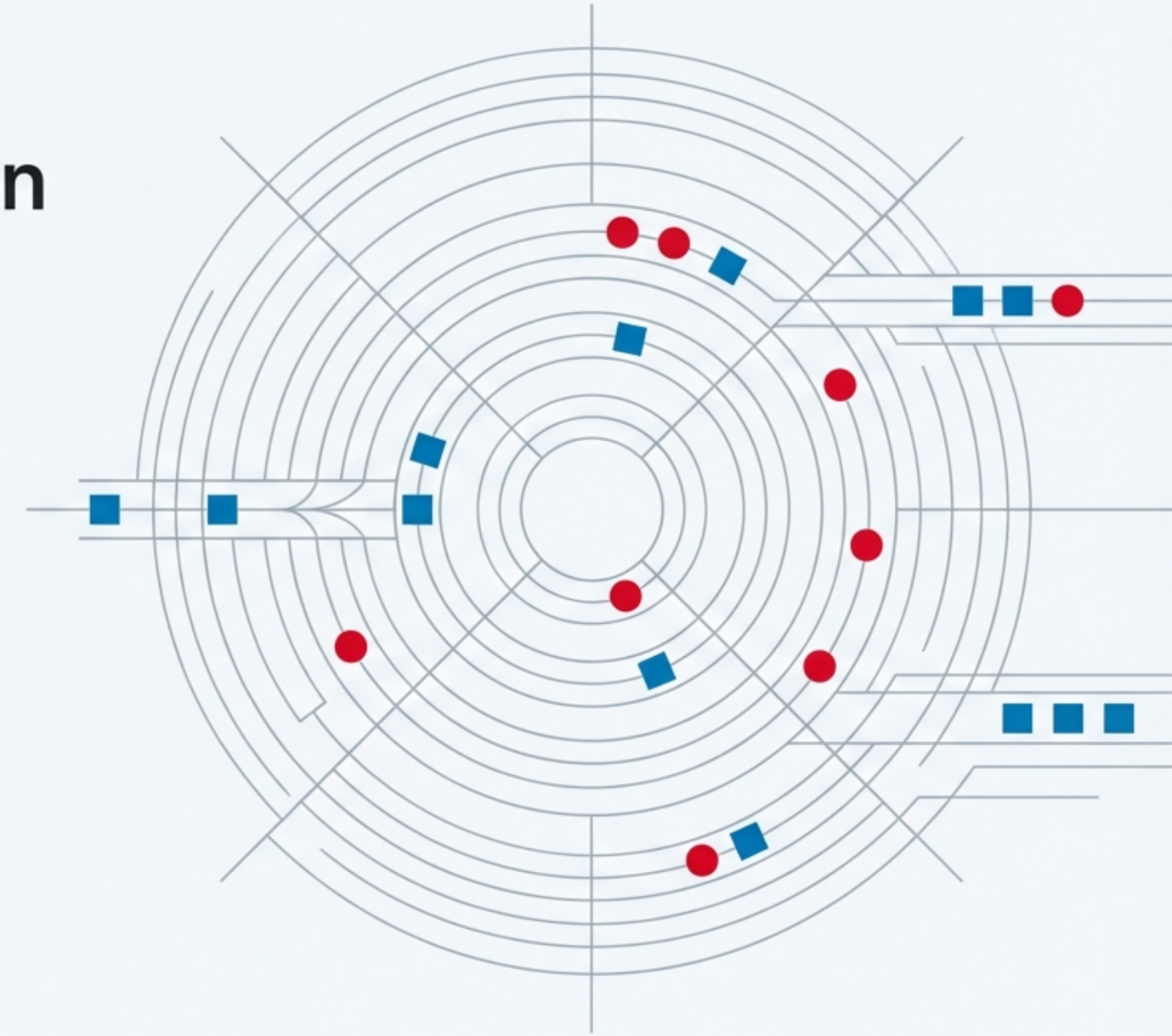
Engineering a High-Performance Transaction Screening Engine

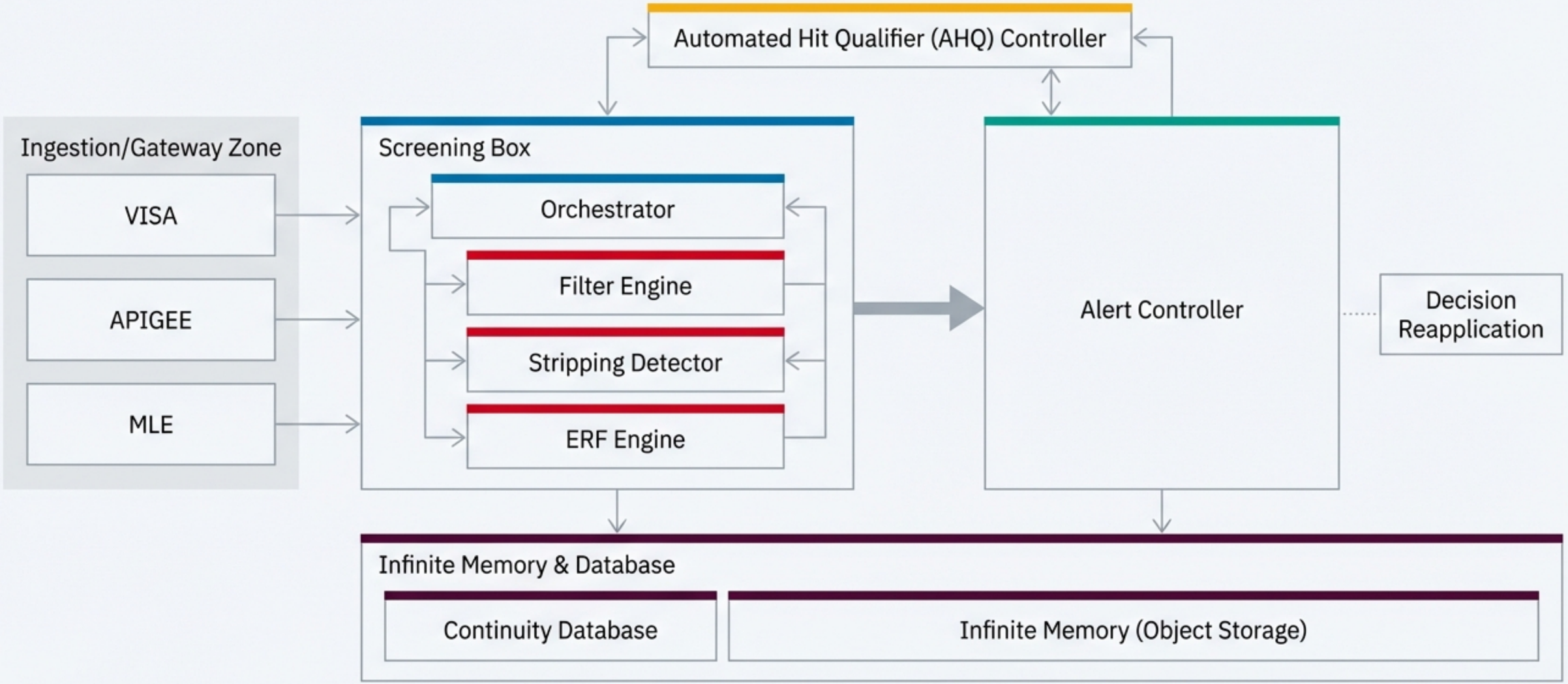
An Architectural Blueprint of the LexisNexis® Firco™ Continuity Platform



A detailed examination of the components, data flows, and intelligent automation that power real-time financial crime compliance.

