# Shared Kitchens

## Assumptions

* Orders input is valid and the temperatures allowed are Hot, Cold, Frozen (made extensible though)
* Courier would arrive only after an order is ready by the kitchen and stored in the shelf
* Orders are not duplicated
* Decay rate is valid number
* (1 + (factor \* decay rate) ) > 0 always holds good

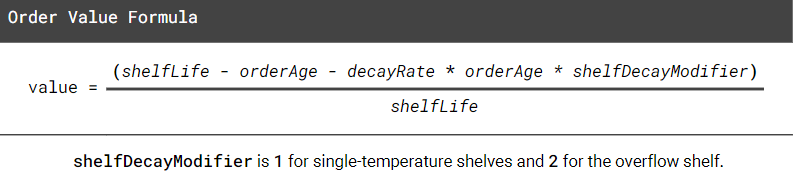
**Build and Run: Please refer ReadMe file**

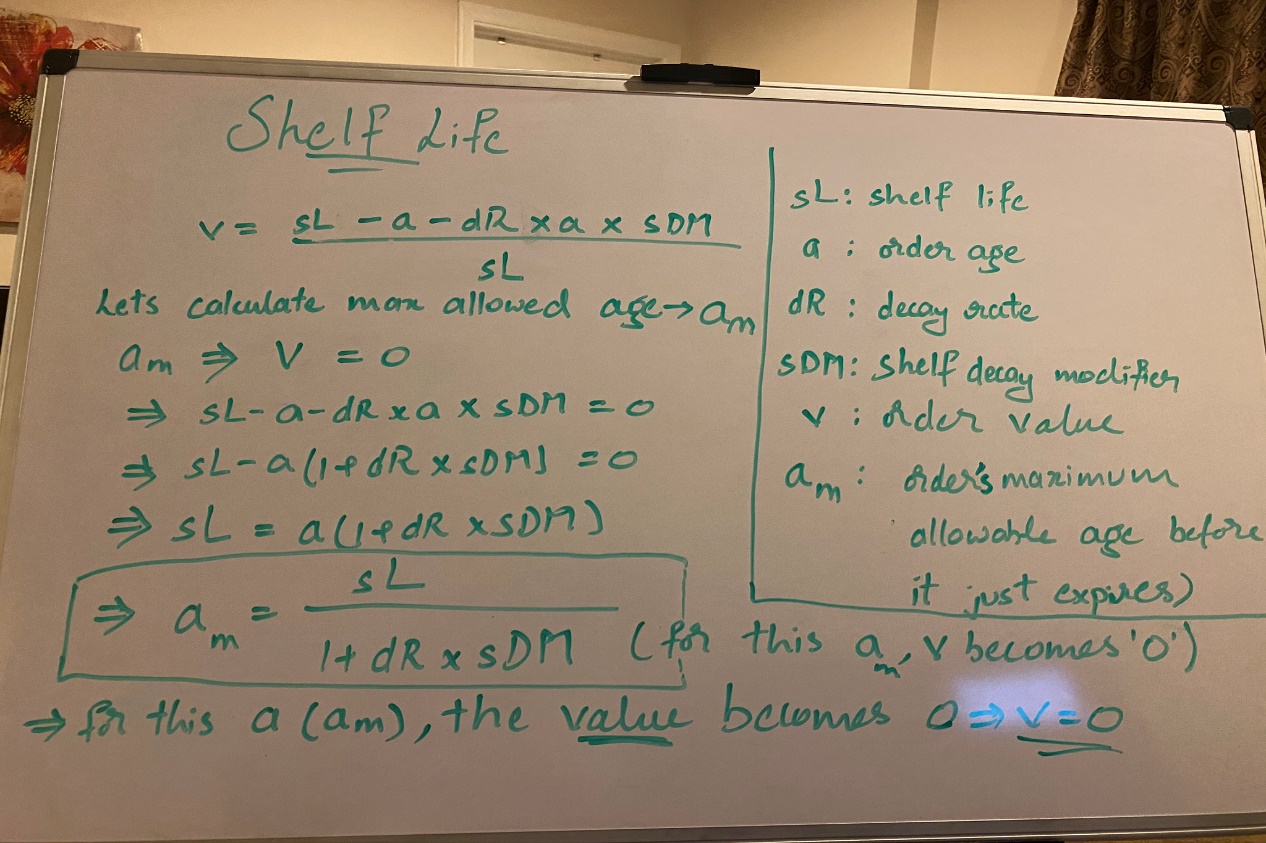
***Some Math regarding the Order Value and Order Age:***

The maximum allowable age of an order is calculated based on the given params solved in the image below:

The given max age of an order for which the *value* becomes 0 is shown below:

Given equation:



Max age of order based on the OrderValue: 

Based on this, an order is termed ***expired*** if:

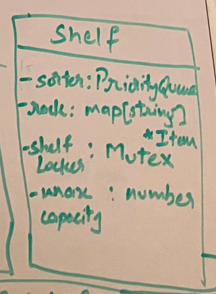
currAge: Current Age of an Order

maxAllowedAge: Maximum allowed age for an order based on above formula (am)

