------------------------------------------+
          |               Matching Engine                   |
          +-------------------------------------------------+
          |               +------------------+              |
          |               | Order Validation|              |
          |               +------------------+              |
          |               +------------------+              |
          |               | Order Matching   |                |
          |               +------------------+                |
          |               +------------------+                |
          |               | Trade Execution  |                |
          |               +------------------+                |
          +---------------------------------------------------+
```

The matching process is the core component of OpenBook V2 that handles order matching and execution. It compares incoming orders with existing orders in the order book to find matches based on price and order size.

Order Validation The order validation step ensures the incoming order meets all required criteria, such as minimum size and price limits, and verifies the owner's balance. If the order fails any of these checks, it is rejected and not added to the order book.

```
// Validation checks
require_gte!(market.max_base_lots(), order_max_base_lots, OpenBookError::InvalidInputLotsSi
require_gte!(market.max_quote_lots(), order_max_quote_lots, OpenBookError::InvalidInputLots
```

Reference: openbook-dex/openbook-v2

These checks ensure the order size and price are within acceptable limits. The function also verifies that the user has sufficient funds to place the order.

Order Matching In the order matching step, the process compares the incoming order with existing orders on the opposite side of the order book. The matching engine uses a price priority algorithm to find the best possible matches, ensuring that orders are matched based on the best available prices and the order in which they were received.

```
// Iterate through book and match against this new order.
for best_opposing in opposing_bookside.iter_all_including_invalid(now_ts, oracle_price_lots)
    if remaining_base_lots == 0 || remaining_quote_lots == 0 {
        break;
    }

    if !best_opposing.is_valid() {
        // Remove the order from the book
        continue;
    }

    if !side.is_price_within_limit(best_opposing_price, price_lots) {
        break;
    }
    // Matching logic
}
```

Reference: openbook-dex/openbook-v2

This loop iterates over the order book to find the best matching orders based on the given order price.

Trade Execution The trade execution step involves executing the matched trades, and updating the order book. This process includes removing matched orders from the order book, and recording the trade in the event heap.

```
// Execute matched trades
process_fill_event(fill, market, event_heap, remaining_accs, &mut number_of_processed_fill_e
```

Reference: openbook-dex/openbook-v2

This function call processes the fill event, updating the necessary state and ensuring all events are handled correctly.

5. Events Iteration and Processing

```
          +--------------------------------------------------+
          |                EventHeap Mechanism                |
          +--------------------------------------------------+
          |              +------------------+                 |
          |              |  Event Storage   |                 |
          |              +------------------+                 |
          |              +------------------+                 |
          |              | Event Iteration  |                 |
          |              +------------------+                 |
          |              +------------------+                 |
          |              |Event Processing  |                 |
          |              +------------------+                 |
          +--------------------------------------------------+
```

EventHeap Mechanism The `EventHeap` in OpenBook V2 handles event processing and settlement. It stores pending events like order fills, ensuring they are processed sequentially.

The `EventHeap` operates through the following steps:

1. **Event Storage**: Incoming events are added to the EventHeap.

```
pub fn push_back(&mut self, value: AnyEvent) {
    // Ensure the heap is not full
    assert!(!self.is_full());
    // Get the slot to store the new event
    let slot = self.header.free_head;
    self.header.free_head = self.nodes[slot as usize].next;

    // Update pointers for the circular buffer / linked list
    let new_next: u16;
    let new_prev: u16;

    if self.is_empty() {
        new_next = slot;
        new_prev = slot;

        self.header.used_head = slot;
    } else {
        new_next = self.header.used_head;
        new_prev = self.nodes[new_next as usize].prev;

        self.nodes[new_prev as usize].next = slot;
        self.nodes[new_next as usize].prev = slot;
    }

    // Increment counters and store the fill event
    self.header.incr_count();
    self.header.incr_event_id();
    self.nodes[slot as usize].event = value;
    self.nodes[slot as usize].next = new_next;
    self.nodes[slot as usize].prev = new_prev;
}
```

Reference: openbook-dex/openbook-v2

2. **Event Processing**: Events are processed in a FIFO (First-In, First-Out) manner, ensuring timely execution.

```
pub fn pop_front(&mut self) -> Result<AnyEvent> {
    self.delete_slot(self.header.used_head()) // Process the first event in the heap
}

pub fn delete_slot(&mut self, slot: usize) -> Result<AnyEvent> {
    // Check if the slot is valid and the heap is not empty
    if slot >= self.nodes.len() || self.is_empty() || self.nodes[slot].is_free() {
        return Err(OpenBookError::SomeError.into());
    }

    // Update pointers for the circular linked list
    let prev_slot = self.nodes[slot].prev;
    let next_slot = self.nodes[slot].next;
    let next_free = self.header.free_head;

    self.nodes[prev_slot as usize].next = next_slot;
    self.nodes[next_slot as usize].prev = prev_slot;

    // Update the head if necessary
    if self.header.count() == 1 {
        self.header.used_head = NO_NODE;
    } else if self.header.used_head() == slot {
        self.header.used_head = next_slot;
    };

    // Decrement counters and mark the slot as free
    self.header.decr_count();
    self.header.free_head = slot.try_into().unwrap();
    self.nodes[slot].next = next_free;
    self.nodes[slot].prev = NO_NODE;

    // Return the processed event
    Ok(self.nodes[slot].event)
}
```

Reference: openbook-dex/openbook-v2

3. **Event Iteration**: Iterate over the events for further processing or querying. This step ensures all events can be accessed in a sequence.

