# Summary of
# Simulating Design & Results

Kenny 10/17/2017

## Notations:

**T**: time interval between time epochs

**R**: the bouncing range of mean valuation in time epochs

**N**: customer number

**sig**: the sigma in Gaussian function. Smaller the sigma is, closer valuations are.

**Ps**: the real-time price of smart token in the market

**Psc**: the price of smart token, broadcasted at every beginning of time slot

## Bancor Market:

The whole simulating time is comprised of **1000** time slots.

In every time slot, Bancor Market processes orders launched by **N** customers.

**Valuation Making -> Transaction Launching -> Transaction Processing**

## Valuation Making in Bancor:

At the beginning of every time slot, **N** customers will be announced about **Psc**, i.e., the current price of smart token.

Based on this price, customers will generate valuations of smart token in Gaussian distribution with **mu** and **sigma.**

**Normal**: **mu** = **Psc,** => 50% valuations larger than **Psc**; while 50% smaller

**In Time Epoch: mu =** random prick from **(Psc/R**, **Psc*R )**

## Code about Random Pick:

```
# getrandbits(1) return False or True randomly
if bool(random.getrandbits(1)):
    valuation_mu = random.uniform(Psc/R, Psc)
else:
    valuation_mu = random.uniform(Psc, Psc*R)
```

By code, the valuation has 50% probability to be larger than Psc; while 50% probability to be smaller.

## Transaction Generating in Bancor:

After customers making their valuations of smart token, they will launch transaction orders with several stipulations:

1. If valuation > **Psc,** customers will launch transaction orders to buy the smart token; otherwise when valuation < **Psc** , sell orders will be generated.

2. If no reserve tokens in hand, customer will not launch orders to buy smart token, though their valuations might be higher than **Psc.** Ditto for sell orders.

3. Customers make valuations and launch orders in every time slot; while in one time slot, one customer can only try to generate one order.

   -- 1000 time slots, totally **<=1000N** transaction orders will be made

4. Customer uses all of their reserve tokens or smart tokens to buy or to sell if they have money in hand.

## Code about Transaction Generation:

```
# self represents a customer (customer class). we name him/her as XXX
if self._valuation > self._market.getCurrentPrice() and self._reserveBalance > 0:
    # XXX issues a buy order
    self._market.buy(self, self._reserveBalance) # all-in policy
elif self._valuation < self._market.getCurrentPrice() and self._tokenBalance> 0:
    # XXX issue a sell order
    self._market.sell(self, self._tokenBalance)
else:
    # nothing to do
    pass
```

After customers generating their transaction orders by valuations and money they hold, they wait to see whether their transactions could be handled by market.

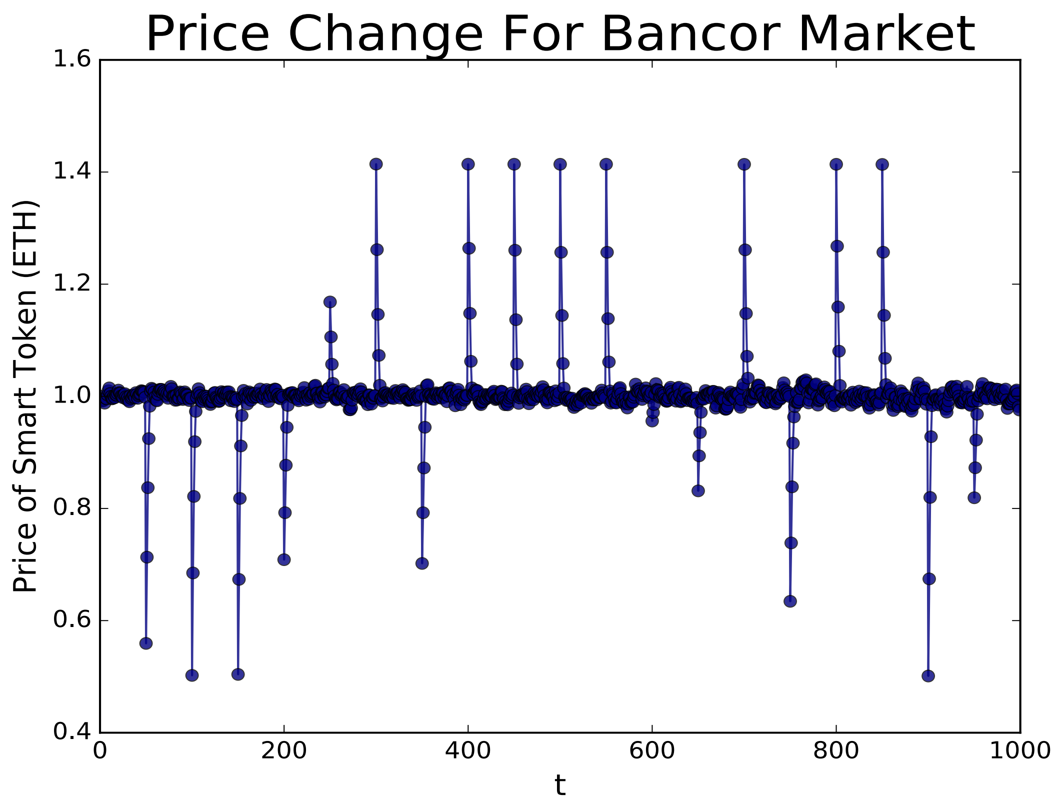
## Transaction Processing in Bancor:

Bancor Market processes customers' transaction orders **one by one**.

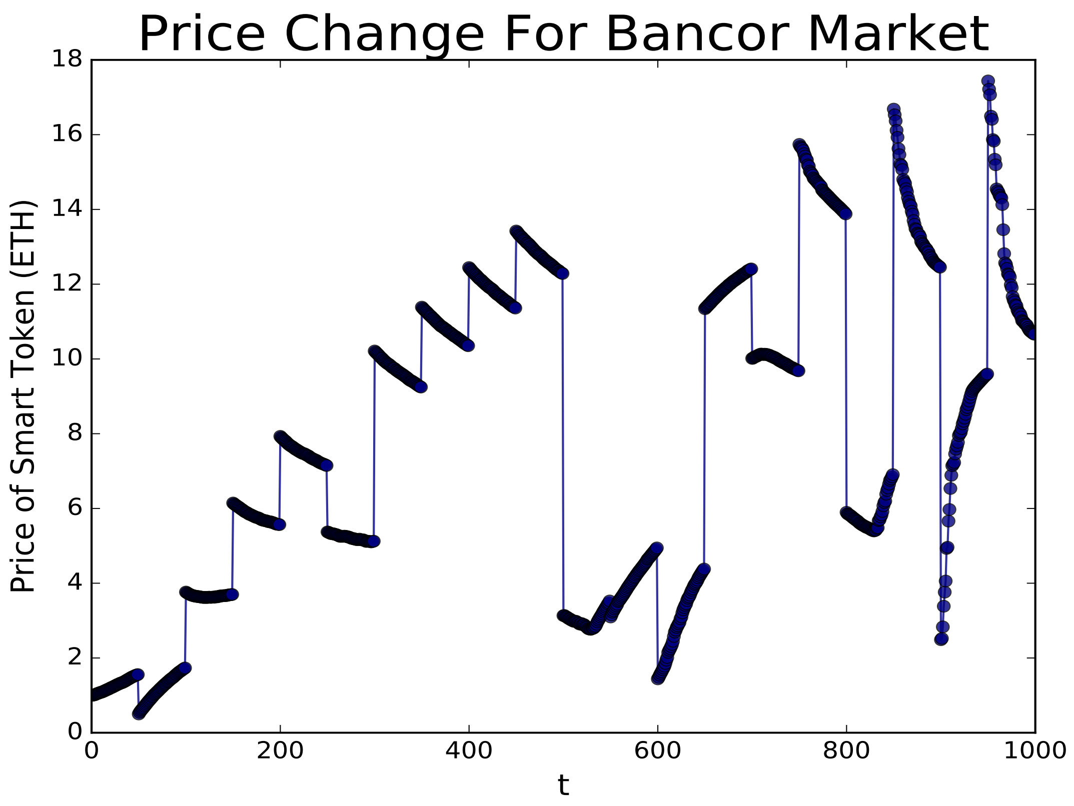
**In market's eyes:**
When dealing with one of customers' transaction orders,
if the real-time price of smart token does not meet this customer's valuation, the market will announce the customer to cancel this order and skip this transaction order to try to deal with the next customer's order.