***currAge – maxAllowedAge >= 0***

***where currAge = Current Time - Order.CreatedTimeS*** *All in seconds units of time*

This maxAllowedAge could be used to set priority on prepared orders. Using a priority queue (using min heap) based on this priority (am), will help know which order is expiring soon or already expired in O(1) complexity. The push operation costs O(logN) and retriving an element costs O(1) with the help of a map (used as reverse index for lookup purposes) (Check **Shelf** **data structure** in the design diagram)

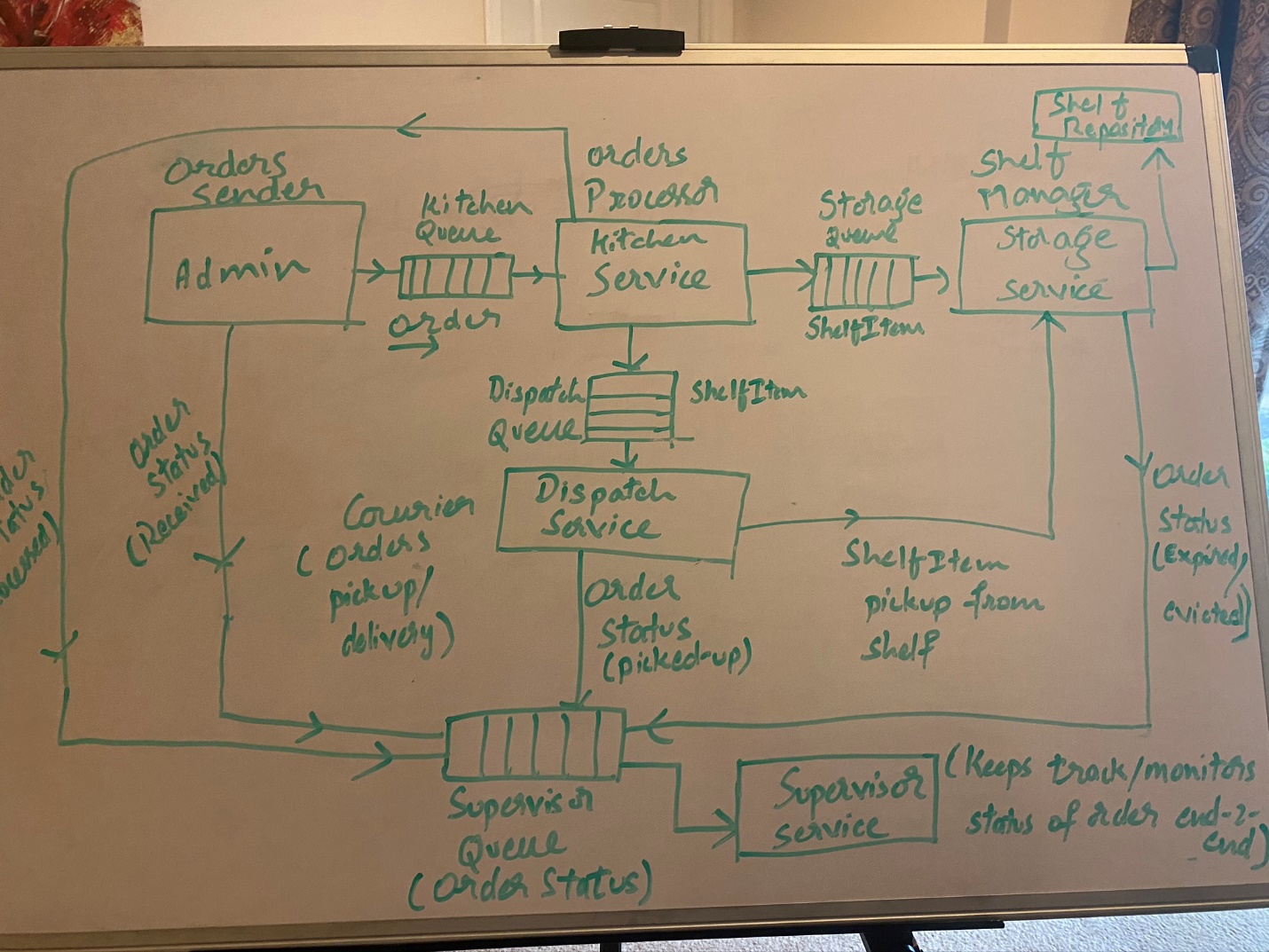


# Architecture

Divided into below components

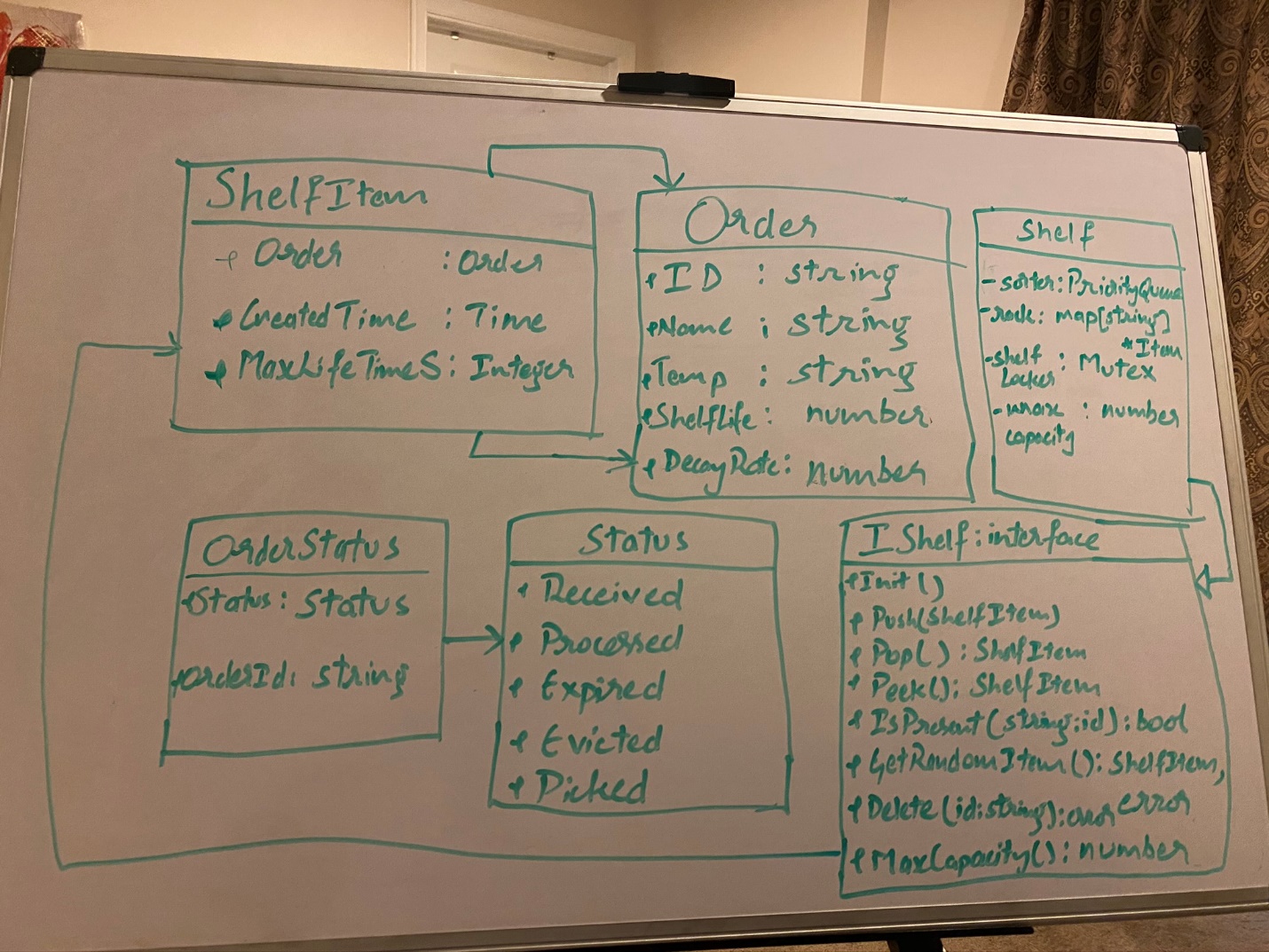
