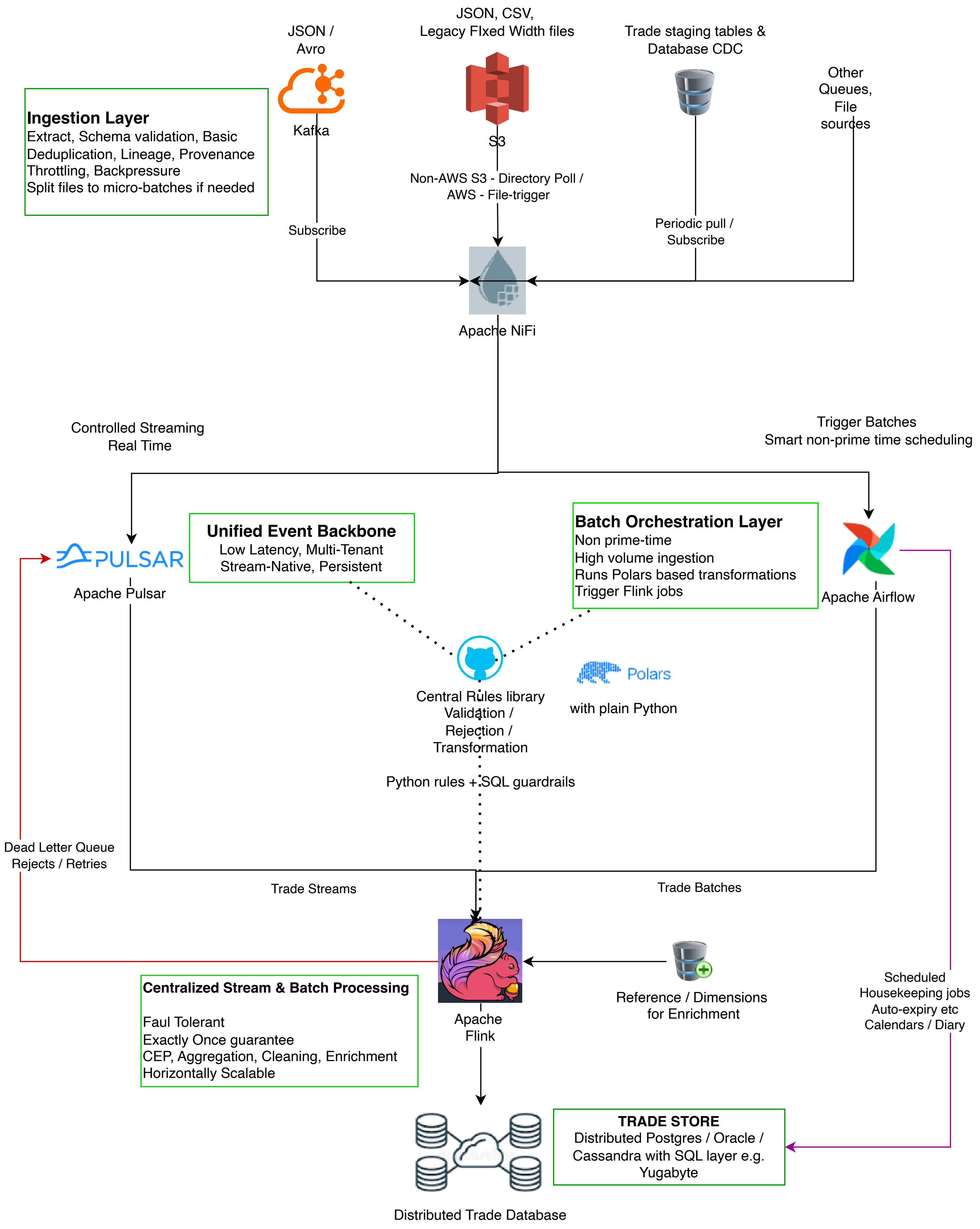
TRADE INGESTION PIPELINE



Ingestion Layer
Extract, Schema validation, Basic
Deduplication, Lineage, Provenance
Throttling, Backpressure
Split files to micro-batches if needed

