# Project Summary: Stock Broker DBMS

I collaborated with two other computer science students to create a stock broker database application throughout our database management systems course. We developed the system in C++ and integrated an Oracle DBMS using SQL queries.

I will be explaining the stock market simulation segment of our project as I was the lead backend developer.

**Application Overview:**

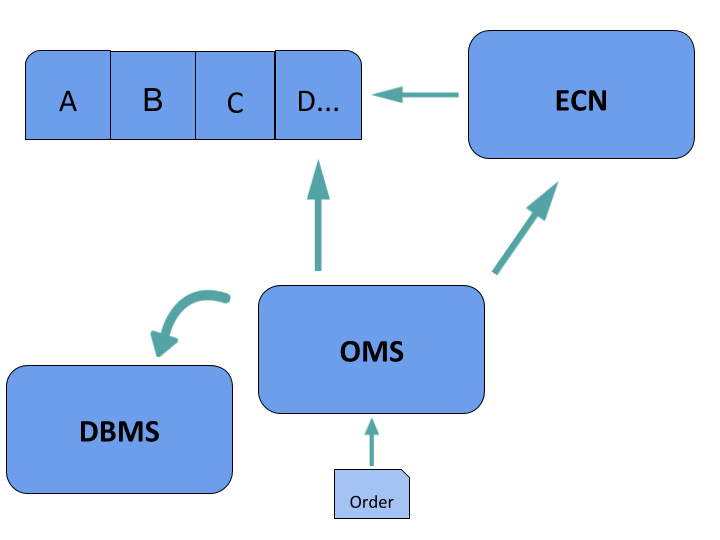
We developed an interface which allows users to:

1. Place orders in a stock market simulation for IBM common stock
2. Deposit cash into their brokerage account
3. View their account information
4. View their portfolio

## Stock Market Simulation: How it works

### Simulation Overview:

I have obtained millisecond market data for IBM stock which includes the details of every bid and ask processed throughout the entire stock market in a single day. The simulation begins by prompting the user to create an order for IBM stock. The user must also specify the time of day the order would be submitted if placed in real-time.

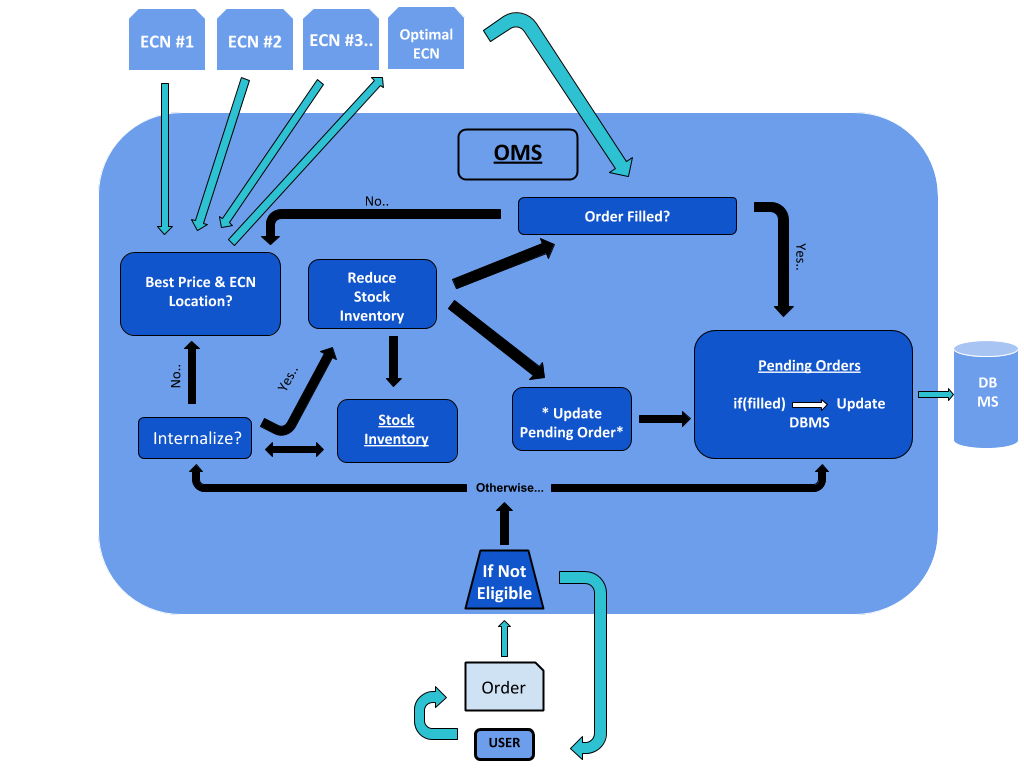


Once the order has been created, the application converts each line of the market data into an Offer, using the chrono library to typecast the string timestamp into the Offer’s millisecond timestamp. The Offer is then submitted to its corresponding ECN before parsing the next line. Each ECN will autonomously match buyers and sellers as each line is streamed, parsed into an Offer and then routed.

