Chronicle SOFTWARE

TECHNICAL REPORT

Building an Exchange with Chronicle Matching Engine

Table of Contents

Introduction

This document discusses the design of a self-contained exchange. The exchange demonstrates how to use the Chronicle Matching Engine. The exchange provides a packaged, modular application framework that supports:

- Distributed gateways for client connectivity
- Order validation and risk checks
- Order matching
- Multiple order types, including worked orders such as Stops, Icebergs, Pegged Orders, Hidden
- Multiple time-in-force types, including IOC, FOK, GTT, GTD
- Configurable matching controls (eg self-match prevention, or controls across specific brokers)
- Market Data feeds
- Reference Data feeds
- Aggregating services such as Risk Controls, Drop Copies, Consolidated Books
- Scalable, distributed architecture
- HA/DR

Component Hierarchy

This section describes the hierarchical organisation of the key components of the exchange, illustrating how each layer builds successively on previous layers to provide scalability, functionality, and resilience.

Order Book

The core of the exchange is the per-symbol Order Book. Each Order Book maintains separate lists of Buy and Sell Orders ordered by price (high to low for Buys, low to high for Sells). Orders with the same Side and Price are grouped into the same level of the book, and prioritised within that level. Priority is initially determined by order entry time, and can change during the order lifecycle depending on how the order evolves (see details below).

Each symbol has a specific tick size which determines the minimum price change, which in turn dictates the separation and grouping of levels in the order book. For any symbol, the tick size can be static (one fixed step size), or dynamic (different tick sizes depending

on current price band). The Tick Size rules for any symbol form part of the Exchange Reference Data feed.

Orders within an Order Book are matched according to the following rules:

- A Buy order matches any Sell with a price less than or equal to the price on the Buy order, from lowest sell price to highest. Matching stops either when the Buy order is fully filled, or there are no more matching Sells.
- A Sell order matches any Buy with a price greater than or equal to the price on the Sell order, from highest buy price to lowest. Matching stops either when the Sell order is fully filled, or there are no more matching Buys.
- The Order Book is swept for matches on each new order, and also when a resting order is amended with a more aggressive price (Buy order price increases, Sell order price decreases). In this context, the term "new order" includes activation of resting orders, such as Stops.
- For any single Order Book (ie for one symbol) only one action is active at any one time (single threaded)

An example Order Book for one symbol would be as follows. For each level the quantity may be formed by one or more orders at the same price (not shown).

Symbol: ABC		
Quantity (Total)	Price	Quantity (Total)
	...	...
	103	2,500
	102.5	3,000
	101	1,000
5,000	100.5	
3,000	99	
2,000	98.5	
7,250	98	
...	...	

Order Entry

The behaviour of an order which is added to the system is principally dictated by its Type and Time-In-Force properties.

> Note, the following table lists the most typical order types which the exchange is designed to support. Not all types will necessarily be supported on initial implementation.

Main Order Type options are listed in the following table. Note the priority rules discussed further below:

Simple/Worked	Type	Comments
Simple Order Types	Market	Executes at best until matching stops. Any remainder is expired
	Limit	Executes at or better than the price specified on the order. Any remainder is added to the order book or expired based on the specified Time-In-Force
Worked Order Types	Stop	A Market order which remains inactive until the market reaches a specified stop price, at which point it is injected as a normal Market order
	Stop Limit	A Limit order which remains inactive until the market reaches a specified stop price, at which point it is injected as a normal Limit order
	Iceberg	An order which has both a Quantity and a (smaller) Disclosed Quantity. Only the Disclosed property is added to/displayed on the order book. Once the displayed quantity reduces to 0 the order is replenished by the system up to the minimum of the Disclosed and Remaining Quantity
	Hidden	A fully hidden order which is active on the book, but with no displayed quantity
	Pegged	A hidden order pegged to some mid point (e.g. best bid, offer)

Note that the Order Types fall into two main categories: “Simple” orders where the lifecycle is entirely controlled by the client (Market and Limit), and “Worked” orders where - in addition to client input - the exchange actively works the order depending on various other parameters (Stop, Stop Limit, Iceberg, Hidden, Pegged). For the remainder of this document an “Order” can be a member of either group; the terms “Simple” and “Worked” will be used whenever a differentiation between the groups is required.

The main Time-In-Force options are:

Type	Comments
Day	Expires at end of current trading day
IOC (Immediate or Cancel)	Executed on receipt. Any remainder is immediately expired
FOK (Fill or Kill)	Fully executed on receipt, or immediately expired
GTT (Good Till Time)	Expires at a specified time (UTC, second resolution)
GTD (Good Till Date)	Expires at the end of specified trading day (local calendar)

Amendments, and Priority Rules

Once an order has been entered into the system, the following properties can be amended:

- Quantity
- Disclosed Quantity
- Price
- Stop Price
- Expiration Time (GTTs)
- Expiration Date (GTDs)

However, Amends are not allowed for the following:

- Changing a Fully Visible to a Hidden order
- Changing a Hidden order to Iceberg or Fully Visible
- Changing an Iceberg to a Hidden order
- Any change to a fully Filled order

When an Order is amended, the priority within its price level is preserved or lost according to the following rules:

Amendment	Time Priority Preserved
Quantity or Disclosed Quantity Increased	X
Price Amended	X
Quantity or Disclosed Quantity Reduced	✓
Expiration Time Amended	✓
Expiration Date Amended	✓

Matching and Priority Rules

For any given price level, the priority order (from high to low) when matching is:

- All displayed orders
- Non-displayed parts of Icebergs (via replenishment)
- Fully Hidden Orders
- Pegged Orders

Within each grouping, the priority is then determined by the time the order was added to the book, or when amended (if the Amend caused the order to lose priority). For Iceberg orders, the time priority assigned to a replenished portion is the time of the replenishment, not the time the Iceberg order itself was added. If multiple Iceberg orders are to be replenished while matching an order, then replenishment would normally involve some form of scaling based on all available hidden quantities across active Icebergs at that level.

