ENGINEERING ATHENA: BUILDING A SCALABLE, RESILIENT, AND COMPLIANT FINANCIAL PLATFORM

Aroma Rodrigues

TRADING

01 Trades

Trade1 => Trader1 bought 100 units of Bond1 from Trader2 on the 5th of August 2025 at 3:00 pm GMT in the regions governed by the legal entities JPMCNA and to be cleared by a clearing house

A trade is a buy or sell transaction in it's atomicity. It usually tells you which parties were involved in the buying and selling of which instruments in which state.

02 Deals

A deal represents the state of the trade according to an actor in that particular trade.

For instance for Trade1, we have a version of the trade that lives in the Book as held by Trader1, Trader2 and sometimes the clearing house.

Trader 1 in his books holds 100 units of Instrument1 and Trader2 in his books holds the price as paid on the day the trade was executed

Both deals are mirrors of each other. This is in the simplest case. Sometimes there is chaining of trades based on how many actors are involved in the trade

Trader2 keeps track of his deal to compute her personal profit and loss

03 Books

Books are usually associated with Traders

A book is an accountant's book - a collection of credits and debits

04 PnL

Based on the national price of the instrument in a moving market where prices change every day a book has its own profit and loss calculations that help the traders strategize their next move

MARKET ANALYSIS

Trade

A trade is a single executed transaction of a financial instrument.

It represents the actual buying or selling event, capturing what happened in the market.

Key Attributes in Athena:

Instrument: e.g., bond, CDS, derivative

Quantity / Notional: size of the trade

Price / Premium: executed price

Counterparties: buyer and seller

Timestamp / Trade Date: when the trade occurred

Status: e.g., pending, settled, corrected, defaulted

Lifecycle in Athena:

Created: when the transaction is executed.

Updated (rarely): to correct errors or mark status changes.

Recorded: always exists in the books, forming the basis for positions, risk, and P&L.

ARCHITECTURE

1. Front Office (FO) – Trade Capture
Source: Traders, Sales, or Electronic Trading platforms.
What happens: A trade (or deal) is entered into the system.
Stored in: A Trade Capture System (could be Athena in your case).

2. Middle Office (MO) – Risk & Control
What happens:
Validate the trade (legal, compliance, credit risk checks).
Assign the trade to a Book (e.g., Corporate Bonds, CDS Book).
Trades get enriched (pricing, risk sensitivities).
Output: Clean, risk-approved trades.

3. Back Office (BO) – Settlement
What happens:
Generate settlement instructions
(payment flows, bond delivery, CDS premium payments).
Reconcile with counterparty.
Corporate actions (maturity, credit events, auctions).

4. Books & PnL
Book = a bucket grouping trades (by desk, trader, or product).
PnL calculated on Book(s):
Daily mark-to-market valuation.
Realized + unrealized gains/losses.
Risk measures (VaR, Greeks, credit exposure).

5. Reference Data & Market Data
Needed across all layers:
Instruments (bonds, CDS definitions,
maturity dates, coupon schedule, recovery rates).
Market Data (prices, credit spreads, interest rates, FX rates).

```
        [ Legacy Feeds ]  [ E-Trade Feed ]  [ CSV Uploads ]
                |               |                 |
                v               v                 v
            (HTTP Polling / File Drops on NFS)
                          |
                          v
                +-----------------------+
                |  Ingestion Service    |
                |  - single instance    |
                |  - sync parsing       |
                +-----------+-----------+
                            |
                            v  (blocking writes)
                +----------------------------+
                |  Single RDBMS (Primary)    |
                |  - one region              |
                |  - no replicas             |
                |  - mixed OLTP + OLAP       |
                +-----------+----------------+
                            |
            +-------------------------+-------------------------+
            |                         |                        |
            v                         v                        v
+----------------------+  +------------------------+  +-----------------------+
| Risk Compute Svc     |  | Reporting Dashboard    |  | Compliance Logger     |
| - sync RPC calls     |  | - ad-hoc SQL queries   |  | - writes audit rows   |
| - runs on app node   |  | - same DB              |  | - same DB             |
| - single worker      |  +------------------------+  +-----------------------+
+----------+-----------+
           |
           v  (nightly batch via cron)
+-----------------------+
| Batch Recalc Job      |
| - exports CSVs        |
| - backfills metrics   |
+-----------------------+
```

```
        [ Legacy Feeds ]  [ E-Trade Feed ]  [ CSV Uploads ]
               |               |                  |
               v               v                  v
          (HTTP Polling / File Drops on NFS)
                          |
                          v
                +-----------------------+
                |  Ingestion Service    |
                |  - single instance    |
                |  - sync parsing       |
                +-----------+-----------+
                            |
                            v  (blocking writes)
                +----------------------------+
                |  Single RDBMS (Primary)    |
                |  - one region              |
                |  - no replicas             |
                |  - mixed OLTP + OLAP       |
                +-------------+--------------+
                              |
      +-----------------------+-----------------------+
      |                         |                       |
      v                         v                       v
+----------------------+  +------------------------+  +------------------------+
| Risk Compute Svc     |  | Reporting Dashboard    |  | Compliance Logger      |
| - sync RPC calls     |  | - ad-hoc SQL queries   |  | - writes audit rows    |
| - runs on app node   |  | - same DB              |  | - same DB              |
| - single worker      |  +------------------------+  +------------------------+
+----------+-----------+
           |
           v  (nightly batch via cron)
+----------------------+
| Batch Recalc Job     |
| - exports CSVs       |
| - backfills metrics  |
+----------------------+
```

Core traits (why it won't scale)

Single everything: one ingestion instance, one app node, one primary DB (no replicas, no sharding). Only vertical scaling possible.