We submit our client’s order to the OMS once we stream the timestamp entered by the user in the market data file. The OMS then processes the order while taking the ‘current’ market conditions and internalization opportunities into consideration. Once the order has been filled, the client’s account will be updated and they will be able to see the average price they have obtained.

This is a very superficial description of the simulation. I will delve into the minute details below.

### OMS Overview:



The diagram above may provide further insight into the inner workings of the OMS.

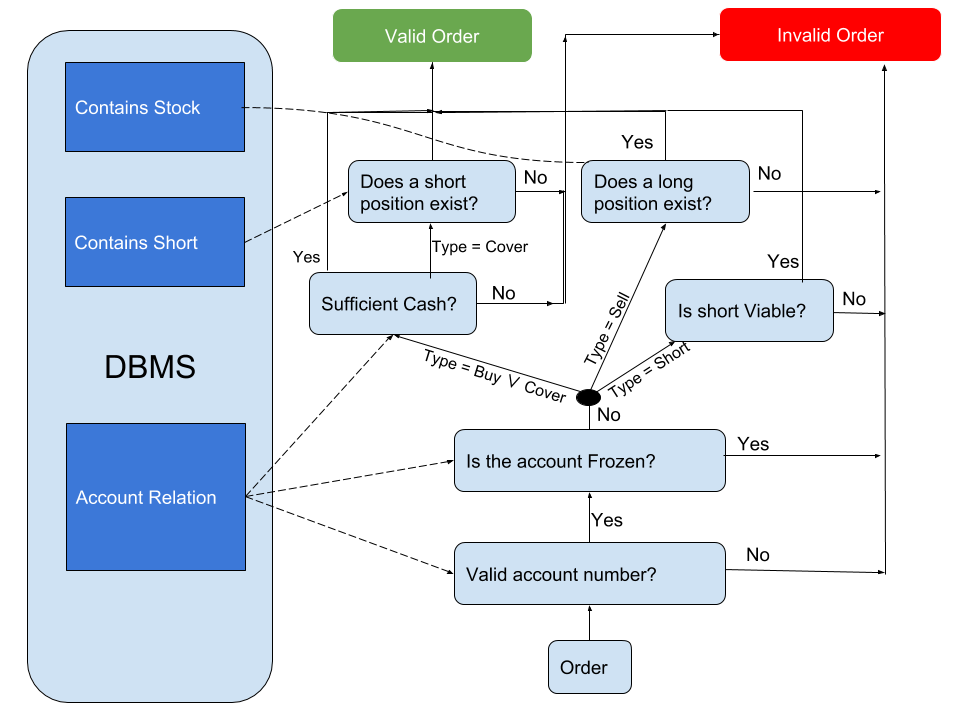
Once an order has been created, it is sent to the stock broker’s OMS (order management system) for processing. The OMS is the middleman between the client and the stock market. First, the OMS parses each order and queries the account that submitted the order to obtain the necessary financial data for determining the eligibility of the order. Eligibility evaluation considers numerous things about an order and its corresponding account (e.g. Does the account have sufficient cash balance? If the order is to sell, does the account have a long position in this stock?, etc.) If the order is eligible, it will be added to the pending orders, otherwise, it is omitted from the OMS and the user receives a notification justifying the ineligibility.

If it is both viable and profitable for the stock broker to internalize the order, the OMS updates the broker’s stock inventory the appropriate order information pending orders. If the broker internalized the entire order, the OMS will remove it from pending orders and update the account appropriately. If the order has not been completely filled, it is then routed to the optimal ECN given the current conditions of the market.

The OMS makes the appropriate changes to the DBMS once the order has been filled.

### Order Eligibility:

Here is a schematic depicting the criteria for determining order eligibility:



Note: The dotted lines above represent the transfer of data between the DBMS and application.

The order is submitted to the OMS once we have reached the timestamp of the order in the file. The first thing the OMS does is determine if the order is eligible. The order will be appended to a vector of pending orders, converted into an *Offer[[1]](#footnote-1)* and then examined for internalization if it is eligible. Otherwise, it will be discarded by the OMS and the user will be notified as to why the order is ineligible.

### Internalization Evaluation

**Background**: Stock market orders can be filled in a variety of ways by a stock broker. The options include:

1. Order to the floor
   1. Route the order to a human trader on the floor of an exchange like the NYSE
2. ECN
   1. Route the order to an Electronic Communication Network that matches buyers and sellers autonomously
3. Market Maker
   1. Route the order to a third market maker
4. Internalization
   1. Fill the order with the broker’s inventory
   2. It may be advantageous to internalize shares for numerous reasons. For instance, the broker may want to sell off a large block of shares without significantly influencing the market price.

