TRC 404 Smart Contract Race Condition Analysis

Currently, TRC 404's smart contract mainly has two operations. One is transfer FT, the entrance of transferring FT operations is the TRC 404 wallet contract. One is transfer NFT, the entrance of transferring NFT operations is the NFT Item contract. Because these two operations are at two different entrances, exceptions may occur due to race conditions.

This article will describe the solution to solve the race condition and analyze in detail whether there is a race condition problem on all nodes of any two operations to ensure that there is no unexpected behavior, or asset loss, or double-spend problem caused by the race condition.

Resolution of race condition

In order to better solve the race condition problem of the TRC404 contract, the following improvements and restrictions are made to the TRC404 contract:

1. For transfer FT

Sender: The jetton_balance of the sender's TRC404 wallet contract will directly deduct the amount of the transfer. For NFTs that need to be burned, the contract will first find the NFT index with the smallest sequence number and the status is not pending_delete, then the TRC 404 wallet contract will not directly delete the NFT The index from owned_nft_dict, but set the NFT index that needs to be burned to pending_delete status. After the NFT is actually burned, a message will be sent to notify the sender's TRC404 wallet contract to delete the NFT index from owned_nft_dict.

Receiver: The jetton_balance of the recipient's TRC404 wallet contract will not directly add the balance of the transfer, but first add the amount of the transfer and then subtract the number of mintNFTs required, and then send a message notification to the receiver's TRC404

wallet contract to increase jetton_balance by one after each mint of one NFT is completed.

2. For transfer NFT

In order to be compatible with these two transactions: A transfer NFT to B and B transfer NFT to C can be executed successfully to be compatible with getgems.io's put-on-sale operation. TRC404 wallet contract adds two attributes: pending_reduce_jetton_balannce and pending_transfer_nft_queue.

When performing the above two operations:

When executing B transfer NFT to C, if the NFT index does not exist in the owned_nft_dict of B's TRC 404 wallet, then this NFT Index will be added to pending_transfer_nft_queue. After A transfers NFT to B is executed, B's TRC 404 wallet will delete the NFT index from the pending_transfer_nft_queue.

If B's balance is 0 when executing B transfer NFT to C, then jetton_balance is set to 0, and pending_reduce_jetton_balance is increased by 1. After A transfer NFT to B is executed, set jetton_balance = jetton_balance +1 - pending_reduce_jetton_balannce, and set pending_reduce_jetton_balannce =0.

3. Gas fee problem

If owned_nft_dict stored too many elements, the gas fee for finding an element from owned_nft_dict may be too high and exit. To avoid this problem, the contract forces a wallet to have a maximum of 5 NFTs. That is, the owned_nft_dict of the TRC 404 wallet contract will only have a maximum of 5 elements. If a user transfers 1 NFT to a wallet that already has 5 NFTs, the recipient will only receive 1 TRC 404 token, and this NFT will be burned by the contract, ensuring that the recipient's wallet will only store up to 5 NFTs.

Notice: When we say jetton_balance increase/decrease by 1 or n, actually, for the implementation of source code, that means jetton_balance + 1 * min_unit() or jetton_balance - 1 * min_unit(), and min_unit() equals 1 * 10^9.

1.Contract core data and status

For the TRC 404 wallet contract, the core data and status include:

jetton_balance :current TRC404 jetton balance
owned_nft_number : current NFT number(user-owned)
owned_nft_limit: The max NFT number that one wallet can hold
owned_nft_dict : NFT dictionary(user-owned). Key: NFT item index, Value: null means normal status, 0 means the status is pending_delete
pending_reduce_jetton_balance: Only addOneFtAnd NFT and reduceOneFtAndNFT operations involve operating this data
pending_transfer_nft_queue: Only addOneFtAndNFT and reduceOneFtAndNFT operations involve operating this data

