Asynchronous Order Management: Documentation & Analysis

Date: January 11, 2026

Topic: Resolving WebSocket Disconnections via Non-Blocking Architecture

1. The Core Problem

The previous implementation of the OrderBookManager was synchronous (blocking).

When the bot decided to place or cancel an order, the main execution thread would pause completely, waiting for the HTTP request to the exchange to finish (which can take 100ms - 2000ms).

Consequence: While the main thread was paused waiting for the API, it stopped processing WebSocket heartbeats (Pings). The exchange server assumed the client was dead and closed the connection.

2. The Solution: "Fire and Forget" Architecture

We refactored place_orders, cancel_orders, and cancel_all_orders to be asynchronous. The main thread now triggers a background task and immediately returns to listening for market data.

Key Technical Changes:

- Removed wait(): We removed all instances of wait(results) and time.sleep(). The bot never explicitly pauses execution anymore.
- Fixed Thread Submission: Fixed a bug where 'submit(func(arg))' was blocking the main thread. Changed to 'submit(func, arg)'.
- Added Callbacks: Attached 'add_done_callback' to background threads to handle errors and cleanup counters automatically.

3. Implementation Logic

Method	Role	Critical Safety Mechanism
place_orders	Places new orders	Uses finally block in callback to ensure counter is decremented.
cancel_orders	Cancels specific orders	Adds IDs to tracking set immediately; Callback removes them on completion.
cancel_all_orders	Nukes all open orders	Replaced dangerous "while True" loop with a single background task.

4. The New "Traffic Light" Strategy

Because the code is asynchronous, the Strategy cannot assume an order is gone the moment 'cancel_orders' returns. We introduced a state-check pattern to prevent 'Insufficient Funds' errors.

Asynchronous Order Management: Documentation & Analysis

The Flow Cycle:

- Tick 1: Strategy sees bad orders -> Calls cancel_orders. (Returns instantly).
- Tick 2: Strategy checks 'has_pending_cancels'. It sees True. It SKIPS this cycle (waiting for funds).
- Tick 3: Background thread finishes. 'has_pending_cancels' becomes False.
- Tick 4: Strategy sees funds are free -> Calls place_orders.