**In customers' eyes**:
Customers generate valuations of product first (by **Psc**), and then accord to the product's price in market (**Ps**) to decide whether to cancel the transaction.

## Results in Bancor:



(a) Without Reseve Aid



(b) With 200 Reserve Aid

Y-axis: Price of smart token (ETH)

X-axis: time of the simulation

## Explanation of the Stable Price:

```
# The Initialization code for example
KennyCoin = Smartcoin(name='Kenny',reservetokenName='ETH',initCRR=0.5,
                      initPrice=1,initIssueNum=800000)
# Bancor Market Initialization
MyBancorMarket = BancorMarket(smartToken = KennyCoin)
# Customer Initialization
custList = []
custNum = 2000
for i in range(custNum):
  Joe = Customer(smartToken = KennyCoin, market = MyBancorMarket,
                 tokenBalance = 200,
                 reserveBalance = 200)
  custList.append(Joe)
```

## Explanation of the Stable Price:

| | | |
|---|---|---|
| | Market:<br>Price of Kenny Coin: 1.0 ETH<br>Smart Token Supply: 800000 | All Customers:<br>Reserve Balance: 400000 # 200 * 2000<br>Smart Token Balance: 400000 |
| Time epoch, everyone wants to buy | Market:<br>Price of Kenny Coin: 1.414 ETH<br>Smart Token Supply: 1131371 | All Customers:<br>Reserve Balance: 0<br>Smart Token Balance: 731371 |
| Normal case, **mu** = 1.414 ETH | Market:<br>Price of Kenny Coin: 0.957 ETH<br>Smart Token Supply: 765686 | All Customers:<br>Reserve Balance: ≈433578<br>Smart Token Balance: ≈365685 # 731371/2 |
| Normal case, **mu** = 0.957 ETH<br><br>now is back close to 1.0 ETH | Market:<br>Price of Kenny Coin: ≈**0.979 ETH**<br>Smart Token Supply: 783148 | All Customers:<br>Reserve Balance: ≈ 383146<br>Smart Token Balance: ≈ 416674 |

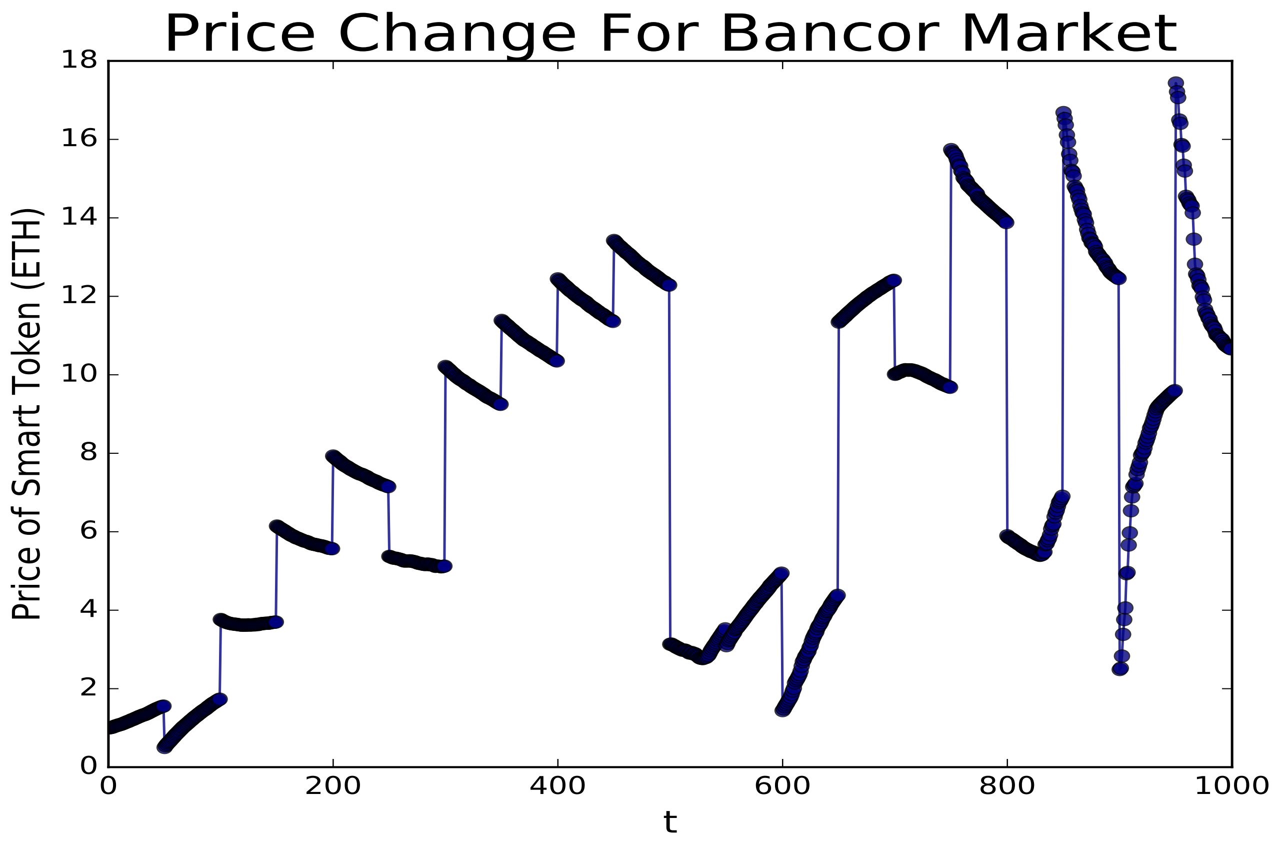
## Explanation of the Stable Price:

Therefore, the reason why in our simulator, the Price of smart token can always go back to average after its bouncing, is that **under all-in policy, almost half of customers cannot launch transaction orders after time epoch.**

## Potential Solution:

We can add customers‘ reserve or smart token balance if they are run out.

e.g. add 200 reserve tokens to customers once they run out reserve



Y-axis: Price of smart token (ETH)
X-axis: time of the simulation

Price from 1.0 ETH rises to about 10 ETH.
However, how much money should we add becomes a tricky problem.
Also, we cannot add too many smart tokens to customers as smart tokens cannot be unlimitedly added. (the total sum should <= Smart Token Supply).

## Explanation of the Stable Price:

In my opinion, the Bancor Market does have its ability to stabilize the price of smart token.
(even in solution, the price of smart token gradually recovers to its original status before the time epoch)

The reason why the price of smart token is stable is that our all-in policy exaggerates the recovering ability of Bancor.

## Cancel Rate in Bancor Market:

Further, since transaction orders might be canceled in market, we also track the cancel rate of transaction orders successfully launched by customers, which is plotted in Figure 2.

