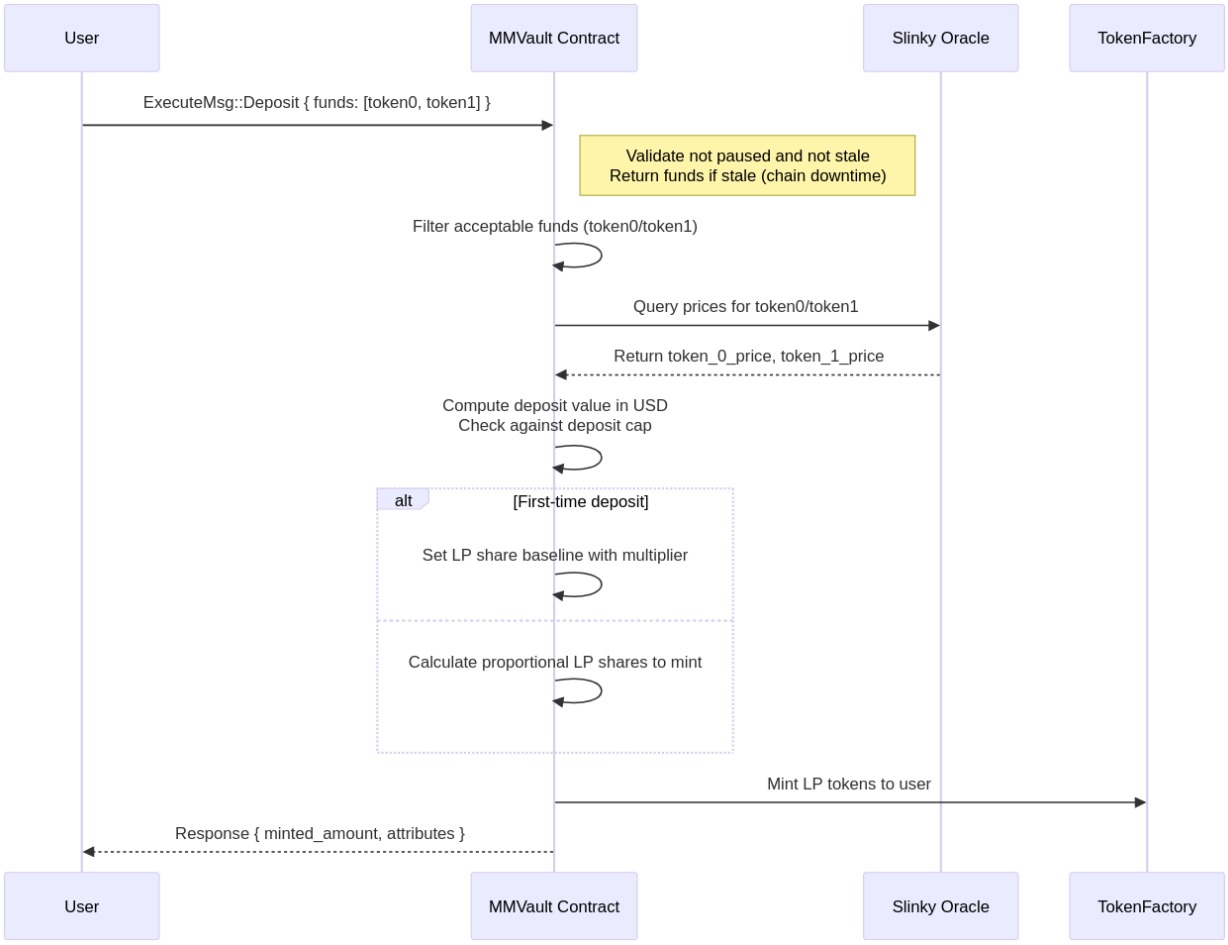
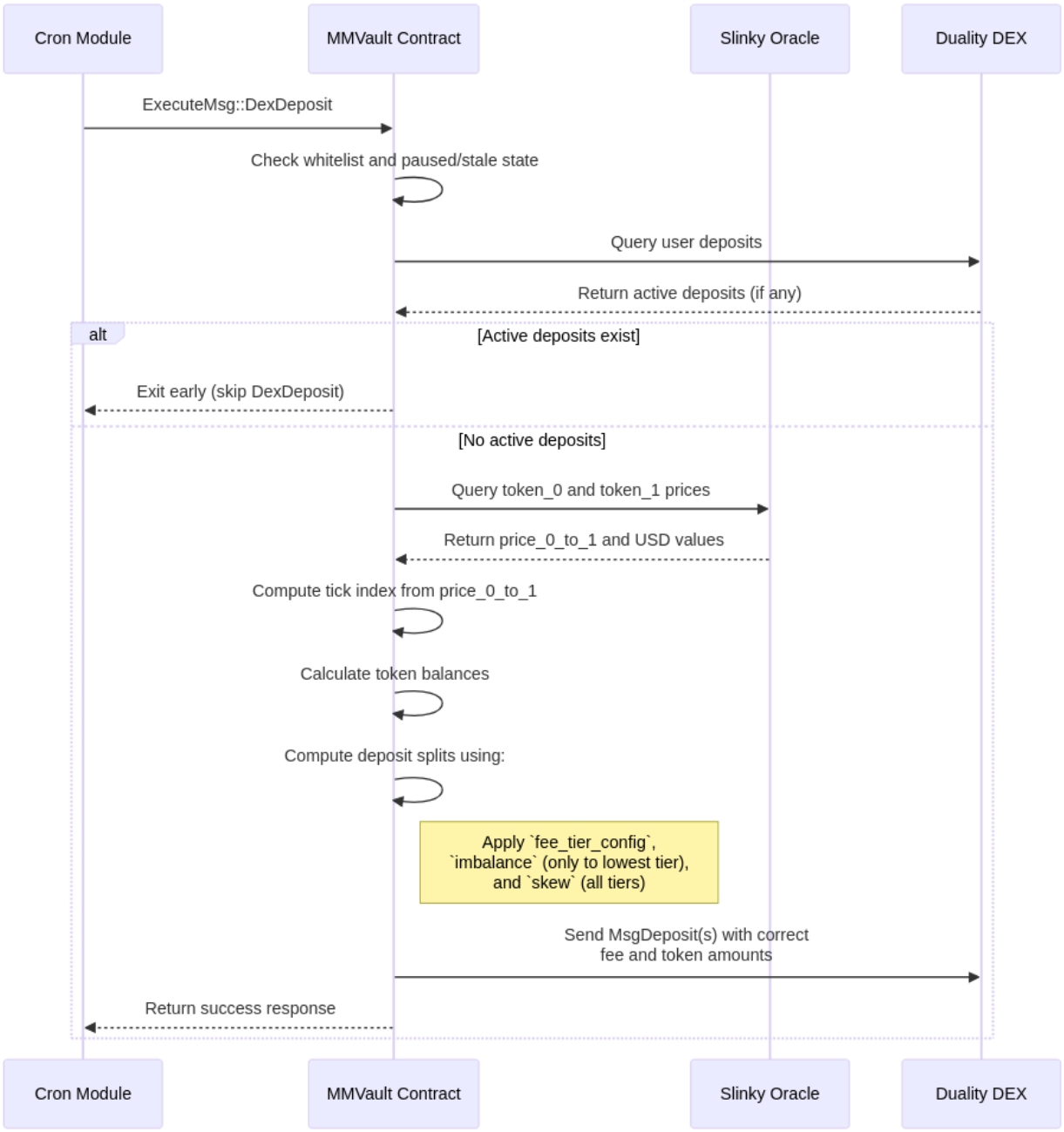
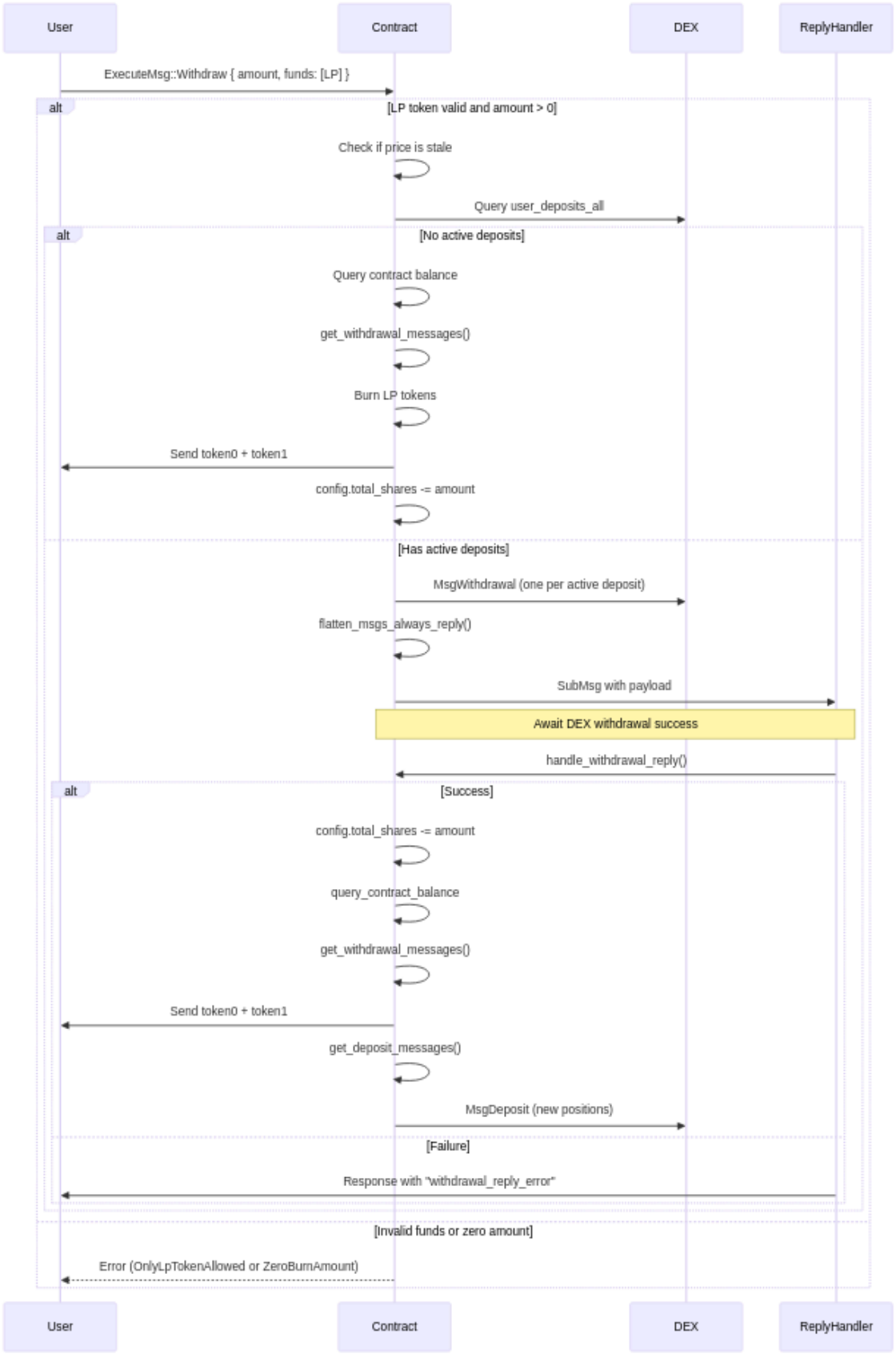
Appendix: Diagrams



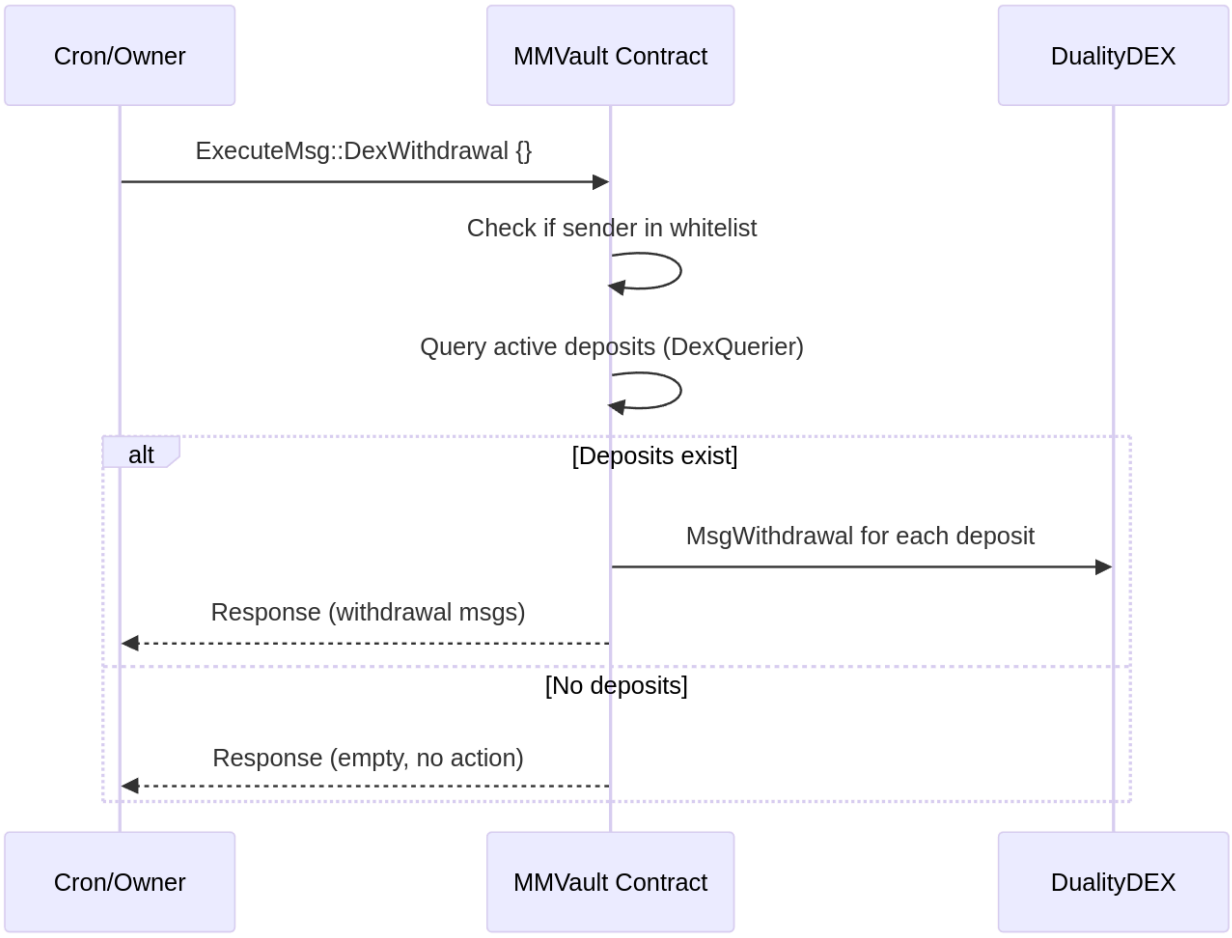
Deposit


Cron Module
MMVault Contract
Slinky Oracle
Duality DEX
ExecuteMsg::DexDeposit
Check whitelist and paused/stale state