* Admin (Orders Receiver)
* Kitchen Service(Orders Processing)
* Storage Service(Shelves Manager)
* Dispatch Service (Courier)
* Supervisor Service(Supervision of order end-to-end)

The components, interaction, communication channels, messages being shared and status reporting shown in the below architecture diagram



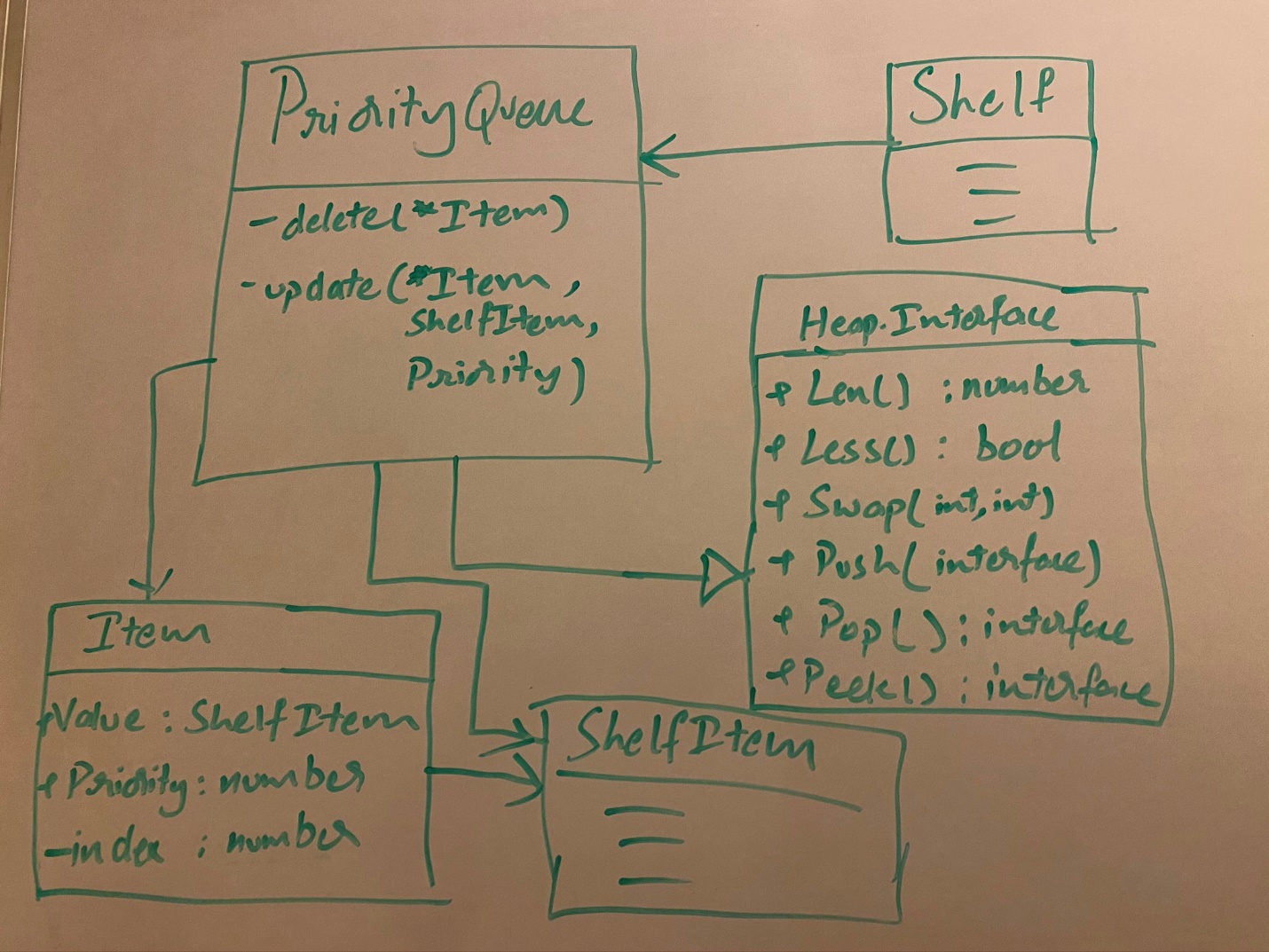
All the above components send/process interact through messages from/to buffered channels (queues) to achieve isolation between those services so they are not coupled. Differed the idea of centralized event aggregator (pub/sub service) due to time limitations. But, that’s left as a future scope of the solution

