Theme 2: Simple E-commerce Inventory Management System (Java)

Step 1- Understand The Problem and Establish Design Scope

✓ A modern inventory management system for an e-commerce platform is a complicated system with stringent requirements on -

1. **Latency**
2. **Throughput**
3. **Robustness**

✓ Before we start, let's ask the interviewer a few questions to clarify the requirements.

1. ***Candidate***: Which type of operations are supported: adding, updating or removing products? Do we need to support limit stocks, market stocks or conditional products?

Interviewer: We need to support the following: **adding, updating and removing products**. For the product type, we need to consider the **limit products**.

2. ***Candidate***: Does the system need to support after-hours inventory update at outlets?

Interviewer: No we just need to support the normal trading hours.

3. ***Candidate***: Could you describe the basic functions of **The Inventory Management System-** such as **how many users, how many products transactions?**

Interviewer: A **digital colleague or a store-assistant** at outlet can place new limit stocks or remove them, and **receive matched stocks in real-time**. Other store-assistants can also view the status of the updated real-time inventory products.

- The system needs to **support at-least tens of thousands(10,000)** of digital-colleagues doing the similar transaction at the same time across the different outlets of the ecommerce platform.
- For the transaction volume, system **should support billions of products update per day**.
- Also the add/remove/update transactions of products is a regulated facility in the inventory so **system should run risk checks**.

4. ***Candidate***: Could you please elaborate on risk checks**?**

Interviewer: Let's just do simple risk checks. For example, a digital-colleague or store-assistant can only update a maximum **of 1 million products of Apple brand in one day**.

Deductions-What caught my attentions?

- Requirements like "at-least 100 symbols".
- Requirements like "tens of thousands of users".

Deduction-1: Interviewer wants to design a **small-to-medium scale** inventory management system for the e-commerce platform.

Deduction-2:Interviewer is not sure about the exact figure. It's better to build the system with focus on extensibility as an area for follow-up questions.

Non-functional Requirements

- ✓ **Availability** – 99.9% since downtime even seconds can harm reputation of the e-commerce platform.
- ✓ **Fault tolerance**- Fault tolerance and a fast recovery mechanism are needed to limit the impact of a production incident.
- ✓ **Latency**- The round-trip latency should be at the millisecond level, with a particular focus on the 99th percentile latency.
- ✓ **Security**-The system should have an account management system. For legal and compliance, the system performs a KYC(Know Your Client for digital-colleagues/shop-assistant) check to verify their identity before a new account is opened. For authorized web pages system should prevent distributed denial-of-service(DDoS) attacks.

Estimation

1. 100 symbols.
2. 1 billion inventory products update per day.
3. E-commerce platform outlets is open Monday through Friday from 9:00 am to 4:00 pm Easter Time. That's 6.5 hours in total.
4. QPS: 1 billion / (6.5 x 3600) = ~43,000
5. Peak QPS: 5 x QPS = 215, 000.The volume is significantly higher in the busy hours of morning and before it closes in the afternoon.