Auxiliary Order Book State

In addition to the book itself, each Order Book component maintains the following state used within the matching process:

- Off-book components of Worked Orders, e.g. for replenishing Icebergs, injecting Stop orders
- Risk controls for the book's Symbol, including awareness of aggregated content
- Reference Data, eg tick size rules, suspension status
- Matching controls, eg self-match prevention, broker-broker matching rules

Specific details of the above are discussed later. For now it is sufficient to note when continuing to build the hierarchy in this section, that an Order Book component should be understood as including some auxiliary state beyond the per-symbol book itself.

Extended Matching Controls

Additional per-order matching rules can optionally be applied on top of the core matching and priority rules described above. The available options are:

- ### Minimum Execution Quantity (MEQ)

  A minimum execution quantity can be specified on an order by setting the minQty field, which sets a minimum execution quantity on each individual fill - the intention being to prevent sweeping multiple small orders. When an aggressing order sweeps the book each available match is checked against the minQty rule where set: if an individual match falls below the minQty (on either side) then the matching resting order is ignored and the book sweep continues on to the next available order.

  If the remaining quantity on an order falls below the specified minQty, the minQty is automatically set to the quantity remaining.

  All orders with a Minimum Execution Quantity specified are hidden. The minQty can be amended, and it is valid to amend the minQty to 0 (however note the order will remain hidden). A lit order (original minQty=0) cannot be amended to a non-zero minQty. The minQty must be explicitly maintained on amend requests (else the default value of 0 will be applied).

- ### Self-Match Prevention (SMP)

  Self-Match Prevention can be used to avoid fills between cooperating orders, and is controlled by the following two parameters:
  - Self-Match Prevention ID (SMPID)
  - SMP Instruction (see below)

  Within the Matching Engine, Self-Match Prevention is detected exclusively by the SMPID attached to orders. SMPIDs themselves are managed externally at the gateway layer, and validated to ensure only authorised firms can submit orders with specific SMPIDs. This in turn means the SMP functionality is kept general and can be used very flexibly to control self-matching at various different levels - e.g. Firm, Desk, Trader Group, Account and even across different Firms where appropriate.

When a self-match is identified between two orders (equal SMPIDs) one of the orders is cancelled according to the SMP Instruction on the aggressing order:

- SMPInstruction=CANCEL_OLD (default): The resting order is cancelled
- SMPInstruction=CANCEL_NEW: The aggressing order is cancelled

Note: If the SMPInstruction=CANCEL_NEW and the aggressing order is already partially filled, the existing fills are retained and only the remaining quantity is cancelled.

A resting order which is cancelled by SMP receives an ExecutionReport with OrdStatus CANCELED. An aggressing order which is cancelled by SMP receives an ExecutionReport with OrdStatus EXPIRED (similar to the unfilled quantity of an IOC).

The SMPID may be amended (including being cleared). The SMPID must be explicitly maintained on amend requests (else the default value on no-SMP will be applied). The SMP Instruction may also be amended.

SMP is not allowed on orders with a TimeInForce of Fill or Kill.

Processing Pipeline

A processing pipeline coordinates the handling of events across one or more Order Books. Note, the pipeline does not itself control the raising of events (e.g. user requests, Stop injections), but is responsible for coordinating the events, and implementing the steps which are applied to any one event.

The main steps in the processing pipeline for each request are:

- Timestamping, Identity stamping, and sequencing of requests
- Validation (eg amend rules, symbol trading status)
- Risk Checks
- Matching (ie the main order book sweep)
- Matching Controls (eg self match, broker-broker rules)
- Worked-Order Replenishment where applicable (eg Icebergs)
- Generating responses to an event (eg Ack, Reject, Fills)

The pipeline is responsible for assigning unique identities to identify Orders and Executions. These identities are guaranteed to be unique across different trading days (for

long-lived orders), and are pseudo-randomised to avoid yielding any information on activity which could potentially be leveraged by users. The Order Identifier is constant throughout the lifecycle of any one order.