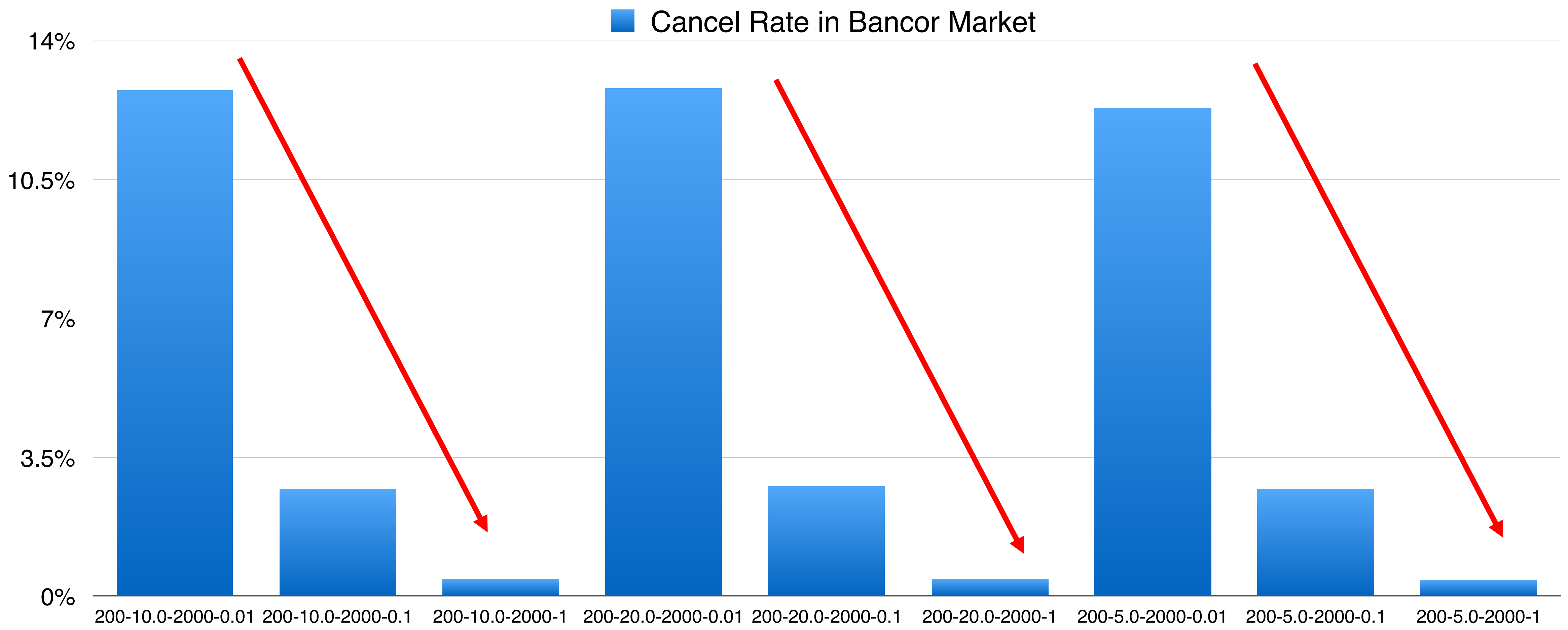


Figure 2. Figure about cancel rate in Bancor Market. Here the cancel rate is calculated by (# of all canceled orders / # of all launched orders). The x-axis: **T-R-N-sig**.

**T**: interval between time epochs, **R**: bouncing range,

**N**: customers' number, **sig**: sigma in Gaussian

## Cancel Rate in Bancor Market:

In Figure 2, what we can learn is that with smaller sigma, the transactions‘ cancel rate is much higher,
which indicates more tightly customers make their valuations, more likely they need to cancel their transaction orders.

Next several slides present the mathematical proof of this result with our model design.

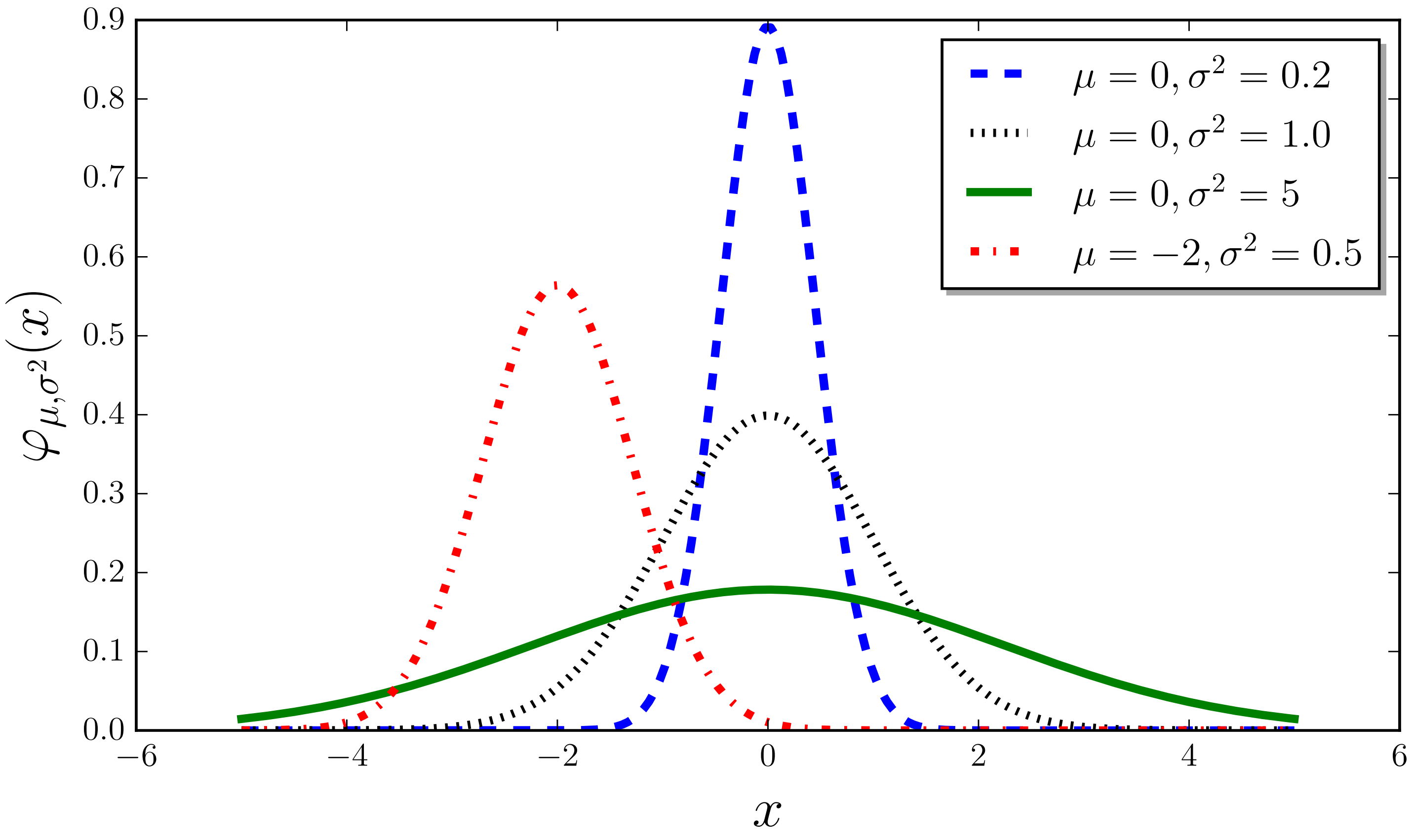
## Review of Valuation Making in Bancor:

At the beginning of every time slot, **N** customers will be announced about **Psc**, i.e., the current price of smart token.

Based on this price, customers will generate valuations of smart token in Gaussian distribution with **mu** and **sigma.**