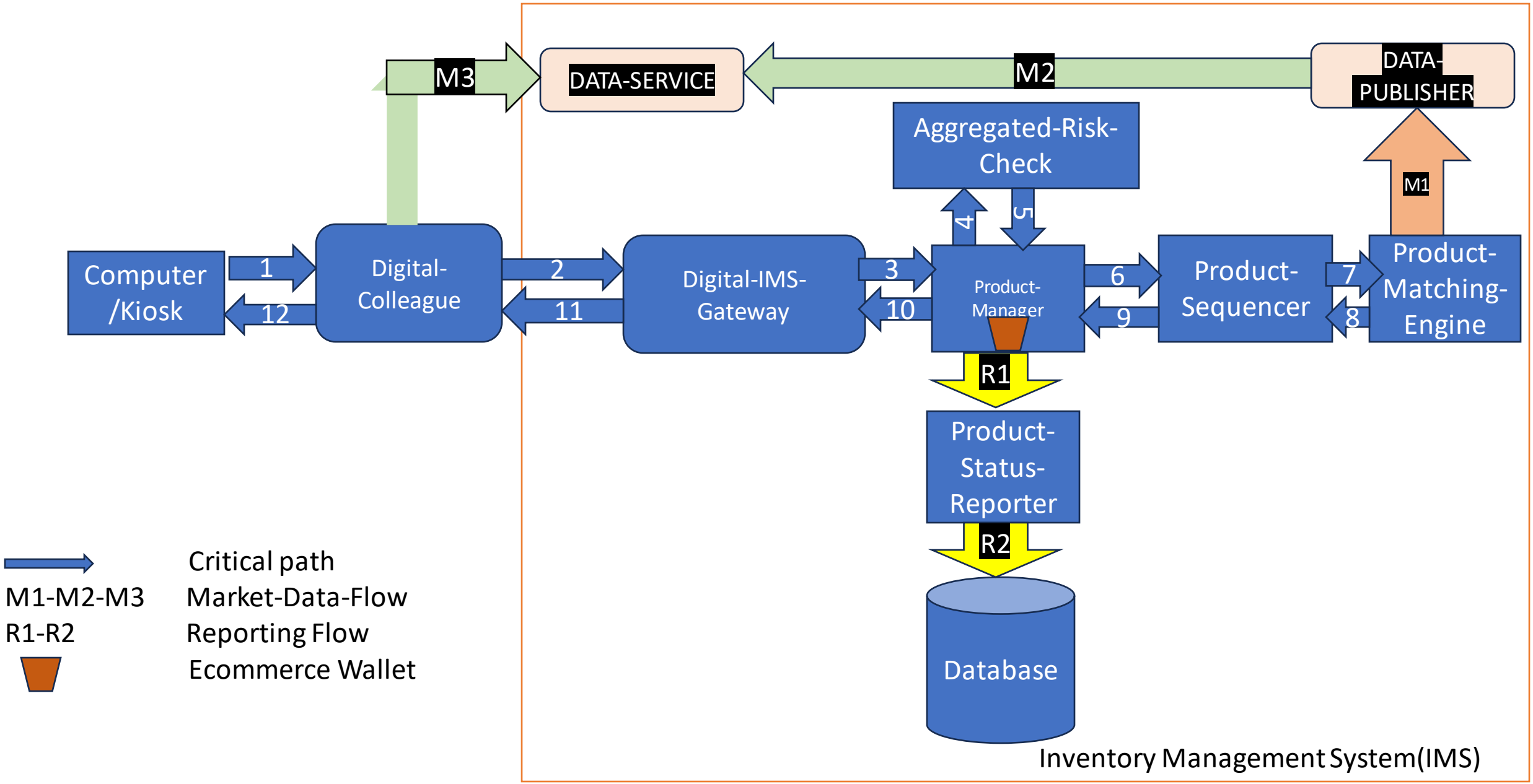


Fig1-HIGH-LEVEL DESIGN

High Level Workflow

1. Digital colleague or store-assistant adds/removes a product via the web-portal of the System at the Kiosk of the Ecommerce Outlet.
2. The digital colleague sends the update of product to the IMS.
3. The product update enters the system through the Digital-IMS-Gateway. It performs basic gatekeeping functions such as-

- Input validation
- Rate limiting
- Authentication.

4. The Digital-IMS-Gateway then forwards the product update to the Product-Manager.
5. The Product Manager performs risk checks based on rules by the risk manager or Admin.

- After passing the checks, the Product-Manager verifies there are sufficient funds in the E-commerce wallet for the product transactions.

6~7. The product update is sent to the Product-Match-Engine. When a match is found, the matching engine does two executions(also called fills), with one each for the Add/Update(Inflow in Inventory) and Remove(Outflow from the Inventory) sides.

To guarantee that matching results are deterministic when replayed, both orders and executions are sequenced in the Product-Sequencer.

8~12: The executions are returned to the digital colleague/store assistant.

Market Data Flow

D1: The Product-Matching-engine generates a stream of executions(fills) as matches are made. The stream is sent to Data-Publisher.

D2:The Data-Publisher constructs the snapshot charts(data-view of products at certain interval generally one-minute, five-minute, one-hour, one-week and one-month) from the stream of executions. It then sends data to the Data-Service.

D3: The Data is saved to specialized storage for real-time analytics. Ecommerce-CEO/Admin connect to Data Service to relay to concerned stakeholders in the business.

Reporting Flow

R1~R2: The Product-Status-Reporter collects all the necessary reporting fields(e.g, client_id, price, quantity, product_type, filled_quantity, remaining_quantity) from product updates and executions, and writes consolidated records to the database.

Note: Steps 1 to 12(Product-Update Flow) is marked as a critical flow because they have different latency requirements.

1. Product-Update Flow

- Critical path of IMS.
- Everything must happen fast.
- Heart of Product-Update Flow is the Product-Matching-Engine.

1. Product-Matching-Engine

- Maintain the "Order Book" against each products. The "Order Book"

is a list of "Inflow/Outflow" for a product.

- Match Inflow and Outflow of products. The matching function must be fast and accurate.
- Distribute the execution stream as Market-data.

- Product-Sequencer also functions as a message queue. It is also an event store for the Product-updates and executions.
- Product-Sequencer confirms Timeliness and fairness.
- Product-Sequencer confirms fast recovery / replay.
- Product-sequencer confirms exactly-once guarantee.

3. Product-Manager

- Product-manager receives product-updates on one end and receives executions on the other. It manages the product's states.
- It receives inbound product updates from the Digital-IMS-Gateway and performs the following-

a. It sends the Products for risk checks that is, verifying the Digital-Colleague's trade volume is below $1M a day.

b. It checks the order against the Digital-Colleague's wallet and verifies that there are sufficient funds to cover the Product's trade.

c. Product-Manager sends the product-update to the Product-Sequencer where the product is stamped with a sequence-ID. The sequence-ID is then processed by the Product-Matching Engine.(only sending necessary attributes to the matching engine).

3. Product-Manager(Continued)

d. On the other hand Product-Manager receives executions from the Product-Matching-Engine via the Product-Sequencer. The Product-Manager returns the executions for the filled products to the Digital-Colleagues via the Digital-IMS-Gateway.

- Product-Manager should be fast, efficient and accurate.
- Product-Manager maintains the current states for the products updates.