The Firco Continuity Solution Architecture

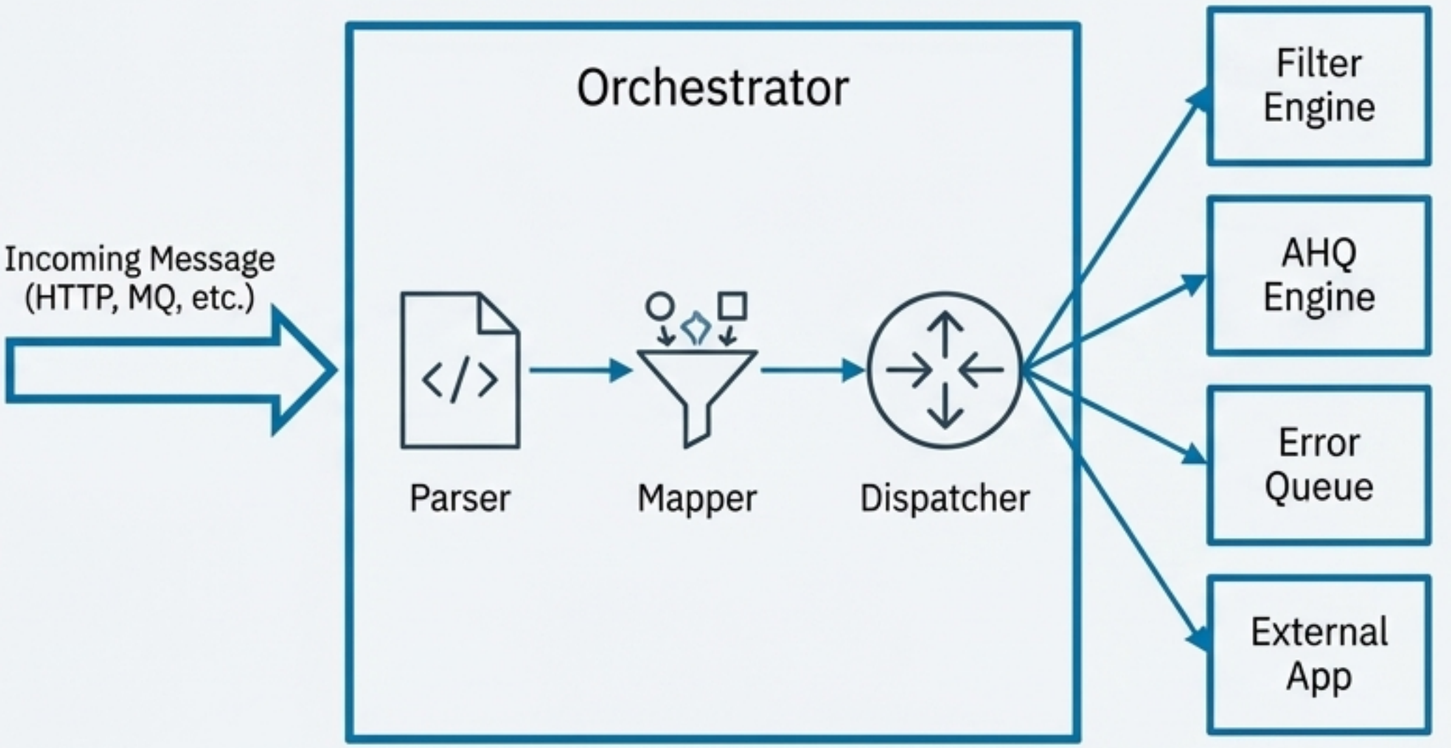
Firco Continuity is a complete, modular solution for transaction filtering. We will deconstruct this architecture, examining each component's role in building a resilient and intelligent screening process.



The Orchestrator is the System's Central Conductor

The **Orchestrator** is a Java-based component that replaces legacy acquisition and routing modules. It is responsible for parsing, routing, and orchestrating the flow of all messages through the system.

- **Core Purpose**: Manages the end-to-end processing flow of messages based on configurable rules.
- **Key Responsibilities**:
  - **Parsing & Mapping**: Consumes messages from various endpoints (HTTP, MQ, Directory) and transforms them into a standardized format.
  - **Routing & Dispatching**: Uses a powerful "Dispatcher" configured with Firco Common Language (FCL) to route messages to the correct processing engines.
  - **Error Management**: Handles exceptions and routes problematic messages to error queues for investigation.
  - **Endpoint Management**: Configures and manages all input (consumer) and output (producer) connection points.



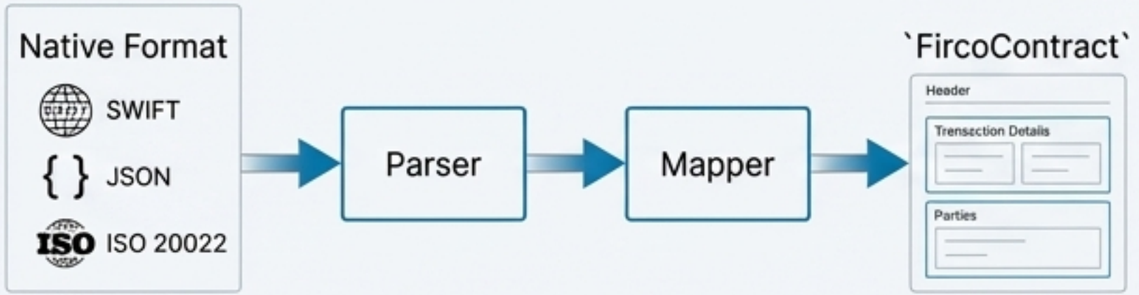
Universal Mapper and Requester Provide Ultimate Flexibility

Built into the Orchestrator, these components ensure the system can adapt to any data format and integrate with any screening or decisioning engine.

Universal Mapper

Purpose: Transforms incoming messages from their native format (e.g., SWIFT, ISO 20022, JSON) into the internal `FircoContract` format.

Key Feature: Can handle complex message structures, including ISO 20022 with Targeted Matching, ensuring all relevant data is available for screening.



Universal Requester

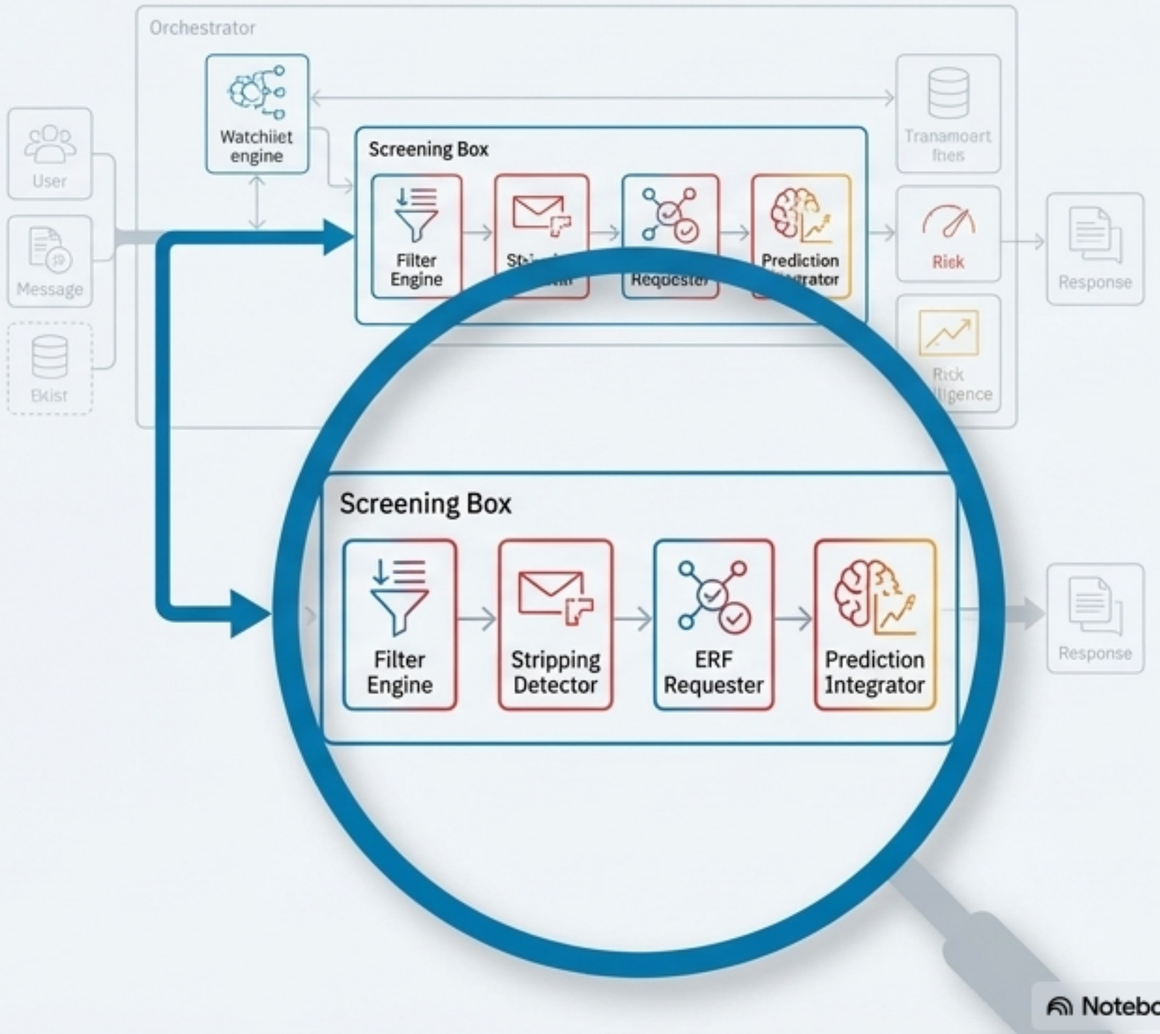
Purpose: Sends messages from the Orchestrator to various processing engines (like the Filter or AHQ) and receives the response.