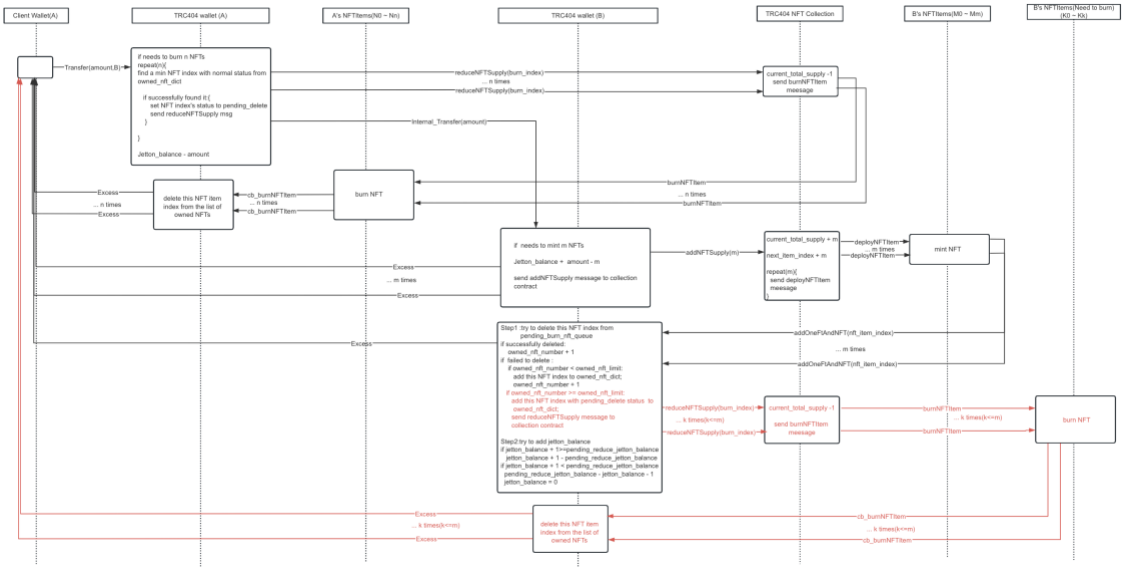
For the NFT Item contract, the core data and status include:

owner: owner of this NFT
contract active status: after minting NFT, NFT contract status will be Active. After burning FT, NFT contract status will be non-active

For the NFT Collection contract, the core data and status include:

current_total_supply: current total NFT supply, burn NFT will decrease this number, mint new NFT will increase this number
next_item_index: The next NFT item index will increase when minting a new NFT

2. Transfer FT



The picture above is the timing diagram of Transfer FT.

The abstract and simplified steps are:

1. Sender TRC 404 wallet contract:

```
jetton_balance -= amount;

repeat( need_to_burn_nft_number){

    find a min and valid NFT index from owned_nft_dict;

    set this NFT index's status to pending_delete;

    send a reduceNFTSupply message to collection contract;

}
```

(Involving data modification: **jetton_balance** , **owned_nft_number** , **owned_nft_dict**)

2.1 TRC 404 NFT collection contract :

```
current_total_supply -= 1;

send burnNFT message to NFT Item;
```

(Involving data modification: **current_total_supply**)

2.2 NFT Item contract:

```
burn NFT;

send cb_burnNFT message to sender TRC404 wallet;
```

(Involving data modification: **contract active status**)

2.3 Sender TRC 404 wallet contract:

```
delete this NFT item index from owned_nft_dict;
```

(Involving data modification: **owned_nft_dict**)

3.1 Receiver TRC 404 wallet contract:

```
jetton_balance = jetton_balance    + amount - need_to_mint_nft_number;

send addNFTSupply to collection;
```

(Involving data modification: **jetton_balance**)

3.2 TRC 404 NFT collection contract :

```
current_total_supply += amount;

repeat(need_to_mint_nft_number) {

    next_item_index += 1;

    send deployNFTItem message to new NFT Item contract;

}
```

(Involving data modification: **current_total_supply , next_item_index**)

3.3 NFT Item contract:

```
mint NFT

send addOneFtAndNft message to receiver TRC 404 wallet
```

(Involving data modification: owner , contract active status)

3.4 Receiver TRC 404 wallet contract:

```
Step1:try to delete this NFT index from Pending_transfer_nft_queue;

 if (successfully deleted){

       owned_nft_number += 1;

 }

 if (failed to delete){

       if (owned_nft_number < owned_nft_limit){

             add this NFT index to owned_nft_dict;

             owned_nft_number += 1;

       }

       if (owned_nft_number >= owned_nft_limit){

             add this NFT index with pending_delete status to owned_nft_dict;

             send reduceNFTSupply message to collection contract;

        }

 }



Step2: try to add jetton_balance;
```

```
if (jetton_balance + 1 >= pending_reduce_jetton_balance){

      jetton_balance = jetton_balance + 1 - pending_reduce_jetton_balance;

}

if (jetton_balance + 1 < pending_reduce_jetton_balance){

       pending_reduce_jetton_balance -= (jetton_balance + 1);

       jetton_balance = 0;

}
```

(Involving data modification: **jetton_balance , owned_nft_number , owned_nft_dict , pending_reduce_jetton_balance , Pending_transfer_nft_queue**)

3.4.1 TRC 404 NFT collection contract:

```
current_total_supply -= 1;

send burnNFT message to NFT Item;
```

(Involving data modification: **current_total_supply**)