4. Digital-IMS-Gateway

- Digital-IMS-Gateway is the lightweight-gatekeeper for the IMS system.
- It receives Product-Updates placed by Digital-Colleagues at KIOSK of Inventory outlets and routes them to Product-Manager.
- It performs the following functions -

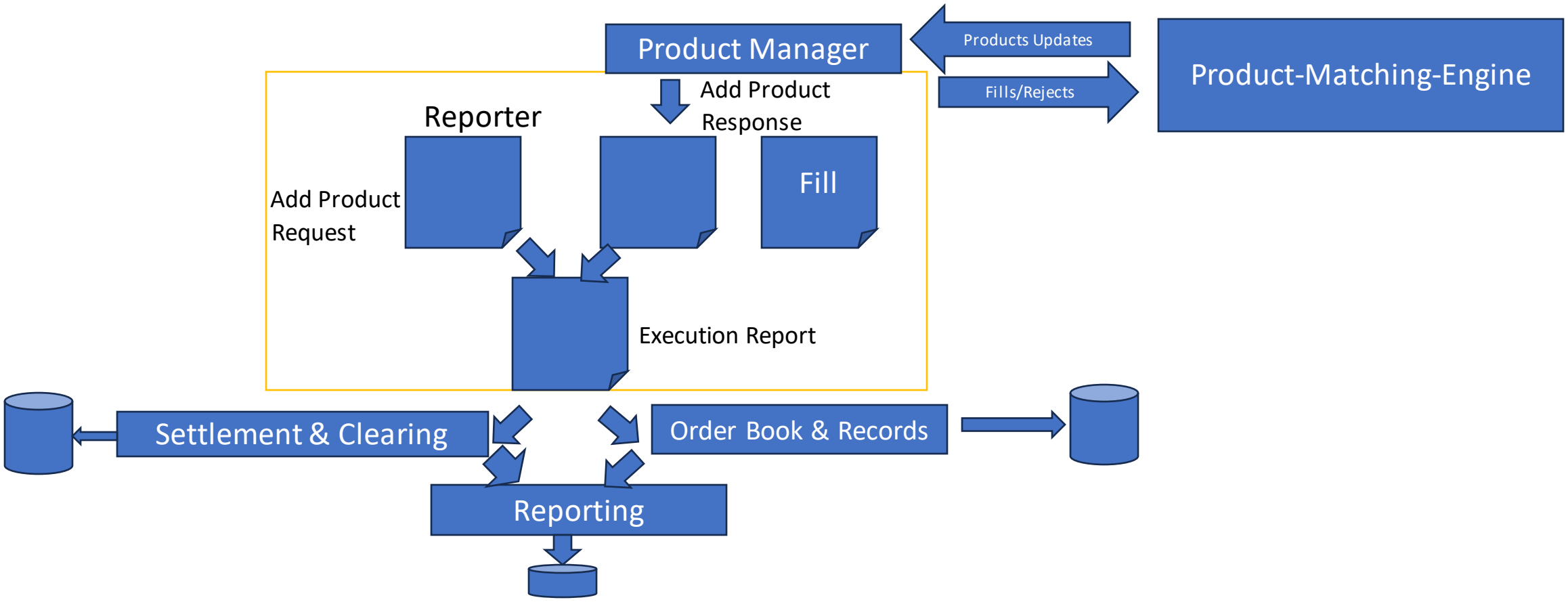
1. Authentication
2. Validation
3. Rate Limit
4. Normalization

5. Market data flow

- Receives executions from the Product-Matching-Engine and builds "Order Book" and "Snapshots" from the stream of executions.
- The Market-Data is sent to the Data-Service where they are made available to the subscribers for Data-Analytics.