Key feature: Manages communication, including resiliency (retries, timeouts) and endpoint configuration (HTTP, etc.), abstracting the connection logic from the main flow.

The Screening Box is the Primary Message Filtering Environment

The Screening Box is a specialized package built upon the Orchestrator, dedicated to the real-time filtering of incoming transactions against sanctions lists and internal rules.

- **Core Purpose**: To screen messages against one or more filter engines and detect potential illicit modifications.
- **Key Components**: It orchestrates a chain of specialized engines:
  - **Filter Engine**: The primary sanctions and watchlist screening engine.
  - **Stripping Detector**: Detects attempts to alter or omit information within a message.
  - **Entity Resolution Filter (ERF) Requester**: Integrates with external tools to reduce false positives.
  - **Prediction Integrator**: Connects to AI/ML models for predictive scoring.



Inside the Screening Box: The Filter and Stripping Engines

Two core engines work in tandem to perform comprehensive message analysis: the Filter Engine identifies potential matches, while the Stripping Detector looks for evidence of tampering.

Filter Engine

Purpose: Scans message content against configured watchlists (e.g., sanctions, PEPs) to identify potential matches or ‘hits’.

Functionality: Utilizes advanced matching algorithms and scoring to generate alerts. Can be configured in a ‘Multi-Filter” setup to run parallel screening processes with different configurations.

Related Jobs: The **`requester`** job sends messages to the Filter.

Stripping Detector

Purpose: Detects if a message has been intentionally altered to bypass screening controls (e.g., removing the name of a sanctioned entity).

Functionality: Compares incoming messages against historical patterns and rules to identify suspicious omissions or modifications. Generates a specific “Stripping alert” if tampering is suspected.

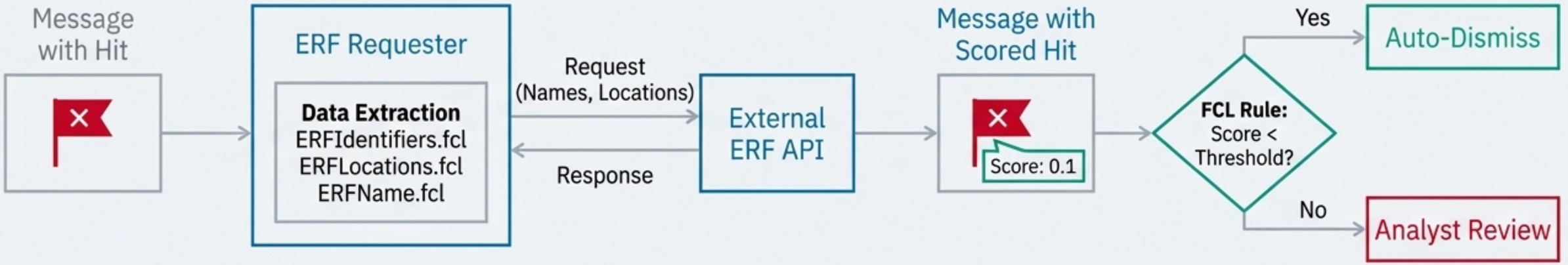
Related Jobs: **`stripping.rule.recorder`**, **`stripping.checker`**.

The Entity Resolution Filter Engine Reduces False Positives

The ERF Requester component sends message hits to an external Entity Resolution Filter (ERF) tool, which uses advanced analytics to score the likelihood of a true match, significantly reducing analyst workload.

- **Workflow:**
  1. The Screening Box generates a standard hit.
  2. If the hit is eligible, the ERF Requester packages relevant data (Names, Locations, Identifiers) from the message.
  3. The request is sent to the external ERF tool for scoring.
  4. The ERF response score is integrated back into the message. FCL rules can then be used to automatically dismiss low-score hits.

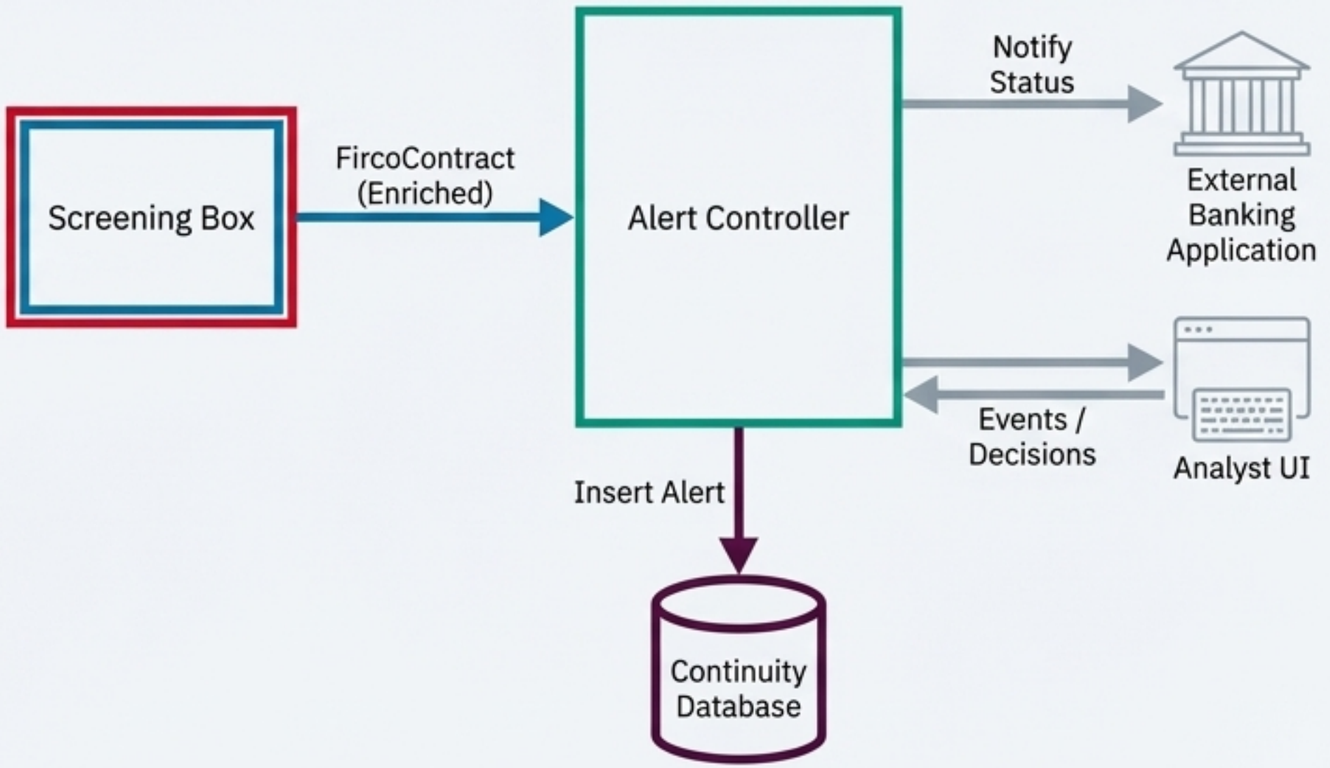
- **Data Extraction:** FCL scripts (`ERFIdentifiers.fcl`, `ERFLocations.fcl`, `ERFName.fcl`) intelligently extract structured and unstructured data from SWIFT MT and ISO 20022 messages for the ERF request.



