**Documentação do Coordenador.**

**Objetivo Geral do Sistema.**

Desenvolver um sistema baseado em worker\_threads do Node.js para detecção e execução de arbitragens triangulares de criptoativos em tempo real. O sistema é projetado para operar com baixa latência, modularidade e escalabilidade para suportar 6-10 exchanges, garantindo resiliência e eficiência na execução de ordens.

**Estrutura Geral**

O sistema é organizado em três níveis de workers:

* **Main**: Nível 1. Inicializa workers de dados e o coordenador, detecta arbitragens com base em dados de mercado e gera a matriz de operações. (Desenvolvido, com pequenas adaptações pendentes.)
* **Workers de Dados**: Nível 2. Um por exchange, coletam dados de mercado (order books, preços) via WebSocket público e enviam ao main para análise. (Desenvolvido, com pequenas adaptações pendentes.)
* **Worker Coordenador**: Nível 2. Recebe a matriz de operações e gerencia a arbitragem. Distribui as ordens (T1, T2, T3) para os workers de execução. No estado atual, o coordenador opera como um script principal (coordenador.js) e simula a matriz de operações, mas será ajustado para funcionar como um worker subordinado ao main durante a integração futura, recebendo a matriz via comunicação (parentPort ou MessageChannel).
* **Workers de Execução**: Nível 3. Um por exchange (ex.: OKX, Bybit, Binance), gerenciam WebSockets privados para postagem e monitoramento de ordens.
* **Workers Opcionais**: Nível 2. Ex.: workerSaldos, para monitoramento de saldos via WebSocket privado, planejado para futura implementação.

A comunicação segue a cadeia: main → workers de dados/coordenador → workers de execução, utilizando MessageChannel para baixa latência. Cada worker opera de forma independente, com conexões WebSocket mantidas ociosas (com pings regulares) até a execução de ordens. Durante a integração com o main, a inicialização do coordenador será ajustada para ser gerenciada pelo main, alinhando-se à arquitetura modular de três níveis.

**Dinâmica de Arbitragem**

O sistema executa arbitragens triangulares com três operações sequenciais (T1, T2, T3), como comprar BTC, vender BTC e comprar USDT. As arbitragens acontecem esporadicamente, e o conjunto de operações é enviado somente uma vez, não exigindo, a princípio, um sistema complexo de rastreamento.

O fluxo é o seguinte:

* O main analisa dados de mercado, detecta uma oportunidade de arbitragem e gera a matriz de operações, enviada ao coordenador. No estado atual, o coordenador simula a matriz (operacoes em coordenador.js), mas isso será substituído pela comunicação real com o main durante a integração.
* O coordenador envia T1 ao worker de execução correspondente e monitora o status retornado. A ordem enviada passa por duas etapas: Validação e Execução.
  + **Validação**: A plataforma verifica se a ordem atende os requisitos da exchange (sintaxe, parâmetros, formatos) e responde via WebSocket. A mensagem informa:
    - **accepted**: Ordem registrada no order book, passando para execução. O worker notifica o coordenador.
    - **rejected**: Ordem falhou (ex.: saldo insuficiente, parâmetro inválido). O worker informa o coordenador com a mensagem de erro, e a arbitragem é abortada, não enviando T2 e T3.
  + **Execução**: A ordem validada chega ao mercado, com status reportado via WebSocket (dependendo da exchange). Possíveis estados:
    - **filled** (ou executed): Ordem executada com sucesso. O worker notifica o coordenador, que prossegue enviando T2 e T3.
    - **live** (ou new): Ordem no order book, aguardando execução.
    - **cancelled**: Ordem não executada (ex.: IOC sem liquidez). O coordenador aborta a arbitragem.
* Um timeout de 3 segundos é aplicado a T1. Se não houver resposta final (ex.: filled, cancelled), a arbitragem é abortada, e a tabela de resultados é atualizada com o motivo "Timeout". Durante a integração com o main, a lógica de timeout será revisada para garantir que cubra todos os cenários, incluindo ordens "accepted" ou "live" sem resposta final.
* Após T1 ser preenchida, o coordenador envia T2 e T3 aos workers correspondentes, que reportam seus status. A arbitragem termina com o log dos resultados, exibindo uma tabela consolidada.

Nem todas as exchanges seguem o mesmo padrão de canais:

* OKX e Bybit usam dois canais: um para validação técnica (accepted/rejected) e outro para execução no mercado (live/filled/cancelled).
* Outras exchanges podem usar um único canal, combinando validação e execução, a ser considerado em expansões futuras. Durante a integração, o tratamento de mensagens será ajustado para lidar com mensagens inválidas ou erros críticos, garantindo robustez.