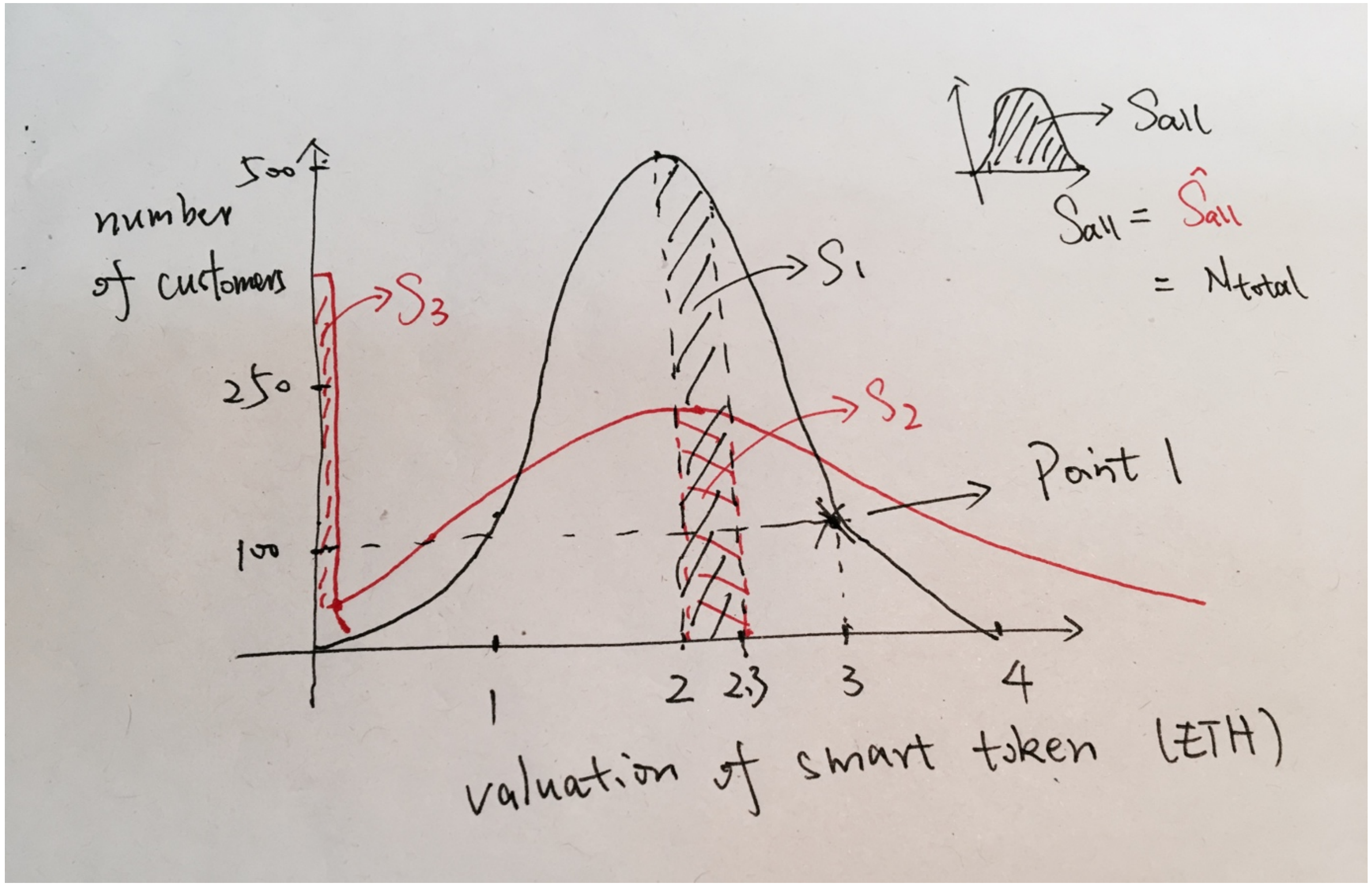
**Normal**: **mu** = **Psc,**
**In Time Epoch: mu =** random prick from **(Psc/R**, **Psc*R )**

## Proof of smaller sigma, higher cancel rate:

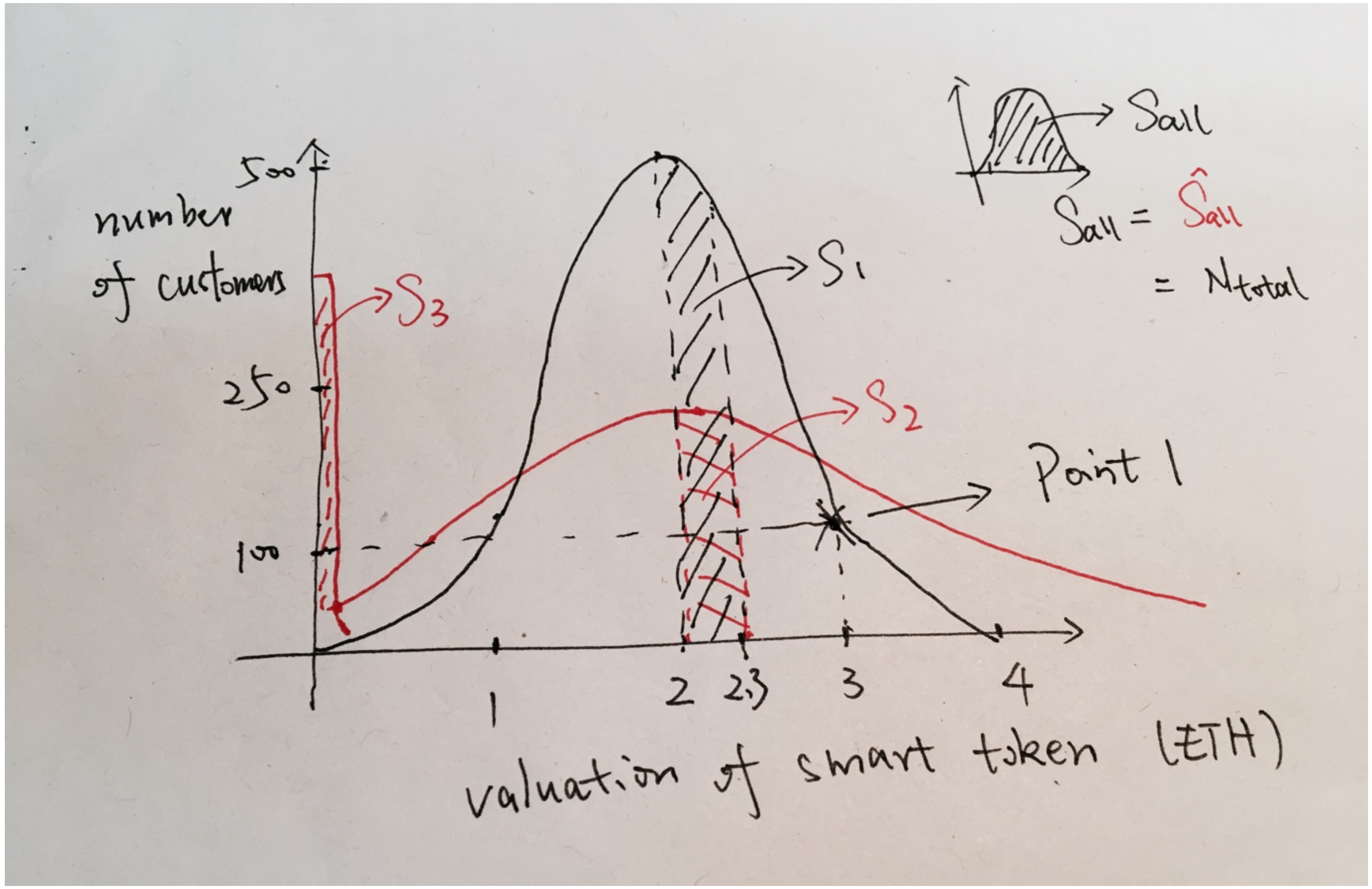


The Gaussian function in different sigma settings. Smaller the sigma is, steeper the Gaussian curve is.