## Design



Rest of the models continued below:

Below models are used by the Storage Service



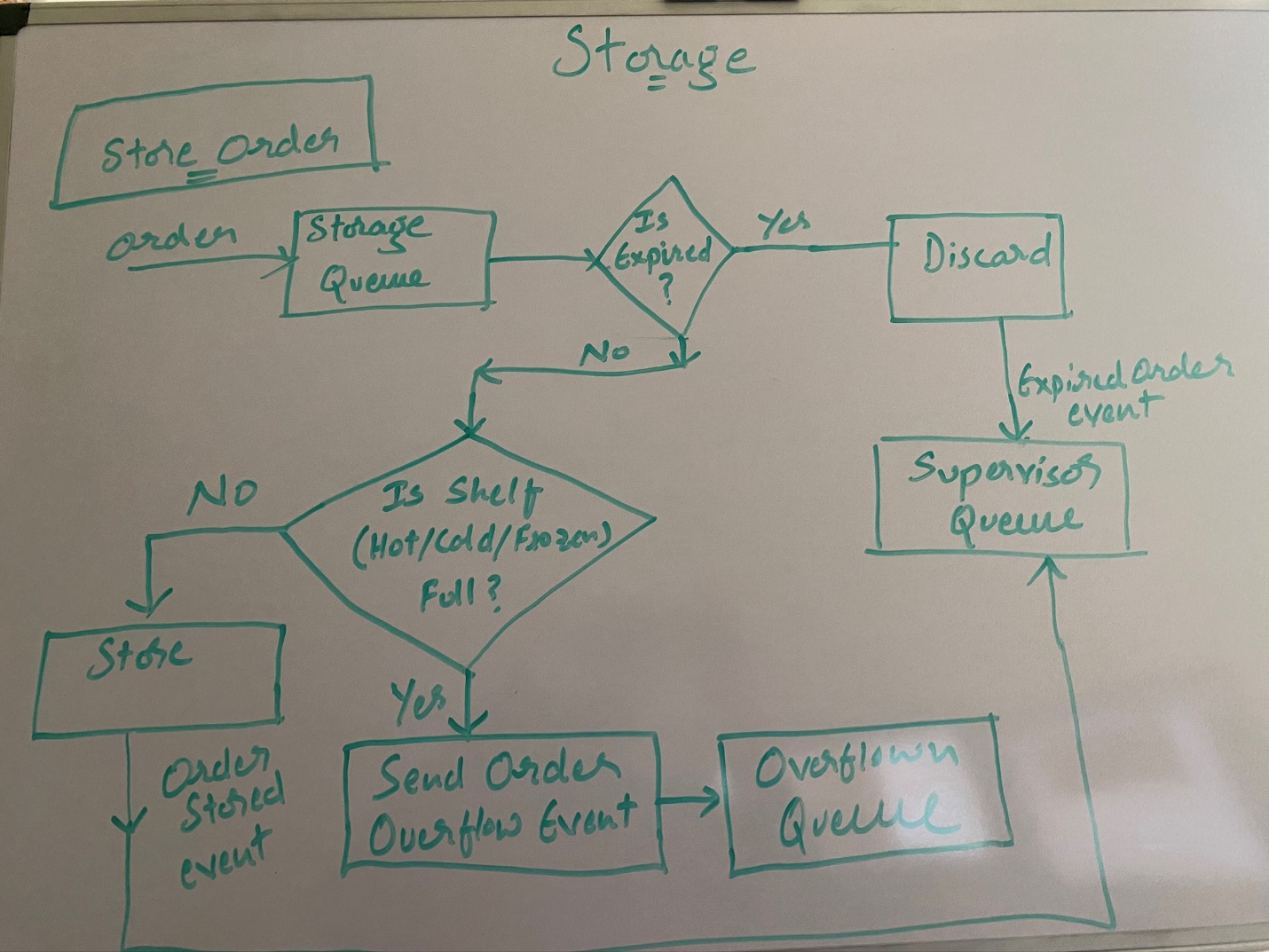
## Order Life Cycle

### Kitchen

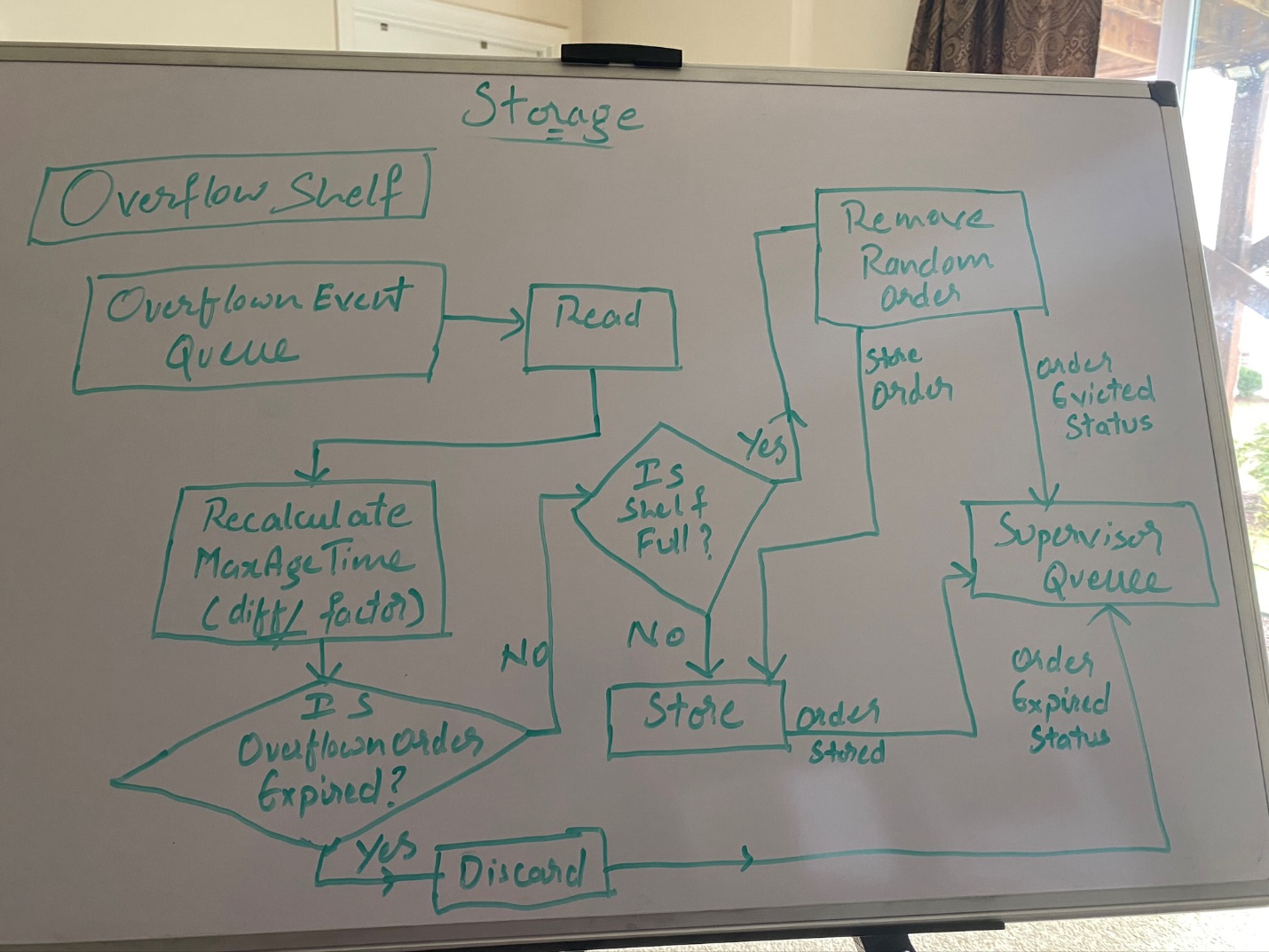
The Kitchen is straightforward. It receives order from Admin, processes it and Sends “Order Processed” event and “OrderReady” events to Storage and Dispatch Channels respectively