The evaluation procedure I have implemented is simple and theoretical due to the lack of available internalization policy documentation. This internalization procedure will not consider sell orders or short orders because the criteria for making these internalization decisions is contingent upon many variables not included in our database and well beyond the scope of this project.

We assume a stock broker is not willing to internalize all the securities they own (e.g. they may reserve some shares to lend to short selling clients). Therefore, we will only consider the set of securities they are willing to internalize in the evaluation. Our OMS will internalize all orders which are able to be internalized and are profitable.

A stock broker is able to internalize an order with a security it owns if it fills the order at a price equivalent to or better than the best market price.

Here is the internalization algorithm I have developed to process an incoming order, o:

1. Query the DBMS to obtain the quantity of shares owner by the broker with ticker = o.ticker and the average price the firm paid for those shares.
2. If the quantity of shares > 0, then continue to 3. Otherwise, we cannot internalize the order.
3. If o.Price > avg price of shares owned, then internalization is profitable.
4. Now we need to determine if internalization is viable, that is, if we can internalize the order and provide the best possible price to the client. The OMS then obtains all the market prices for the security by communicating with the ECNs. If the broker’s price for the shares is lower than the best market price, then it is considered viable.
5. If it is both viable and profitable to internalize the order, the OMS will generate a price at which to internalize the order. Since the broker wants to maximize profit, we will set the price as follows:
   1. For market orders, we will fill the client’s order at $0.01 below the best market price.
   2. For limit orders, we will fill the client’s order at the price they entered.
6. Update the stock broker’s stock inventory with respect to the number of shares filled.
7. Update order information in pending orders.
8. If the order is not filled, send to an ECN.

### OMS-ECN Protocol

I developed the OMS-ECN “protocol” to transmit the order information necessary to route orders between the OMS and an ECN. The Offer object is a subset of order information which is used in conjunction with order\_info objects as a rudimentary communication framework between the OMS and each ECN.

The OMS – ECN Protocol is primarily comprised of the *Offer* and *order\_info* objects.

Offers contain the order data required to be processed by an ECN. Order\_info objects are generated by an ECN whenever an order is used in a transaction.

**Demonstration of Order\_info**

CODE:

int main(){

ifstream read\_data("/Users/thomasciha/Documents/2nd Year/CS 275/IBM\_Trades\_N\_Quotes.csv"); // note: there are 1353590 lines

string line;

getline(read\_data,line); //get header, throw it out

class ECN Test\_ECN("ECN1");

// creating sample order to test system:

hours hrs (stoi("9"));

minutes mns (stoi("45"));

milliseconds ms = duration\_cast<milliseconds> (hrs) + duration\_cast<milliseconds>(mns);

cout << " ms = " << ms.count() << endl;

Order sample\_buy(1,2, 700, Stock, "IBM", ms, Market, Day, Buy, 0);

Order sample\_ask(2,3, 500, Stock, "IBM", milliseconds(500), Market, Day, Sell, 0);

vector<class ECN \*> OMS\_Initializer;

OMS\_Initializer.push\_back(&Test\_ECN);

OMS my\_broker(OMS\_Initializer);

int count = 0;

Restricts number of offers to 15

while(getline(read\_data, line)){

Offer temp = create\_offer(line);

if(count > 200000 && count < 200015)

Test\_ECN.ParseOffer(temp);

count +=1;

}

cout << "CURRENT BIDS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

order\_info of = my\_broker.ProcessOrder(sample\_buy);

of.Print();

cout << "CURRENT BIDS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

read\_data.close();

}

OUTPUT:

**CURRENT BIDS:**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

Note: the format of the offers is [price, qty, timestamp]

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.81 200 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.59 100 37672790|**

Offers 200,000 – 200,015

**-------------------------------**

**CURRENT ASKS:**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.88 100 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

Order\_info returned from ECN illustrates the market order was completely filled at an average price of 153.86 and that 700 shares were filled at this ECN.

**Avg Price: 153.864**

**Shares Filled: 700**

**Order Status: Filled**

**CURRENT BIDS: AFTER TRADE**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

The OMS just processed a market order with a quantity of 700 shares. These are the remaining bids and asks.