3.4.2 NFT Item contract:

```
    burn NFT;

    send cb_burnNFT message to sender TRC404 wallet;
```

(Involving data modification: **contract active status**)

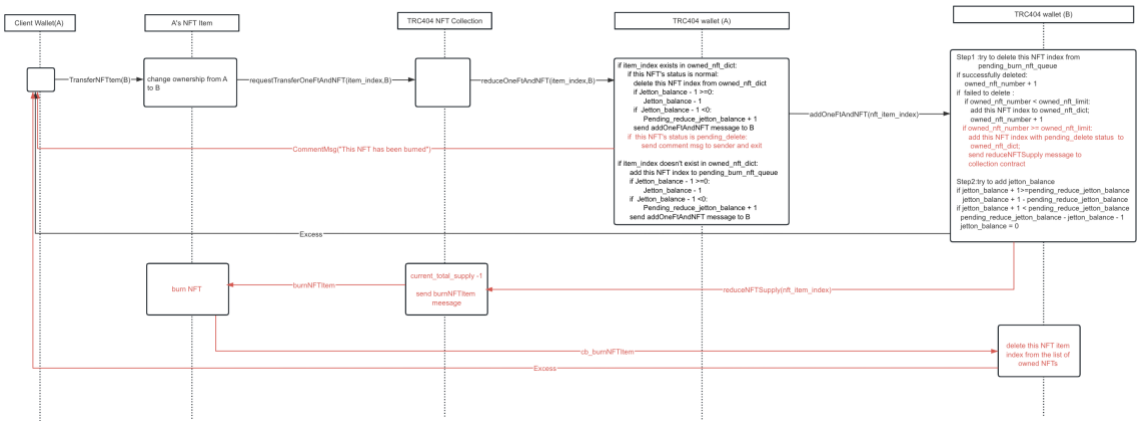
3.4.3 Receiver TRC 404 wallet contract:

```
  delete this NFT item index from owned_nft_dict;
```

(Involving data modification: **owned_nft_dict**)

Note: Execution of 3.4.1 , 3.4.2, and 3.4.3 will only be triggered when executing 3.4 and owned_nft_number >= owned_nft_limit.

3. Transfer NFT



The picture above is the timing diagram of transfer NFT.

The abstract and simplified steps are:

1. NFT Item contract:

```
change ownership from sender to receiver;

send requestTransferOneFtAndNFT message to collection;
```

(Involving data modification: **owner**)

2. TRC 404 NFT collection contract:

```
send reduceOneFTAndNFT message to sender TRC 404 wallet;
```

(Involving data modification: none)

3. Sender TRC 404 wallet contract:

```
if (item_index exists in owned_nft_dict){

    if (this NFT's status is normal){
```

```
            delete this NFT index from owned_nft_dict;
            if (Jetton_balance - 1 >=0){
                    Jetton_balance -= 1;
            }
            if (Jetton_balance - 1 < 0){
                    Pending_reduce_jetton_balance += 1;
                    send addOneFtAndNFT message to B's TRC404 wallet;
            }
        }
        if (this NFT's status is pending_delete) {
            send comment msg to sender and exit;
        }


 }
if (item_index doesn't exist in owned_nft_dict){
            add this NFT index to Pending_transfer_nft_queue;
            if (Jetton_balance - 1 >= 0){
                    Jetton_balance -= 1;
            }
            if (Jetton_balance - 1 < 0){
```

```
            Pending_reduce_jetton_balance += 1;

            send addOneFtAndNFT message to B;

        }

}
```

(Involving data modification: jetton_balance , owned_nft_number , owned_nft_dict , pending_reduce_jetton_balance , Pending_transfer _nft_queue)

4. Receiver TRC 404 wallet contract:

```
Step1:try to delete this NFT index from Pending_transfer_nft_queue;

  if (successfully deleted){

       owned_nft_number += 1;

  }

  if (failed to delete){

      if (owned_nft_number < owned_nft_limit){

          add this NFT index to owned_nft_dict;

          owned_nft_number += 1;

      }

      if (owned_nft_number >= owned_nft_limit){

          add this NFT index with pending_delete status to owned_nft_dict;

          send reduceNFTSupply message to collection contract;

      }
```

```
    }



Step2: try to add jetton_balance;

    if (jetton_balance + 1 >= pending_reduce_jetton_balance){

        jetton_balance = jetton_balance + 1 - pending_reduce_jetton_balance;

    }



    if (jetton_balance + 1 < pending_reduce_jetton_balance){

        pending_reduce_jetton_balance -= (jetton_balance + 1);

        jetton_balance = 0;

    }
```