The Alert Controller Manages Screening Results and Notifications

After the Screening Box completes its analysis, the Alert Controller takes charge of the results. It is responsible for database insertion, event management, and communication with downstream applications.

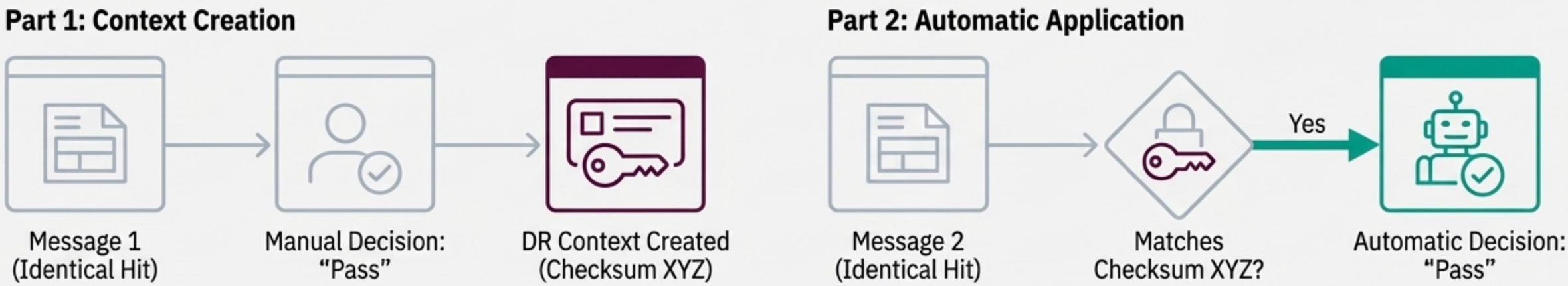
- **Core Purpose:** To process the output from the Screening Box, persist alerted messages, and notify banking applications of the final status.
- **Key Responsibilities:**
  - **Database Injection:** Inserts alerted messages and their full context into the Continuity database for review.
  - **Notification:** Formats and sends notifications (e.g., message status updates) to the original banking application using mappers like `WSJWMQOUT_V2`.
  - **Event Handling:** Receives events from the user interface (e.g., a decision made by an analyst) and orchestrates subsequent actions, such as re-screening (`recheck`).



Decision Reapplication Automates Repetitive Alert Handling

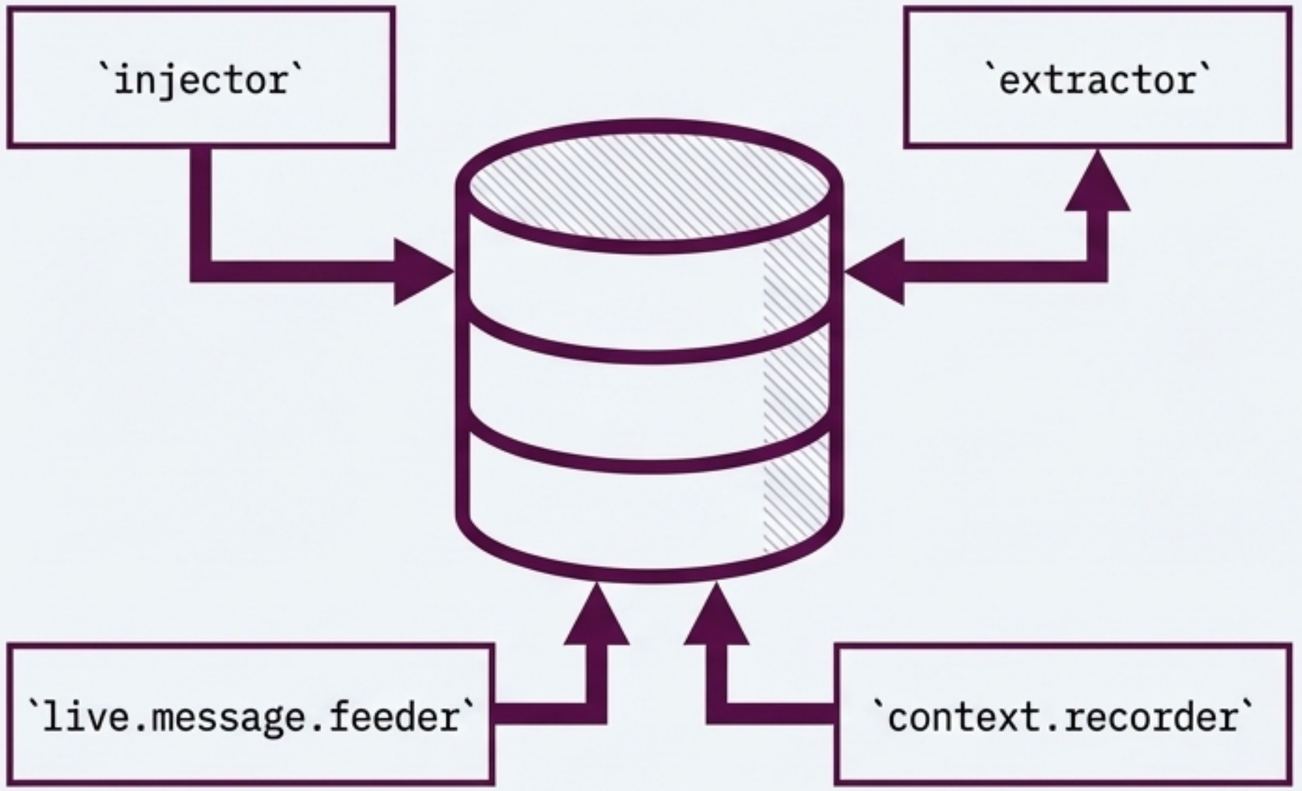
Decision Reapplication automatically reapplies a final decision made by an operator to subsequent, identical alerts. This eliminates the need to manually review the same recurring false positive hits.

- **How it Works**:
  1. A rule defines which messages are eligible for Decision Reapplication.
  2. For an eligible message, a unique **checksum** is calculated based on message fields and hit information (e.g., tag, score, entity ID).
  3. This checksum creates a **Decision Reapplication Context**.
  4. When an operator makes a final decision (e.g., 'Passed'), it is stored in the context.
  5. The next time a message arrives with the *exact same checksum*, the system automatically applies the stored decision.
- **Rule-Driven**: The entire process is controlled by highly configurable rules written in Core Engine language.
- **Related Jobs**: `rule.recorder` (loads rules), `decision.reapplication` (applies decisions).



The Database and DB Client Form the System's Core Record

The DB Client is a standard back-end package that provides a suite of tools to handle all communication with the primary LexisNexis® Firco™ Continuity database.