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.81 200 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.59 100 37672790|**

**-------------------------------**

**CURRENT ASKS: AFTER TRADE**

**-------------------------------**

**| 153.88 100 37672789|**

**-------------------------------**

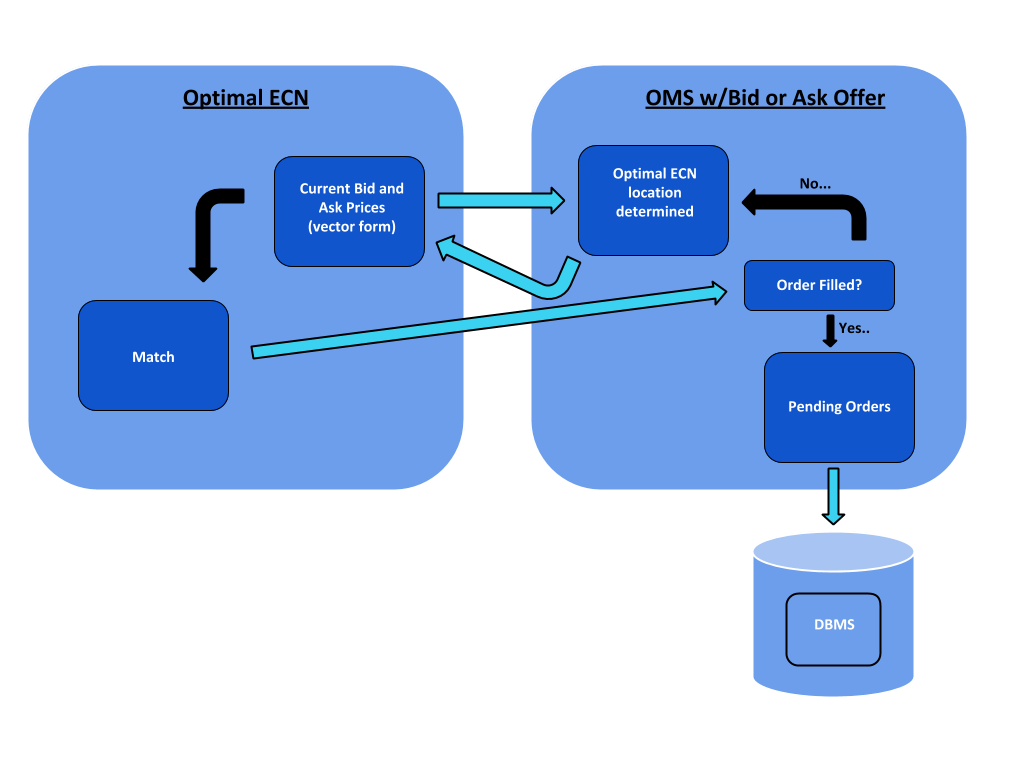
**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

**Program ended with exit code: 0**

### ECN: How it works



Current\_bids and current\_asks are vectors of offers that store the bids in descending order and asks in ascending order. Together, the current bid and current ask vectors constitute a market. Each ECN is comprised of many markets, one market for each stock; however, since we were only able to obtain market data for IBM, each ECN has one market in this project.

## Troubleshooting:

### Using Pointers to update ECNs:

This code streams the time-series market data for IBM and converts it into Offer objects. Those Offers are then added to the ECN object Test\_ECN, which will autonomously match bids and asks based on their parameters. Orders are sent from the user to the stock broker’s OMS (Order Management System). In reality, a stock broker has many venues available for which they can submit an order for processing. The OMS attribute Available\_ECNS is representative of these venues. As we parse the data, we will add offers to their respective ECNS (in this example there’s only one ECN). The OMS then should be able to obtain the market data of each ECN. However, the output of the code below shows this does not occur. This fails to happen because a copy of the ECN, including all bids and asks, is used to initialize the OMS object. Hence, the OMS object will only have access to the market bids and asks that were present in each ECN at the time at which the OMS was initialized.

CODE:

int main(){

ifstream read\_data("/Users/thomasciha/Documents/2nd Year/CS 275/IBM\_Trades\_N\_Quotes.csv"); // note: there are 1353590 lines

string line;

getline(read\_data,line); //get header, throw it out

class ECN Test\_ECN("ECN1");

// creating sample order to test system

hours hrs (stoi("9"));

minutes mns (stoi("45"));

milliseconds ms = duration\_cast<milliseconds> (hrs) + duration\_cast<milliseconds>(mns);

cout << " ms = " << ms.count() << endl;

Order sample\_order(1,2, 1000, Stock, "IBM", ms, Market, Day, Buy, 0);

Order sample\_ask(2,3, 500, Stock, "IBM", milliseconds(500), Market, Day, Sell, 0);

vector<class ECN> OMS\_Initializer;

OMS\_Initializer.push\_back(Test\_ECN);

OMS my\_broker(OMS\_Initializer);

int count = 0;

while(count < 50 && getline(read\_data, line)){

Offer temp = create\_offer(line);

Test\_ECN.ParseOffer(temp);

count +=1;

}

cout << Test\_ECN.market.current\_bids.size() << endl;

cout << Test\_ECN.market.current\_asks.size() << endl;

cout << "CURRENT BIDS: " << endl;

my\_broker.available\_ECNs.begin()->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: " << endl;

my\_broker.available\_ECNs.begin()->market.Print\_Offers(Ask);

//nothing shows up in the OMS! ^^

read\_data.close();

}

OUTPUT:

**ms = 35100000**

**25**

**25**

**CURRENT BIDS:**

**CURRENT ASKS:**

**Program ended with exit code: 0**

Proposal:

We want the Available\_ECNS bid and ask values to update dynamically. Instead of copying the ECN to members of the OMS, we can utilize ECN pointers as variables that point to the memory of each ECN. We cannot create a vector of references because references cannot be re-assigned and must be assigned upon initialization. References cannot be assigned NULL like pointers.