**Matriz de Operações**

A matriz de operações é uma lista de ordens gerada pelo main, contendo os detalhes de T1, T2 e T3. Cada ordem inclui campos genéricos para compatibilidade entre exchanges:

* **exchange**: Identificador da exchange (ex.: OKX, Bybit).
* **symbol**: Par de negociação (ex.: BTCUSDT, BTCBRL).
* **side**: Lado da ordem (buy, sell).
* **type**: Tipo da ordem (market, limit).
* **amount**: Quantidade a negociar (em moeda base ou cotada, conforme a exchange).
* **price**: Preço da ordem (obrigatório para limit, ignorado para market).
* **timeInForce**: Política de execução (ex.: ioc para Immediate or Cancel, null para padrão).

No estado atual, a matriz é simulada estaticamente no coordenador.js (ex.: variável operacoes) para testes. Durante a integração com o main, o coordenador será ajustado para receber a matriz dinamicamente via parentPort ou MessageChannel, alinhando-se à arquitetura descrita.

Exemplo de matriz simulada:

const operacoes = [

{ exchange: "Bybit", symbol: "BTCUSDT", side: "sell", type: "limit", amount: "0.0001", price: "95000", timeInForce: "IOC" },

{ exchange: "OKX", symbol: "BTCBRL", side: "buy", type: "market", amount: "0.0001", price: null, timeInForce: null },

{ exchange: "Binance", symbol: "USDTBRL", side: "sell", type: "market", amount: "10", price: null, timeInForce: null },

];

**Mapeamento de Campos da Matriz de Operações**

Os campos genéricos da matriz são mapeados para os padrões específicos de cada exchange, já que as APIs variam em nomenclatura e formatação. A tabela abaixo resume a correspondência para as exchanges atuais (OKX, Bybit, Binance) e futuras (KuCoin, BitPreço, etc.):

| **Exchange** | **Symbol** | **Side** | **Type** | **Quantity** | **TimeInForce** |
| --- | --- | --- | --- | --- | --- |
| OKX | instId (ex.: BTC-USDT) | buy/sell | ordType (market/ioc/limit) | sz | ioc (via ordType) |
| Bybit | symbol (ex.: BTCUSDT) | Buy/Sell | Market/Limit | qty | GTC/IOC/FOK |
| Binance | symbol (ex.: BTCUSDT) | BUY/SELL | MARKET/LIMIT | quantity | GTC/IOC/FOK |
| KuCoin\* | symbol (ex.: BTC-USDT) | buy/sell | market/limit | size | GTC/IOC/FOK |
| BitPreço\* | symbol (ex.: BTC-BRL) | buy/sell | market/limit | amount | Não especificado |
| NovaDAX\* | symbol (ex.: BTC\_BRL) | BUY/SELL | MARKET/LIMIT | amount | Não especificado |
| Foxbit\* | symbol (ex.: BTC\_BRL) | buy/sell | market/limit | quantity | Não especificado |
| Mercado Bitcoin\* | symbol (ex.: BTCBRL) | buy/sell | market/limit | quantity | Não especificado |

\*Outras exchanges são considerações futuras, a serem detalhadas conforme integração.

**Detalhes de Mapeamento**

* **OKX**:
  + symbol → instId: Adiciona hífen (ex.: BTCUSDT → BTC-USDT).
  + type/timeInForce → ordType: Combina em um campo (ex.: limit + ioc → ioc, market + null → market).
  + amount → sz: Quantidade em BTC (para spot, com tgtCcy: base\_ccy em ordens a mercado).
  + price → px: Obrigatório para limit/ioc, ignorado para market.
  + tgtCcy: base\_ccy para ordens a mercado, omitido para ordens limite.
* **Bybit**:
  + symbol → symbol: Mantém sem hífen (ex.: BTCUSDT).
  + side → side: Capitalizado (ex.: Buy, Sell).
  + type → orderType: Market ou Limit.
  + amount → qty: Quantidade em moeda base.
  + timeInForce → timeInForce: IOC, GTC ou FOK.
* **Binance**:
  + symbol → symbol: Mantém sem hífen (ex.: BTCUSDT).
  + side → side: Maiúsculo (ex.: BUY, SELL).
  + type → type: MARKET ou LIMIT.
  + amount → quantity: Quantidade em moeda base.
  + timeInForce → timeInForce: IOC, GTC ou FOK.

Este mapeamento é implementado nos workers de execução, mas centralizado aqui para facilitar manutenção e adição de novas exchanges. Durante a integração com o main, o suporte a novas exchanges será ajustado para ser dinâmico, permitindo o registro de novos workers sem modificações significativas no código.