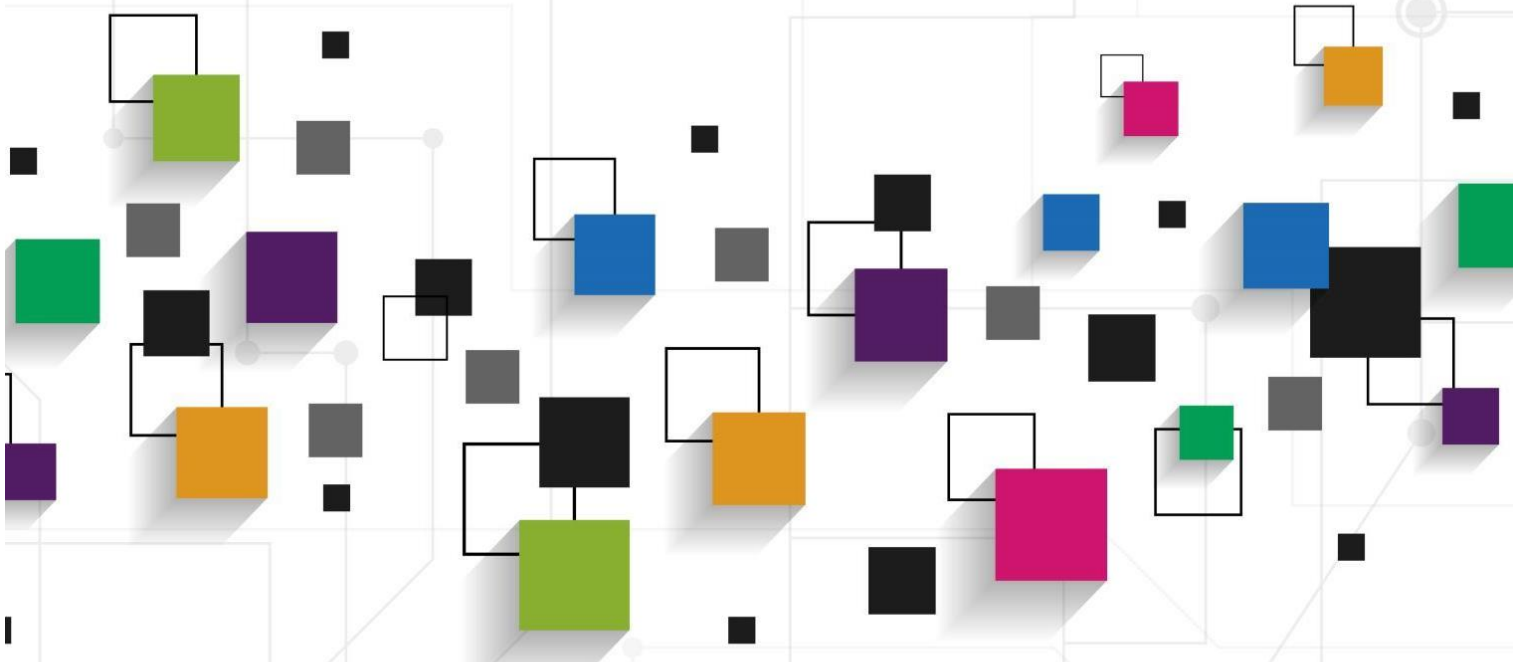
6. Reporting Flow

- Provides trading history, accumulation of data services from various data tables for report generation, settlements.
- Less sensitive to latency.



API Design

- RESTful conventions for the API below to specify the interface between Digital-Colleagues and the Digital-IMS-Gateway.



No.	Service/Functionalities	Endpoint	Parameters	Response-Body	Response-Code
1.	**transact**– endpoint places a transaction. Requires authentication.	**POST /v1/transact**	**side**: inflow(add/update) or outflow(remove)(*String*) **price**: price of the limit product(*String*) **quantity**: Quantity of product(*Long*)	**id**: ID of Product(*Long*) **creationTime**: System Creation Time of the Product(*Long*) **filledQuantity**: the quantitythat has been successfully executed.(*Long*) **remainingQuantity**: The quantity still to be executed(*Long*). **status**: add/removed/updated/filled.(*String*) *Rest attributes are same as input parameteres.	**200**:successful **40x**: parameter error /access denied/unauthorized **500**:server error

No.	Service/Functionalities	Endpoint	Parameters	Response-Body	Response-Code
2.	**execute**– endpoint queries execution info. It requires authentication.	GET /v1/execution?symbol={:symbol}&orderId={:orderId}&startTime={:startTime}&endTime={:endTime}	**productId**: ID of the Product.Optional(*String*) **startTime**: query start time in epoch.(*Long)* **endTime**: query end time in epoch.(*Long)*	**[Each Execution array has-]** **id**: ID of execution(*Long*) **productId**: ID of the Product.(*String*) **side**: inflow(add/update)or outflow(remove)(*String*) **quantity**: the filled quantity(*Long*) **price:** price of the execution**.***(Long)*	**200**:successful **40x**: parameter error/access denied/unauthorized **500**:server error

No.	Service/Functionalities	Endpoint	Parameters	Response-Body	Response-Code
3.	**Order book**– endpoint queries order book information for a symbol.	**GET /v1/marketdata/orderBook?symbol={:symbol}**	**symbol**: ID of the Product.(*String*) **startTime**: query start time in epoch.(*Long)* **endTime**: query end time in epoch.(*Long)*	**bids**: array with prize and size(*Array*) **asks**: array with prize and size(*Array*)	**200**:successful **40x**: parameter error/access denied/unauthorized **500**:server error

No.	Service/Functionalities	Endpoint	Parameters	Response-Body	Response-Code
4.	**Historical prices**– endpoint queries chart data for a given time range and resolution.	**GET /v1/marketdata/snapshot?symbol={:symbol}&resolution={:resolution}&startTime={:startTime}&endTime={:endTime}**	**symbol**: ID of the Product.(*String*) **resolution**: window length of the snapshot chart in seconds.(*Long)* **startTime**: query start time in epoch.(*Long)* **endTime**: query end time in epoch.(*Long)*	**[Each Historical Snapshot price array has-]** **open**: open price of each data set(*Double*) **close**: close price of each data set(*Double*) **high**: high price of each data set(*Double*) **low**: low price of each data set(*Double*)	**200**:successful **40x**: parameter error/access denied/unauthorized **500**:server error

Data Models

1. Product, order and execution.
2. Order book.
3. Historical snapshot chart.

- In the critical trading path, orders and executions are not stored in a database for high performance.
- Orders and executions are stored in sequencer for fast recovery, and data is archived after the outlet closes.
- Reporter writes orders and executions to the database for reporting use cases like reconciliation and report generation.

2. Order Book

- An Order Book is a list of Inflow and Outflow Products into/from the IMS.
- It is a key data structure used for fast Product matching. Inside Product-Matching-Engine.