Synchronous coupling: every call is blocking; compute and reporting sit directly on the OLTP tables.

Mixed workloads on one DB: ingestion writes, risk analytics, dashboards, and compliance all hammer the same tables/indices.

Batch mentality: heavy "recalc" runs nightly via cron instead of streaming/incremental computation.

No backpressure/queuing: spikes in feed volume stall ingestion threads and cascade to users.

Single region: higher latency for global users; any regional outage = platform outage.

Ad-hoc compliance: audit is just extra rows in the same DB; noisy neighbors + easy to break invariants.

Schema rigidity: no event versioning; schema changes require coordinated downtime.

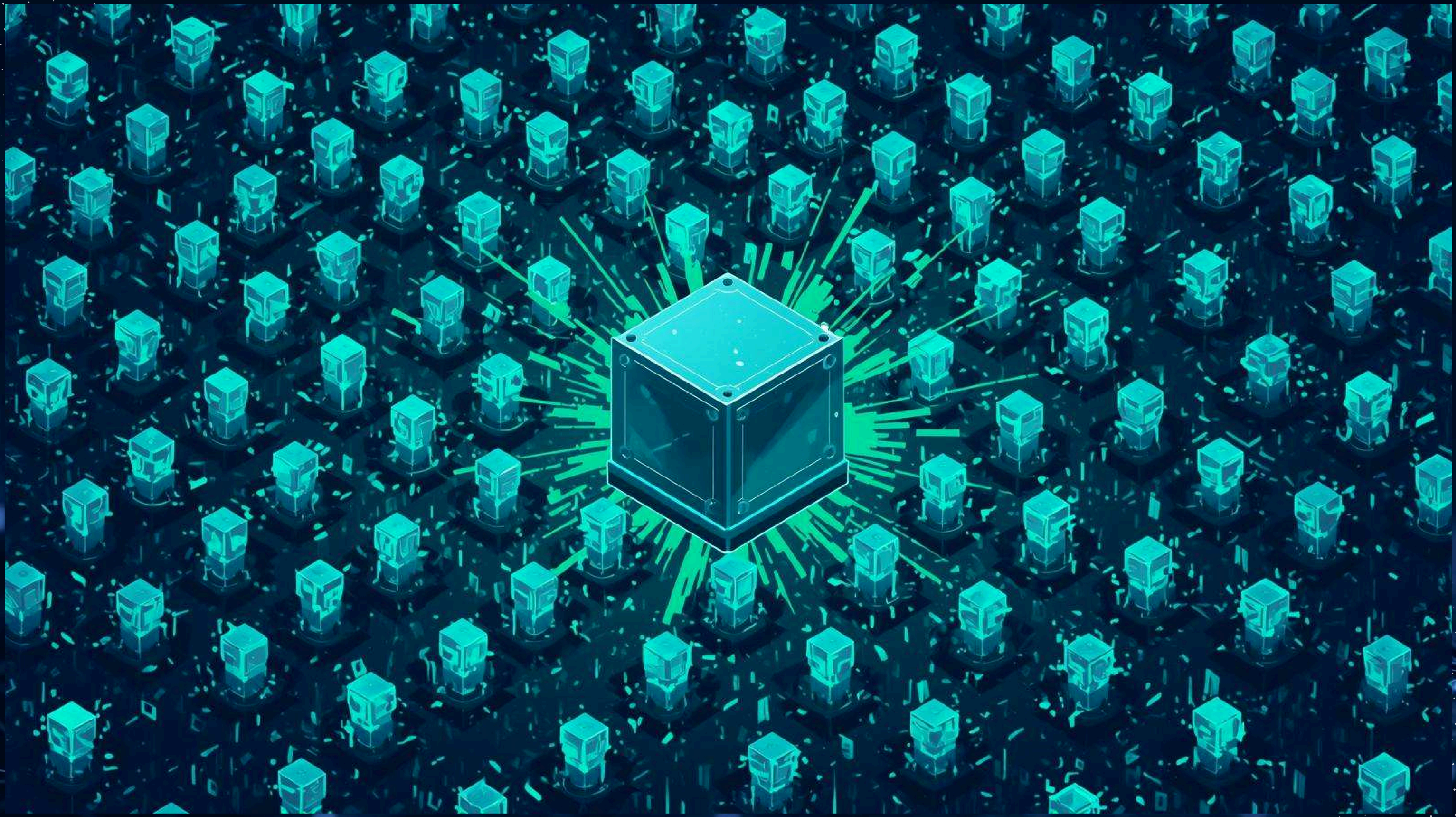
ADVENTURE TIME

SCALE OF MILLIONS

SCALE OF MILLIONS

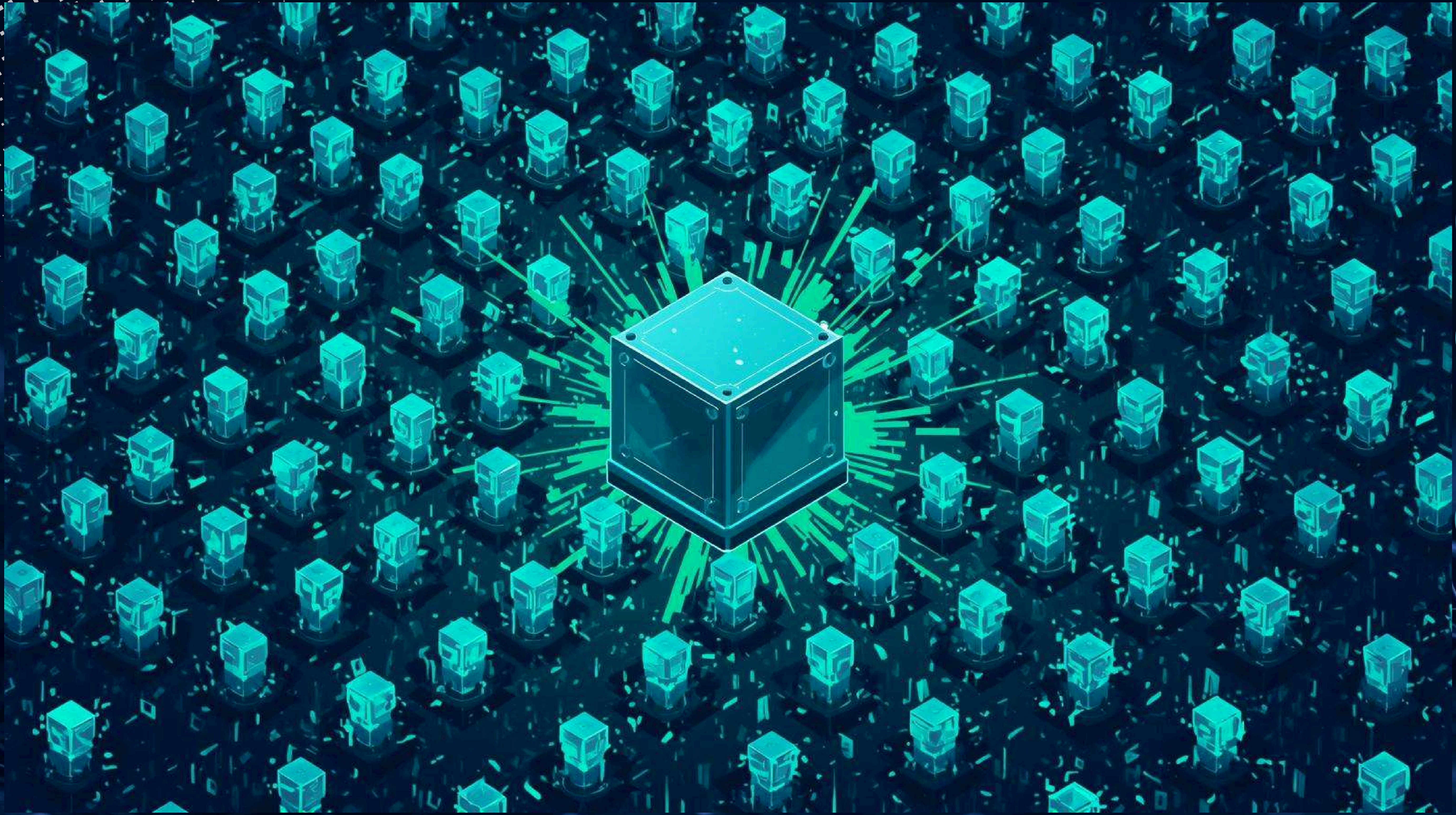
What your ingesting service feels like

SCALE OF MILLIONS



What your data looks like

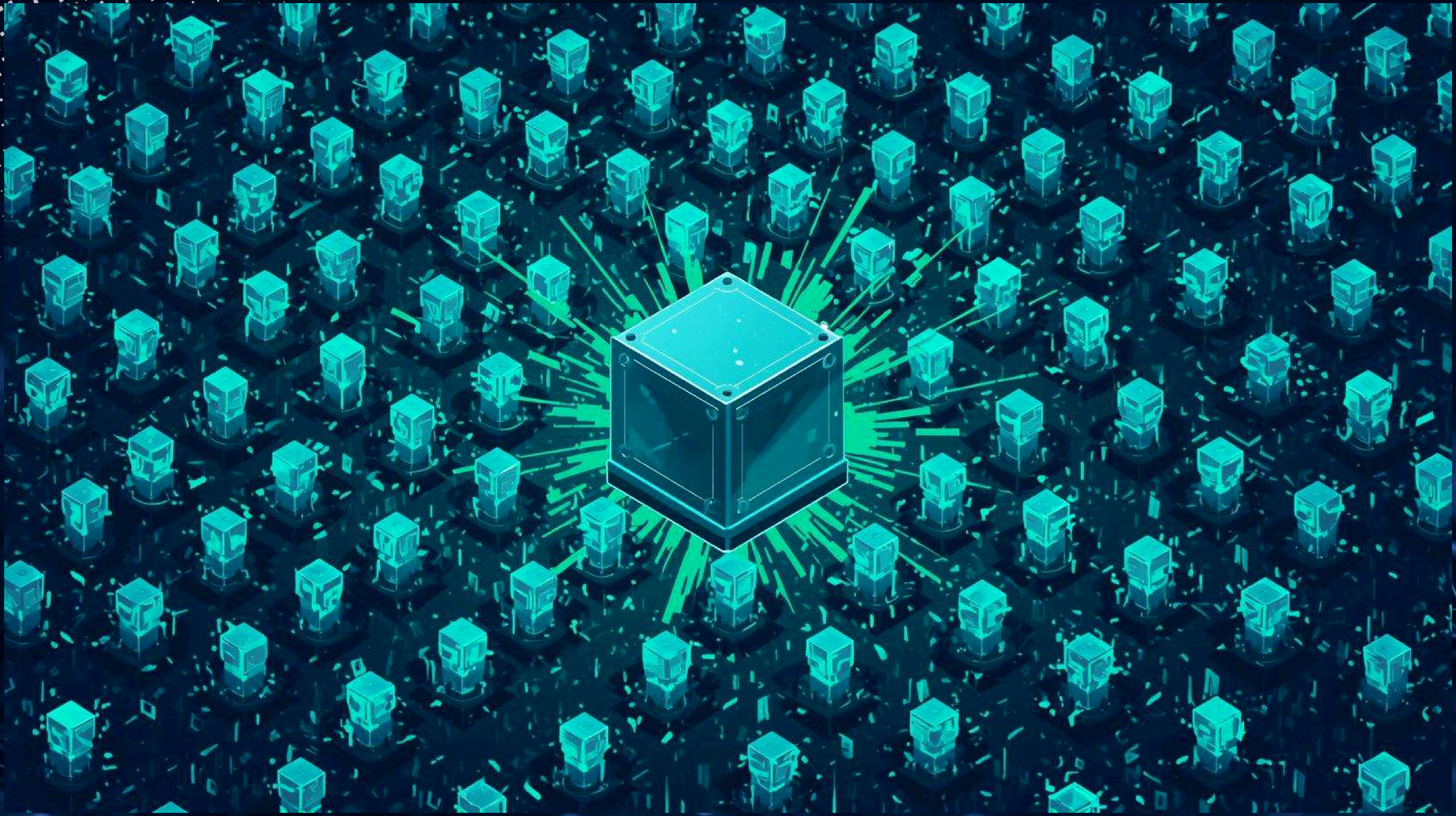
SCALE OF MILLIONS



Deal - 1