NEW CODE:

int main(){

ifstream read\_data("/Users/thomasciha/Documents/2nd Year/CS 275/IBM\_Trades\_N\_Quotes.csv"); // note: there are 1353590 lines

string line;

getline(read\_data,line); //get header, throw it out

class ECN Test\_ECN("ECN1");

// creating sample order to test system:

hours hrs (stoi("9"));

minutes mns (stoi("45"));

milliseconds ms = duration\_cast<milliseconds> (hrs) + duration\_cast<milliseconds>(mns);

cout << " ms = " << ms.count() << endl;

Order sample\_order(1,2, 1000, Stock, "IBM", ms, Market, Day, Buy, 0);

Order sample\_ask(2,3, 500, Stock, "IBM", milliseconds(500), Market, Day, Sell, 0);

vector<class ECN \*> OMS\_Initializer;

OMS\_Initializer.push\_back(&Test\_ECN);

OMS my\_broker(OMS\_Initializer);

int count = 0;

while(count < 50 && getline(read\_data, line)){

Offer temp = create\_offer(line);

Test\_ECN.ParseOffer(temp);

count +=1;

}

cout << Test\_ECN.market.current\_bids.size() << endl;

cout << Test\_ECN.market.current\_asks.size() << endl;

cout << "CURRENT BIDS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

read\_data.close();

}

OUTPUT:

**ms = 35100000**

**25**

**25**

**CURRENT BIDS:**

**-------------------------------**

**| 152.82 400 16622986|**

**-------------------------------**

**-------------------------------**

**| 152.79 400 16385336|**

**-------------------------------**

**-------------------------------**

**| 152.79 400 16414380|**

**-------------------------------**

**-------------------------------**

**| 152.76 400 16150210|**

**-------------------------------**

**-------------------------------**

**| 152.76 400 16174540|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14400125|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14400142|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14404739|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405457|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405457|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405457|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405458|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405928|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14405929|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14406065|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 14406066|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15209642|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15209643|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15588410|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15590461|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15591571|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 15591572|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 16051313|**

**-------------------------------**

**-------------------------------**

**| 152.73 400 16051793|**

**-------------------------------**

**-------------------------------**

**| 152.59 100 14400012|**

**-------------------------------**

**CURRENT ASKS:**

**-------------------------------**

**| 156.04 400 16414380|**

**-------------------------------**

**-------------------------------**

**| 156.04 400 16622986|**

**-------------------------------**

**-------------------------------**

**| 156.07 400 16174540|**

**-------------------------------**

**-------------------------------**

**| 156.07 400 16385336|**

**-------------------------------**

**-------------------------------**

**| 156.1 400 15591572|**

**-------------------------------**

**-------------------------------**

**| 156.1 300 16051313|**

**-------------------------------**

**-------------------------------**

**| 156.1 400 16051793|**

**-------------------------------**

**-------------------------------**

**| 156.1 400 16150210|**

**-------------------------------**

**-------------------------------**

**| 156.12 100 15591571|**

**-------------------------------**

**-------------------------------**

**| 156.28 400 15209643|**

**-------------------------------**

**-------------------------------**

**| 156.28 300 15588410|**

**-------------------------------**

**-------------------------------**

**| 156.28 400 15590461|**

**-------------------------------**

**-------------------------------**

**| 156.3 100 15209642|**

**-------------------------------**

**-------------------------------**

**| 156.31 400 14406066|**

**-------------------------------**

**-------------------------------**

**| 161 100 14405457|**

**-------------------------------**

**-------------------------------**

**| 157.64 200 14405457|**

**-------------------------------**

**-------------------------------**

**| 156.32 200 14400142|**

**-------------------------------**

**-------------------------------**

**| 156.32 400 14404739|**

**-------------------------------**

**-------------------------------**

**| 156.32 200 14405457|**

**-------------------------------**

**-------------------------------**

**| 156.33 100 14406065|**

**-------------------------------**

**-------------------------------**

**| 156.34 100 14400012|**

**-------------------------------**

**-------------------------------**

**| 156.34 100 14400125|**

**-------------------------------**

**-------------------------------**

**| 157.65 400 14405458|**

**-------------------------------**

**-------------------------------**

**| 157.66 400 14405929|**

**-------------------------------**

**-------------------------------**

**| 161 100 14405928|**

**-------------------------------**

**Program ended with exit code: 0**

However, we need to be careful with vector<class ECN \*> OMS\_Initializer; The vector contains pointers which point to an ECN object. If we delete one of the ECNs outside of this vector, the pointer pointing to the ECN which was deleted will be invalid.