**Rastreamento de Tempo**

O rastreamento de tempo é essencial para monitorar latência e garantir consistência:

* **t0**: Timestamp da detecção da arbitragem no main, propagado ao coordenador e workers. No estado atual, t0 é simulado pelo coordenador ao enviar T1, mas será recebido do main durante a integração.
* **t**: Tempo relativo (t = abs - t0), calculado para cada evento (envio de ordem, recebimento de status). Durante a integração, o cálculo de t será centralizado no coordenador para evitar discrepâncias, garantindo que os workers forneçam apenas abs.
* **abs**: Timestamp absoluto de cada evento (ex.: Date.now()).
* O coordenador centraliza o rastreamento, logando eventos com t e abs e usando o status de T1 para decidir o fluxo (prosseguir com T2/T3 ou abortar). Os workers reportam abs (e opcionalmente t) em mensagens de status. Durante a integração, a sincronia será reforçada para evitar cálculos inconsistentes de t.

**Worker Coordenador**

* **Função**: Gerencia a arbitragem, recebendo (ou simulando) a matriz de operações do main. Distribui ordens (T1, T2, T3) aos workers de execução via MessageChannel, monitora status e decide o fluxo:
  + Envia T1 e aguarda status (accepted, rejected, filled, cancelled).
  + Se T1 for filled, envia T2 e T3; se rejected ou cancelled, aborta imediatamente e loga o motivo.
  + Aplica timeout de 3s para T1, abortando se não preenchida.
* **Dependência**: No estado atual, simula a matriz e t0. Durante a integração, receberá a matriz e t0 do main via parentPort ou MessageChannel. Interage com workers de execução (OKX, Bybit, Binance).
* **Estado**: Mantém conexões ociosas com workers, rastreia ordens via clientOrderId (gerado dinamicamente com timestamp) e loga eventos com tempos relativos (t) e absolutos (abs). Durante a integração, o coordenador será ajustado para operar como um Worker, inicializado pelo main.
* **Comunicação**: Envia { type: 'executeOrder', order, clientOrderId } aos workers e recebe { type: 'orderStatus', clientOrderId, status, instId, avgPx, fillSz, cancelSource, abs, t }. Durante a integração, o tratamento de mensagens inválidas ou erros críticos será implementado para garantir robustez.

**Workers de Execução: Generalidades**

Os workers de execução (OKX, Bybit, Binance) compartilham características comuns:

* **Comunicação**: Usam dois canais:
  + parentPort: Comunicação com o main para inicialização, configuração e logs gerais.
  + portToCoord: MessageChannel dedicado para troca de mensagens com o coordenador (executeOrder, orderStatus).
* **Formatação de Ordens**: Recebem a matriz padronizada e convertem para o formato da exchange (ex.: symbol → instId para OKX).
* **Gestão de IDs**: Usam clientOrderId para correlacionar ordens internas com IDs da exchange (ordId para OKX, orderId para Bybit/Binance). Mantêm um orderIdMap para evitar conflitos.
* **Logs**: Incluem instId, status, motivo de rejeição/cancelamento e tempos (t, abs) em mensagens de status. No estado atual, logs são exibidos apenas na tela, mas durante a integração, serão implementados em arquivo para auditoria.
* **Modo de Operação**: Suportam ordens spot (tdMode: cash). Ajustam amount (sz/qty) para atender mínimos da exchange.
* **Reconexão**: Implementam reconexão após 5s em caso de falha no WebSocket, com ping periódico (20-25s).

**Workers de Execução: Especificidades**

**Worker de Execução: OKX**

* **Função**: Gerencia um WebSocket privado para postagem e monitoramento de ordens, formatando a matriz para a API da OKX e retornando status ao coordenador.
* **Canais**:
  + Canal de Postagem (op: 'order'): Valida ordens tecnicamente, retorna:
    - accepted (code: "0", com ordId): Ordem válida.
    - rejected (code !== "0", ex.: 51020): Erro técnico (ex.: tamanho insuficiente).
  + Canal de Ordens (orders): Reporta o estado no mercado, retorna:
    - live: Ordem ativa no livro (opcional).
    - filled: Ordem executada, com avgPx e fillSz.
    - cancelled: Ordem não executada (ex.: IOC sem liquidez).
* **Conexão**: WebSocket (wss://ws.okx.com:8443/ws/v5/private), autenticado via HMAC-SHA256. Ping a cada 25s, reconexão após 5s.
* **Mapeamento de Campos**:
  + symbol → instId (ex.: BTCUSDT → BTC-USDT).
  + side → side (buy/sell).
  + type/timeInForce → ordType (ex.: limit + ioc → ioc).
  + amount → sz (em BTC).
  + price → px (ignorado para mercado).
  + tgtCcy: base\_ccy (mercado), omitido (limite).
* **Especificidades**:
  + Suporta ordens spot (tdMode: cash).
  + Ajusta sz para mínimos (ex.: 0.00001 BTC para BTC-USDT).
  + Corrige erro 51020 ("Order amount too low") com tgtCcy: 'base\_ccy'.