Day	
1	Instl, Price:100, 100
2	Instl, Price:100, 300
3	
n	Instl, Price:100, 500

SCALE OF MILLIONS



PnL needs to be calculated daily
Actually every minute sometimes
Indexing trades and only sourcing those from date indexed as last end of day computation will work

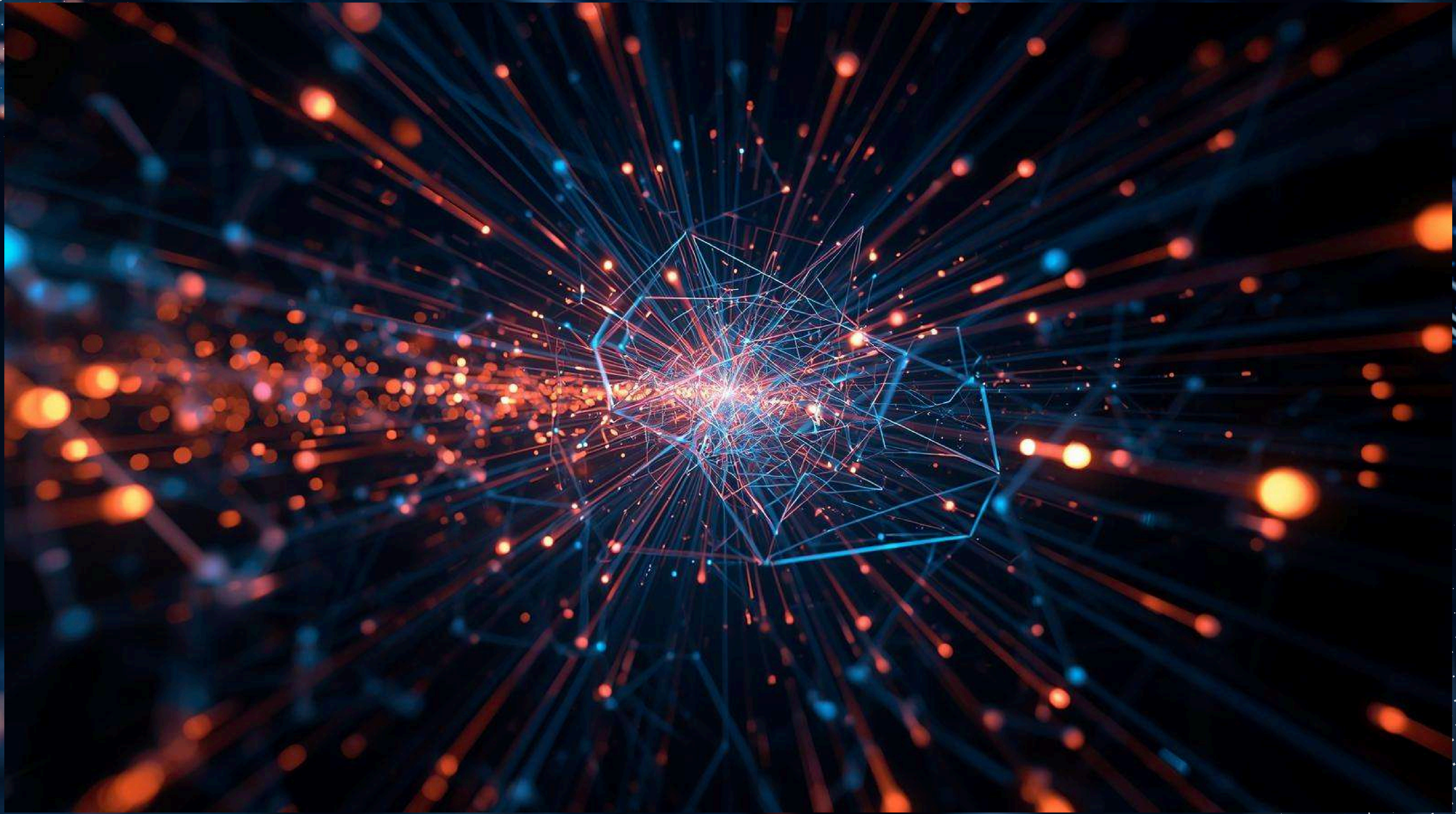
Redundancy of previous records can be addressed by RDBMS calls based on dates

Normalized table of changes cannot be guaranteed to function well because of what factors in deals change.