## Partially Filled Market Order:

* In the rare event there are not enough bids or asks to fulfill a market order, the ECN will fill as many shares as possible and then return order\_info with number of shares filled, avg price obtained, and a flag indicating that the market order should be routed to another ECN.

Here’s the code:

int main(){

ifstream read\_data("/Users/thomasciha/Documents/2nd Year/CS 275/IBM\_Trades\_N\_Quotes.csv"); // note: there are 1353590 lines

hours hrs (stoi("9"));

minutes mns (stoi("45"));

milliseconds ms = duration\_cast<milliseconds> (hrs) + duration\_cast<milliseconds>(mns);

string line;

getline(read\_data,line); //get header, throw it out

class ECN Test\_ECN("ECN1");

// creating sample order to test system:

Order sample\_buy(1,2, 700, Stock, "IBM", ms, Market, Day, Buy, 0);

Order sample\_ask(2,3, 5750, Stock, "IBM", milliseconds(500), Market, Day, Sell, 153.80);

vector<class ECN \*> OMS\_Initializer;

OMS\_Initializer.push\_back(&Test\_ECN);

OMS my\_broker(OMS\_Initializer);

int count = 0;

while(getline(read\_data, line)){

Offer temp = create\_offer(line);

if(count > 200000 && count < 200015)

Test\_ECN.ParseOffer\_TEST(temp);

count +=1;

}

cout << "CURRENT BIDS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

vector<order\_info> of = my\_broker.ProcessOrder(sample\_ask);

cout << "CURRENT BIDS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

for(auto it : of){

it.Print();

}

read\_data.close();

}

Here is the OUTPUT:

**CURRENT BIDS:**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.81 200 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.59 100 37672790|**

**-------------------------------**

**CURRENT ASKS:**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.88 100 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

**CURRENT BIDS: AFTER TRADE**

**CURRENT ASKS: AFTER TRADE**

**-------------------------------**

**| 0 4950 500|**

**-------------------------------**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.88 100 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

**Program ended with exit code: 0**

The problem here is that there is insufficient market liquidity to fill the market order. Since there are no bids left, the system defaults to inserting the market order into the current asks with a price of 0 (since it’s a sell market order). This is illogical. A better solution would be to throw a flag when a market order is unfulfilled at a particular ECN and then communicate this with the OMS so the market order can be rerouted to another venue.

Here is the updated output after the trade was made:

**CURRENT BIDS: AFTER TRADE**

**CURRENT ASKS: AFTER TRADE**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.88 100 37672789|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

**ORDER ID: 2**

**Avg Price: 153.84**

**Shares Filled: 100**

**Order Status: Pending**

**incomplete\_market\_order : true**

## Testing the order\_info returned from bids and asks used when processing orders

* First we display the current bids and asks at the current state of the market
* Remember, the format of the offers is [price, qty, timestamp]

**CURRENT BIDS:**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**CURRENT ASKS:**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

* Then we add three buy orders routed from the OMS to the ECN. They have the timestamps 100, 200 and 300.

**CURRENT BIDS: RIGHT BEFORE PROCESSING ASK**

**-------------------------------**

**| 153.85 100 100|**

**-------------------------------**

**-------------------------------**

**| 153.84 200 200|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.83 300 300|**

**-------------------------------**

**CURRENT ASKS: RIGHT BEFORE PROCESSING ASK**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

* Then we submit a sell market order with a quantity of 25 shares (it has order number 2)
* The “erasing order num..” output is generated within the OMS in the update\_orders function. This is how the OMS processes the order\_info returned by the ECN subsequent to submitting an order.
* The OMS uses this order\_info to update the order information and will add the information below “HERE IS THE ORDER TABLE UPDATE” to the DBMS

**erasing order number: 2 from pending orders**

**Before erasing, here's the order\_info of that order:**

**ORDER ID: 2**

**Avg Price: 153.85**

**Shares Filled: 25**

**Order Status: Filled**

**incomplete\_market\_order: false**

**HERE IS ORDER TABLE UPDATE:**

**-------- Order ID: 2 --------**

**Fill Price: 153.85**

**Entered Price: 153.8**

**Size: 25**

**Timestamp: 500**

**Order Action: Sell**

**Order Type: Market**

**Order Status: Filled**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

* Here is the state of the market subsequent to the market order which was just processed

**CURRENT BIDS: AFTER TRADE**

**-------------------------------**

**| 153.85 75 100|**

**-------------------------------**

**-------------------------------**

**| 153.84 200 200|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.83 300 300|**

**-------------------------------**

**CURRENT ASKS: AFTER TRADE**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

* This is the information returned by the ECN from submitting the three bids (sam\_buy1,2,3) and the market sell\_order.
* The avg price of -1 and shares filled of 0 in the buy order\_info objects indicate the orders were unable to be filled at all in the current market conditions.