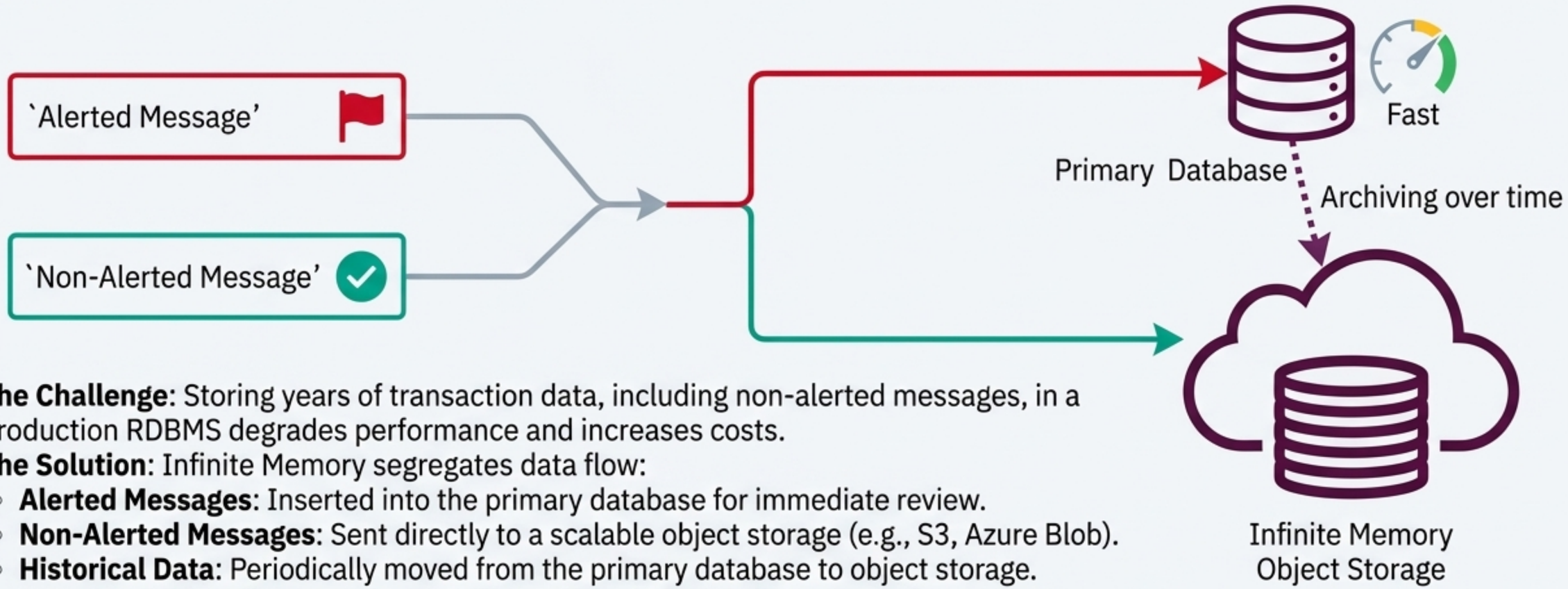
- **Core Purpose**: To manage the insertion, extraction, and reporting of all message and alert data within the relational database.
- **Key Jobs**:
  - `injector`: Inserts transactions, metadata, and hits into the database.
  - `extractor`: Notifies banking applications of decisions and recheck events.
  - `live.message.feeder`: Manages the flow of messages into the 'Live Messages' queue for analyst review.
  - `context.recorder`: Inserts the message's filtering context (rules, lists, settings used) into the database for full auditability.

Note: *The functions of `injector`, `extractor`, and `context.recorder` are increasingly handled by modern Alert Review and Decision Workflow APIs.*

Digital Architect's Blueprint

Infinite Memory Manages Data at Enterprise Scale

As data volumes grow, traditional databases become performance bottlenecks. Infinite Memory is a feature designed to manage vast amounts of data by intelligently routing it between the primary RDBMS and a scalable object storage solution.

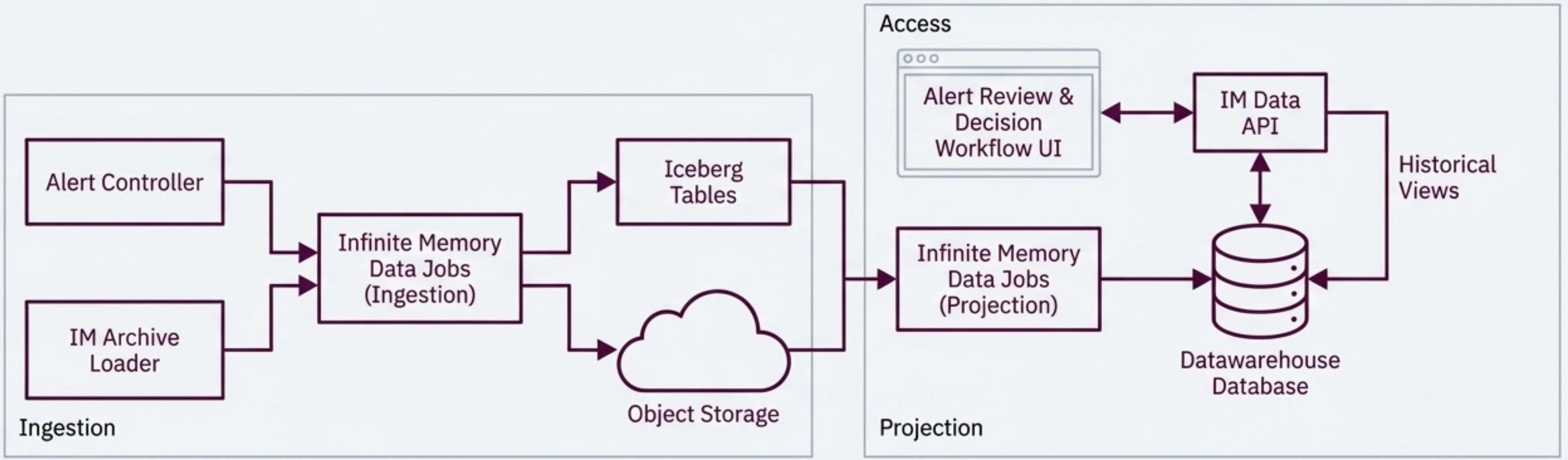


- **The Challenge**: Storing years of transaction data, including non-alerted messages, in a production RDBMS degrades performance and increases costs.
- **The Solution**: Infinite Memory segregates data flow:
  - **Alerted Messages**: Inserted into the primary database for immediate review.
  - **Non-Alerted Messages**: Sent directly to a scalable object storage (e.g., S3, Azure Blob).
  - **Historical Data**: Periodically moved from the primary database to object storage.
- **Benefit**: Keeps the primary database lean and fast, while providing access to a virtually infinite history of transactions for audit, reporting, and analytics.

Digital Architect's Blueprint

The Infinite Memory Architecture: Ingestion and Projection

Infinite Memory uses a two-step process to manage and provide access to data. Ingestion converts and stores raw data, while Projection makes it available for reporting and historical review.



****Key Table Structures****

The datawarehouse utilizes a star schema for optimized reporting, with key tables including:

- FKTA_D_MESSAGES (Message details)
- FKTA_F_HITS (Hit information)
- FKTA_F_ACTIONS (User and system events)
- FKTA_D_LISTED_RECORDS (Listed entity data)

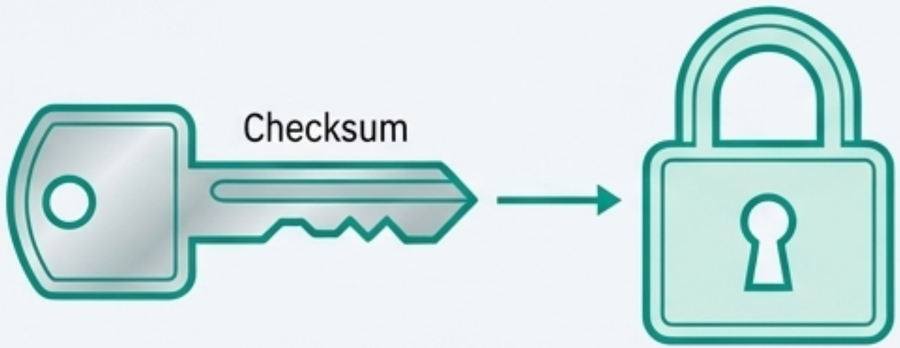
Digital Architect's Blueprint

Automated Hit Qualifier Learns from Operator Decisions

AHQ is a feature that automatically resolves recurring false positive hits based on the historical qualification patterns of human operators. It moves beyond exact checksums to recognize nuanced patterns.

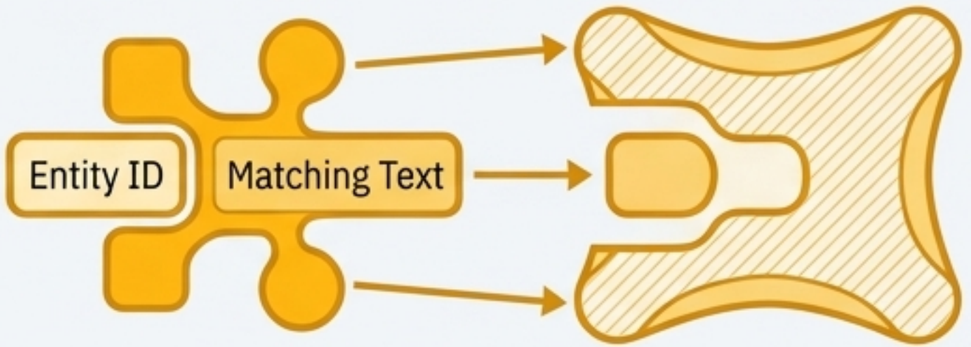
How AHQ differs from Decision Reapplication

Decision Reapplication



Requires an *exact* checksum match on the message and all hits. It is rigid but very safe.