Your regime can change - Legal entity change, instrument can change, pricing changes

In this case data will always be redeundant

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY



HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

Create Dependency Graph data structures that trigger updates automatically when @dep marked parameters change

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

CreditEventl->Instrumentl->All references of Instrumentl change->If changes on Instl,
PnL recalculated based on @dep markers when loading or if called.
Data structure event listening
Creates an index listener on Instrument ID and looks for changes if marked @dep
Because of this structure and varying class data - called using
python get attr→ hydra is object oriented

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

What if the differences are not structured enough to waste a whole RDBMS row

Save only the differences

Use previous state to capture today’s state, mechanism to time travel

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

HOW TO MAINTAIN DATA THAT CHANGES FREQUENTLY

DATA AVAILABILITY AND REPLICATION

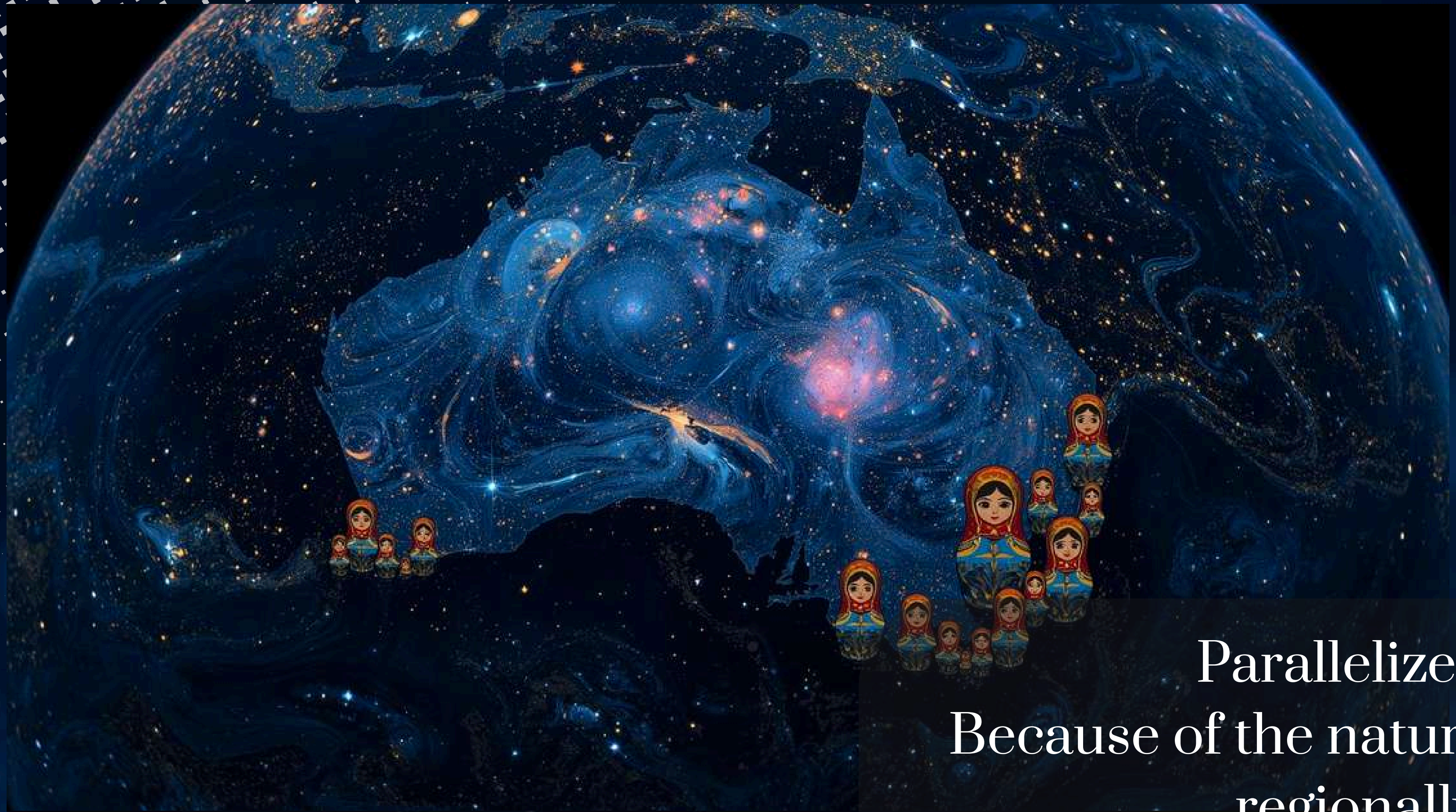
Replication is across geographies because data loss events are geographically connected
Power outages, natural diasters etc
Athena runs sync jobs in its database called hydra to follow the CAP principle

DATA AVAILABILITY AND REPLICATION

Main DB instances are usually located near trading hubs - NYC, LDN
Main instances are replicated upto 3X in the same region to improve I/O bottlenecks
thus requiring both local and global syncs

COMPUTE AT SCALE

COMPUTE AT SCALE



Parallelize as much as possible
Because of the nature of markets, trading is usually regionally legal entity based
Athena maintains a different instance for cross LE trades
Often uses clever techniques to add a third leg to the trade to be able to compute efficiently making for some very clever corner cases

