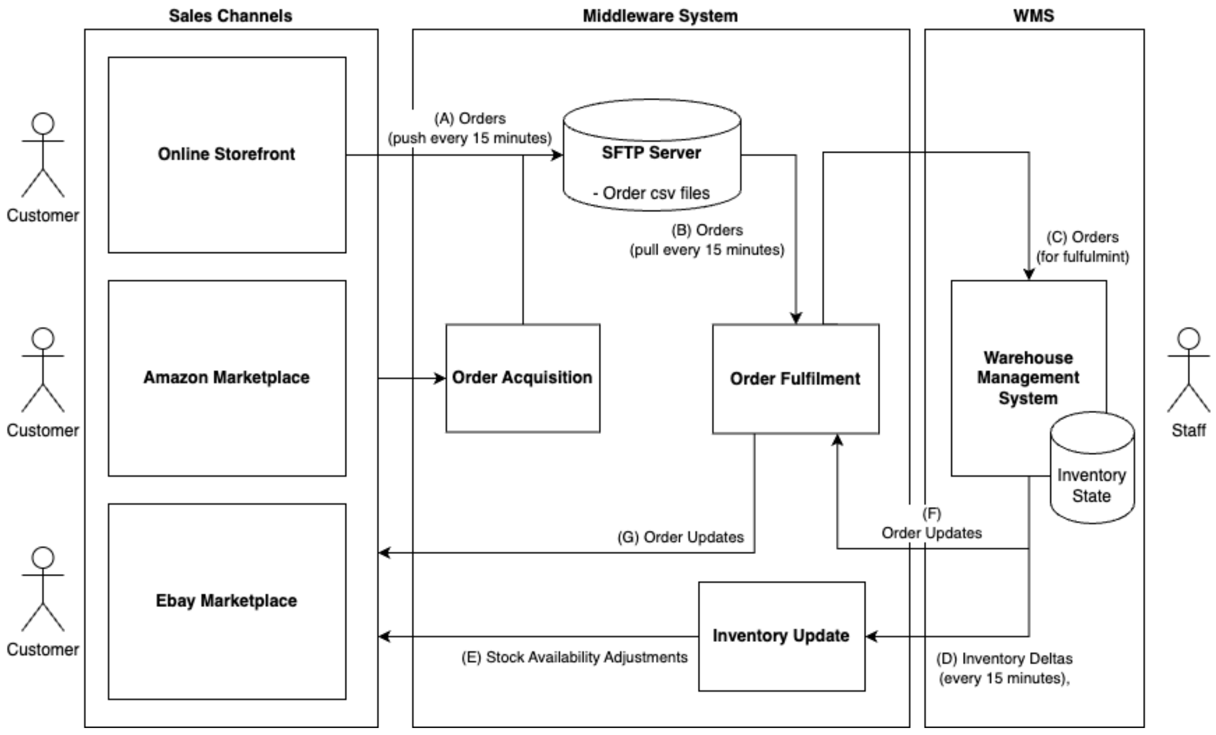
Overselling Across Sales Channels

TLDR

- The primary reason behind overselling across retailer's sales channel is slow feedback loop. Stock availability adjustments in the sales channels can take upwards of 45 minutes following an order.
- In the short term, symptoms of the problem can be reduced by speeding up information flow frequency.
- An event-based architecture with closer integration with sales channels is recommended to further optimise information flow and speed up their feedback loop.