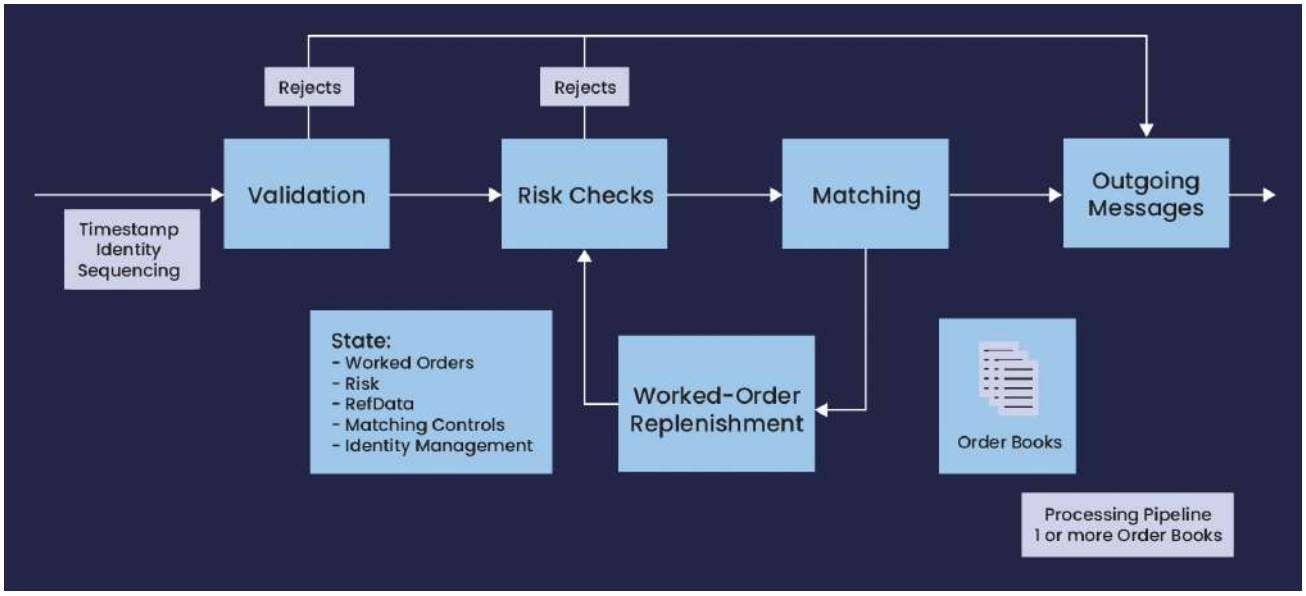

Figure 1. Main processing pipeline of the exchange

Transactions, Status, and Execution Types

The main transactions on an Order handled by a processing pipeline are:

- New
- Amend
- Cancel
- Mass Cancel
- Executions

These can be raised either by user requests or automatically from within Chronicle Matching Engine (see next section).

The status of an order at any one time is one of the following (from high to low precedence):

- Filled
- Cancelled
- Expired
- Partially Filled
- New
- Rejected

- Suspended (not currently on the order book - eg parked Stop/Stop Limit)

Where an order can be in more than one state the state with highest precedence is selected.

An Execution Type contains the result of a specific action and complements the overall state of the order:

- Order Accepted
- Order Rejected
- Order Executed (partially or fully filled)
- Order Expired (GTT or GTD expiry; IOC; FOK)
- Order Cancelled
- Order Cancel/Replaced
- Suspended (not currently on the order book - eg parked Stop/Stop Limit)

Chronicle Matching Engine

A Chronicle Matching Engine instance wraps a single processing pipeline (which may control 1 or more Order Books), and controls the interfaces between the pipeline and the wider exchange environment. A core piece of the instance is a single event loop which controls and coordinates all activity across the instance.

The Event Loop monitors activity across the following 5 generators of events:

- Client Gateway Requests
- Internal Timer Events (eg GTT Expiries, Exchange Open/Close)
- Worked Order Events (eg Stop/Stop Limit Injection)
- Aggregated Risk Updates (ie updates from centralised Risk controls)
- External Market and Reference Data Events (eg symbol suspended)

and provides an ordered sequence of events to the Processing Pipeline, where each event is actioned.

For each event processed by the pipeline, the Chronicle Matching Engine instance atomically appends a single change record to a Chronicle Queue, known as the Transaction Queue. There is 1 Transaction Queue per instance, and each individual record written to the Queue contains complete information related to the processing of the event which raised the transaction, in the form:

[Incoming Event | Order Book and State Changes | Outgoing Events]

The Transaction Queue can be accessed on the same host as the Chronicle Matching Engine instance, or alternatively be continuously replicated across any number of hosts. A Transaction Queue is self-contained and provides complete, ordered, information about an instance which can be used to monitor as well as clone any instance. It is principally used to drive the following:

- Market Data updates (from Additions, Amends, Executions)
- Drop Copies
- Feeds to aggregation services (such as system-wide risk)
- HA/DR