**order\_info: sam\_buy1-----**

**ORDER ID: 1**

**Avg Price: -1**

**Shares Filled: 0**

**Order Status: Pending**

**incomplete\_market\_order: false**

**order\_info: sam\_buy2-----**

**ORDER ID: 100009**

**Avg Price: -1**

**Shares Filled: 0**

**Order Status: Pending**

**incomplete\_market\_order: false**

**order\_info: sam\_buy3-----**

**ORDER ID: 100010**

**Avg Price: -1**

**Shares Filled: 0**

**Order Status: Pending**

**incomplete\_market\_order: false**

**order\_info: sam\_ask-----**

**ORDER ID: 1**

**Avg Price: 153.85**

**Shares Filled: 25**

**Order Status: Pending**

**incomplete\_market\_order: false**

**ORDER ID: 2**

**Avg Price: 153.85**

**Shares Filled: 25**

**Order Status: Filled**

**incomplete\_market\_order: false**

* Here are the pending orders remaining in the OMS. The fill price for Order ID = 1 is extremely high because we compute the average price by multiplying the avg\_fill price of each respective order\_info object by the quantity that was filled. At this point in time, the order with ID = 1 is not filled, hence, the fill price is not accurate.
* The fill price of 0 for the other two orders indicates neither have been filled at all.

**Displaying pending\_orders:**

**-------- Order ID: 1 --------**

**Fill Price: 3846.25**

**Entered Price: 153.85**

**Size: 100**

**Timestamp: 100**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Pending**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

**-------- Order ID: 100009 --------**

**Fill Price: 0**

**Entered Price: 153.84**

**Size: 200**

**Timestamp: 200**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Pending**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

**-------- Order ID: 100010 --------**

**Fill Price: 0**

**Entered Price: 153.83**

**Size: 300**

**Timestamp: 300**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Pending**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

* Next we submit another market sell order for 100 shares. This order will eliminate 25 shares from the second best bid and the rest of the best bid since the best bid only comprises 75 shares.
* Since the best\_bid (order ID 1) has had its 75 remaining shares filled, it is removed from pending orders by the OMS and an order table update is conducted.

**erasing order number: 1 from pending orders**

**Before erasing, here's the order\_info of that order:**

**ORDER ID: 1**

**Avg Price: 153.85**

**Shares Filled: 75**

**Order Status: Filled**

**incomplete\_market\_order: false**

**HERE IS ORDER TABLE UPDATE:**

**-------- Order ID: 1 --------**

**Fill Price: 153.85**

**Entered Price: 153.85**

**Size: 100**

**Timestamp: 100**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Filled**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

* Since the market sell order (order ID 25) was completely filled, we remove it from pending orders as well and update the DBMS.

**erasing order number: 25 from pending orders**

**Before erasing, here's the order\_info of that order:**

**ORDER ID: 25**

**Avg Price: 153.848**

**Shares Filled: 100**

**Order Status: Filled**

**incomplete\_market\_order: false**

**HERE IS ORDER TABLE UPDATE:**

**-------- Order ID: 25 --------**

**Fill Price: 153.848**

**Entered Price: 153.84**

**Size: 100**

**Timestamp: 600**

**Order Action: Sell**

**Order Type: Market**

**Order Status: Filled**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

**-** This is the state of the market after processing the second sell order

**CURRENT BIDS: AFTER TRADE 2**

**-------------------------------**

**| 153.84 175 200|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.84 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.83 300 300|**

**-------------------------------**

**CURRENT ASKS: AFTER TRADE 2**

**-------------------------------**

**| 153.86 200 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.87 100 37672788|**

**-------------------------------**

**-------------------------------**

**| 153.9 100 37672788|**

**-------------------------------**

* Here is the order\_info returned from submitting the second sell order.
* The object with order\_ID 1 indicates that the sell order had filled 75 shares of order ID 1 at an avg price of 153.85. The order status indicates the order has been completely filled.
* Order\_info object with order ID 100009 indicates that 25 shares were just filled at an average price of 153.84
* Order\_info object with order ID 25 corresponds to the sell order which was just processed and indicates the entire order has been filled at an average 153.848. This is because it was filled by 25 shares of order number 100009 at a price of 153.84 and 75 shares of order number 1 with a price of 153.85.