```
pub fn iter(&self) -> impl Iterator<Item = (&AnyEvent, usize)> {
    EventHeapIterator {
        heap: self,
        index: 0,
        slot: self.header.used_head(),
    }
}

struct EventHeapIterator<'a> {
    heap: &'a EventHeap,
    index: usize,
    slot: usize,
}

impl<'a> Iterator for EventHeapIterator<'a> {
    type Item = (&'a AnyEvent, usize);
    fn next(&mut self) -> Option<Self::Item> {
        if self.index == self.heap.len() {
            None
        } else {
            let current_slot = self.slot;
            self.slot = self.heap.nodes[current_slot].next as usize;
            self.index += 1;
            Some((&self.heap.nodes[current_slot].event, current_slot))
        }
    }
}
```

Reference: openbook-dex/openbook-v2

The `EventHeap` is designed to handle a high volume of events efficiently. By processing events in a FIFO manner, the system ensures that all events are handled in the order they are received, maintaining the integrity and fairness of the exchange. This is particularly important in high-frequency trading environments, where the timely processing of events is critical for market efficiency.

6. Orders Tree Operations

Similar to OpenBook v1, tree node insertion and removal follow a comparable process.

Insert Operation: Insert LeafNode with Key 8

```
                 +-----------------------+
                 |        RootNode       |
                 |-----------------------|
                 |  tag: 0               |
                 |  padding: [0, 0, 0]   |
                 |  prefix_len: 2        |
                 |  key: 10              |
                 |  children: [0, 1]     |
                 |  child_earliest_expiry|
                 | [1844674407370..]     |
                 |  reserved: [0,..]     |
                 +-----------------------+
                            |
            +---------------+----------------+
            |                                |
+-----------v-----------+      +-------------v-------------+
|        InnerNode      |      |         InnerNode         |
|  tag: 1               |      |  tag: 1                   |
|  padding: [0, 0, 0]   |      |  padding: [0, 0, 0]       |
|-----------------------|      |---------------------------|
             |  prefix_len: 3        |      |  prefix_len: 4          |
             |  key: 5               |      |  key: 6                 |
             |  children: [2, 2]     |      |  children: [4, 5]       |
             |  child_earliest_expiry|      |  child_earliest_expiry  |
             | [184467440737095, ..] |      | [184467440737095, ..]   |
             |  reserved: [0, ...]   |      | reserved: [0, ...]      |
             +---------+--------------+     +-------------+-----------+
                       |                                  |
          +------------v----------+            +----------v---------------+
          |    LeafNode           |            |        LeafNode          |
          |-----------------------|            |--------------------------|
          |  tag: 2               |            |  tag: 2                  |
          |  padding: [0, 0, 0]   |            |  padding: [0, 0, 0]      |
          |  owner_slot: 1        |            |  owner_slot: 1           |
          |  time_in_force: 2     |            |  time_in_force: 2        |
          |  key: 7               |            |  key: 8                  |
          |  owner: [0x21..],     |            |  owner: [0x21..],        |
          |  quantity: 22         |            |  quantity: 50            |
          |  timestamp: 12345     |            |  timestamp: 6789         |
          |  peg_limit: 0         |            |  peg_limit: 0            |
          |  client_order_id: 0   |            |  client_order_id: 0      |
          +-----------------------+            +--------------------------+
                    |
    +---------------v---------------+
    |           FreeNode            |
    |-------------------------------|
    |  tag: 3                       |
    |  padding: [0,]                |
    |  next: 0                      |
    |  reserved: [0,]               |
    +-------------------------------+
```

Before Insertion:

```
                              +-----------------------+
                              |        RootNode       |
                              |-----------------------|
                              |  tag: 0               |
                              |  padding: [0, 0, 0]   |
                              |  prefix_len: 2        |
                              |  key: 10              |
                              |  children: [0, 1]     |
                              |  child_earliest_expiry|
                              | [1844674407370..]     |
                              |  reserved: [0,..]     |
                              +-----------------------+
                                         |
                         +---------------+---------------+
                         |                               |
             +-----------v-----------+      +------------v------------+
             |       InnerNode       |      |        LeafNode         |
             |  tag: 1               |      |-------------------------|
             |  padding: [0, 0, 0]   |      |  tag: 2                 |
             |-----------------------|      |  padding: [0, 0, 0]     |
             |  prefix_len: 3        |      |  owner_slot: 1          |
             |  key: 5               |      |  time_in_force: 2       |
             |  children: [2, 2]     |      |  key: 7                 |
             |  child_earliest_expiry|      |  owner: [0x21..],       |
             | [184467440737095, ..] |      |  quantity: 22           |
             |  reserved: [0, ...]   |      |  timestamp: 12345       |
             +--------+--------------+      |  peg_limit: 0           |
                      |                     |  client_order_id: 0     |
          +-----------v----------+          +-------------------------+
          |    LeafNode          |
          |----------------------|
          |  tag: 2              |
          |  padding: [0, 0, 0]  |
          |  owner_slot: 1       |
          |  time_in_force: 2    |
          |  key: 8              |
          |  owner: [0x21..],    |
          |  quantity: 50        |
          |  timestamp: 6789     |
          |  peg_limit: 0        |
          |  client_order_id: 0  |
          +----------------------+
                    |
    +---------------v---------------+
    |           FreeNode            |
    |-------------------------------|
    |  tag: 3                       |
    |  padding: [0,]                |
    |  next: 0                      |
    |  reserved: [0,]               |
    +-------------------------------+
```

After Insertion (Insert LeafNode with Key 8):

Remove Operation: Remove LeafNode with Key 7

```
                          +-----------------------+
                          |        RootNode       |
                          |-----------------------|
                          |  tag: 0               |
                          |  padding: [0, 0, 0]   |
                          |  prefix_len: 2        |
                          |  key: 10              |
                          |  children: [0, 1]     |
                          |  child_earliest_expiry|
                          | [1844674407370..]     |
                          |  reserved: [0,..]     |
                          +-----------------------+
                                     |
                     +---------------+---------------+
                     |                               |
         +-----------v-----------+      +------------v------------+
         |       InnerNode       |      |       LeafNode          |
         |  tag: 1               |      |-------------------------|
         |  padding: [0, 0, 0]   |      |  tag: 2                 |
         |-----------------------|      |  padding: [0, 0, 0]     |
         |  prefix_len: 3        |      |  owner_slot: 1          |
         |  key: 5               |      |  time_in_force: 2       |
         |  children: [2, 2]     |      |  key: 8                 |
         |  child_earliest_expiry|      |  owner: [0x21..],       |
         | [184467440737095, ..] |      |  quantity: 50           |
         |  reserved: [0, ...]   |      |  timestamp: 6789        |
         +--------+--------------+      |  peg_limit: 0           |
                  |                     |  client_order_id: 0     |
      +-----------v----------+          +-------------------------+
      |    LeafNode          |
      |----------------------|
      |  tag: 2              |
      |  padding: [0, 0, 0]  |
      |  owner_slot: 1       |
      |  time_in_force: 2    |
      |  key: 7              |
      |  owner: [0x21..],    |
      |  quantity: 50        |
      |  timestamp: 6789     |
      |  peg_limit: 0        |
      |  client_order_id: 0  |
      +----------------------+
                |
+---------------v---------------+
|           FreeNode            |
|-------------------------------|
|  tag: 3                       |
|  padding: [0,]                |
|  next: 0                      |
|  reserved: [0,]               |
+-------------------------------+
```

Before Removal:

```
                          +-----------------------+
                          |        RootNode       |
                          |-----------------------|
                          |  tag: 0               |
                          |  padding: [0, 0, 0]   |
                          |  prefix_len: 2        |
                          |  key: 10              |
                          |  children: [0, 1]     |
                          |  child_earliest_expiry|
                          | [1844674407370..]     |
                          |  reserved: [0,..]     |
                          +-----------------------+
                                     |
                     +---------------+---------------+
                     |                               |
         +-----------v-----------+      +------------v------------+
         |      InnerNode        |      |        LeafNode         |
         |  tag: 1               |      |-------------------------|
         |  padding: [0, 0, 0]   |      |  tag: 2                 |
         |-----------------------|      |  padding: [0, 0, 0]     |
         |  prefix_len: 3        |      |  owner_slot: 1          |
         |  key: 5               |      |  time_in_force: 2       |
         |  children: [2, 2]     |      |  key: 8                 |
         |  child_earliest_expiry|      |  owner: [0x21..],       |
         | [184467440737095, ..] |      |  quantity: 50           |
         |  reserved: [0, ...]   |      |  timestamp: 6789        |
         +--------+--------------+      |  peg_limit: 0           |
                  |                     |  client_order_id: 0     |
                  |                     +-------------------------+
                  |
                  |
                  |
  +---------------v---------------+
  |           FreeNode            |
  |-------------------------------|
  |  tag: 3                       |
  |  padding: [0,]                |
  |  next: 0                      |
  |  reserved: [0,]               |
  +-------------------------------+
```

After Removal (Remove LeafNode with Key 7):