## Proof of smaller sigma, higher cancel rate:

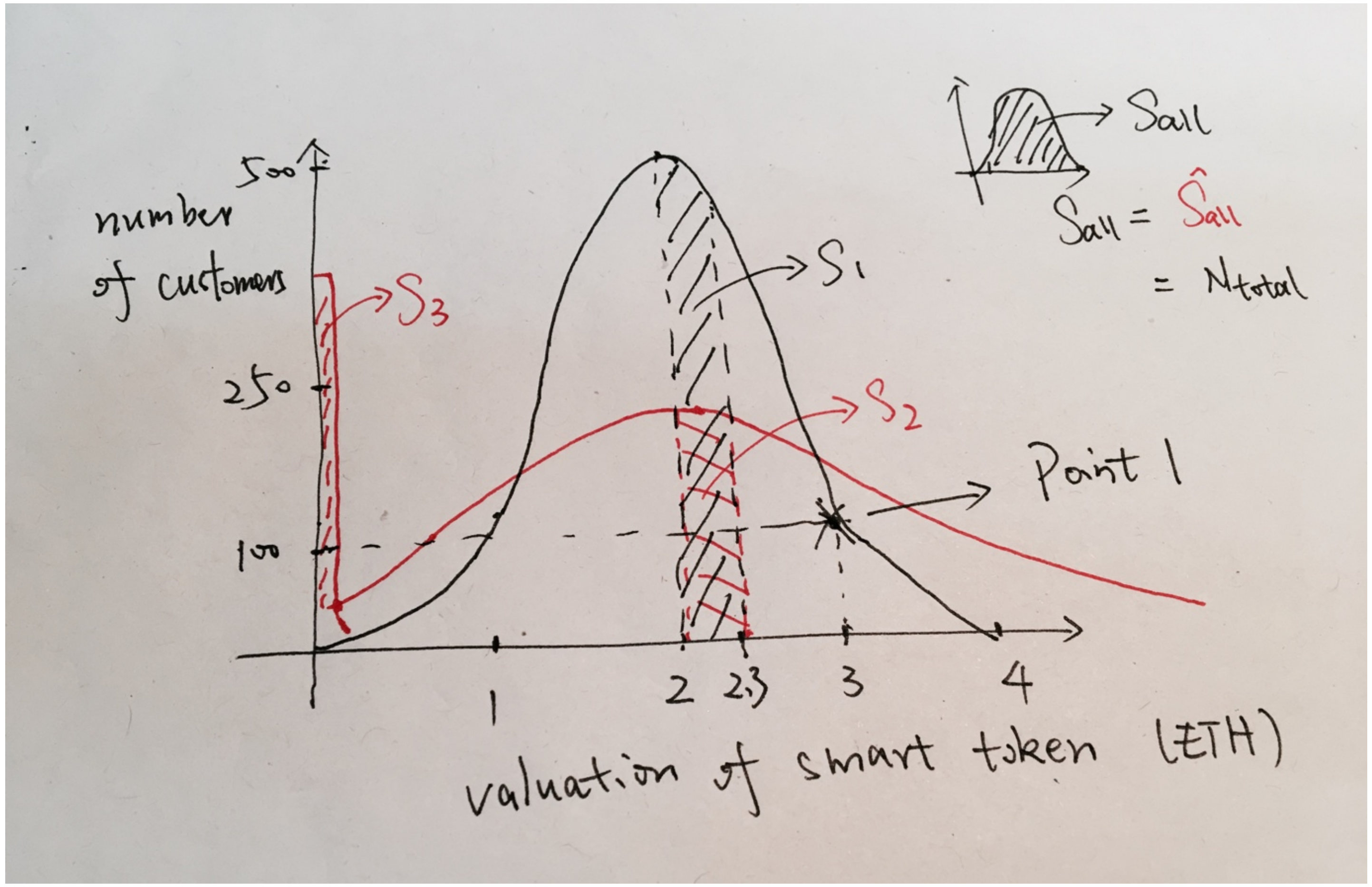


The black curve shows the valuation distribution with **sig1**, the red curve shows with **sig2**. By Figure 1, we know **sig2** > **sig1.** The mean valuation is 2.



The point 1 means there are 100 customers making valuation as 3 ETH.

Thus, by doing a simple calculus, like small plot in the right-top corner of Figure, we know that the total area rounded up by x-axis and Gaussian curve is the total number of customers.

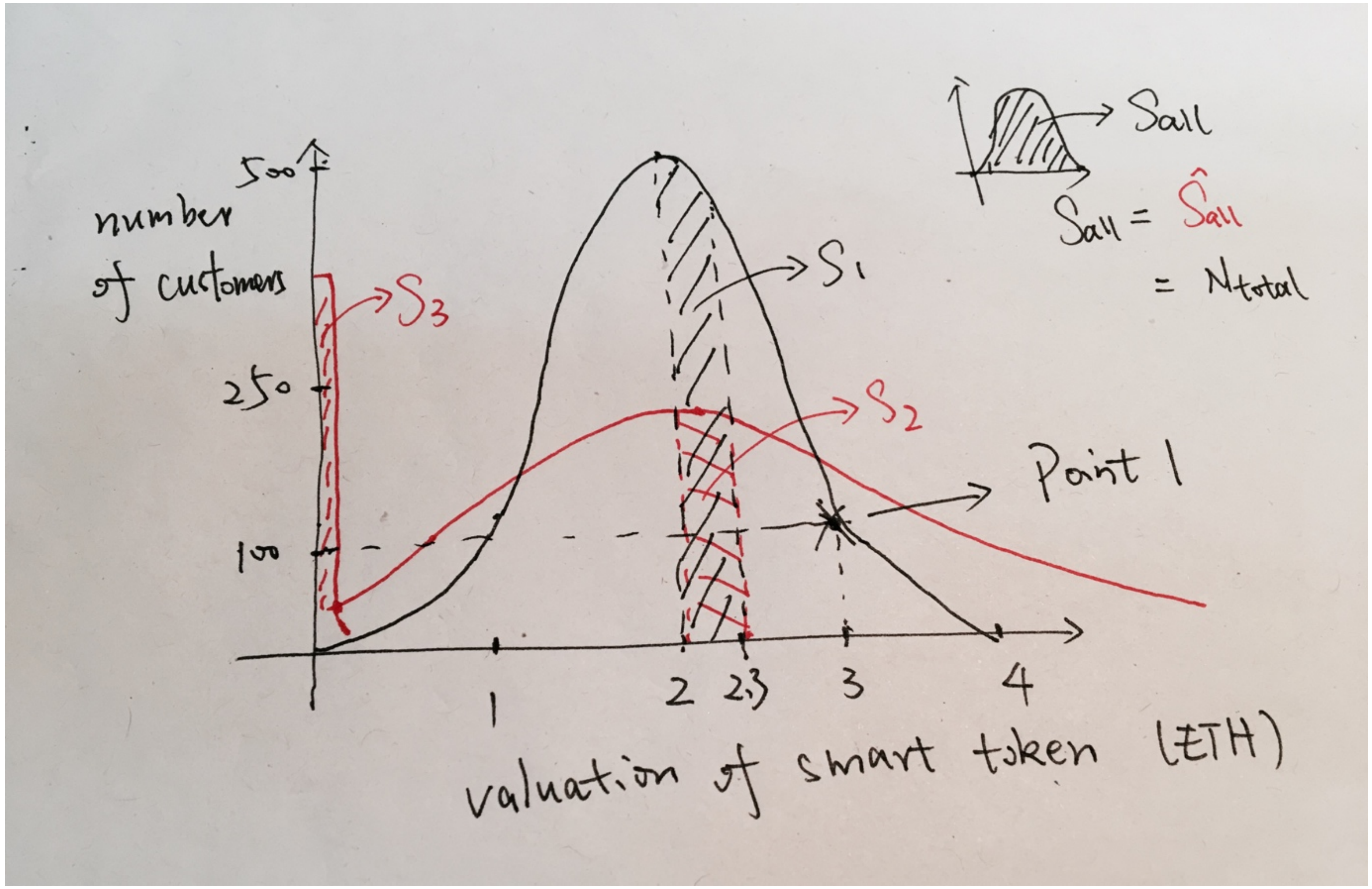


Since both in red curve and black curve, there are totally 2000 customers coming into market,