The Problem

Current architecture of the system is something like the following:



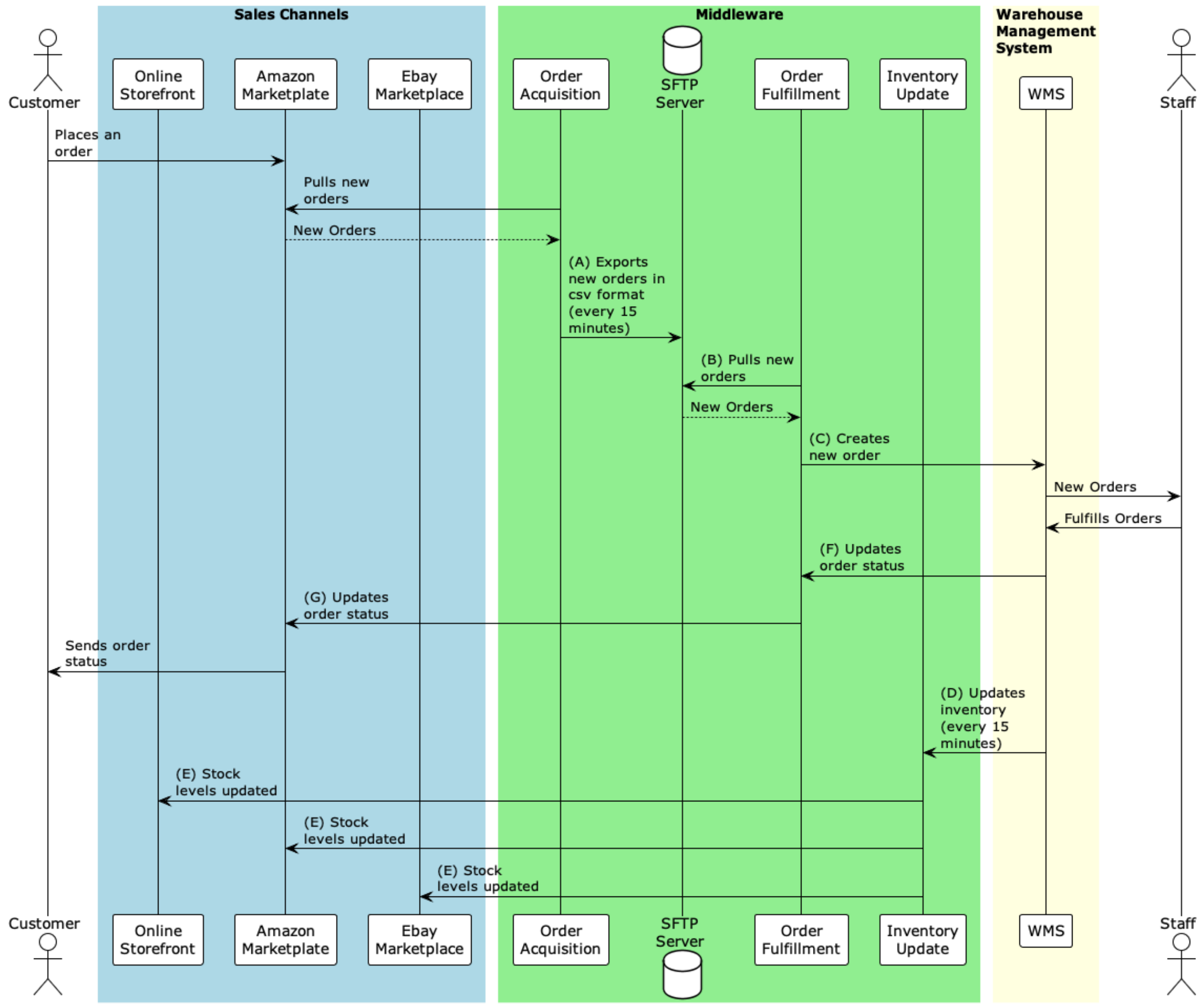
In this diagram:

- Customers interact with sales channels on the left.
- These systems either push orders to the SFTP server in 15-minute or as part of middleware, there is an Order Acquisition system that either receives or pulls orders. **(A)**
- The Order Fulfilment function in middleware pulls orders from this SFTP server **(B)** in another 15-minute interval and pushes them to WMS **(C)**.
- Once orders are fulfilled in WMS, orders also need to be updated in the sales channels to notify customers. This is shown in **(F)** and **(G).**

- Stock levels also need to be adjusted in sales channel. This is done through 15-minute interval updates from WMS **(D)** that is forwarded to the sales channels through an Inventory Update function **(E)** in middleware.