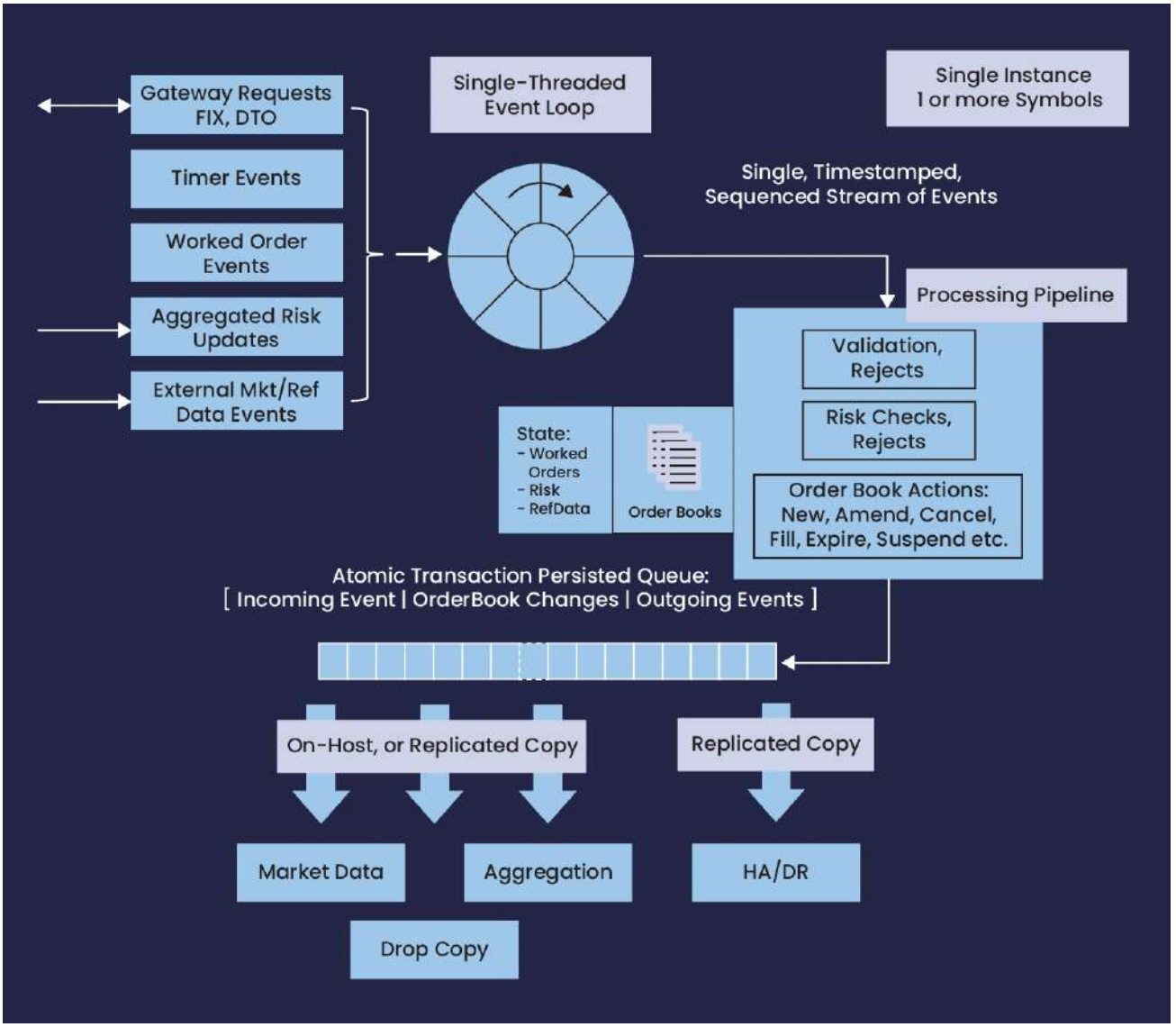

Figure 2. A Chronicle Matching Engine instance processing events

The use of a queue to decouple the instance and other feeds across the wider exchange environment enables great flexibility on how components are distributed in order to best

use available resources, and also to segregate services based on latency profiles (e.g. drop copies are likely less latency sensitive than market data and can be handled off the critical path).

Distributed Matching Engines Instances, & Aggregating Services

Matching Engine instances can be freely scaled to meet design load: the only requirement is that all transactions for any one symbol are handled by a single instance. The Transaction Queue feeds from multiple instances can be used to feed other services, and these services can either themselves be distributed, or aggregated into a single instance. Where aggregation is required, Transaction Queues from multiple instances are replicated into the aggregating service, then polled round-robin for updates.

Risk Controls

Risk checks form part of the processing pipeline. A number of checks are per-symbol or per-order and can be applied using local state within any given Chronicle Matching Engine instance. Other checks may require a global view of a client's exposure across all symbols, and for these a separate, aggregating Risk Service is required. This service builds required full state by listening to the individual Transaction Queues from all instances, and in turn communicates this overall state through its own Risk Queue, which is replicated to and read by each instance to update local Risk controls.

This is illustrated in the below diagram where three transaction engines replicate their transaction queue to the Aggregating Risk service host. The Risk service then uses those queues to maintain a global risk state which is persisted by a series of atomic updates to an Aggregated Risk Queue (directly analogous to how Transaction Queues are built by the Chronicle Matching Engine instances). This Aggregated Risk Queue is then replicated to each of the instance's hosts, where it is polled by the instance event loop for updates.