GitHub Copilot's Contribution

- Making of Product Matching Algorithms-FIFO with Lead market Maker Approach-(High Level View)

```
Context handleProduct(Order orderBook, OrderEvent orderEvent) {
if (orderEvent.getSequence()!= nextSequence) {
   return ERROR(OUT_OF_ORDER, nextSequence);
}
if (!validateOrder(symbol, price, quantity)) {
   return ERROR(OUT_OF_ORDER, orderEvent);
}

Order order = createOrderFromEvent(orderEvent);
switch (msgType);
    case NEW:
          return handleNew(orderBook, order);
    case CANCEL:
          return handleCancel(orderBook, order);
    default:
          return ERROR(INVALID_MSG_TYPE, msgType);
}
```

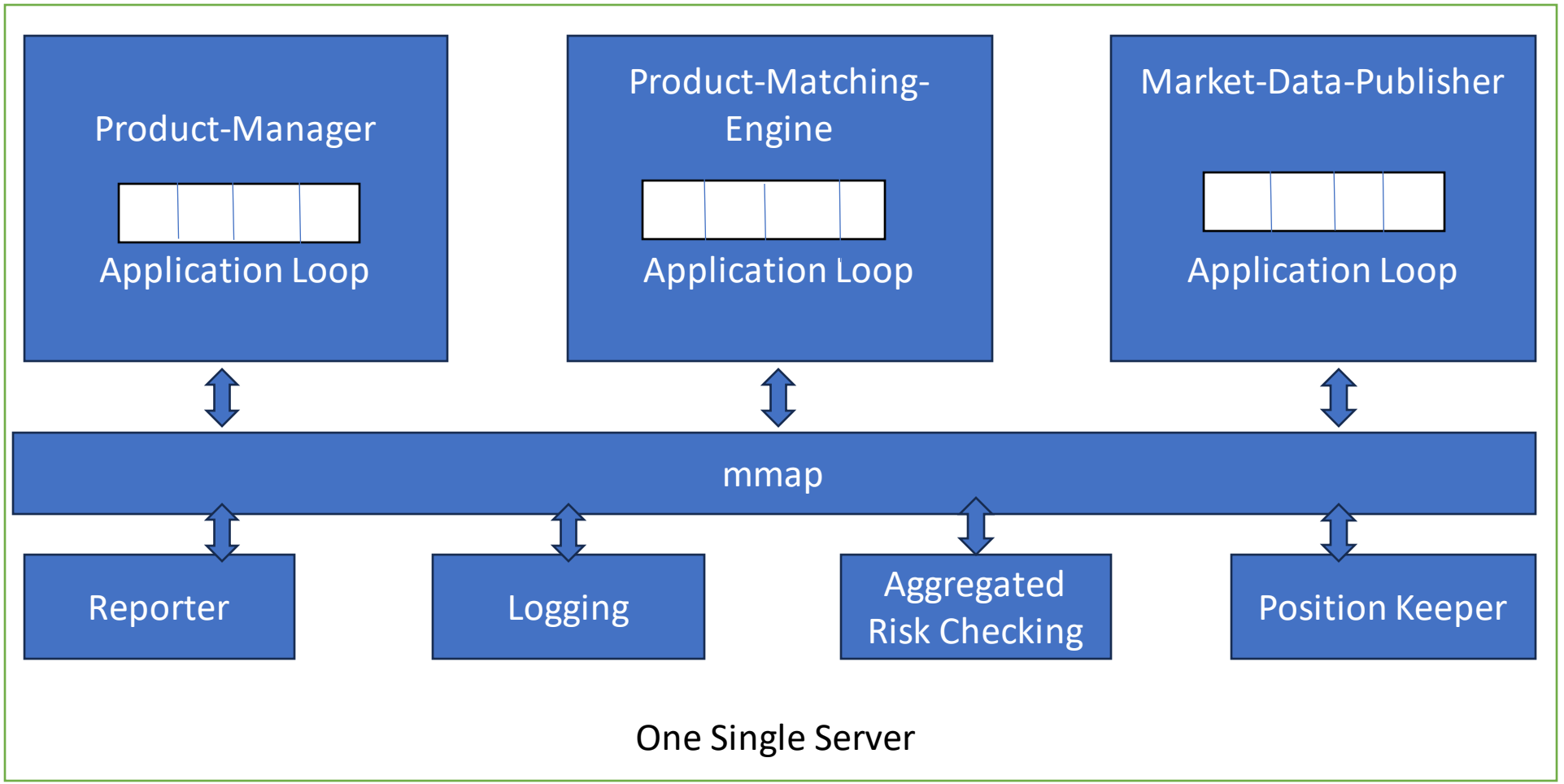
GitHub Copilot's Contribution

```
Context handleNew(Order orderBook, Order order) {
if (BUY.equals(order.side) {
   return match(orderBook.sellBook, order);
} else {
return match(orderBook.buyBook, order);
}
}
Context handleCancel(Order orderBook, Order order) {
if (!orderBook.orderMap.contains(order.orderId) {
   return ERROR(CANNOT_CANCEL_ALREADY_MATCHED, order);
}
removeOrder(order);
setOrerStatus(order, CANCELED);
return SUCCESS(CANCEL_SUCCESS, order);
}
```

GitHub Copilot's Contribution

```
Context match(OrderBook book, Order order) {
 Quantity leavesQuantity = order.quantity - order.matchedQuantity;
 Iterator<Order> limitIter = book.limitMap.get(order.price).orders;
 while (limitIter.hasNext() && leavesQuantity > 0) {
    Quantity matched – min(limitIter.next.quantity, order.quantity);
    Order.matchedQuantity += matched;
    leavesQuantity = order.quantity - order.matchedQuanity;
    remove(limitIter.next);
    geneerateMatchedFill();
  }
  return SUCCEMATCH_SUCCESS, order);
}
```

Step 3- Low Level Design



A LOW LATENCY SINGLE SERVER IMS DESIGN

Application Loop

- ✓ It keeps polling for tasks to execute in aa while loop and is primary task execution mechanism.
- ✓ Each box in the diagram represents a component.
- ✓ A component is a process on the server.
- ✓ To maximize CPU efficiency, each application loop(think of it as the main processing loop) is single-threaded and the thread is pinned to a fixed CPU core.
- ✓ Benefits of pinning the application loop is ensuring no context switch.
- ✓ CPU1 is fully allocated to the order manager's application loop.
- ✓ No locks and therefore no lock contention, since there is only one threads that updates states.

