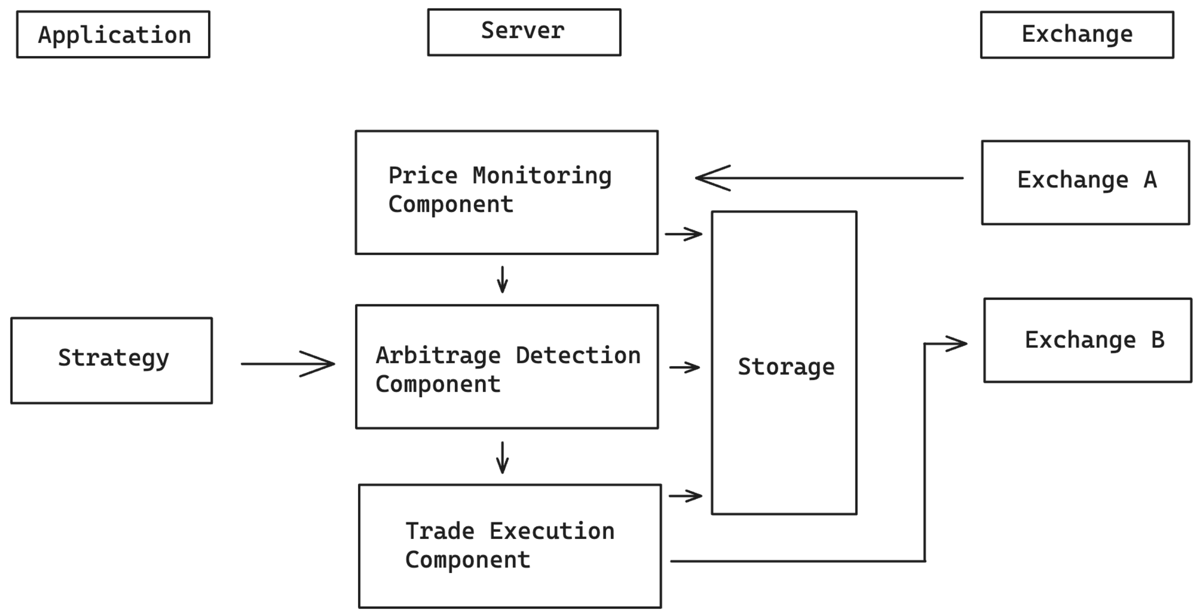
Part 1: System Design

Chia-Wei (Eric) Lin

Personal Website: https://weitude.tech/

Diagram


Application
Server
Exchange
Price Monitoring Component
Exchange A
Storage
Strategy
Arbitrage Detection Component
Exchange B
Trade Execution Component

System Design

To design a simple trading bot that arbitrages Bitcoin prices across different centralized exchanges, we can outline the following system components:

1. Price Monitoring Component:
   This component is responsible for continuously monitoring the prices of Bitcoin on different centralized exchanges. It interacts with the exchanges' API endpoints to fetch the current prices using the `price_bitcoin(timestamp)` endpoint.
2. Arbitrage Detection Component:
   The arbitrage detection component analyzes the prices fetched from different exchanges and identifies potential arbitrage opportunities. It compares the prices across exchanges and determines if there is a significant price difference that can be exploited for arbitrage.

3. Trade Execution Component:
   Once an arbitrage opportunity is identified, the trade execution component takes action by executing the buy and sell orders on the respective exchanges. It interacts with the exchanges' API endpoints to place orders using the `buy_bitcoin(price, amount)` and `sell_bitcoin(price, amount)` endpoints.

Design Choices and Trade-offs

1. Centralized vs. Decentralized Architecture:
   - Centralized Architecture: In a centralized architecture, the trading bot runs on a single server or infrastructure, and all components are integrated into a monolithic system. This approach simplifies development and deployment but may lack scalability and fault tolerance.
   - Decentralized Architecture: A decentralized architecture distributes the components across multiple servers or services. It allows for better scalability, fault tolerance, and modular development. However, it adds complexity and requires additional infrastructure and coordination.
2. Synchronous vs. Asynchronous Execution:
   - Synchronous Execution: In a synchronous approach, the bot waits for the response from each API call before proceeding to the next. This simplifies the logic but can lead to delays if there are network issues or slow API responses.
   - Asynchronous Execution: Asynchronous execution allows the bot to make concurrent API calls and process responses asynchronously. This improves performance and responsiveness but adds complexity to handle asynchronous programming, error handling, and synchronization.
3. Error Handling and Resilience:
   - Error Handling: The system should have robust error handling mechanisms to handle network errors, API failures, and exceptions. Proper error logging, retries, and fallback strategies should be implemented to minimize the impact of errors on the bot's performance.
   - Resilience: The system should be designed to handle failures of individual components or exchanges. Redundancy, failover mechanisms, and monitoring can help ensure the bot's availability and continuity of operations.

4. Security:
    - API Security: The bot should securely handle API keys and ensure that the communication between the bot and the exchanges is encrypted using HTTPS.
    - Data Security: If sensitive data is stored, appropriate measures should be taken to protect it, such as encryption and access controls.
5. Scalability and Performance:
    - Scaling: Considerations should be made for scalability to handle a growing number of exchanges, increasing trade volume, and higher frequency of price updates.
    - Performance Optimization: The system should be designed for efficient data retrieval, processing, and trade execution to minimize latency and maximize the bot's effectiveness in capturing arbitrage opportunities.