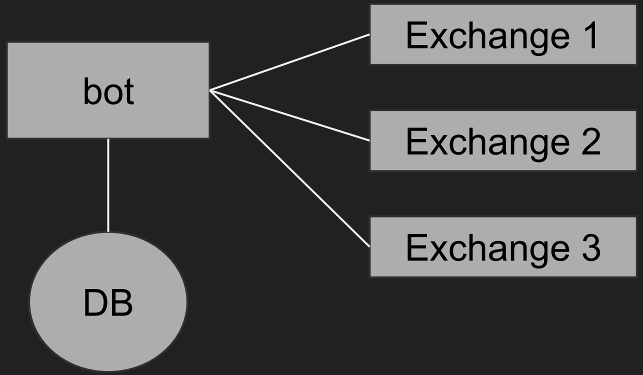
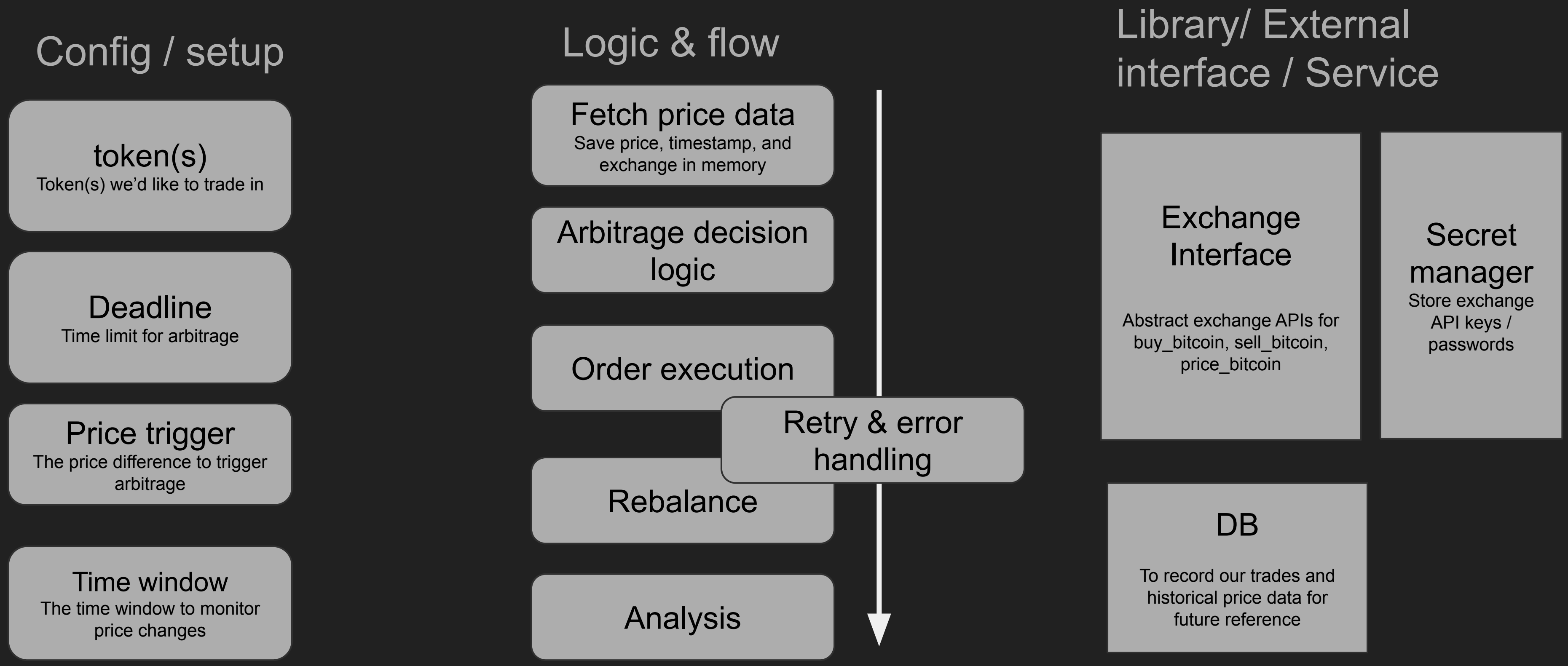
High level design



System components


Config / setup
token(s)
Token(s) we'd like to trade in
Deadline
Time limit for arbitrage
Price trigger
The price difference to trigger arbitrage
Time window
The time window to monitor price changes
Logic & flow
Fetch price data
Save price, timestamp, and exchange in memory
Arbitrage decision logic
Order execution
Retry & error handling
Rebalance
Analysis
Library/ External interface / Service
Exchange Interface
Abstract exchange APIs for buy_bitcoin, sell_bitcoin, price_bitcoin
Secret manager
Store exchange API keys / passwords
DB
To record our trades and historical price data for future reference

Design Explanation

For our simple arbitrage bot, first we’ll start it up with configurations like tokens, price trigger for arbitrages, and trading window to setup our bot.

Once we start up our bot, the bot will start to monitor prices on exchanges, and save prices and associated information in **memory**.

Once the bot has found an arbitrage opportunity (through our decision logic, which here is defined by the price trigger config), it will attempt to make the trade on exchanges in a blocking manner (so we don’t trigger multiple trade attempts before completion). Specifically, the bot will calculate if the difference between the max price and the minimum price of all the exchanges pass the execution threshold.