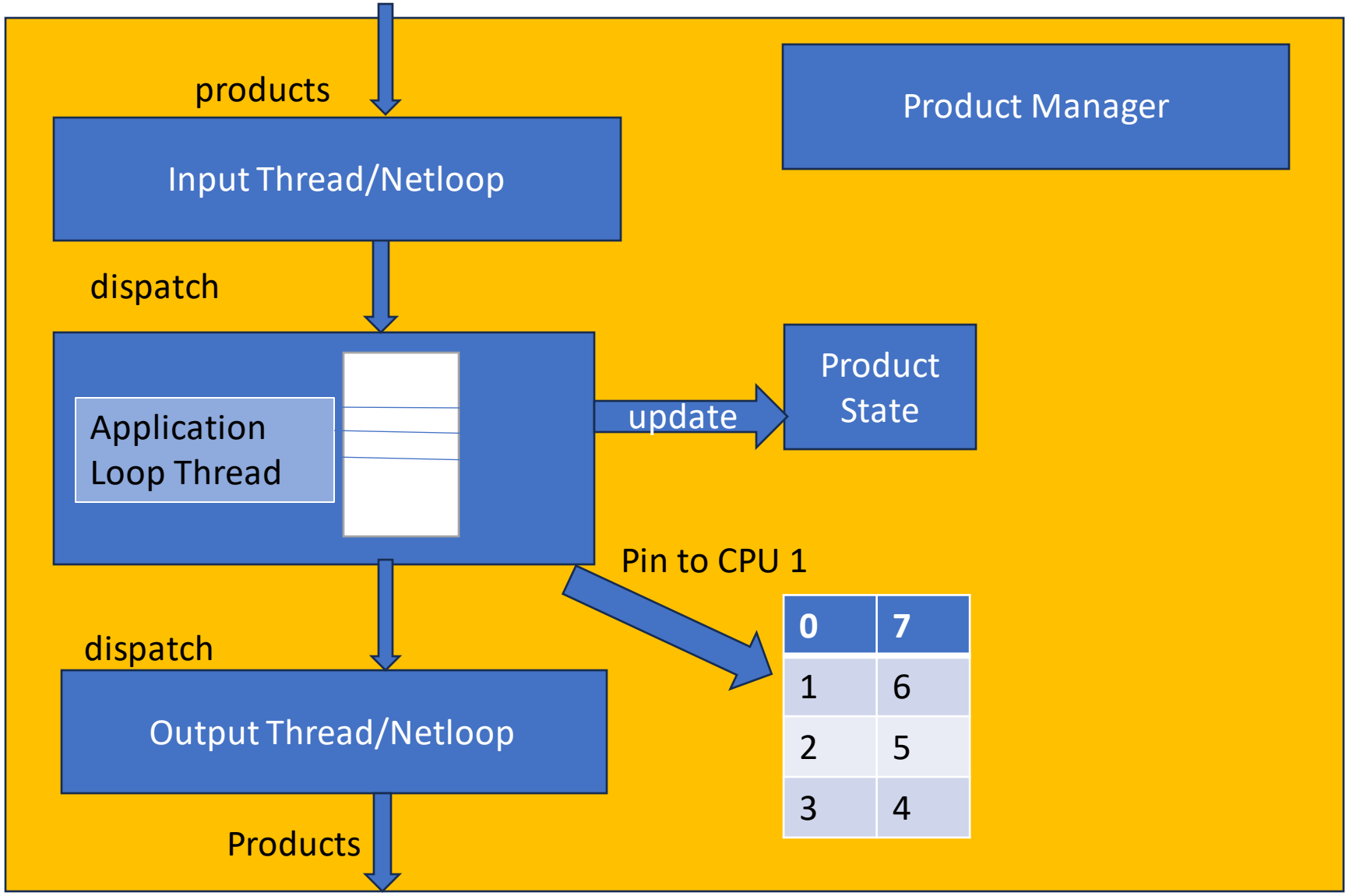


Fig4-Application Loop Diagram

mmap

- ✓ Refers to a POSIX-compliant UNIX system call named mmap that maps a file into the memory of a process.
- ✓ mmap provides a mechanism for high-performance sharing of memory between processes.
- ✓ The performance advantage is compounded when the backing file is in /dev/shm.
- ✓ /dev/shm is amemory-backed file system.
- ✓ When mmap is done over a file in /dev/shm, the access to the shared memory does not result in any disk access at all.
- ✓ Mmap is used in the server to implement a message bus over which the components can communicate in sub-microsecond.

Event Sourcing

Non-Event Sourcing	Event-Sourcing
1. Used in classical database schema.	1. Used in this System Design.
2. It keeps track of the order status for an order, but it does not contain any information about how an Purchase Update arrives at the current state.	2. It tracks all the events that change the order states and it can recover order states by replaying all the events in sequence.