Unified Event Backbone
Low Latency, Multi-Tenant
Stream-Native, Persistent

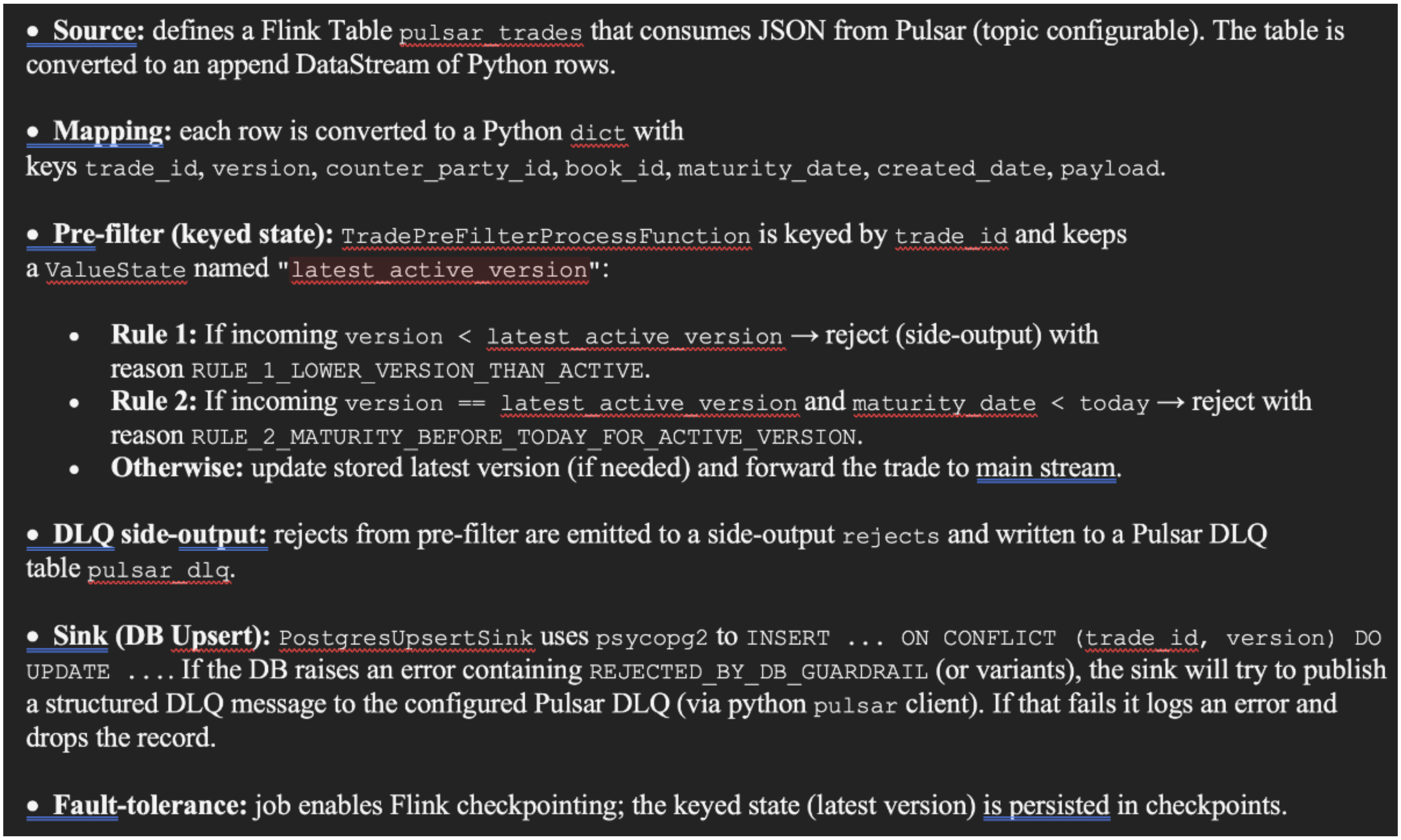
Batch Orchestration Layer
Non prime-time
High volume ingestion
Runs Polars based transformations
Trigger Flink jobs

Centralized Stream & Batch Processing

Faul Tolerant
Exactly Once guarantee
CEP, Aggregation, Cleaning, Enrichment
Horizontally Scalable

TRADE STORE
Distributed Postgres / Oracle /
Cassandra with SQL layer e.g.
Yugabyte

Flink Job Flow



- **Source:** defines a Flink Table `pulsar_trades` that consumes JSON from Pulsar (topic configurable). The table is converted to an append DataStream of Python rows.

- **Mapping:** each row is converted to a Python `dict` with keys `trade_id`, `version`, `counter_party_id`, `book_id`, `maturity_date`, `created_date`, `payload`.

- **Pre-filter (keyed state):** `TradePreFilterProcessFunction` is keyed by `trade_id` and keeps a `ValueState` named `"latest_active_version"`:

  - **Rule 1:** If incoming `version < latest_active_version` → reject (side-output) with reason `RULE_1_LOWER_VERSION_THAN_ACTIVE`.
  - **Rule 2:** If incoming `version == latest_active_version` and `maturity_date < today` → reject with reason `RULE_2_MATURITY_BEFORE_TODAY_FOR_ACTIVE_VERSION`.
  - **Otherwise:** update stored latest version (if needed) and forward the trade to main stream.

- **DLQ side-output:** rejects from pre-filter are emitted to a side-output `rejects` and written to a Pulsar DLQ table `pulsar_dlq`.

- **Sink (DB Upsert):** `PostgresUpsertSink` uses `psycopg2` to `INSERT ... ON CONFLICT (trade_id, version) DO UPDATE ...`. If the DB raises an error containing `REJECTED_BY_DB_GUARDRAIL` (or variants), the sink will try to publish a structured DLQ message to the configured Pulsar DLQ (via python `pulsar` client). If that fails it logs an error and drops the record.

- **Fault-tolerance:** job enables Flink checkpointing; the keyed state (latest version) is persisted in checkpoints.