END OF DAY CALCULATIONS

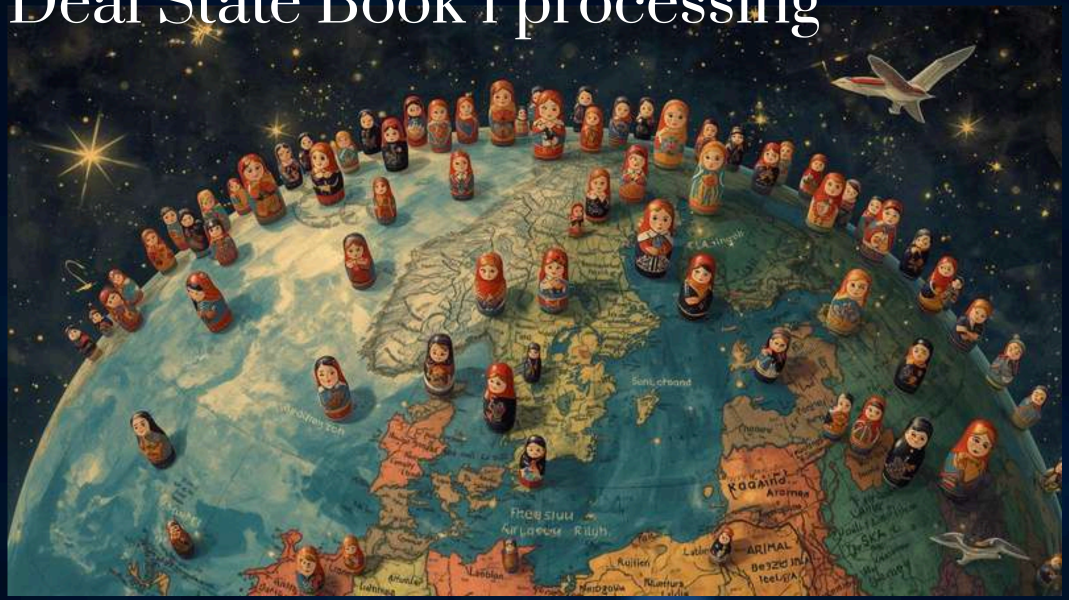
These batches run in parallel and often can share resources
Each batch has dependent jobs
Services run asynchronously to prevent single points of failures and tight couplings
This means that if job 5 fails, the state returned from a job it is dependent upon, say job 4 can be reused to recompute job 5 if the data/results in job 4 were not the issue and doesn't need recomputation

END OF DAY CALCULATIONS

Instrument A ref data processing

Instrument B processing

Deal State Book 1 processing



Deal State Book 2 processing

EOD PnL compilation

Break down processes into dependent steps and chunk/shard data into independent processing units

For a faster compute time and async states

For instance both Book 1 and 2 may have positions for Instrument B - so dependency and cron jobs

For the last step - integrate chunks - using MapReduce

Use recon mechanisms to reduce errors

END OF DAY CALCULATIONS

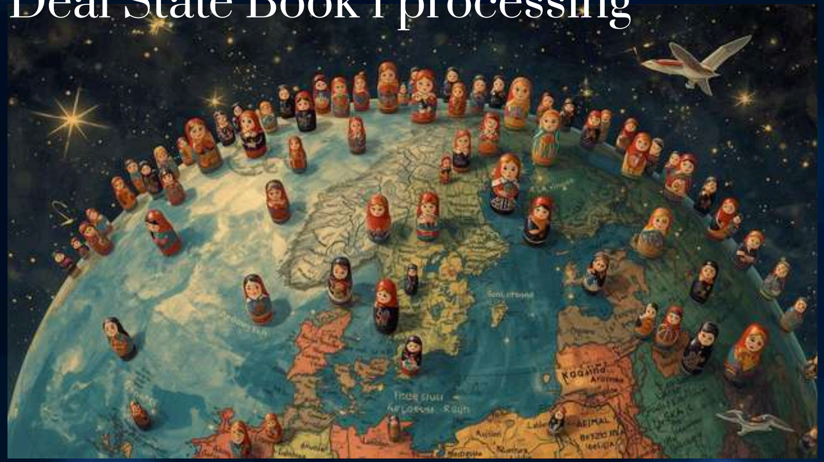
For every computation, in order to optimize for in memory computations and more parallelism
every operation is chunked into independent computable chunks - which form a Lambda like function
which executes upon its own CBB node and the results are collated
For instance instead of pulling refs from Instl to compute metrics on the server itself, another compute block
is shown a functional hydra call to load data - so it loads in memory without affecting the compute of the main server
Each doll gets their own playfield to unravel and be put together

END OF DAY CALCULATIONS

Instrument A ref data processing

Instrument B processing

Deal State Book 1 processing



Deal State Book 2 processing

EOD PnL compilation

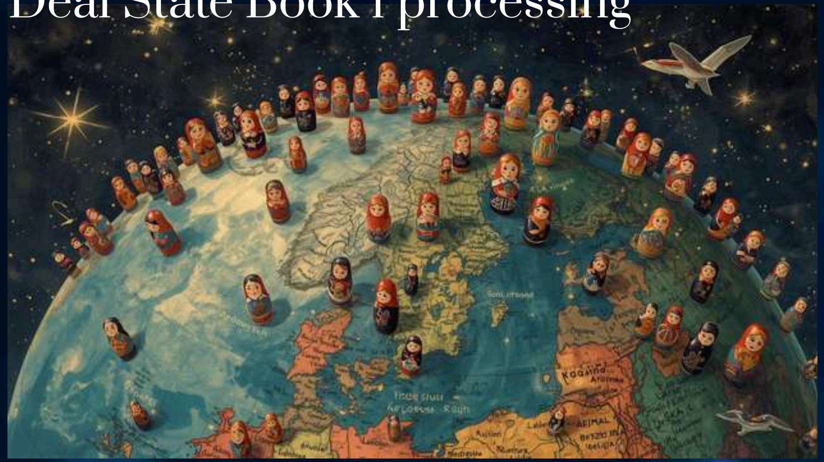
Compliance Feeds

END OF DAY CALCULATIONS

Instrument A ref data processing

Instrument B processing

Deal State Book 1 processing



Deal State Book 2 processing

EOD PnL compilation

Compliance Feeds

Importance of Compliance Feeds

BACK TO EARTH

WHY PYTHON?