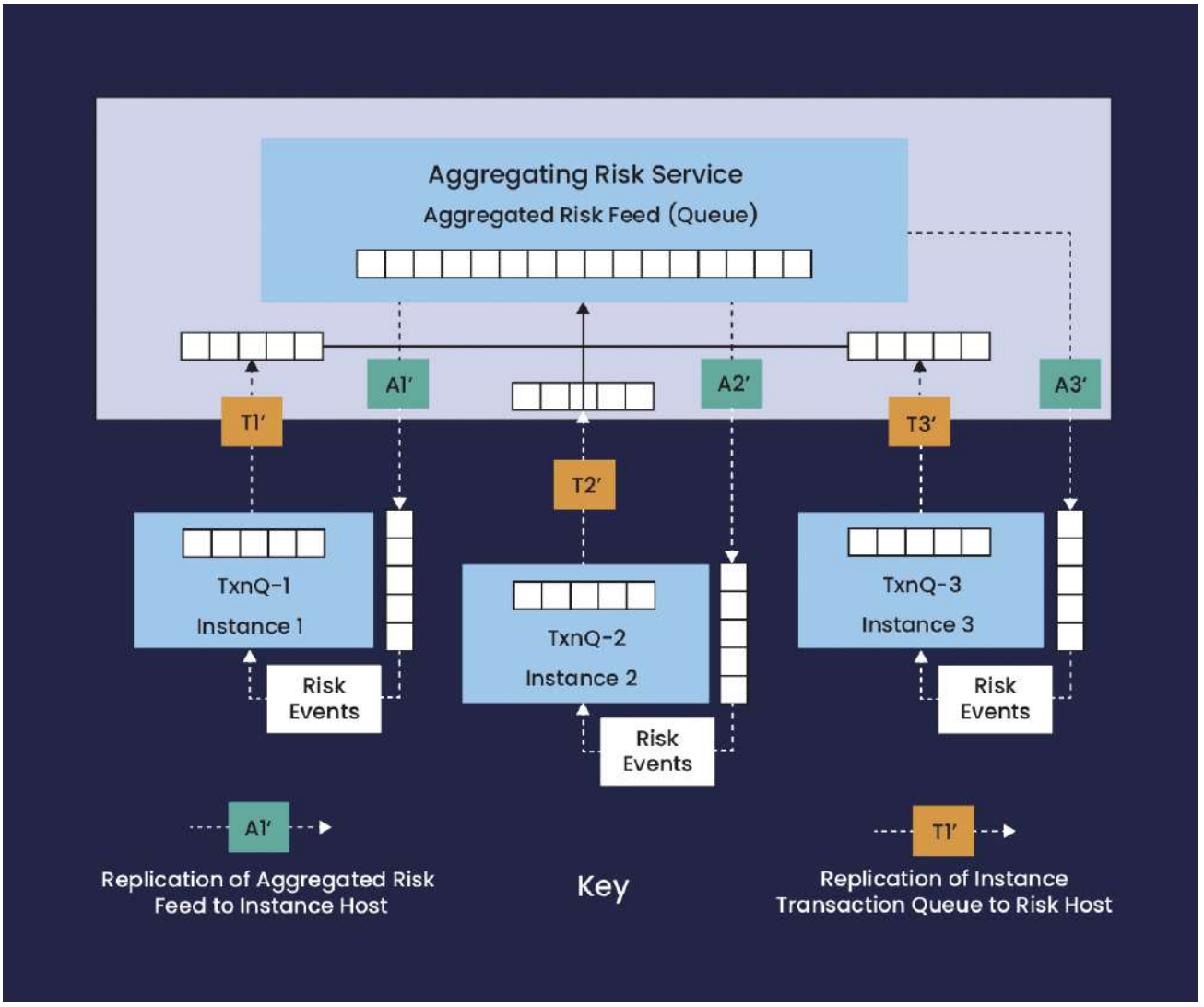

Figure 3. Three transaction engines replicating their transaction queue to the Aggregating Risk service host

Typical risk controls include:

- Maximum Gross (sum of all trades and all open orders). Global or per instrument.
- Maximum Net (the difference between the sum of all Buy trades and open orders, and the sum of all Sell trades and open orders). Global or per instrument.
- Maximum Order Value. Can be set per Order Type
- Order Type per Security. Allowed Order Types per security; includes blocking a security entirely

Further risk controls can be added easily as needed.

Drop-Copy Service

A Drop-Copy feed can be built in a directly similar fashion to an Aggregating Risk service, although in this instance there is no need to replicate back to the Chronicle Matching

Engine instances: any output from the Drop-Copy Service is sent directly to the connected clients.

Again, replicated copies of each instance's Transaction Queue are set up on the Drop-Copy Service host and updates are processed to generate Drop-Copy reports of all relevant execution reports to connected clients, specifically any of the following type:

- Order Ack
- Reject
- Cancel
- Cancel/Replace
- Expired
- Suspended
- Rejected
- Execution

It is worth noting, that as the input data to the Drop-Copy Service is the same as that for the Aggregating Risk Service, both can optionally be run on the same host and share a single set of Transaction Queues replicated from the Chronicle Matching Engine instances' hosts.