Automated Hit Qualifier



Creates a “context” based on a flexible **signature** defined by rules (e.g., just the entity ID and matching text). It learns from multiple operator decisions on hits matching this signature to automate future qualifications.

Key Concepts

- **Context Lifecycle**: Contexts move from `CANDIDATE` to `ENABLED` after meeting thresholds (e.g., qualified as false by 2 different operators 5 times).
- **Real-Time Disposition**: When a context is enabled, AHQ can proactively find and qualify other pending hits that match the same context.

AHQ Architecture Enables High-Speed, In-Memory Qualification

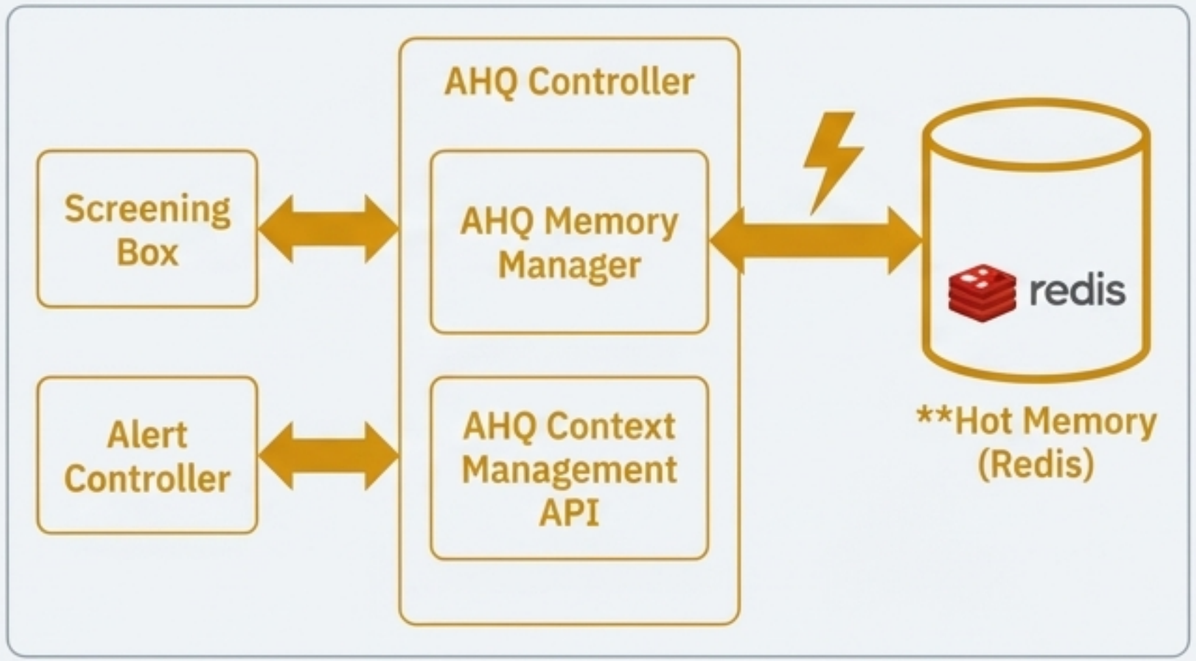
AHQ relies on a dedicated controller package with an in-memory database (Redis) to achieve the high performance required for real-time payments and large-scale learning.

Architectural Components

- **AHQ Controller**: An Orchestrator-based package that manages the AHQ flows.
- **AHQ Memory Manager**: The core engine that handles context creation, counter increments, and events. It connects to the 'Hot Memory' (Redis).
- **AHQ Context Management API**: Handles administrative operations on contexts from the user interface.
- **AHQ Requester**: An engine in the Screening Box and Alert Controller that sends events to the AHQ Controller.

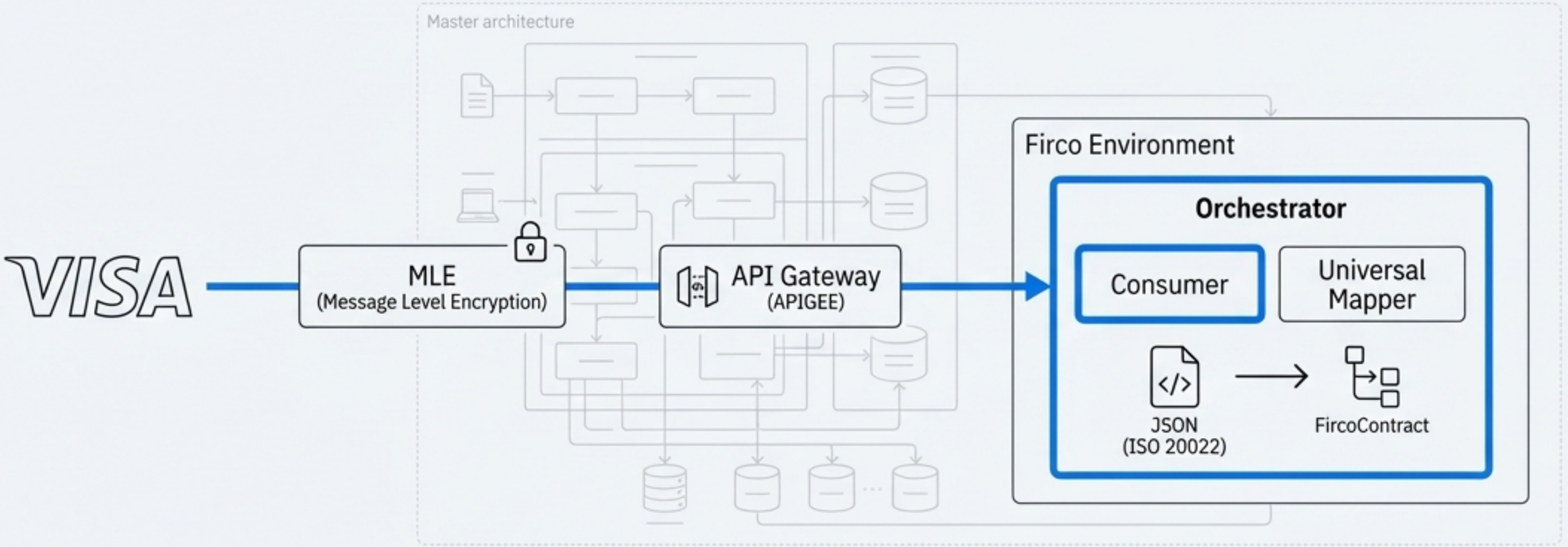
Operating Modes

- **Production**: Live creation of contexts and automated qualification.
- **Learning**: Ingests historical message archives to learn from past decisions and pre-build the context memory.
- **Simulation**: A read-only mode used for impact assessment without affecting production data.



Digital Architect's Blueprint

A Transaction's Journey Begins: Secure Ingestion and Preparation



Step 1: Ingestion

- A financial message (e.g., a VISA transaction) is initiated by an upstream system.
- It is securely transmitted, often with Message Level Encryption (MLE).
- An API Gateway (like APIGEE) manages the inbound traffic and authenticates the request.
- The request is forwarded to a specific Orchestrator endpoint (e.g., HTTP).

Step 2: Parsing and Mapping

- The Orchestrator's **Consumer** receives the message.
- The **Universal Mapper** parses the message from its native format (e.g., JSON representation of an ISO 20022 message).
- It is transformed into the internal, standardized **FircoContract** format. This canonical object contains all message data, metadata, and will be enriched at each subsequent step.