### Storage

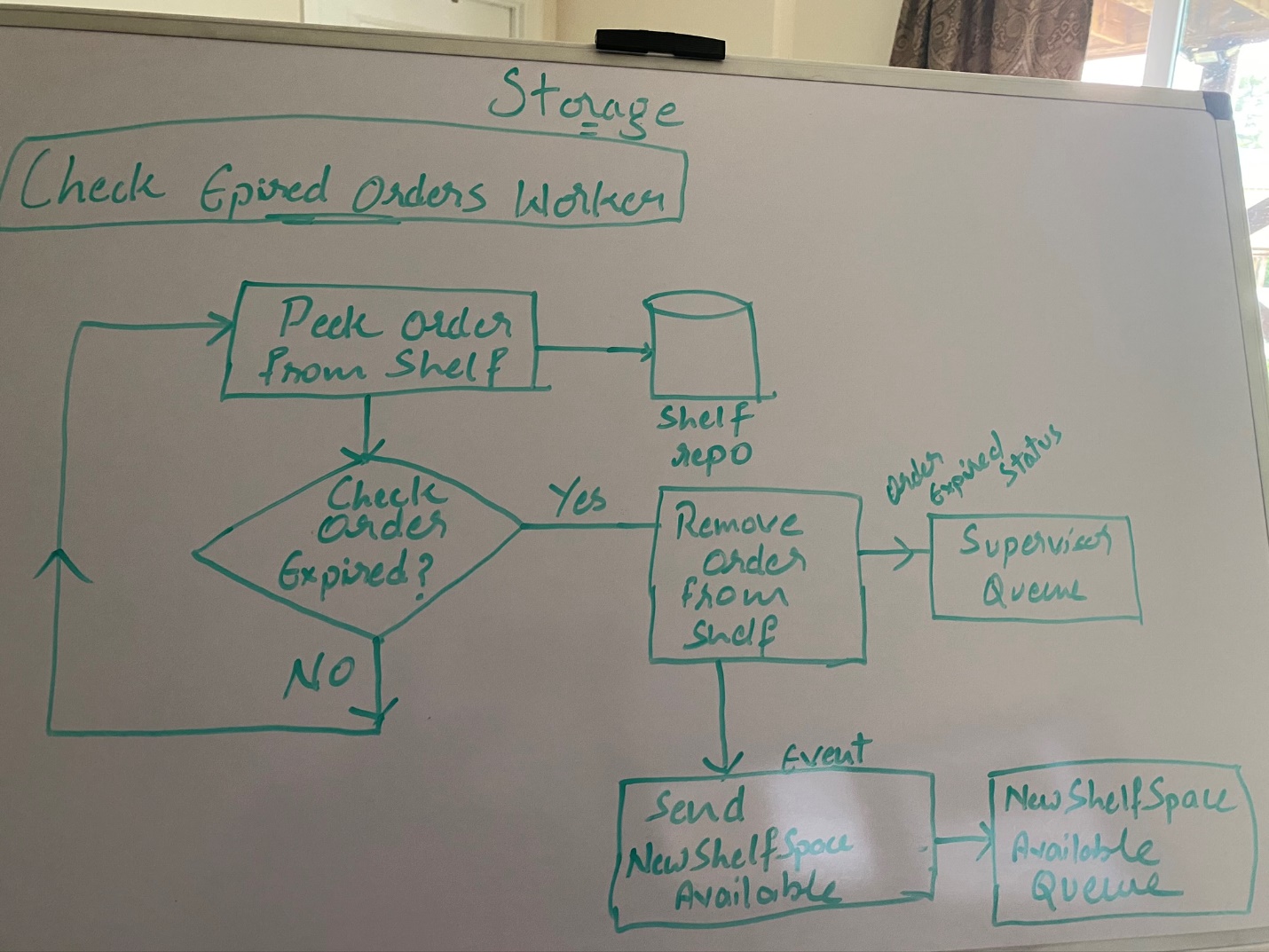
Normal Shelves (Hot/Cold/Frozen)