Query user deposits
Return active deposits (if any)
alt
[Active deposits exist]
Exit early (skip DexDeposit)
[No active deposits]
Query token_0 and token_1 prices
Return price_0_to_1 and USD values
Compute tick index from price_0_to_1
Calculate token balances
Compute deposit splits using:
Apply `fee_tier_config`, `imbalance` (only to lowest tier), and `skew` (all tiers)
Send MsgDeposit(s) with correct fee and token amounts
Return success response
Cron Module
MMVault Contract
Slinky Oracle
Duality DEX

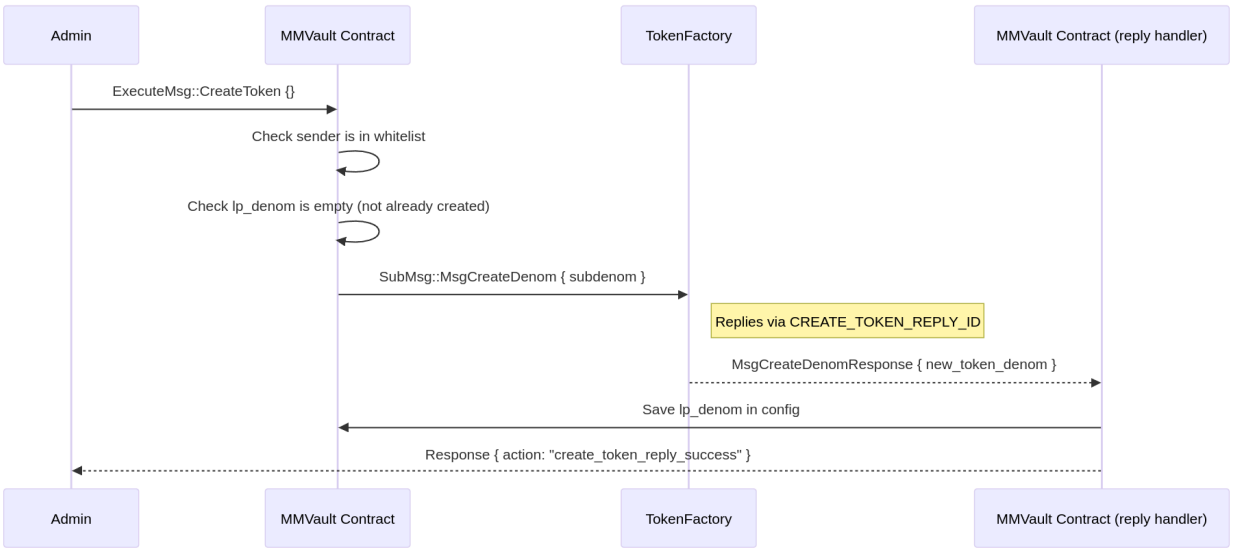
DEX Deposit


User
Contract
DEX
ReplyHandler