(Involving data modification: jetton_balance , owned_nft_number , owned_nft_dict , pending_reduce_jetton_balance , Pending_transfer _nft_queue)

4.1 TRC 404 NFT collection contract:

```
current_total_supply -= 1;

send burnNFT message to NFT Item;
```

(Involving data modification: current_total_supply)

4.2 NFT Item Contract:

```
burn NFT;

send cb_burnNFT message to sender TRC404 wallet;
```

(Involving data modification: contract active status)

4.3 Receiver TRC 404 wallet contract:

```
delete this NFT item index from owned_nft_dict;
```

(Involving data modification: **owned_nft_dict**)

Note: Only when executing Step 4, and owned_nft_number >= owned_nft_limit, the execution of Step 4.1, 4.2, and 4.3 will be triggered.

3. Race condition analysis

To simplify, the transactions of transfer FT are described using FT1 and FT2 , for transactions of transfer NFT are described using NFT1 and NFT2 . The following will provide a detailed description of whether each step of Transfer FT and Transfer NFT has potential competitive conflicts.

From a classification point of view, conflicts can be divided into three major categories: transfer FT and transfer FT conflict, transfer FT Conflicts with transferNFT, transferNFT conflicts with transferNFT. In addition, for the same type of conflicts, in addition to considering concurrent execution, such as A transfer 1 FT / NFT to B and A transfer 1 FT / NFT to B, you also need to consider whether there is a problem with this kind of case: such as A transfer 1 FT / NFT to B, B transfer 1 FT/NFT to C.

3.1 transfer FT's competitive analysis

Consider whether there is a potential conflict issue when a user concurrently performs a Transfer FT operation. Assuming the user executes two Transfer FT transactions simultaneously, namely FT1 and FT2 The following will analyze whether there are conflicts in each step of the transaction. The so-called conflicts refer to whether there will be abnormal situations such as transaction failure, asset reduction, or asset reuse(like double spend) due to concurrent execution.

3.1.1 A transfer FT to B and A transfer FT to B competition analysis

FT1: A transfer FT to B, FT2 : A transfer FT to B. The analysis of possible competing nodes for transaction B is shown in the following table:

	FT2(A transfer FT to B)	Step 1(Sender TRC404wallet)	Step 2.1(collection)	Step 2.2(NFT Item)	Step2.3(Sender TRC404wallet)	Step3.1(Receiver TRC404 wallet)	Step3.2(collection)	Step3.3(NFT Item)	Step3.4(Receiver TRC404 wallet)	Step3.4.1 (collectiton)	Step3.4.2(NFT Item)	Step3.4.3(Receiver TRC404 wallet)
1	FT1(A transfer FT to B)											
2	Step 1(Sender TRC404wallet)	jetton_balance - owned_nft_number + owned_nft_dict:set NFT index to pending_delete			owned_nft_dict							
3	Step 2.1(collection)		current_total_supply - 1				current_total_supply			current_total_supply		
4	Step 2.2(NFT Item)			contract active status:non-active				contract active status				
5	Step 2.3(Sender TRC404wallet)	owned_nft_dict			owned_nft_dict: delete NFT index							
6	Step 3.1(Receiver TRC404 wallet)					jetton_balance +			jetton_balance			
7	Step 3.2(collection)		current_total_supply				current_total_supply+ next_item_index +			current_total_supply		
8	Step 3.3(NFT Item)			contract active status				owner :receiver contract active status:active			contract active status	
9	Step 3.4(Receiver TRC404 wallet)					jetton_balance			jetton_balance + owned_nft_number + owned_nft_dict: add NFT index pending_reduce_jetton_balance: + pending_burn_nft_queue:add NFT index			owned_nft_dict
10	Step 3.4.1(collectiton)		current_total_supply				current_total_supply			current_total_supply - 1		
11	Step 3.4.2(NFT Item)							contract active status			contract active status: non-active	
12	Step 3.4.3(Receiver TRC404 wallet)								owned_nft_dict			owned_nft_dict: delete NFT index
13												

The steps involved in updating the current_total_supply and next_item_index of the collection contract include :
FT1 Step 2.1 and FT2 Step 2.1 ,
FT1 Step 2.1 and FT2 Step 3.2 ,
FT1 Step 2.1 and FT2 Step 3.4.1 ,
FT1 Step 3.2 and FT2 Step 3.2 ,
FT1 Step 3.2 and FT2 Step 3.4.1 ,
FT1 Step 3.4.1 and FT2 Step 3.4.1 ,
Whether these steps perform addition or subtraction operations, they are atomic operations and will not involve race condition conflicts, so they will not be analyzed again. The following focuses on analyzing the possible conflicts that may arise between the TRC404 Wallet contract and the NFT Item contract.