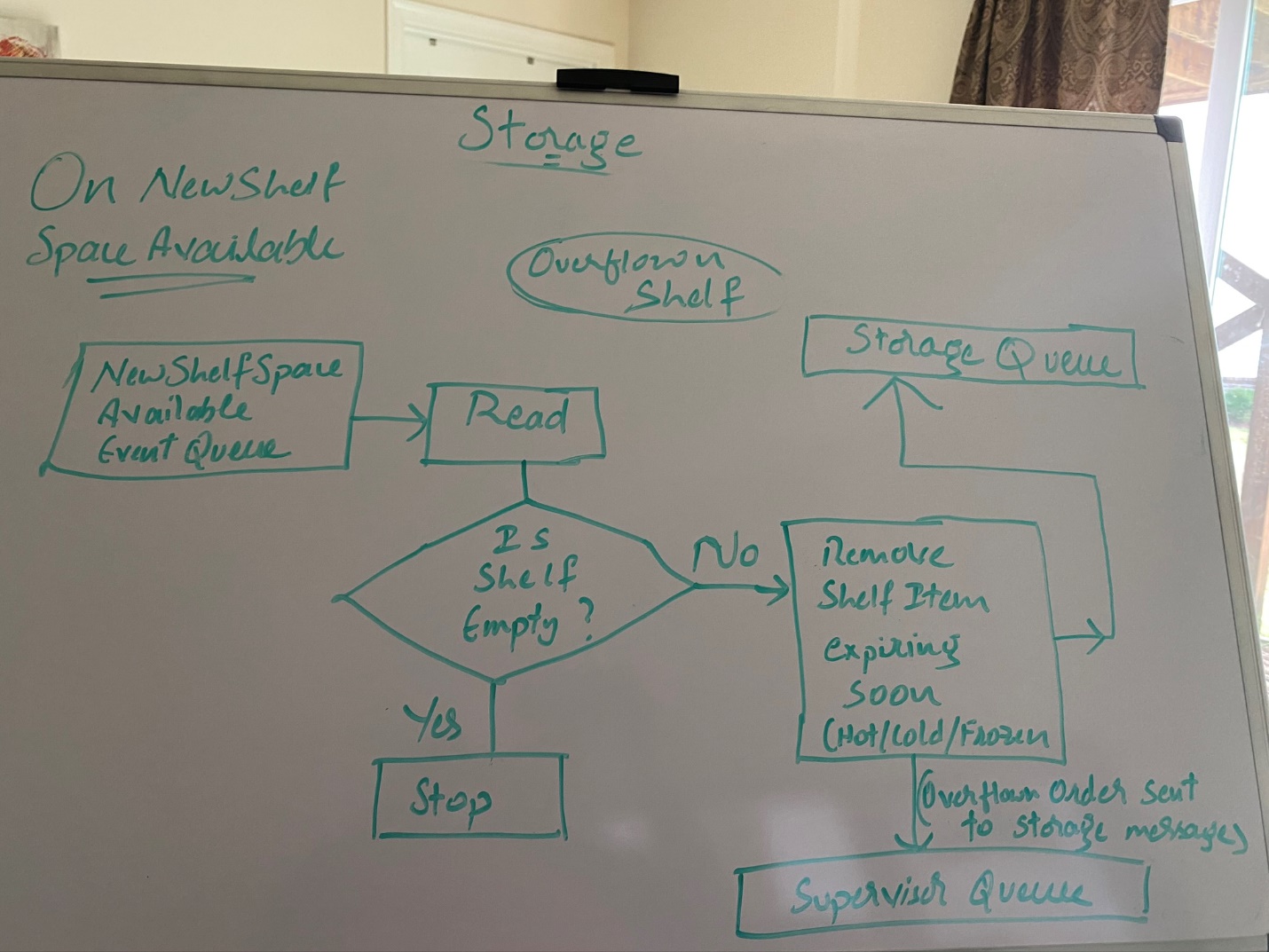
Overflow Shelf:



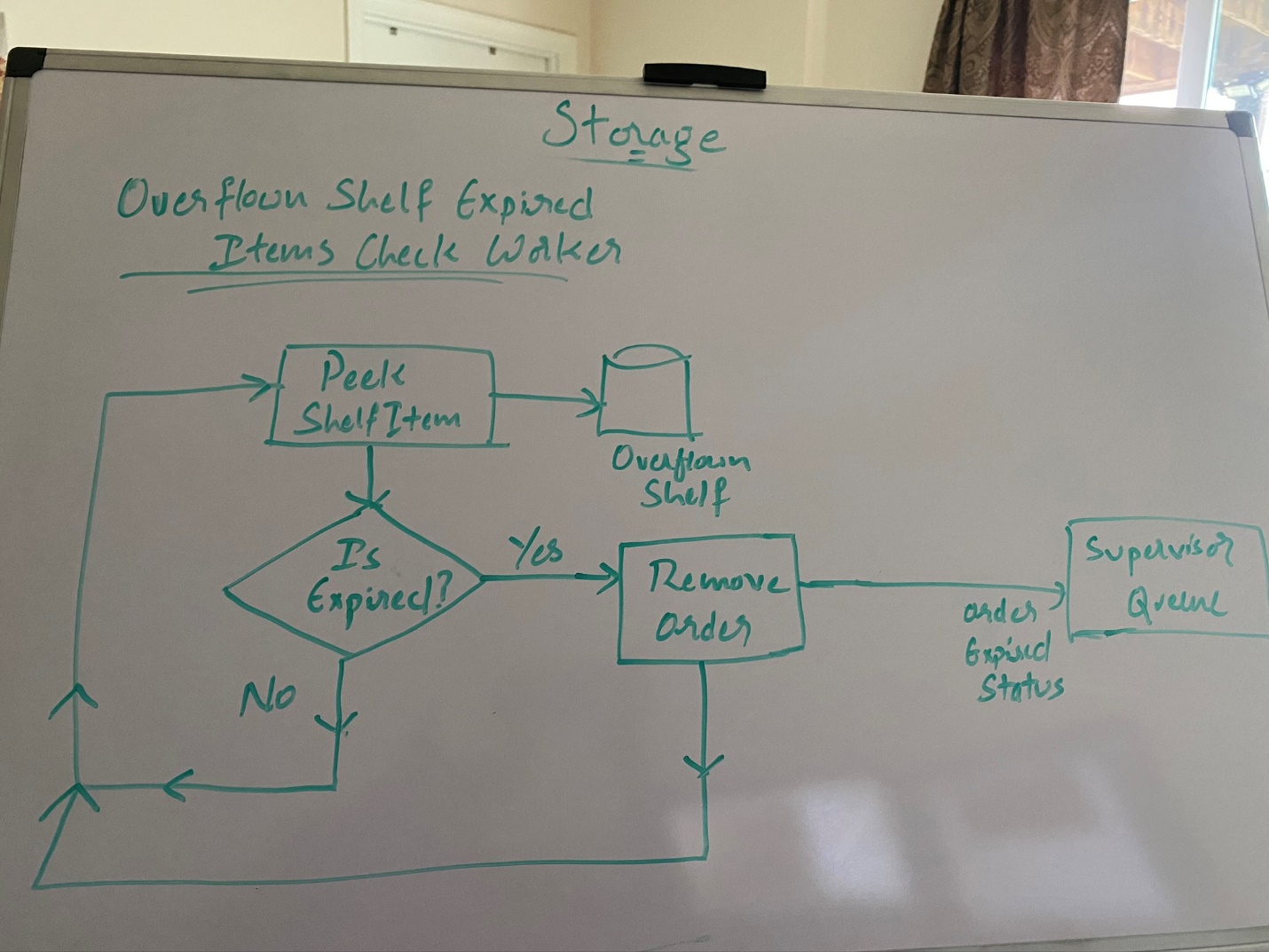
Check Expired Orders Worker – Normal Shelves (Hot/Cold/Frozen):