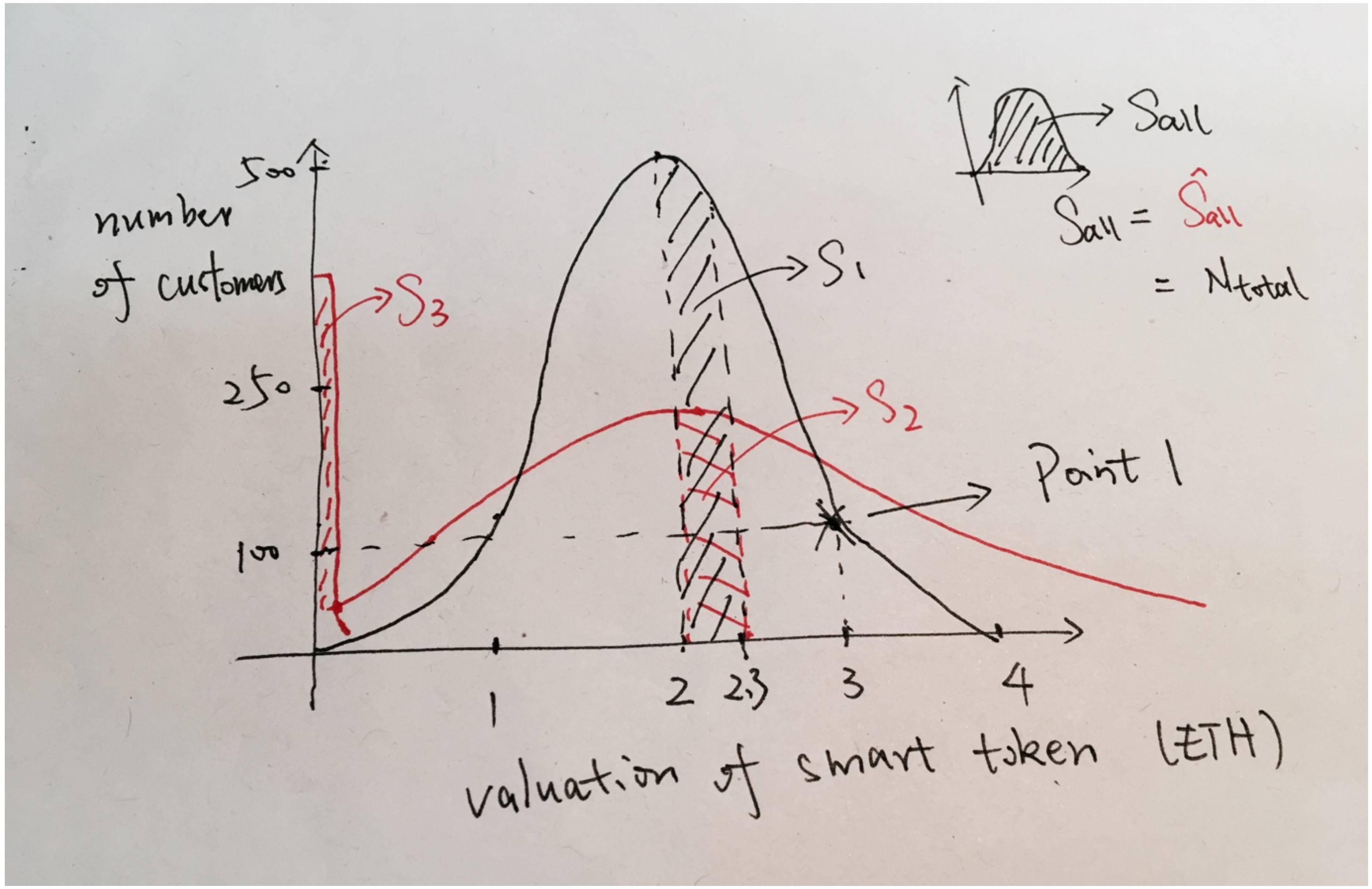
we know that:

**S all** = **S all** = **N total**

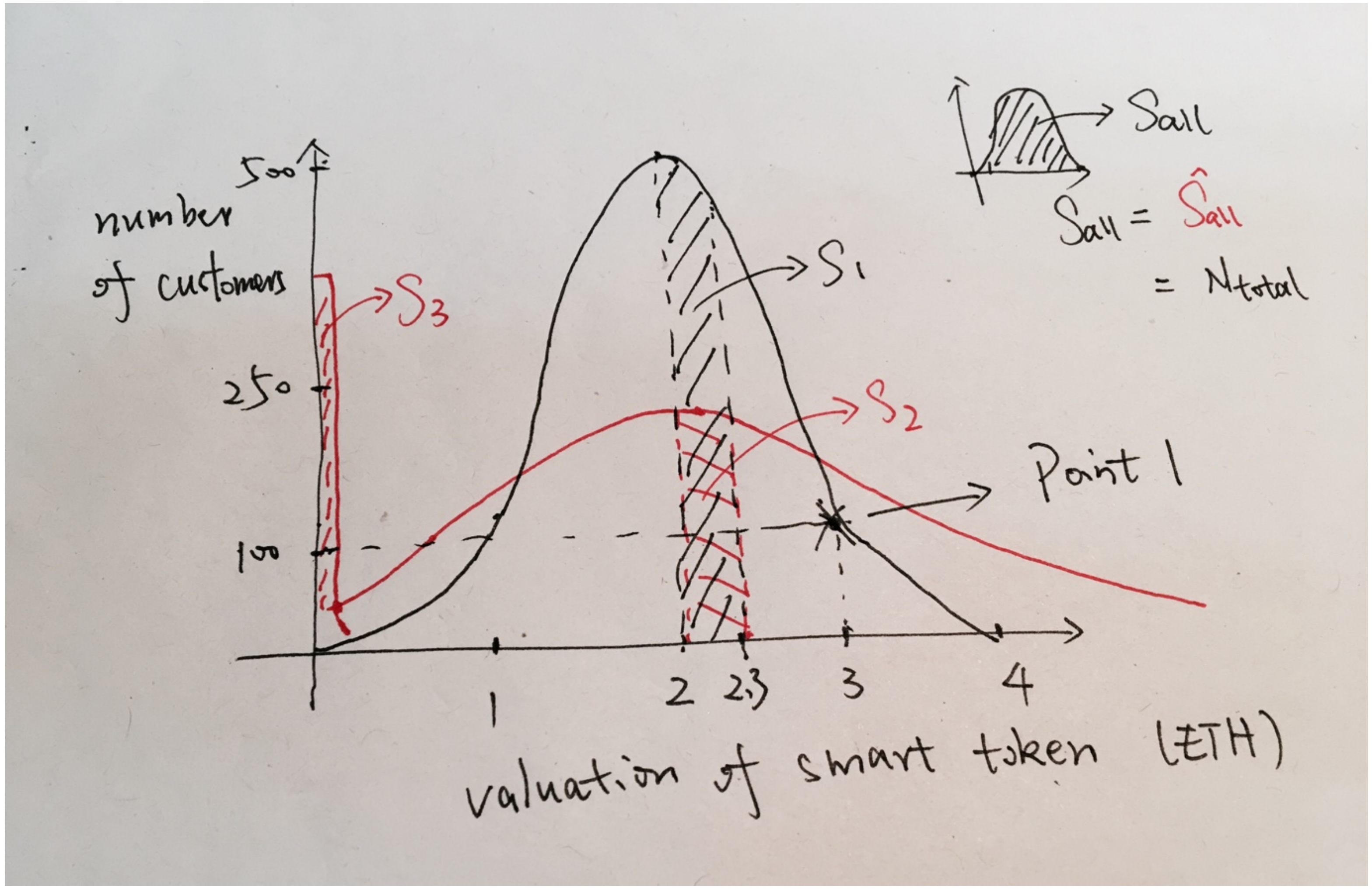
= 2000 (# of customers)



When one customer successfully making his transaction, the price of smart token in market will fluctuate, from 2 ETH to 2.3 ETH.



In this case, the **blacked shadowed area S1** or **the red shadowed area S2** presents the number of customers who now cannot make transactions.
(buy-order valuation smaller than current price of smart token)



Hence, the current probability of customers' order being canceled in black-curve distribution:

**Pr = S1/S all = S1/N total.**

Similarly,

**Pr = S2/S all = S2/ N total.**

Apparently, **S1 > S2**.

Therefore, in current state,

**Pr > Pr.**

## Proof of smaller sigma, higher cancel rate:

In fact, whether the price of smart token is increasing or decreasing, **Pr** is **always** larger than **Pr**. This indicates, **at every time**, the probability of transaction being canceled in Black curve Gaussian distribution (with **sig1**) is larger than it in red curve (with **sig2**).

Combining with the fact that **sig2** > **sig1**, the proof of smaller sigma causing higher cancel rate is done.

## Proof of smaller sigma, higher cancel rate:

If you are careful enough, you might notice the weird S3 area.
This is because when the valuation by Gaussian function is smaller than 0, we set this valuation to be 0.001 * mean valuation (in our example is 0.002). Therefore, S3 actually equals with number of customers who generate valuation smaller than 0.

```python
for i in range(custNum):
    if custValuation_list[i] < 0:
        # Customer does not want to sell their token in free.
        # Here we give them a small valuation when valuation < 0
        custList[i].changeValuation(0.001*currentMarketPrice)
    else:
        custList[i].changeValuation(custValuation_list[i])
```

Actually, S3 does not disturb the prove at all, since larger the S3 is, smaller the S2 is.

## Classic Market:

The whole simulating time is comprised of **1000** time slots.

In every time slot, Classic Market processes the orders launched by **N** customers by managing the order list in the market.