ExecuteMsg::Withdraw { amount, funds: [LP] }
alt
[LP token valid and amount > 0]
Check if price is stale
Query user_deposits_all
alt
[No active deposits]
Query contract balance
get_withdrawal_messages()
Burn LP tokens
Send token0 + token1
config.total_shares -= amount
[Has active deposits]
MsgWithdrawal (one per active deposit)
flatten_msgs_always_reply()
SubMsg with payload
Await DEX withdrawal success
handle_withdrawal_reply()
alt
[Success]
config.total_shares -= amount
query_contract_balance
get_withdrawal_messages()
Send token0 + token1
get_deposit_messages()
MsgDeposit (new positions)
[Failure]
Response with "withdrawal_reply_error"
[Invalid funds or zero amount]
Error (OnlyLpTokenAllowed or ZeroBurnAmount)
User
Contract
DEX
ReplyHandler

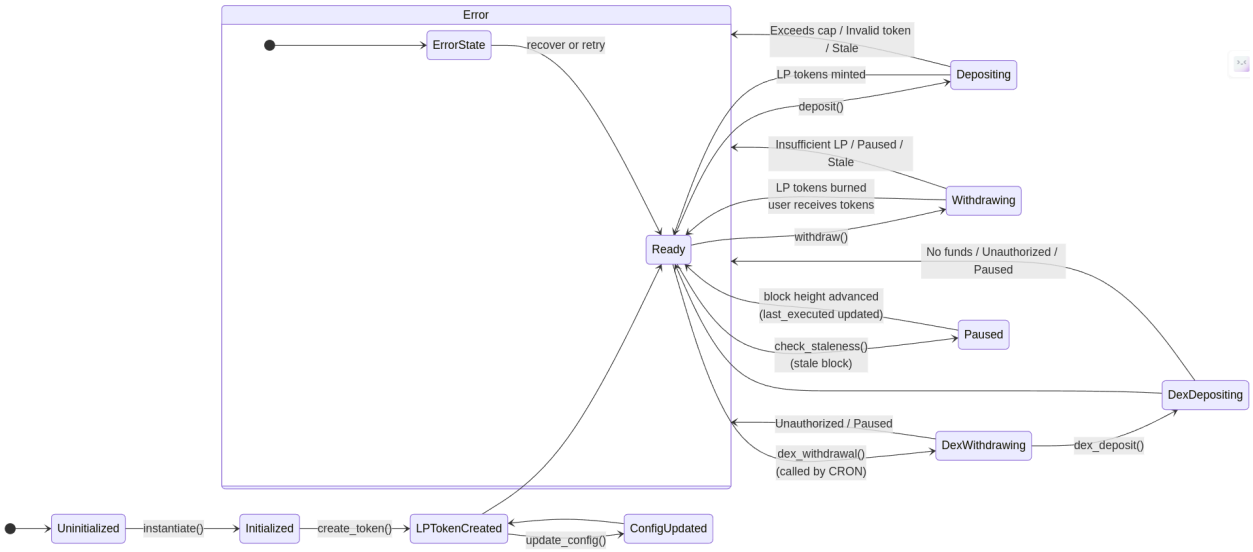
Withdraw



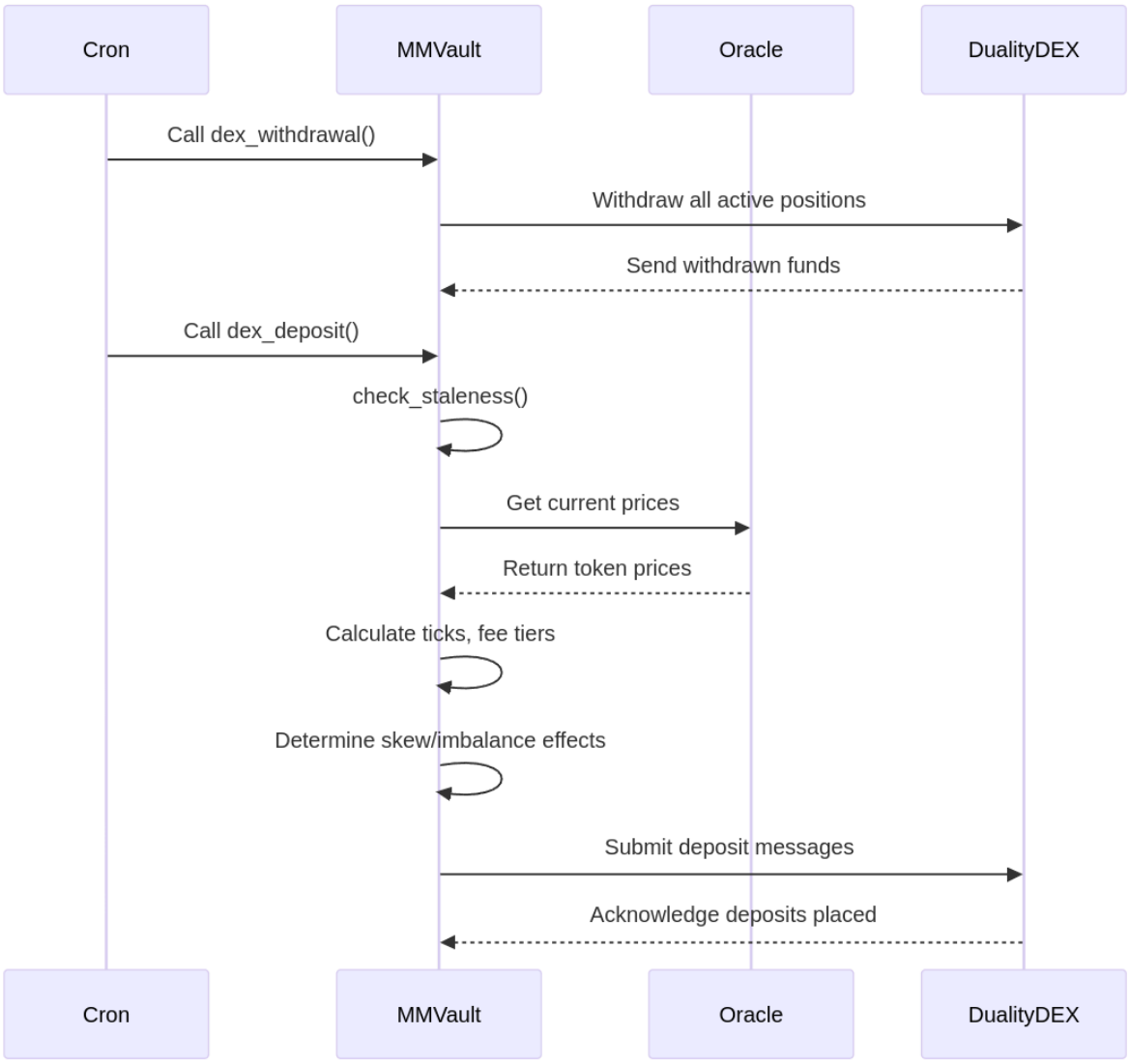
DEX Withdraw



Create Token



State Machine Diagram



Interaction with DEX