The Journey Continues: Analysis within the Screening Box

Digital Architect's Blueprint

Step 3: Dispatching

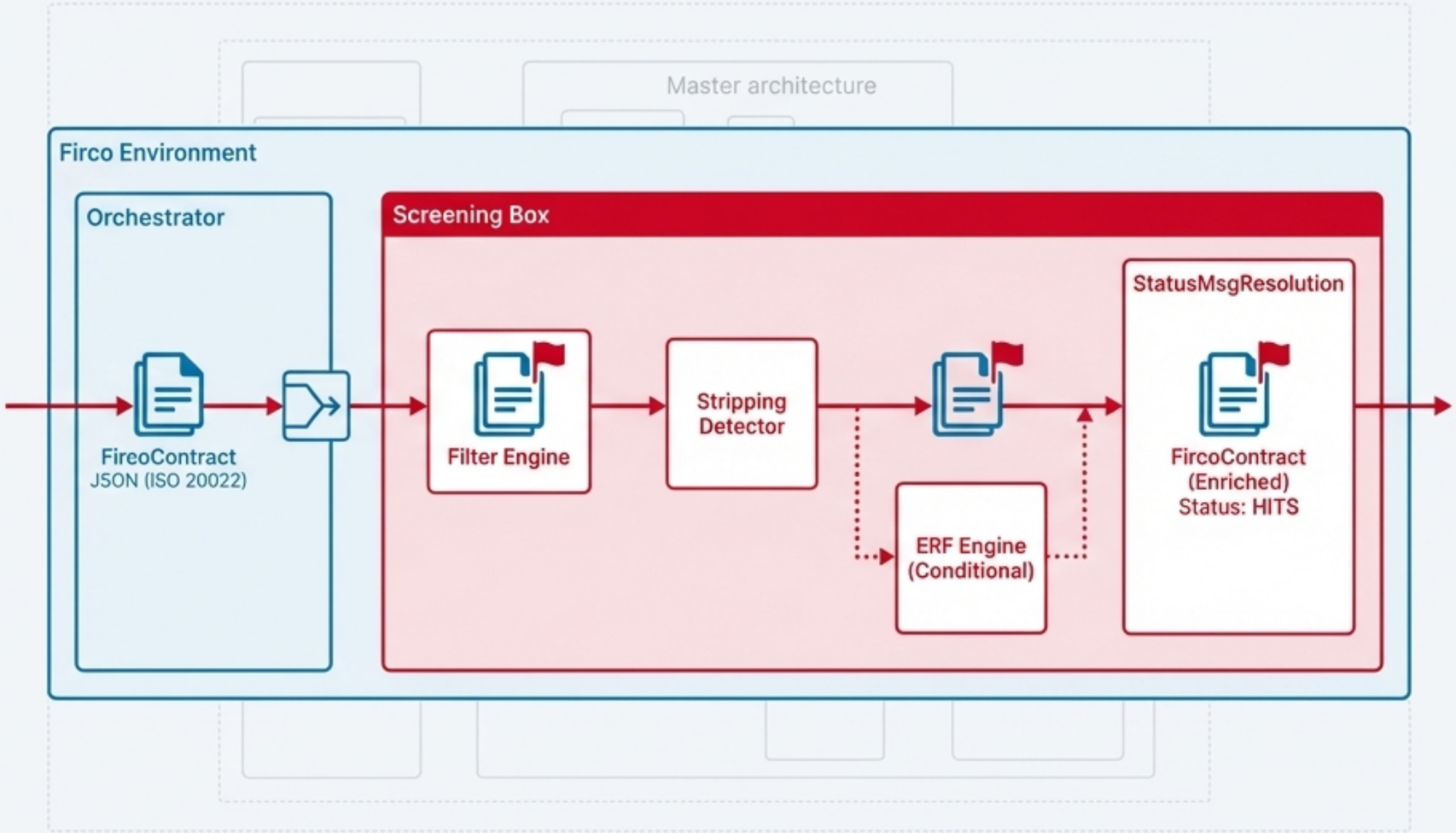
- The Orchestrator's **Dispatcher** evaluates its FCL rules and routes the `FircoContract` to the main screening flow.

Step 4: Multi-Engine Screening

- **Filter Engine**: The message is screened against watchlists. A potential **hit** is identified and appended to the `FircoContract`.
- **Stripping Detector**: The message is analyzed for potential tampering. The result is added to the `FircoContract`.
- **ERF Engine (Conditional)**: If the hit is eligible, the ERF Requester sends it for external scoring. The score is returned and added to the hit data within the `FircoContract`.

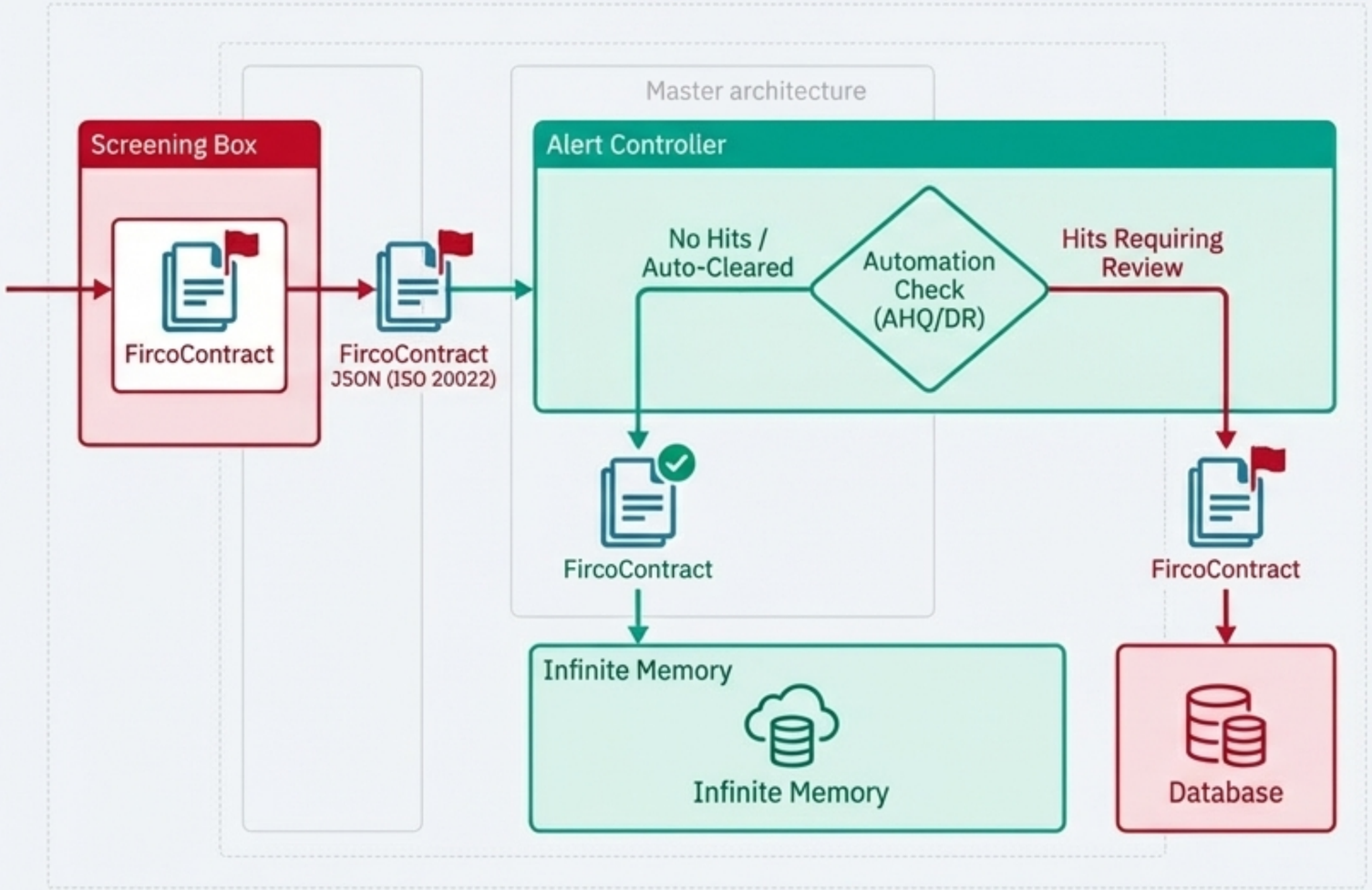
Step 5: Status Resolution

- A final FCL function, `StatusMsgResolution`, calculates the initial status of the message based on the results (e.g., `HITS`, `NON_BLOCKING`). The `FircoContract` is now fully enriched with all screening results.



A Fork in the Road: Decisioning and Intelligent Automation

Digital Architect's Blueprint



Step 6: Routing to Alert Controller

- The Screening Box sends the fully enriched `FircoContract` to the Alert Controller's input endpoint.