Please note that boxes in the middleware section are functions. It is not known whether they are all separate services or are part of one or more split services.

This information flow is summarised in the following sequence diagram for an example order from amazon:



If we imagine an order is being placed for the last stock item, there will be:

- Delay of pulling orders from Amazon – unknown
- Delay of pushing orders to SFTP server **(A)** – up to 15 minutes
- Delay of pulling orders from SFTP server **(B)** – up to 15 minutes
- Order creation delay **(C)** – unknown
- Order fulfilment delay **(D)** – unknown
- Inventory update following order fulfillment **(E)** – up to 15 minutes
- Inventory update delay to sales channel **(F)** – unknown

Even if we assume all unknown quantities to be instant, there can be up to 45-minute delay simply due to periodic sync steps in the flow of information. In this time, more orders can be placed for this item across the three different sales channels.

Quick Wins

To reduce symptoms of overselling, frequency of update across the three system should be speed up. Assuming reasonable rate limit constraints, 15-minute sync interval (Points A, B and D) should all be reduced to 15s or faster (4 times a minute). This will reduce the 45-minute known delay to less than a minute.

Benefits of this approach include:

- Very little additional development effort is required – existing mechanism is being speed up.[1]
- Infrastructure requirements are unchanged.

Potential drawbacks of this approach include:

- Overselling may still occur. Primarily because:
    - A minute delay may still be a long time for popular or promoted items
    - The primary mechanism for stock adjustment in sales channel is inventory update after order fulfilment. Stock is not reserved upon placement of order. Since stock fulfilment is a manual step by staff, there may still be significant delays (e.g. staff having a long queue of orders being fulfilled or orders being placed at off hours)
- May not be possible due to API rate limits. For example, if the initial reason behind coming up with 15-minute sync delays is not hitting API rate limits of sales channels or WMS.

Another tactic to implement on top maybe to hold a small percentage of extra stock as buffer. In this scenario, if only a small amount of overselling occurs, this buffer could absorb it. This however has additional inventory, management and cost implications.