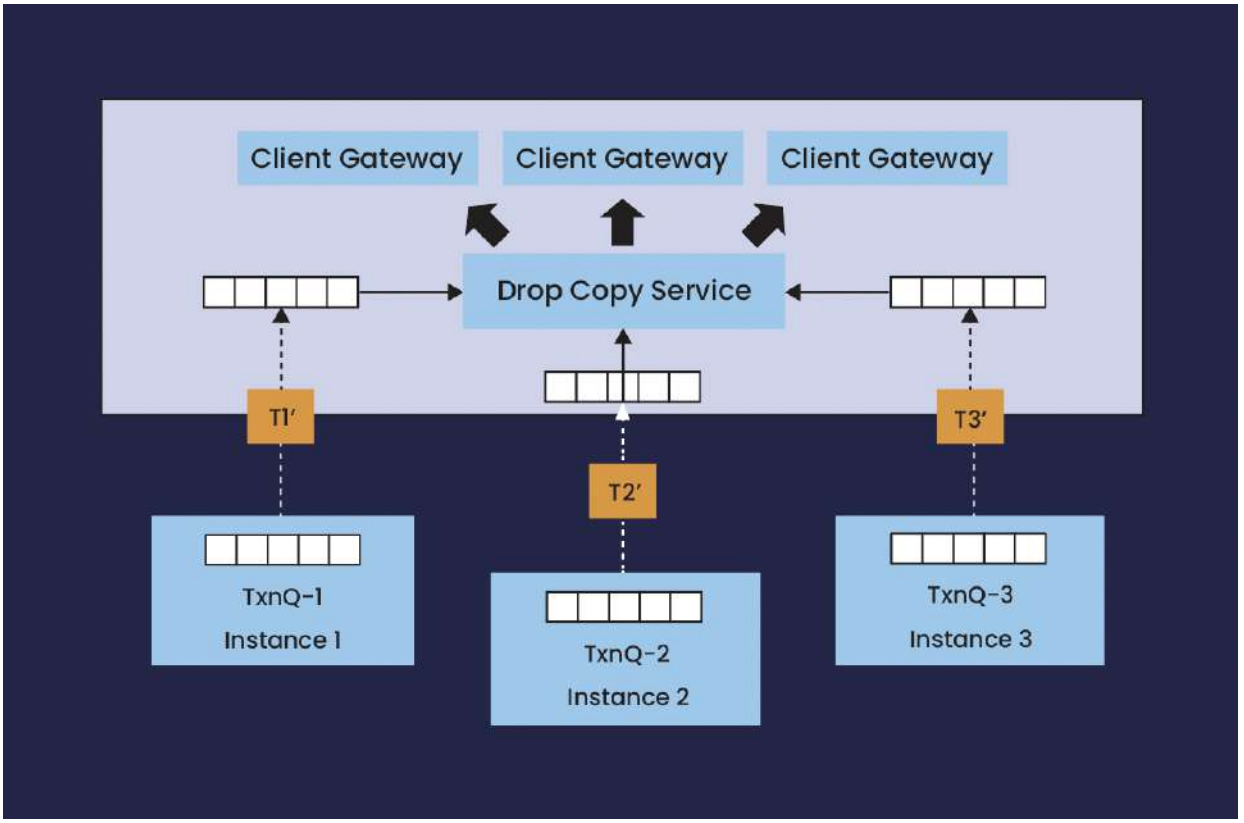

Figure 4. Generating Drop-Copy reports of all relevant execution reports

Consolidated Book

A Consolidated Book Service providing a single view across all Books (for example for driving front-end GUIs) can be assembled following exactly the same approach as a Drop Copy service. In this case the Consolidated Book service monitors (replicated copies of) the Chronicle Matching Engine instances' queues and assembles a real-time summary picture across all books, which can be used to drive queries and live updates to connected clients. Again optionally this service can share the same input queues as Risk and/or Drop Copy.

Market Data Service

A Market Data service can be constructed in a similar fashion to the above as an aggregating service using replicated copies of each Matching Engine's Transaction Queue. Alternatively, each Matching Engine host could publish market data individually given all information is available from the local Transaction Queue for the subset of symbols traded on that instance. Market Data updates would normally be published via UDP, (although the initial POC will be TCP/IP) which in turn would require the use of sequence numbers for gap detection, so if publishing independently each Chronicle Matching Engine instance would either need to publish to a separate topic, or sequence numbers could be grouped into separate channels by setting distinct high-order bits or similar per Chronicle Matching Engine instance. The full implementation details of the Market Data service will be finalised in due course.

The Market Data service can provide three levels of data of increasing depth as follows:

- Level 1: Best bid and offer (quantity and price), plus all executions
- Level 2: Periodic snapshots of the order book to a fixed depth (~10 levels)
- Level 3: Full order book, and lowest latency:
    - Every visible order on the book
    - All executions

A separate TCP/IP interface will be provided to enable recovery of messages following a gap detection on the UDP stream. This mechanism will provide the ability to recover messages only up to a limited age (e.g. last N messages, or last hour, or trading day), after which the message should be considered lost.

Client Gateways

A distributed cluster of client gateways forms the client-facing surface of the exchange. client gateways can be distributed arbitrarily to match demand and resources, with one or more gateways on each of one or more hosts.

A client is assigned a specific number of sessions and can connect any session to any gateway. Each session can only be connected to one gateway at any time. All symbols can be reached from any one session, and each session is logically independent from any other. Sessions can be configured as part of a group, enabling one session to enter orders on-behalf-of another session if needed.

Optional throttling controls can be set up on a per-session basis to limit the number of messages sent per second (sliding window). Messages exceeding the limit will be rejected, and continued breaches will result in the session being dropped (via a controlled Logout as opposed to TCP/IP disconnect). Heartbeats and administrative sessions do not form part of the throttling count.

A connected client may optionally enable Cancel-On-Disconnect for the session, to ensure all open orders submitted through that session are cancelled in the event of a disconnect or logout. Client gateways support FIX.
All state for a client gateway is incrementally written to a Queue, and can be replicated across one or more hosts for HA/DR.