**Worker de Execução: Bybit**

* **Função**: Gerencia dois WebSockets privados para postagem e monitoramento, formatando a matriz para a API da Bybit.
* **Canais**:
  + tradeWs (op: 'order.create'): Posta ordens, retorna:
    - accepted (retCode: 0, com orderId): Ordem válida.
    - rejected (retCode !== 0): Erro técnico.
  + orderWs (topic: 'order'): Monitora o estado, retorna:
    - live (orderStatus: "New"): Ordem ativa.
    - filled (orderStatus: "Filled"): Ordem executada.
    - cancelled (orderStatus: "Cancelled"): Ordem não executada.
* **Conexão**:
  + tradeWs: wss://stream.bybit.com/v5/trade.
  + orderWs: wss://stream.bybit.com/v5/private.
  + Autenticados via HMAC-SHA256. Ping a cada 20s, reconexão após 5s.
* **Mapeamento de Campos**:
  + symbol → symbol (ex.: BTCUSDT).
  + side → side (Buy/Sell).
  + type → orderType (Market/Limit).
  + amount → qty (em moeda base).
  + timeInForce → timeInForce (ex.: IOC).
* **Especificidades**:
  + Usa orderIdMap para correlacionar clientOrderId com orderId.
  + Suporta ordens spot, com side capitalizado.

**Worker de Execução: Binance**

* **Função**: Gerencia um WebSocket privado para postagem e monitoramento, formatando a matriz para a API da Binance.
* **Canais**:
  + Canal de Postagem: Posta ordens, retorna:
    - accepted: Ordem válida, com orderId.
    - rejected: Erro técnico.
  + Canal de Status: Monitora o estado, retorna:
    - live: Ordem ativa (opcional).
    - filled: Ordem executada.
    - cancelled: Ordem não executada.
* **Conexão**: WebSocket privado, autenticado via HMAC-SHA256. Ping periódico, reconexão após 5s.
* **Mapeamento de Campos**:
  + symbol → symbol (ex.: BTCUSDT).
  + side → side (BUY/SELL).
  + type → type (MARKET/LIMIT).
  + amount → quantity (em moeda base).
  + price → price (ignorado para mercado).
  + timeInForce → timeInForce (ex.: IOC).
* **Especificidades**:
  + Suporta ordens spot.
  + Ajusta quantity para mínimos (ex.: 0.00001 BTC).

**Mapeamento de Campos da Matriz de Operações**

| **Campo** | **OKX** | **Bybit** | **Binance** |
| --- | --- | --- | --- |
| symbol | instId (BTC-USDT) | symbol (BTCUSDT) | symbol (BTCUSDT) |
| side | side (buy/sell) | side (Buy/Sell) | side (BUY/SELL) |
| type | ordType (ioc) | orderType (Market/Limit) | type (MARKET/LIMIT) |
| amount | sz (BTC) | qty (BTC) | quantity (BTC) |
| price | px | price | price |
| timeInForce | Parte de ordType | timeInForce (IOC) | timeInForce (IOC) |
| tgtCcy | base\_ccy (mercado) | - | - |