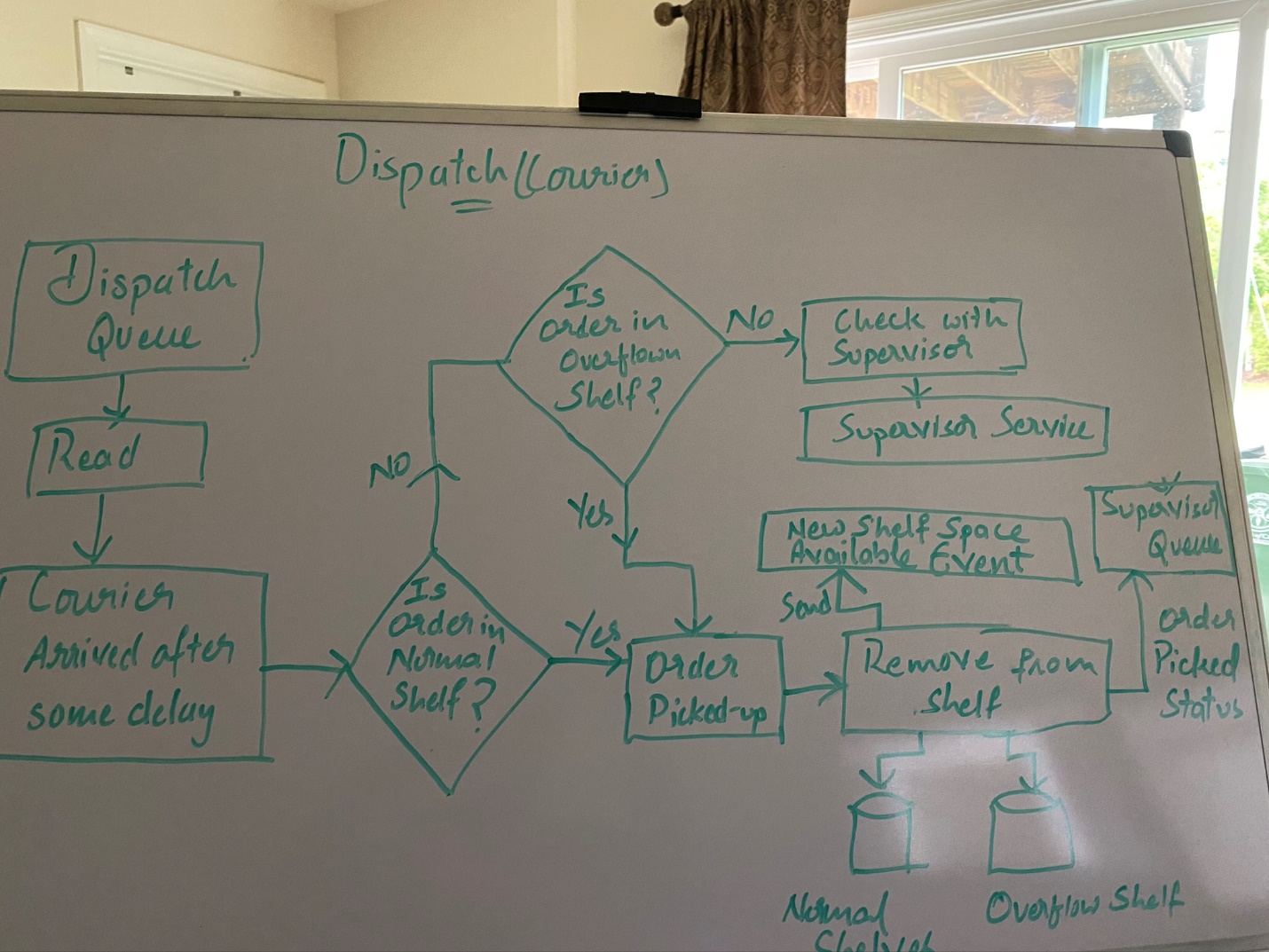
On New shelf space available event received – Overflown Shelf:



Overflown Shelf Expired Orders Checking Worker – Overflown Shelf:



### Dispatch (Courier)



If an order was not available while a courier picking up, the status is checked with Supervisor service. Supervisor service informs if that order is either evicted or expired or never even present (because it does book keeping of processed records).

The supervisor finally would be able to print the overall report of all orders processed with how many of them were expired, evicted, processed and picked-up