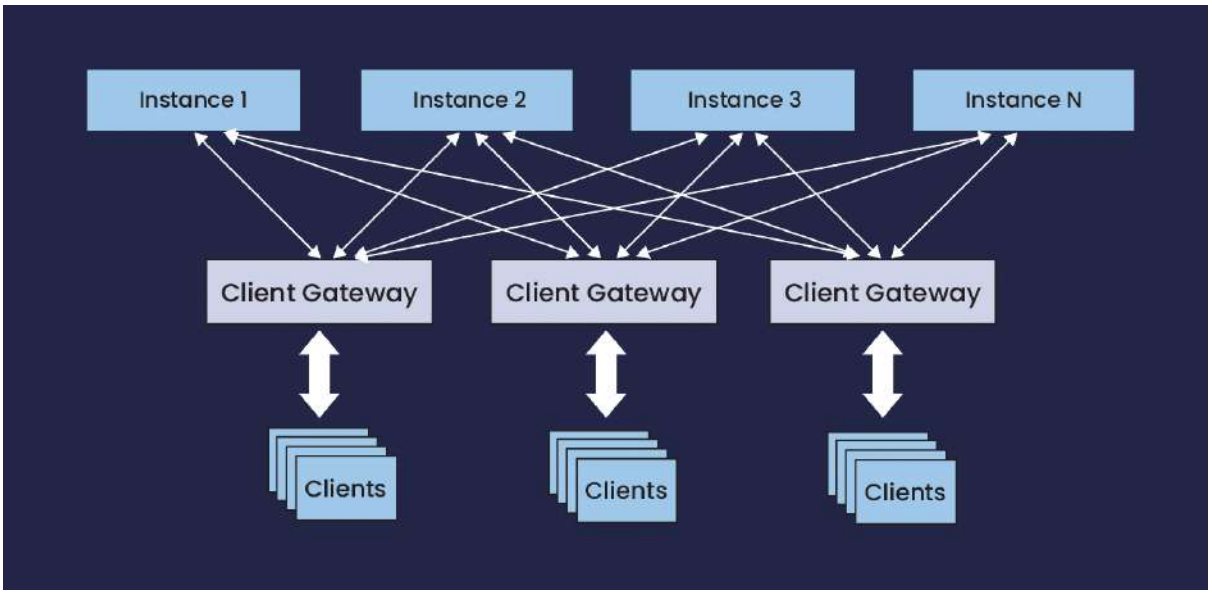

Figure 5. Clients connecting sessions to client gateways

Distribution, Scalability, and HA/DR

Throughout the preceding discussion emphasis has been placed on the modular and hierarchical nature of the design:

- The core per-symbol Order Book
- Processing Pipeline with 1 or more Order Books per Matching Engine
- 1 or more Matching Engines, with Aggregating services providing global views
- All coupling between hosts based on replicated Queues
- Distributed cluster of client gateways

This approach provides great flexibility in how components are distributed over available hosts, and can be freely scaled as needed to match growth in demand.

All state required by an individual component is contained in the component's Queue, and the Queue can be replicated in real time to any number of hosts, which in turn provides considerable flexibility for HA/DR options. Replication can optionally be configured to require acknowledgements from some or all of the secondary instances which minimises the risk of message loss at the cost of higher latencies. This aspect is again freely configurable, so parts of the system which demand the highest latencies but can tolerate some potential message loss can be run without acknowledgement, whereas other parts of the system which demand no message loss but which are less latency sensitive can run with acknowledgement.

For completeness it is noted that for some components an alternative approach may be to use distributed maps to share state, rather than an explicitly queue-based mechanism. The queue approach is favoured for two main reasons:

1. A map-based approach loses time information - i.e. multiple updates to the same key necessarily lose time history. By contrast, a queue-based approach maintains full time-ordered history of events, which not only enables strong coordination as the data is distributed around the system, but also naturally provides a replay mechanism (eg for simulation and test environments)
2. A map-based approach ultimately uses a form of replication built on the same mechanism as queue replication. By keeping everything explicitly using queues and queue replication, a single common mechanism is used throughout the system, making maintenance simpler

In this model also, only a single component is responsible for appending data to any one Queue. Once a Queue is available on a host, any number of services can read data from

the Queue completely independently. This in turn allows - for example - multiple aggregation services on one host to be driven off a single replicated copy of the set of queues from the Chronicle Matching Engine instances. Going the other way, multiple services can be run on the same host as the source Queue without the need for replication: all that is required is access to the Queue data. The number of Queue readers does not impact write performance.

The diagram below illustrates the main components of a typical full exchange environment. Components within the shaded grey boxes can be arranged arbitrarily across any number of hosts to match available resources. The entire environment is then replicated to a secondary DR site. The DR site will require the same components as the primary in one-to-one correspondence, but these components may be arranged across available resources independently of the primary arrangement.

Clients can connect to any available gateway. Gateways in the secondary site will not accept any connections until the site becomes primary.