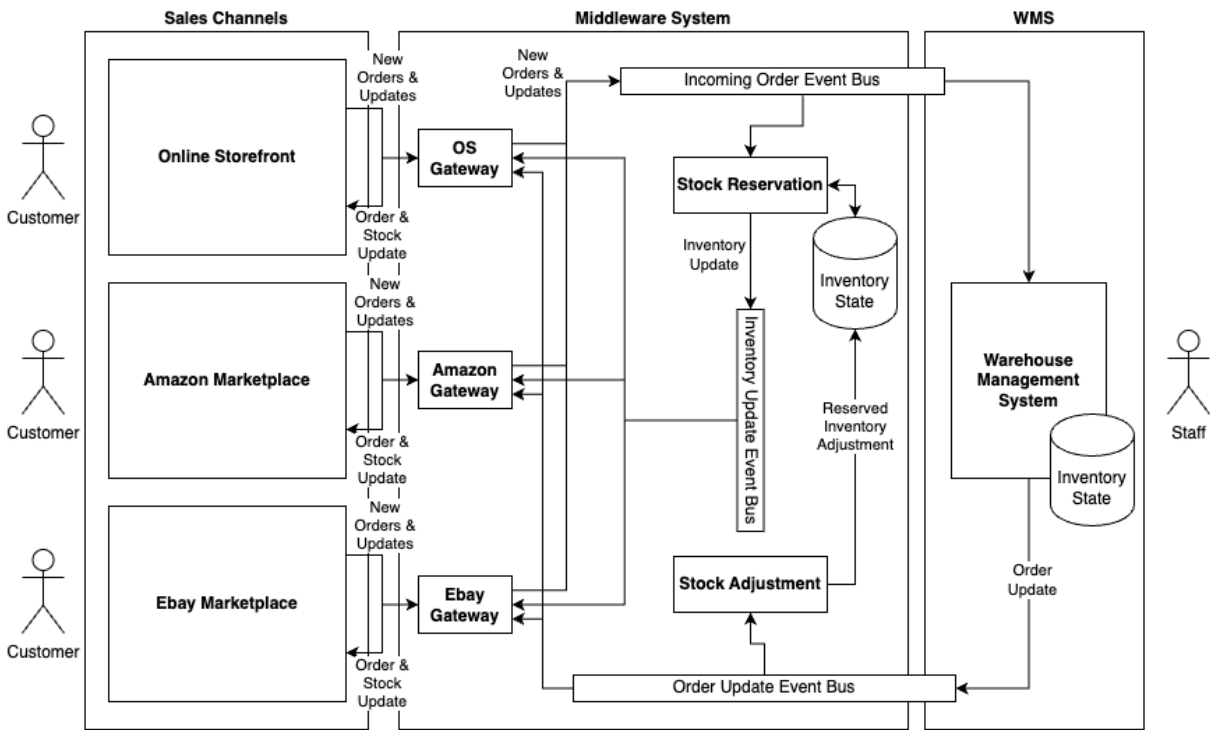
Better Solution

A better solution for speedy feedback to sales channels and avoidance of overselling should feature the following things:

- Faster forms of communication across system (push/webhook APIs) in critical parts of the information flow – instead of periodic pulls.
- A stock reservation system for orders across the sales channels – instead of waiting for orders to be fulfilled manually by staff.

Without prescribing any specific technologies, I propose the following:

[1] There may still be minor development effort required if existing service(s) cannot handle new update frequency.



I propose introducing:

- A Gateway component specific to each sales channel. Each gateway is responsible for dealing with their corresponding sales channel.
    - The gateway should receive new orders and customer updates (e.g. order cancellations) through a push update. For example, Amazon has Amazon SP-API Notification API. Push/Webhook APIs are contemporary industry standard for timely event delivery over cloud.
    - Once an order is received, Gateway will convert it to a sales channel independent representation event format and publish it to the incoming orders event bus.
    - Gateway is also responsible for picking up events from order update (e.g. order shipped) and inventory update event busses (e.g. an item is no longer available) and adjusting orders / stock availability at their relevant sales channel.
- A stock reservation component for reserving inventory for orders as they come in.
    - It should listen for new orders, reserve stock for said orders and publish to inventory update event bus for sales channel gateways to adjust item availability.
    - This means that middleware must keep inventory level and reservation state. This duplication of data enables middleware to speed up feedback loop to sales channel.
- A stock adjustment component for adjusting reserved/available stock in inventory state by listening to Order updates from WMS.
- Events inside the event busses should designed to be vendor neutral – not tied to current sales channels or current WMS provider.

- Event delivery should be set up with at least once delivery. This also adds requirements for idempotence for the components.

Other adjustments not drawn in the diagrams:

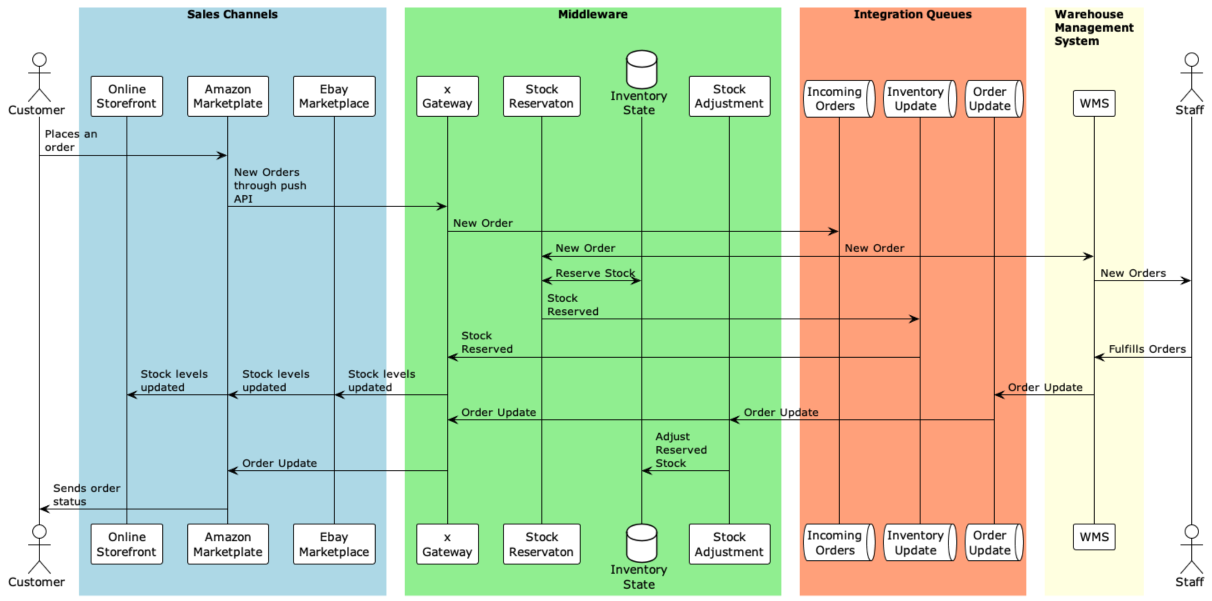
- In this proposal, communication with WMS happen via events published in event busses. If this is not an option for integration (e.g. if WMS only has an API through which we are supposed to create and manage orders), we could introduce an Order Creation component that listens to new order events and adjusts orders in WMS.
- Similar treatment should be applied with an Order Updater component for getting order updates out of WMS if required.
- As inventory state is now being kept in Middleware, an audit component may be required to identify potential desynchronisation between WMS and Middleware. It should periodically monitor, provide alerts and attempt to fix discrepancies for safe scenarios.
- As push API is used to for receiving updates from sales channel, another audit component may be required to do similar sync between Middleware state and Sales channel (e.g. If an order was missed due to a push call failing). It can also be programmed to attempt to fix issues by publishing missed updates to relevant event bus.
- Incoming order event bus could also be split into 2 event busses if we want to reserve stock before promoting order to WMS.

This has been a logical proposal. I haven't prescribed any deployment/implementation organisation on purpose as that will depend on organisational technical strategy and scalability requirements of the business. Some example configurations:

- All components shown of the middleware could sit in a single monolith for a new startup without a lot of volume. Event busses could be implemented as database tables or even in memory queues for prototype.
- For a company with a private cloud, these components could be implemented as microservices deployed in a local Kubernetes cluster. Kafka or similar could be used for event delivery.
- For a cloud native strategy, this could be implemented as a serverless set of functions/lambdas, joined using cloud vendor pub/sub.

There are other variations possible. Such as doing synchronous processing rather than using asynchronous queues. Potential scalability of this is not as great as what is proposed. Under the current architecture, for example, if demand across a particular sales channel is very high, we could run more copies of that gateway component to cope with increased volume of requests. This would not be possible with a synchronous approach due to a need for consistency.

The following diagram shows order flow through the new system:



It shows that with a stock reservation system, available stock for sale can be adjusted well before the order is fulfilled by staff. And with no periodic syncs, information can flow between system components in parallel without delay.

Benefits of this architecture include:

- This proposed an event-based architecture. All components are decoupled, easier to develop, test and scale independently.
- This should have a faster feedback loop than periodic polling. Events are typically propagated in few hundred milliseconds through event streaming services. This should reduce feedback loop to few seconds at most.
- Vendor neutral event format will help easily add future sales channels and adopt when WMS is changed / upgraded.

Drawback of this architecture:

- It relies on event buses for integration. Whatever event streaming/queueing system is used is a new potential single point of failure.
- There are a lot more hops information flows through. Even though information flows much faster than current architecture, overall system is more complex with more system components. For example, it comes with configuration management and DevOps overhead. It can also be more expensive operationally.