Step 7: Automated Decisioning Check

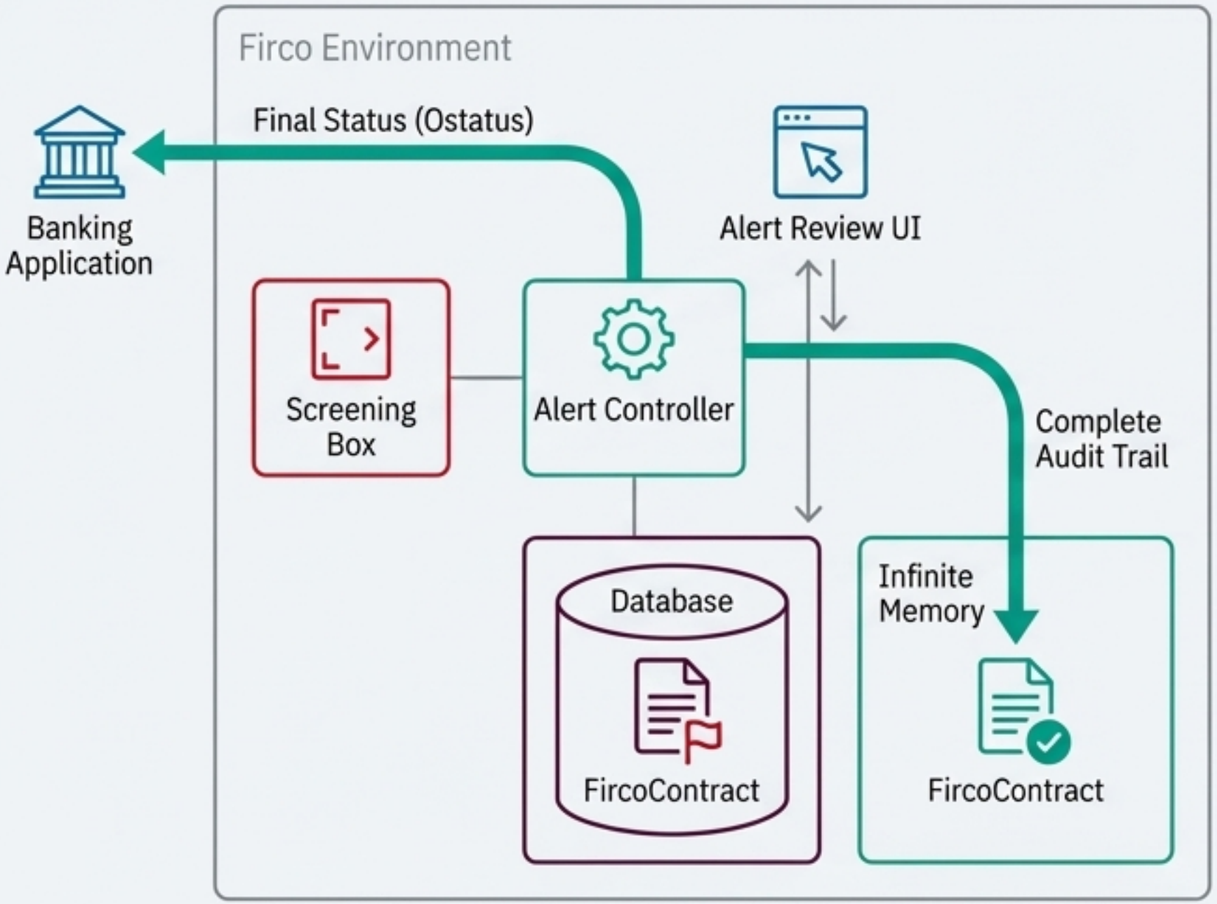
- The Alert Controller's dispatcher checks for automation opportunities before human review:
  - **Decision Reapplication:** Does the message's checksum match an existing `ENABLED` context? If yes, the stored decision is applied.
  - **Automated Hit Qualifier**: Does the hit match an `ENABLED` AHQ context? If yes, the hit is automatically qualified (e.g., as a false positive).

Step 8: Persistence and Routing

- **Path A (No Hits / Auto-Cleared):** If the message has no hits, or all hits were automatically cleared by AHQ/DR, the Alert Controller routes it directly to **Infinite Memory** for long-term storage. The process ends here.
- **Path B (Hits Requiring Review):** If the message has unresolved hits, the Alert Controller uses the **DB Client injector** (or API equivalent) to insert the message and its hits into the primary **database**.

The Final Step: Human Review and Final Disposition

Digital Architect’s Blueprint



Step 9: Analyst Review

- The message appears in the "Live Messages" queue within the Alert Review and Decision Workflow UI.
- An analyst reviews the message details, hit information, and any automated context (from AHQ, DR, etc.).

Step 10: Final Decision

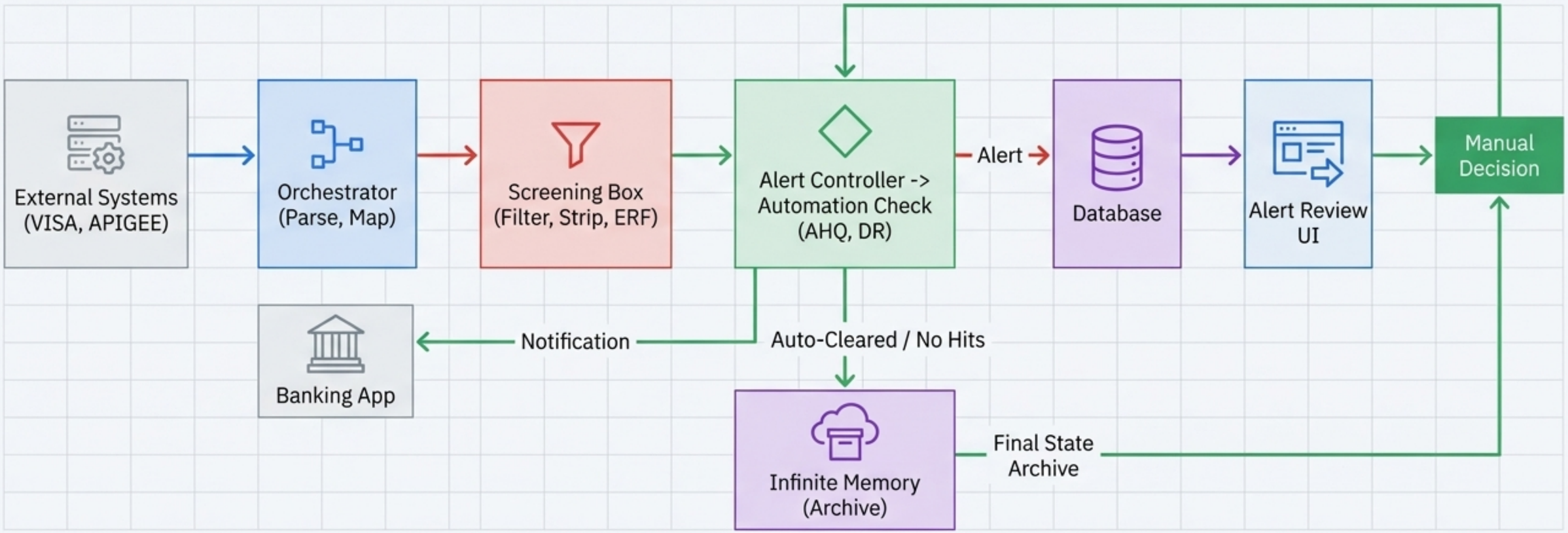
- The analyst makes a final decision, for example, qualifying the hit as a false positive and setting the message state to "Passed". This action creates a new event.

Step 11: Notification and Archiving

- The decision event is processed by the Alert Controller.
- An "extractor" job formats a notification and sends the final status (Ostatus) back to the originating banking application.
- The final message state and all associated events are propagated to Infinite Memory, ensuring a complete, permanent audit trail. The message is now fully resolved.

The Complete End-to-End Transaction Flow

From secure ingestion to intelligent analysis and final disposition, the Firco Continuity platform provides a robust, auditable, and efficient workflow for every transaction.



An Integrated, Intelligent Screening Ecosystem

The Firco Continuity architecture is more than a collection of components; it is a fully integrated ecosystem designed for performance, scalability, and intelligence.