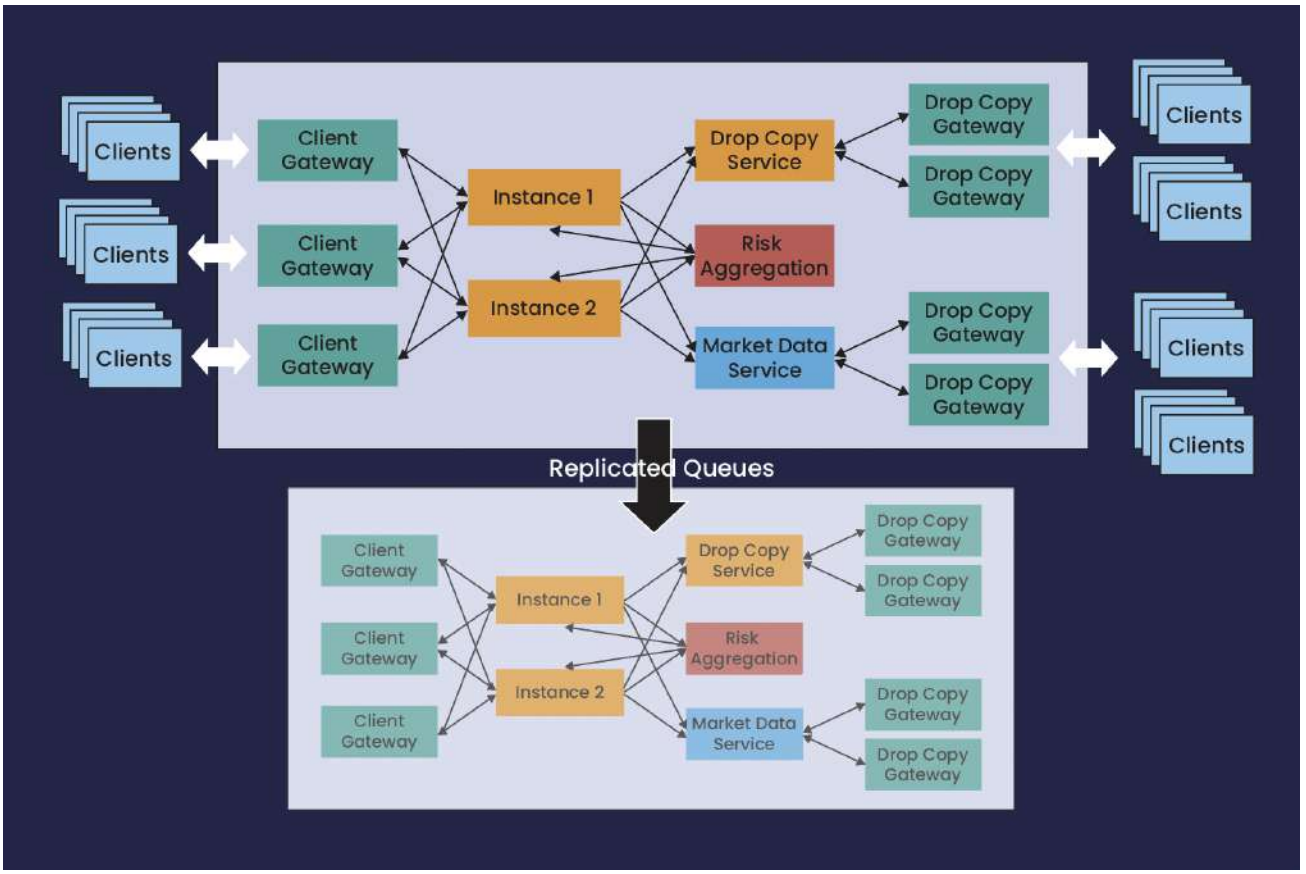

Figure 6. Main components of a typical full exchange environment

Minimal Demonstrator

A minimal exchange environment demonstrating core matching functionality with all symbols supported in one Matching Engine (but neither Drop Copy nor Market Data services) can leverage the flexibility of the design to collapse the client gateway and Risk Aggregation services into the same environment/host as the Chronicle Matching Engine instance, providing a fully self-contained package. The components remain coupled via Queues, and replication to a secondary instance remains as above.

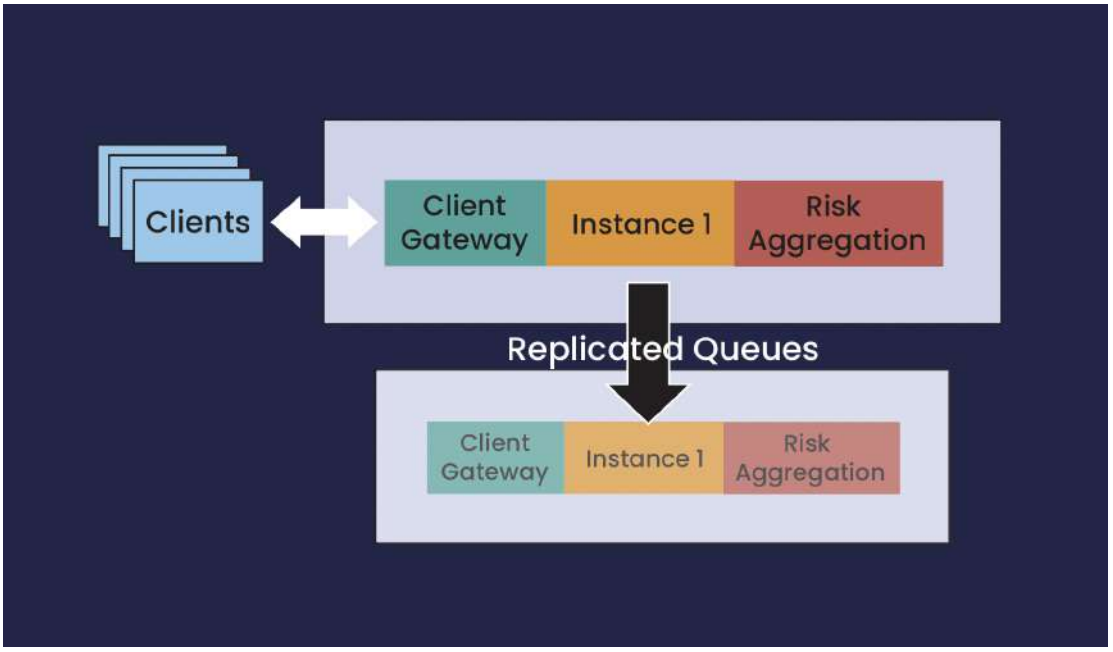

Figure 7. Minimal exchange environment with all symbols supported in one instance of a Matching Engine

Glossary of Terms

Term	Definition
Client	Any computer hardware or software device that requests access to a service provided by a server
Client Gateway	A software component that directs requests from a client to the appropriate service or service instance, possibly in addition performing authentication.
Drop-Copy service	A service which receives copies of trading activity via a separate channel from main order entry. Designed to facilitate real-time monitoring of trading activity (eg back-office, regulatory feeds) off the latency-critical path
ExecutionReport	A report on the status of the execution of Order(s) and Transaction(s)
Order	An instruction to buy or sell on a trading venue
Order Book	A price-ordered list of buy and sell Orders for instruments on an exchange
Risk Aggregation	The real-time collection of multiple metrics across client trading activities in order to maintain an overall risk status and corresponding limits for each client
Time-In-Force	Indicates how long an Order will remain active before it is executed or expires
Symbol	An abbreviation used to uniquely identify an instrument that is being traded on an exchange