**Observações para Futuro Desenvolvimento**

* **Integração com o Main**: Substituir a simulação da matriz e t0 no coordenador pela comunicação real com o main. Recomenda-se implementar a comunicação via parentPort ou MessageChannel para que o coordenador receba a matriz de operações dinamicamente, alinhando-se à arquitetura modular.
* **Estruturação do Coordenador como Worker**: Ajustar o coordenador.js para operar como um Worker inicializado pelo main, em vez de um script principal. Isso garantirá que o coordenador seja subordinado ao main, respeitando a hierarquia de três níveis e facilitando a escalabilidade.
* **Suporte Dinâmico a Novas Exchanges**: Implementar um registro dinâmico de workers de execução, permitindo adicionar novas exchanges sem modificações significativas no código do coordenador. Isso pode ser feito com uma configuração centralizada de portas e mapeamentos, aumentando a escalabilidade para 6-10 exchanges.
* **Expansão de Exchanges**: Incorporar KuCoin, BitPreço, NovaDAX, etc., atualizando o mapeamento de campos e adaptando workers de execução.
* **Modularidade**: Workers de execução são agnósticos, mas o coordenador precisa de ajustes para suportar novas exchanges dinamicamente.
* **Exclusão de Exchanges Ocupadas**: Implementar mecanismo para evitar arbitragens em exchanges com operações em andamento.
* **Saldos Insuficientes**: Logar avisos para T2/T3 sem ação imediata, já que T1 captura o lucro principal.
* **Múltiplas Arbitragens**: Gerenciar sobreposições futuramente, assumindo raridade no estado atual.
* **Logs em Arquivo**: Implementar persistência de logs em arquivo para auditoria, com opções configuráveis de nível de detalhe, além dos logs na tela.
* **Tratamento de Erros**: Durante a integração, implementar tratamento robusto para mensagens inválidas ou erros críticos dos workers, garantindo que a arbitragem seja abortada adequadamente e a tabela de resultados exibida.

**Roteiro Padrão das Fases Para Inclusão de Workers de Novas Exchanges**

1. Produzir um código que conecte com a exchange e subscreva os WebSockets e canais necessários.
2. Acrescentar ao código a postagem de uma ordem fixa de um coordenador simulado.
3. Integrar ao coordenador esse código gerado.

Alternativamente:

1. Produzir um código que conecte com a exchange e subscreva os WebSockets e canais necessários.
2. Criar um mainProxy simplificado que simule o coordenador, contendo apenas as instruções necessárias.
3. Acrescentar ao código a postagem de uma ordem fixa de um coordenador simulado.
4. Integrar o código ao coordenador, observando as especificidades do mainProxy.

**Tabela com Resultado das Operações e da Arbitragem**

Após o envio completo da matriz, o código exibe uma tabela consolidada com os resultados das operações (T1, T2, T3) em um formato estruturado. A tabela contém as colunas: T (identificador: T1, T2, T3), Exchange (nome da exchange), Par (par de negociação), Side (buy/sell), Type (limit/market), Amount (quantidade), Askd.Price (preço solicitado), Exec.Price (preço executado), TIF (time in force: IOC, GTC ou ---), Result (status: filled, rejected, cancelled, live, aborted), Time(ms) (tempo desde o início até o status final), Reason (motivo de rejeição/cancelamento ou ---).

Os dados são preenchidos a partir da matriz operacoes e atualizados com base nas mensagens de status recebidas dos workers via MessageChannel. A tabela é exibida ao final da arbitragem, após todas as operações serem concluídas ou abortadas, com uma linha em branco após T3. Durante a integração, será garantido que a tabela seja exibida em todos os cenários de falha ou abortamento, incluindo erros críticos.

Exemplo da Tabela:

[coordenador - Resultado] [abs=1745764097220 t=351] Tabela de Resultados da Arbitragem:

T | Exchange | Par | Side | Type | Amount | Askd.Price | Exec.Price | TIF | Result | Time(ms) | Reason

---+----------+---------+------+--------+--------+------------+------------+-----+-----------+----------+------------------------

T1 | Bybit | BTCUSDT | sell | limit | 0.0001 | 95000 | --- | IOC | cancelled | 350 | EC\_NoImmediateQtyToFill

T2 | OKX | BTCBRL | buy | market | 0.0001 | --- | --- | --- | aborted | 350 | T1 Failed

T3 | Binance | USDTBRL | sell | market | 10 | --- | --- | --- | aborted | 350 | T1 Failed

Gerada com a matriz:

[

{ exchange: "Bybit", symbol: "BTCUSDT", side: "sell", type: "limit", amount: "0.0001", price: "95000", timeInForce: "IOC" },

{ exchange: "OKX", symbol: "BTCBRL", side: "buy", type: "market", amount: "0.0001", price: null, timeInForce: null },

{ exchange: "Binance", symbol: "USDTBRL", side: "sell", type: "market", amount: "10", price: null, timeInForce: null }

]

**Final: Estado Operacional em 27/04/2025**

O coordenador (coordenador.js) e os workers de Binance, Bybit e OKX estão operacionais, mas requerem ajustes durante a integração com o main geral para:

* Receber a matriz de operações dinamicamente do main.
* Operar como um worker subordinado ao main.
* Suportar novas exchanges dinamicamente.
* Implementar logs em arquivo.
* Garantir robustez no tratamento de erros e exibição da tabela de resultados.

Os workers funcionais estão na pasta tester, com pequenas mudanças em relação aos códigos da subpasta \versoesAnterioresOk, que foram os primeiros a serem totalmente operacionais.