If an error has occurred such that our arbitrage cannot be fulfilled completely, the bot will default to the retry logic to try and complete the arbitrage.

Once the arbitrage is completed, the bot will aim to rebalance the accounts across multiple exchanges (if needed)

Lastly, the arbitrage result and price information will be saved to a database for future reference and analysis

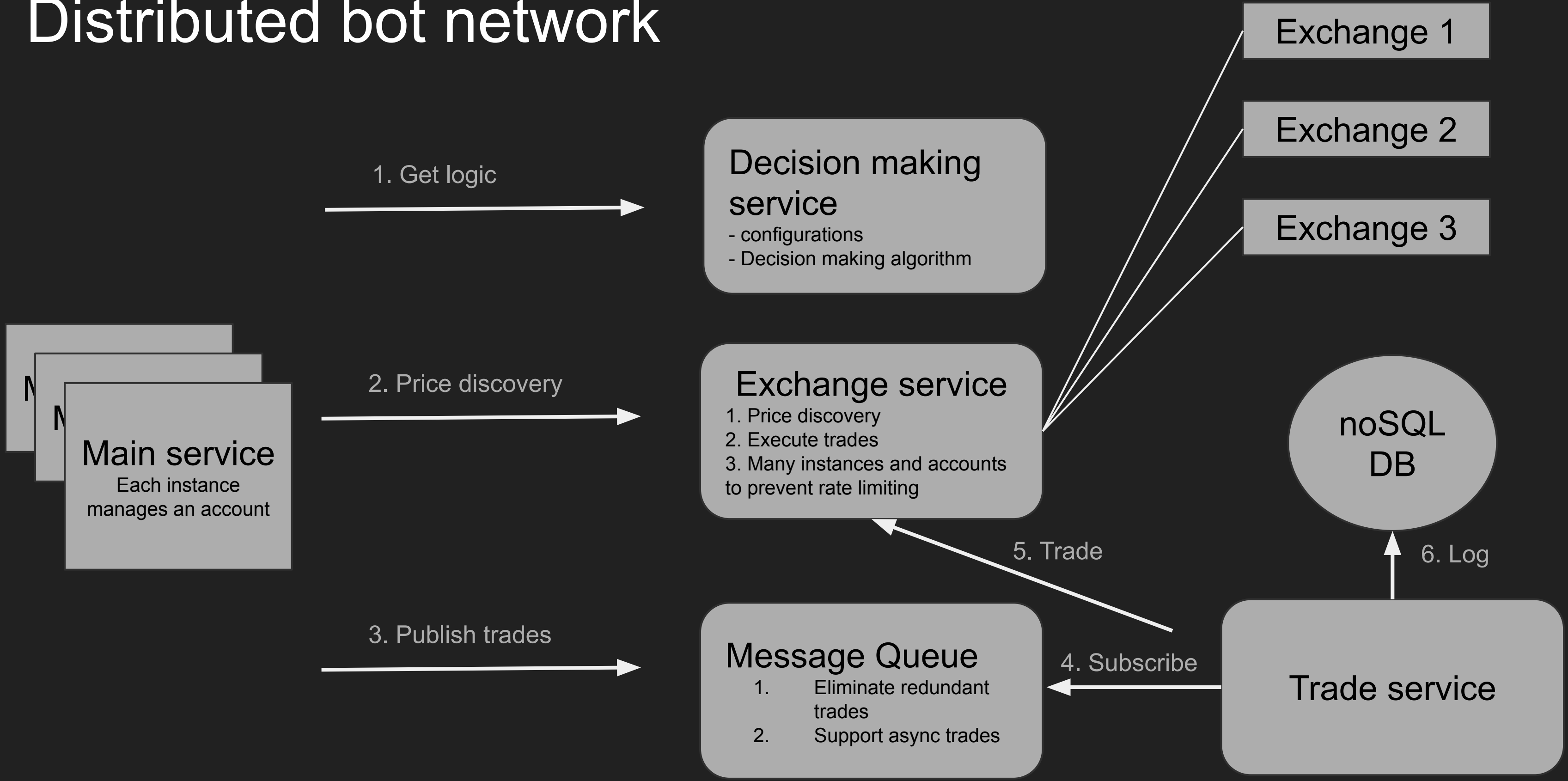
Design Choices

- We use a single bot server to handle all the tasks since it’s simple and easy to maintain
  - Drawback: this architecture will not scale, see the last section for a more scalable solution for distributed design
- We choose at first to save price information in memory because it is fast and easy to work with
  - Drawback: we lose the data if server crashes
- We use a simple price difference as a trigger to execute arbitrage trades
  - Drawback: this is not very dynamic nor adaptable upon market conditions

Design Choices 2: Tech Spec

- Use Node.js for the bot for speedy development and high maintainability
    - Downsides: Not ideal for high concurrency trades. For that, we can switch over to use something like Golang, and work on goroutines which are efficient and great for concurrency

Distributed bot network


1. Get logic
Decision making service
- configurations
- Decision making algorithm
Exchange 1
Exchange 2
Exchange 3
2. Price discovery
Exchange service
1. Price discovery
2. Execute trades
3. Many instances and accounts to prevent rate limiting
Main service
Each instance manages an account
noSQL DB
5. Trade
6. Log
3. Publish trades
Message Queue
1. Eliminate redundant trades
2. Support async trades
4. Subscribe
Trade service