PYTHON'S DUCK TYPING AND DYNAMIC TYPING ALLOW REPRESENTING VARIED INSTRUMENTS WITHOUT RIGID SCHEMAS.

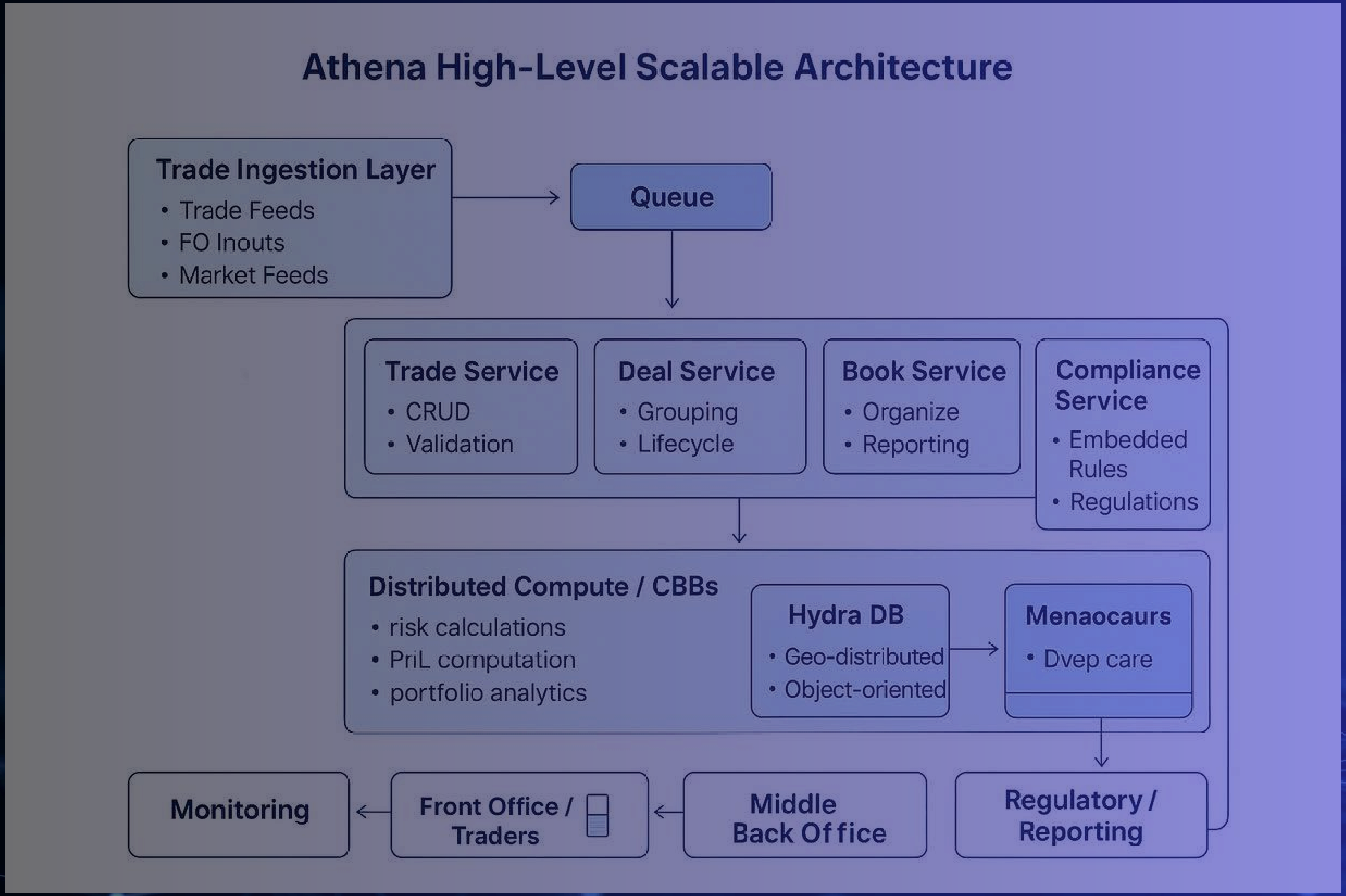
PYTHON'S DICTIONARY STRUCTURE MAKES IT EASY TO USE INDICES IN HYDRA

PYTHON INTEGRATES WELL WITH C++ CBBS

ATHENA

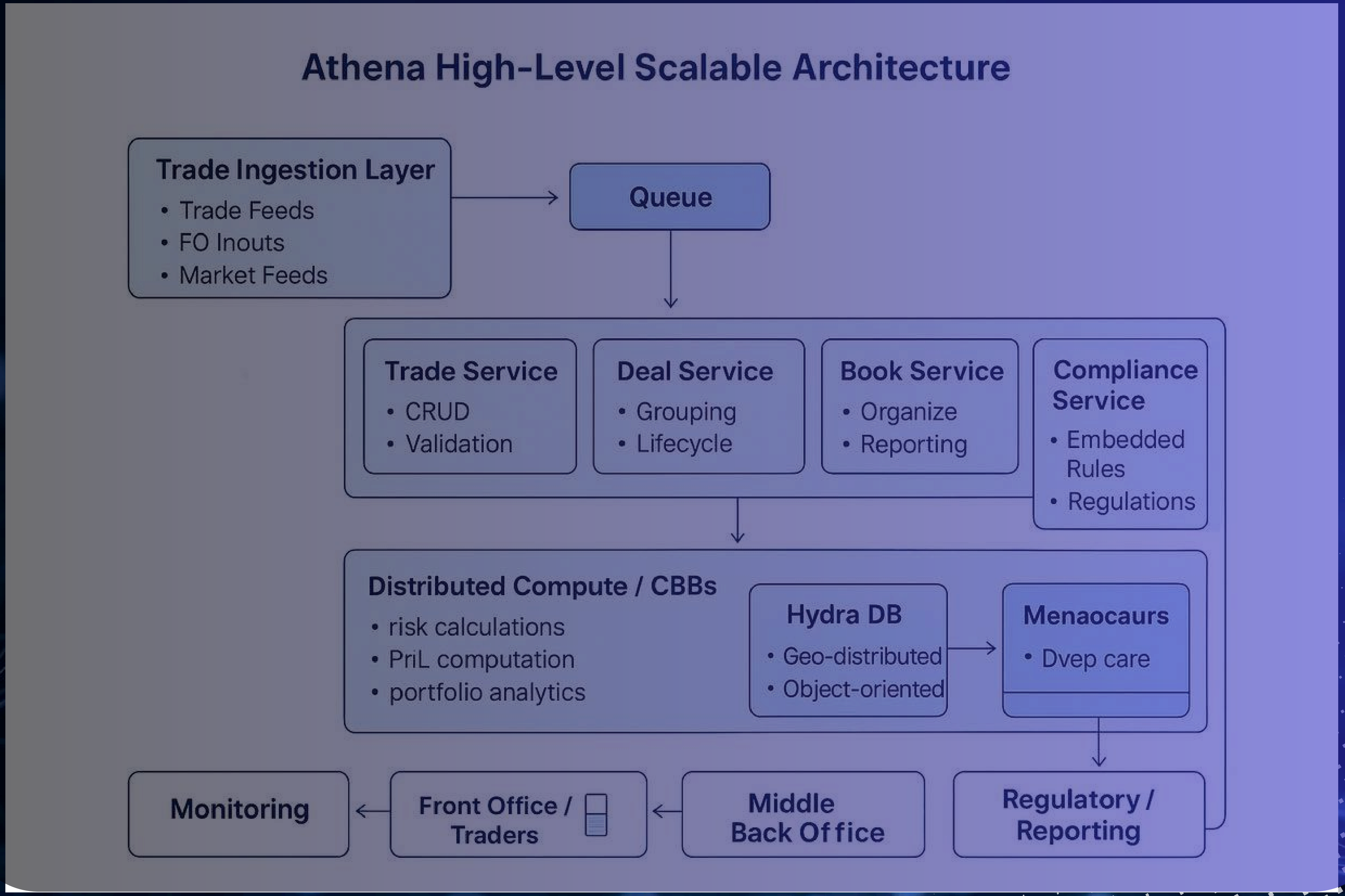
```
[Trade Feeds / FO Inputs]
          |
          ▼
[Trade Ingestion Layer / Queue]
          |
          ▼
[Trade & Deal Service] → [Book Service] → [Compliance Service]
          |
          ▼
[Distributed Compute / CBBs] → [Hydra DB] → [PnL & Risk Outputs]
          |
          ▼
[Event Daemons] → [Alerts, Notifications, Corporate Actions]
          |
          ▼
[Monitoring / Dashboards / Regulatory Reports]
```

ATHENA


Athena High-Level Scalable Architecture
Trade Ingestion Layer
• Trade Feeds
• FO Inouts
• Market Feeds
Queue
Trade Service
• CRUD
• Validation
Deal Service
• Grouping
• Lifecycle
Book Service
• Organize
• Reporting
Compliance Service
• Embedded Rules
• Regulations
Distributed Compute / CBBs
• risk calculations
• PriL computation
• portfolio analytics
Hydra DB
• Geo-distributed
• Object-oriented
Menaocaurs
• Dvep care
Monitoring
Front Office / Traders
Middle Back Office
Regulatory / Reporting

ATHENA

```
   [ Legacy Feeds ]  [ E-Trade Feed ]  [ CSV Uploads ]
          |               |                  |
          v               v                  v
        (HTTP Polling / File Drops on NFS)
                    |
                    v
          +-----------------------+
          |  Ingestion Service    |
          |  - single instance    |
          |  - sync parsing       |
          +-----------------------+
                     |
                     v  (blocking writes)
          +----------------------------+
          |  Single RDBMS (Primary)    |
          |  - one region              |
          |  - no replicas             |
          |  - mixed OLTP + OLAP       |
          +----------------------------+
                      |
  +----------------------------------------+
  |                         |                        |
  v                         v                        v
+----------------------+  +------------------------+  +------------------------+
| Risk Compute Svc     |  | Reporting Dashboard    |  | Compliance Logger      |
| - sync RPC calls     |  | - ad-hoc SQL queries   |  | - writes audit rows    |
| - runs on app node   |  | - same DB              |  | - same DB              |
| - single worker      |  +------------------------+  +------------------------+
+----------------------+
          |
          v  (nightly batch via cron)
+-----------------------+
| Batch Recalc Job      |
| - exports CSVs        |
| - backfills metrics   |
+-----------------------+
```



Thank You