**Valuation Making -> Transaction Launching -> Transaction Processing**

## Valuation Making in Classic:

The valuation making in Classic Market is similar with Bancor Market. However, since the real-time price of the product, i.e., **Ps** does not change, the mean valuation is constant in normal cases. That is that the **mu** always equals **Ps**.

**Normal**: **mu** = **Ps, Ps** is a constant.

**In Time Epoch: mu =** random prick from **(Ps/R**, **Ps*R )**

## Transaction Generating in Classic:

1~4 sipulations are similar with Bancor Market.

5. A customer will not launch new order if his order has not been fulfilled.

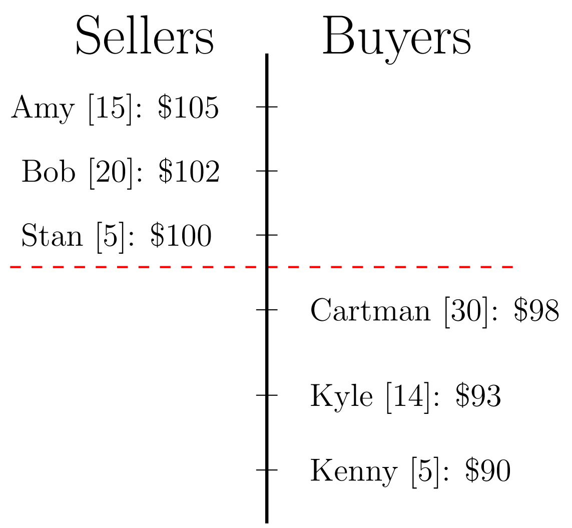
For instance, in a certain time slot, a customer launches a sell order at valuation 5 ETH to sell all of his 200 smart tokens.
However, in the next time slot, he finds that only 120 smart tokens have been sold. Therefore, he will not launch new order and continue to wait for his remaining 80 smart tokens being sold at valuation 5 ETH.
In the end of simulation, i.e., 1000 time slots have passed, if this transaction order is still unfinished in market, we say this order should be canceled.

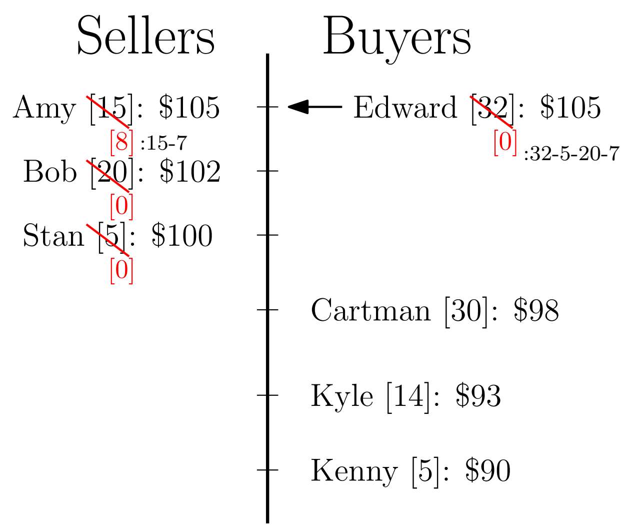
## Transaction Processing in Classic:

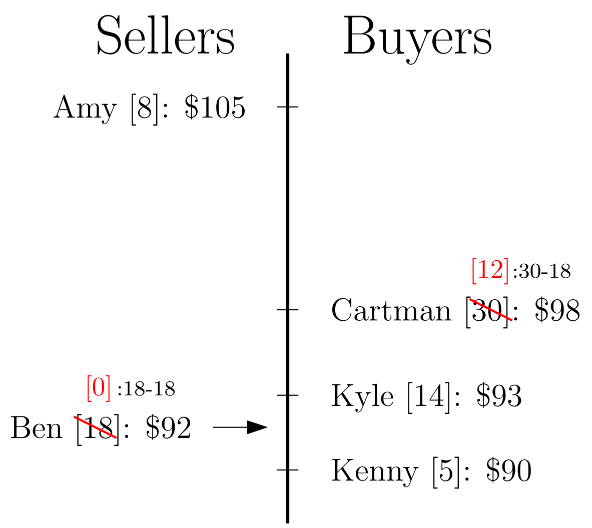
Classic Market manages an order list to process all transaction orders launched by customers.

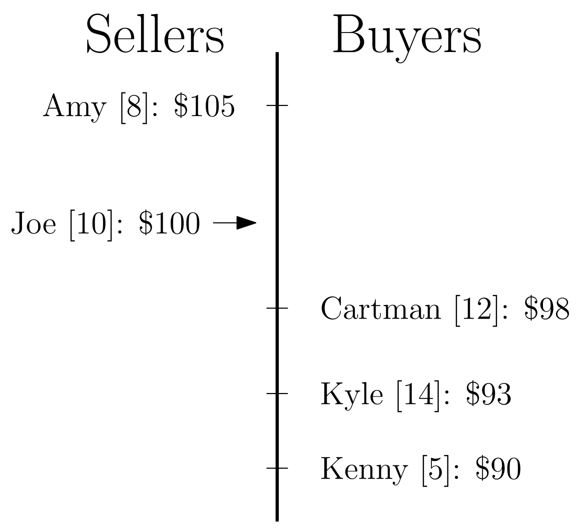
In short, all transaction orders from customers will be separated into two sub-list, named as sell list and buy list. In each list, orders will be sorted by the valuation of these orders.

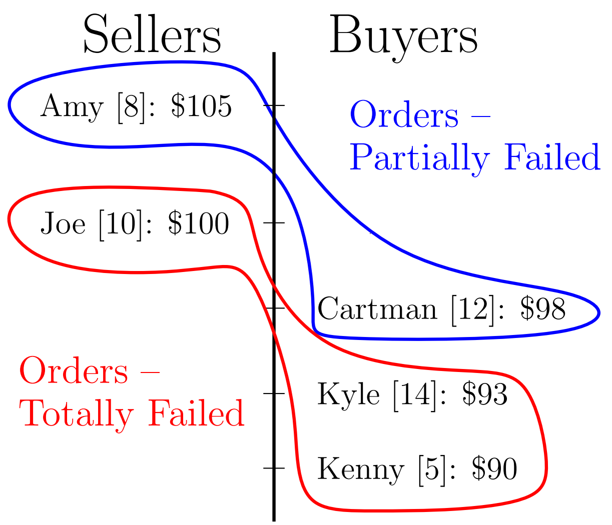
# Transaction Processing in Classic:











## Transaction Processing in Classic:

Both Partially Failed orders and Totally Failed orders will be remained in order list and expect in the time slot they can be finished.

However, if these orders can never be finished -- after 1000 time slot, they still are remained in the market, we then say these orders should be canceled.
Thus, we calculate the **cancel rate** by:
    # of orders remained in market / # of all launched orders.

The Classic Market's cancel rate statistic graph is plotted in Figure 4:

## Cancel Rate in Classic Market:

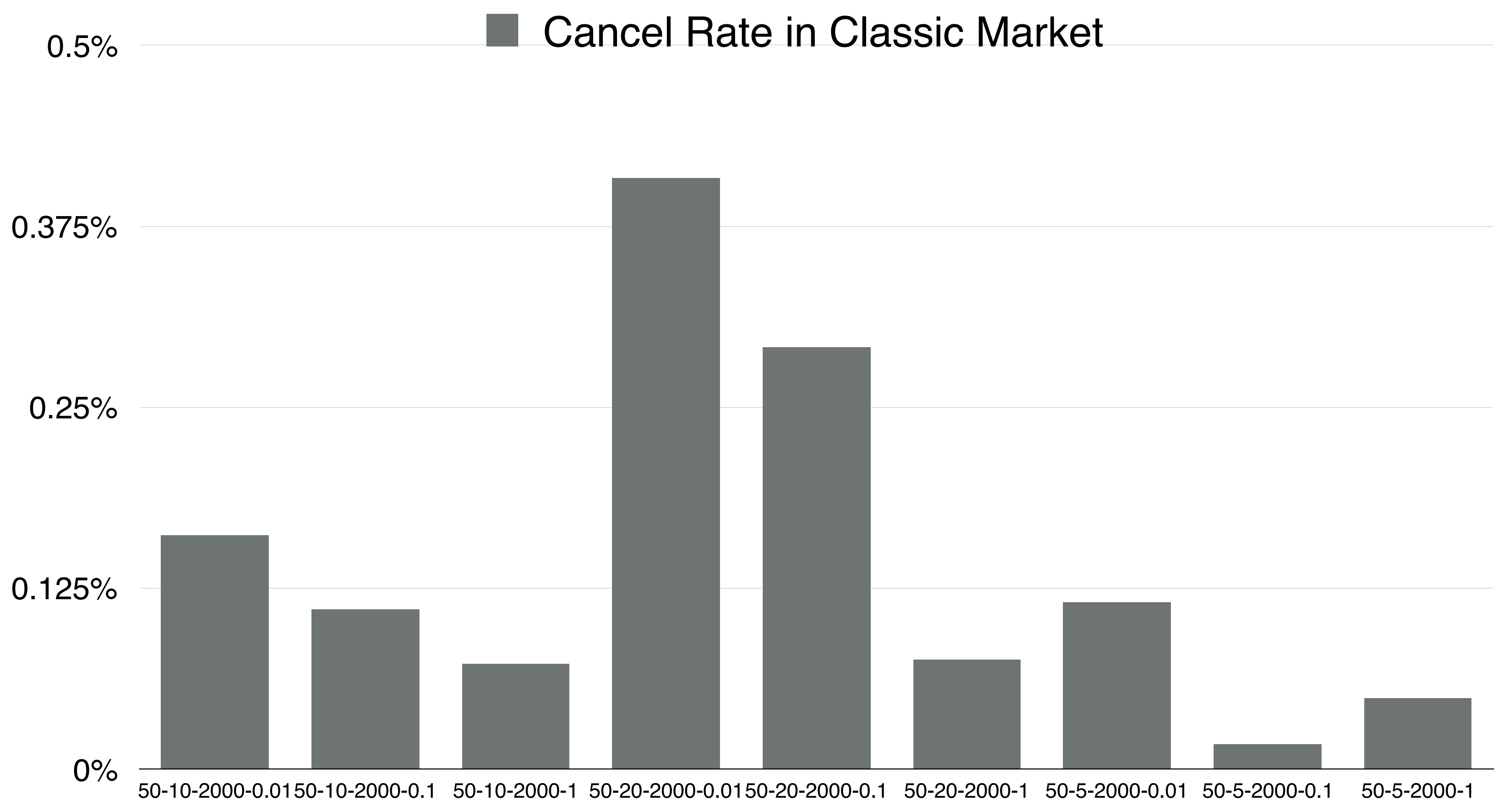


Figure 4: The transaction orders' cancel rate in classic market. The x-axis: **T-R-N-sig**.

## Comparison Between Bancor and Classic's Cancel Rate :

By comparing the transactions' cancel rate between Bancor Market and Classic Market, the result is shown in Figure 5.

The review of Cancel Rate:

In Bancor:

- Why: The price of smart token cannot meet customer's valuation in order.
- Cal: cancel rate = # of all canceled orders / # of all launched orders

In Classic:

- Why: Customer's order cannot be finished before the end of simulation.
- Cal: cancel rate = # of orders unfinished / # of all launched orders.

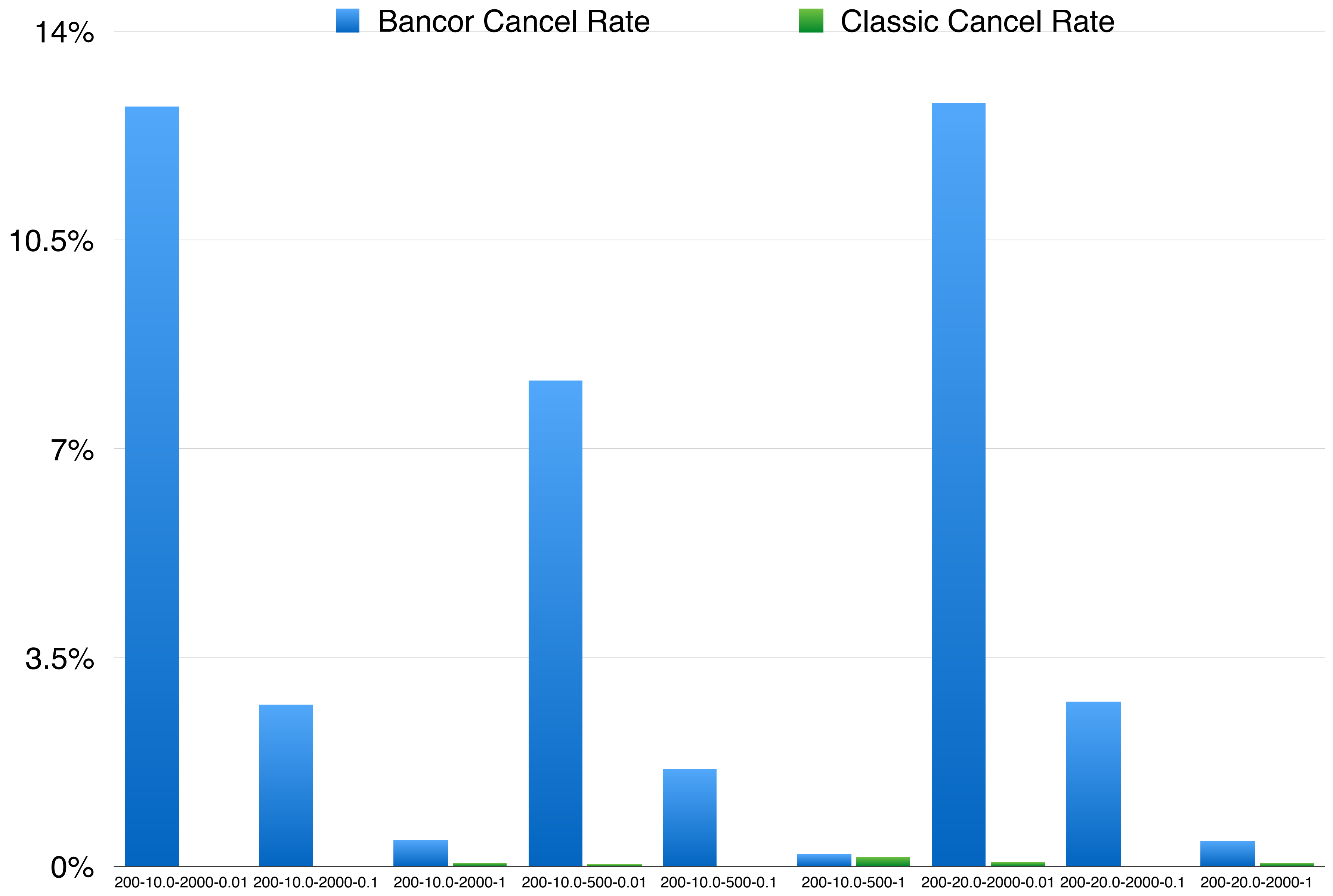


Figure 5: The comparison of cancel rate between Bancor Market and Classic Market. The x-axis: **T-R-N-sig**.

By Figure 5, we can know the cancel rate in Bancor Market is much higher than in Classic Market

## Summary:

1. Silde 8: if customers keep investing money in Bancor Market (never run out of reserve tokens), the market craze in time epoch can lead the price of smart token bouncing around and finally to extremely high.

2. Silde 15: the order‘s cancel rate in Bancor Market can be quite high -- reaching beyond 10% in several parameter settings, especially when customers’ valuations are made tightly, i.e., sigma is small.

3. Slide 34: The low classic cancel rate in Figure 4 indicates the "co-incidence of double wants" might not be a problem in Classic Market.

4. Slide 36: The cancel rate in Bancor Market is always much higher than in Classic Market.

## Summary:

The code as well as code‘s tutorial are presented on:

https://github.com/Ohyoukillkenny/Bancor-Simulator

## Flaw and Future Work:

Up to now, what we have fully discussed in Bancor market is based on **limited order**.
Also, our evaluating metrics might not be representative enough.

Now, I am still working on modifying our simulating model in order to make it feasible in more general cases.