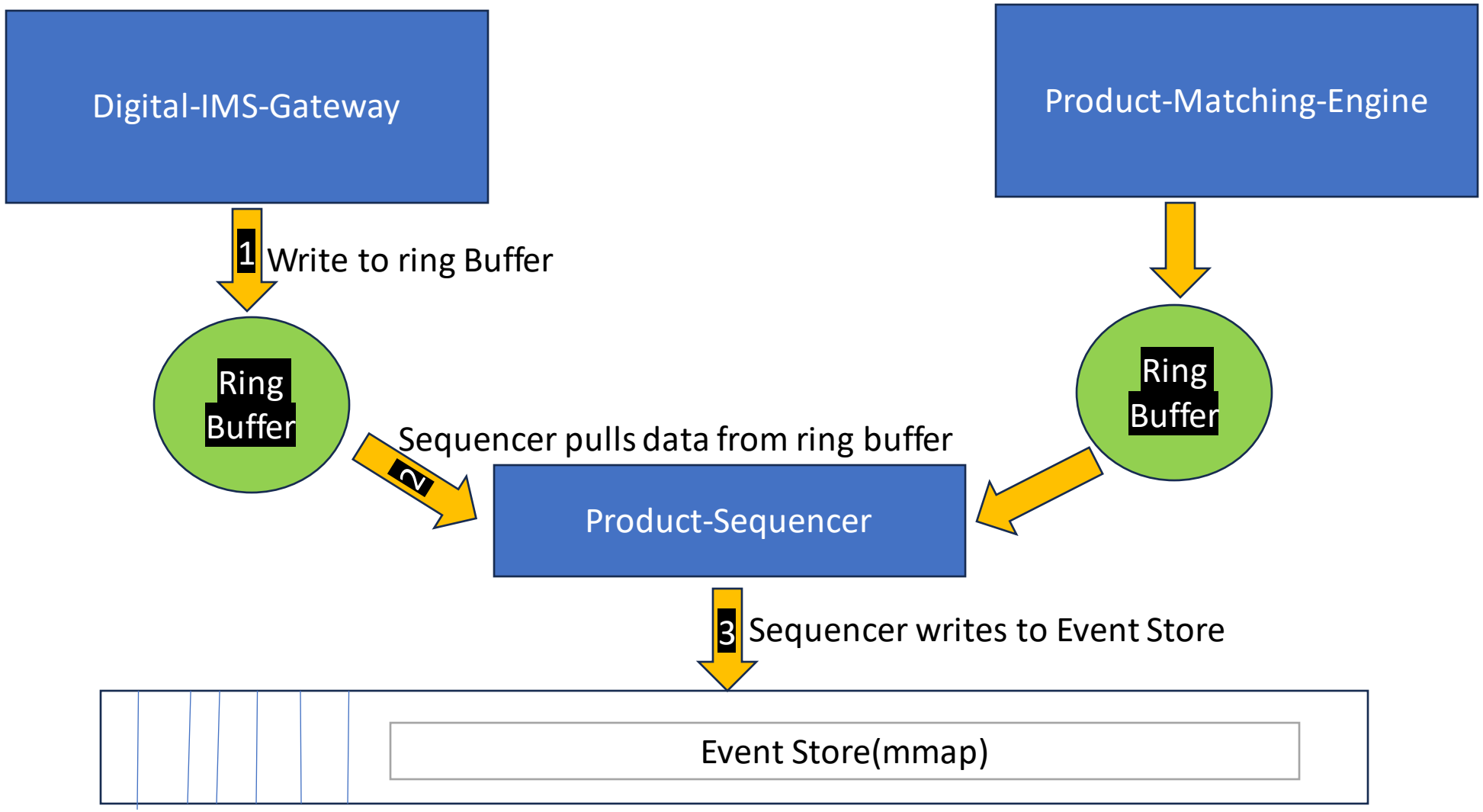
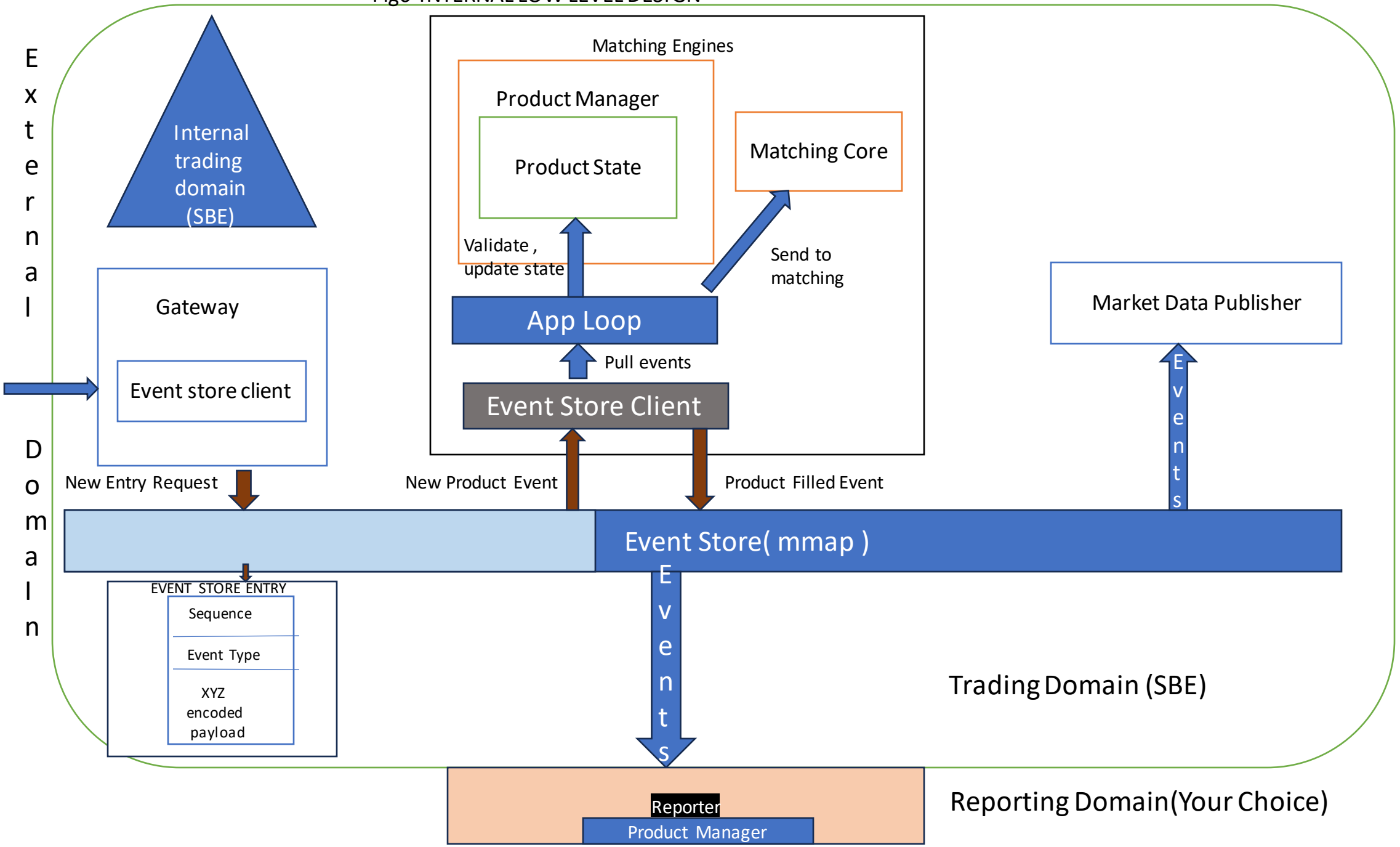