**Sam Ask 2**

**ORDER ID: 1**

**Avg Price: 153.85**

**Shares Filled: 75**

**Order Status: Filled**

**incomplete\_market\_order: false**

**ORDER ID: 100009**

**Avg Price: 153.84**

**Shares Filled: 25**

**Order Status: Pending**

**incomplete\_market\_order: false**

**ORDER ID: 25**

**Avg Price: 153.848**

**Shares Filled: 100**

**Order Status: Filled**

**incomplete\_market\_order: false**

**Displaying Pending orders again:**

**Displaying pending\_orders:**

**-------- Order ID: 100009 --------**

**Fill Price: 3846**

**Entered Price: 153.84**

**Size: 200**

**Timestamp: 200**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Pending**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

**-------- Order ID: 100010 --------**

**Fill Price: 0**

**Entered Price: 153.83**

**Size: 300**

**Timestamp: 300**

**Order Action: Buy**

**Order Type: Limit**

**Order Status: Pending**

**Ticker: IBM**

**Order Term: Day**

**--------------------------------**

**Program ended with exit code: 0**

CODE:

int main(){

ifstream read\_data("/Users/thomasciha/Documents/2nd Year/CS 275/IBM\_Trades\_N\_Quotes.csv"); // note: there are 1353590 lines

hours hrs (stoi("9"));

minutes mns (stoi("45"));

milliseconds ms = duration\_cast<milliseconds> (hrs) + duration\_cast<milliseconds>(mns);

string line;

getline(read\_data,line); //get header, throw it out

class ECN Test\_ECN("ECN1");

// creating sample order to test system:

Order sample\_buy(1,2, 100, Stock, "IBM", milliseconds(100), Limit, Day, Buy, 153.85);

Order sample\_buy2(100009,1, 200, Stock, "IBM", milliseconds(200), Limit, Day, Buy, 153.84);

Order sample\_buy3(100010,4, 300, Stock, "IBM", milliseconds(300), Limit, Day, Buy, 153.83);

Order sample\_ask(2,3, 25, Stock, "IBM", milliseconds(500), Market, Day, Sell, 153.80);

Order sample\_ask2(25,30, 100, Stock, "IBM", milliseconds(600), Market, Day, Sell, 153.84);

vector<class ECN \*> OMS\_Initializer;

OMS\_Initializer.push\_back(&Test\_ECN);

OMS my\_broker(OMS\_Initializer);

int count = 0;

while(getline(read\_data, line) && count < 200010){

Offer temp = create\_offer(line);

if(count > 200000 && count < 200015)

Test\_ECN.ParseOffer\_TEST(temp);

count +=1;

}

cout << "CURRENT BIDS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: " << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

vector<order\_info> sam\_buy1, sam\_buy2, sam\_buy3, sam\_ask, sam\_ask2;

sam\_buy1 = my\_broker.ProcessOrder(sample\_buy);

sam\_buy2 = my\_broker.ProcessOrder(sample\_buy2);

sam\_buy3 = my\_broker.ProcessOrder(sample\_buy3);

cout << "CURRENT BIDS: RIGHT BEFORE PROCESSING ASK" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: RIGHT BEFORE PROCESSING ASK" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

sam\_ask = my\_broker.ProcessOrder(sample\_ask);

cout << "CURRENT BIDS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: AFTER TRADE" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

vector<pair<vector<order\_info>, string>> all\_order\_data;

all\_order\_data.push\_back(make\_pair(sam\_buy1, "order\_info: sam\_buy1-----"));

all\_order\_data.push\_back(make\_pair(sam\_buy2, "order\_info: sam\_buy2-----"));

all\_order\_data.push\_back(make\_pair(sam\_buy3, "order\_info: sam\_buy3-----"));

all\_order\_data.push\_back(make\_pair(sam\_ask, "order\_info: sam\_ask-----"));

for(int i = 0; i < all\_order\_data.size(); i++){

cout << all\_order\_data[i].second << endl;

for(int j = 0; j < all\_order\_data[i].first.size(); j++){

all\_order\_data[i].first[j].Print();

}

cout << endl;

}

my\_broker.DisplayPendingOrders();

cout << "\n\n" << endl;

sam\_ask2 = my\_broker.ProcessOrder(sample\_ask2);

cout << "CURRENT BIDS: AFTER TRADE 2" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Bid);

cout << "CURRENT ASKS: AFTER TRADE 2" << endl;

my\_broker.available\_ECNs[0]->market.Print\_Offers(Ask);

cout << "Sam Ask 2" << endl;

for(auto i : sam\_ask2){

i.Print();

}

cout << "Displaying Pending orders again: " << endl;

my\_broker.DisplayPendingOrders();

cout << "\n\n" << endl;

read\_data.close();

}

1. An *Offer* is part of the OMS – ECN Protocol. See documentation section: OMS – ECN Protocol [↑](#footnote-ref-1)