Fig5- INTERNAL DESIGN OF PRODUCT-SEQUENCER

Fig6-INTERNAL LOW LEVEL DESIGN



Mitigating Challenges & Edge Cases

Expectations	How to Meet Expectations	In case Anything goes Wrong How to Troubleshoot
✓ Aiming High Availability, 4 nines(99.99%) ✓ Detection of failure and the decision to failover to the backup instance should be fast.	1. Identify single point of failure in the architecture by normal monitoring of hardware and processes. 2. We can also send heartbeats from the matching engine, if the heartbeat is not received in time, the matching engine might be experiencing problem.	1. For stateless services such as the client gateway, they could easily be horizontally scaled by adding more servers. 2. For stateful components, such as the Product-Manager and Product-Matching-Engine we need to copy state data across replicas. 3. Using the concept of Hot-Warm matching technique. When the hot matching engine works as the primary instance, the warm engine receives and processes the exact same events but does not send any event out onto the event store. 4. When the primary goes down, the warm instance immediately take over as the primary and send out events. 5. When the warm secondary too goes down it can be restarted and can always recover all the states from the event store.
✓ System must be fault tolerant	1. To make the system fault tolerant – we need t answer many questions • If the primary instance goes down, how and when do we decide to failover to the backup instance? • How do we choose the leader among backup instance? • What is the RTO needed(Recovery Timme Objective)? • What functionalities need to be recover(RPO – Recovery Point Objective)?/ Can our system operate under degraded conditions?	1. When we first release the code we might need failovers manually to gather enough signals and operational experience an gain confidence to surface edge cases. 2. Once the decision of failover is correctly made, we can use any one of the leader-election algorithms such as Raft Algorithm.

Expectations	How to Meet Expectations	In case Anything goes Wrong How to Troubleshoot
✓ Network Security must be ensured.	1. An IMS system for ecommerce platform provides some public interfaces and a DDoS attack is a real challenge.	1. Isolate public services and data from private services, so DDoS attacks don't impact the most important clients. In case same data is served, we can have multiple read-only copies to isolate problems. 2. Use a caching layer to store data that is infrequently updated. With good caching, most queries won't hit the databases. 3. Harden URLs against DDoS attacks. 4. Rate limiting is frequently used to defend against DDoS attacks.
✓ System must be consistent updating products records at IMS across all stores at the same time	1. Smart clients will fight to be the first on the list when the outlet opens since the order of subscribers is decided by the order in which they connect to the publisher, with the first one always receiving data first.	1. Multi-cast using reliable UDP protocol is a good solution to broadcast updates to many participants at once. The MDP could also assign a random order when the subscriber connects it.

Step 4- Wrap Up

- An ideal deployment model for an IMS is to put everything on a single gigantic server or even one single process.

- The convenience provided by the cloud ecosystem changes some of the designs and lowers the threshold for entering the industry.

Thank You.

Meet the Audience soon!