3.1.1.1 FT1 Step1 and FT2 Step1 (TRC 404 Wallet)

There are three data involved in race conditions, jetton_balance, owned_nft_number, owned_nft_dict.

For jetton_balance, in Step 1, the contract directly tries to subtract the transfer amount contained in the request. If the result is

less than 0, it will exit directly and no subsequent operations will be performed. This is an atomic operation and there is no competition conflict.

For owned_nft_number, in Step 1, the contract will subtract the number of nft that needs to be burned. This is an atomic operation and there is no competition conflict.

For owned_nft_dict, in Step 1, the logic that the contract needs to implement is to find out the NFT index needs to be burned, and set the index status to pending_delete . In order to avoid setting the index repeatedly (potential competition conflicts), the conditions for selecting the NFT index that need to be burned is the smallest among owned_nft_limit, and the index status must be normal, that is, it is not pending_delete.

3.1.1.2 FT1 Step1 and FT2 Step 2.3 (TRC 404 Wallet)

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, the operation performed by Step 1 on owned_nft_dict is to find the NFT index that meets the conditions and set the status of the NFT index to pending_delete. Step 2.3 operates on owned_nft_dict to delete the specified NFT index.

If a race condition conflict occurs, it means that the NFTindex of the FT1 Step 1 operation is the same as the NFT index of the FT2 Step 2.3 operation, but as described in 3.1.1.1, Step 1 will find the smallest index with normal status each time, then burn this NFT corresponding to NFT index. So FT1 Step1 index cannot be the same as the index of owned_nft_dict iNFT2 Step 2.3, that is, there will be no conflict.

3.1.1.3 FT1 Step 2.2 and FT2 Step 2.2 (NFT Item)

There is one data involving a race condition, NFT item contract active status.

For contract active status, the operation performed in Step 2.2 is burn NFT, so the contract active status will become non-active.

Similar to the reason in 3.1.1.2, these two steps will not conflict.

3.1.1.4 FT1 Step 2.2 and FT2 Step 3.3 (NFT Item)

There is one data involving a race condition, NFT item contract active status.

For contract active status, the operation performed in Step 2.2 is burn NFT, so the contract active status will become non-active. The operation performed in Step 3.3 is mint NFT, and the contract active status will change to active.

If a race condition conflict occurs, it means that the NFT Item of the FT1 Step 2.2 operation and the NFT Item of the FT2 Step 3.3 operation are the same.

However, in Step 3.2, the new NFT index needs to come from the next_item_index of the collection contract, and every time minting a new NFT will make next_item_index+1. Therefore, the NFT items for FT1 Step 2.2 and FT2 Step 3.3 operations cannot be the same, meaning there will be no conflict.

3.1.1.5 FT1 Step 2.3 and FT2 Step 2.3 (TRC 404 Wallet)

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, the operation performed by Step 2.3 is to delete the specified NFT from the owned_nft_dict item index.

If a race condition conflict occurs, it means that the NFT Item index of the FT1 Step 2.3 operation and the NFT Item index of the FT2 Step 2.3 operation are the same, but as described in 3.1.1.1, Step 1 will find the smallest index with normal state each time, then burn the NFT corresponding to NFT index. Therefore, the NFT Item index operated by FT1 Step 2.3 and FT2 Step 2.3, cannot be the same, that is, there will be no conflict.

3.1.1.6 FT1 Step 3.1 and FT2 Step 3.1 (TRC 404 Wallet)

There is one data involving a race condition, jetton_balance.

For jetton_balance, in Step 3.1, the contract directly attempts to add the amount included in the internal_transfer request, and then subtract the number of NFTs that need to be minted. From an internal perspective of the contract, this is an atomic operation and there is no competition conflict.

3.1.1.7 FT1 Step 3.1 and FT2 Step 3.4 (TRC 404 Wallet)

There is one data involving a race condition, jetton_balance.

For jetton_balance, in Step 3.1, the contract directly attempts to add the amount included in the internal_transfer request, and then subtract the number of NFTs that are required to be minted. In Step 3.4, jetton_balance may execute jetton_balance + 1 - pending_reduce_jetton_balance or set jetton_balance=0.From the inside of the contract, these are all atomic operations and there is no competition conflict.

3.1.1.8 FT1 Step 3.3 and FT2 Step 3.3 (NFT Item)

There is one data involving a race condition, NFT item contract active status.

For contract active status , the operation performed in Step 3.3 is mint NFT, and contract active status will change to active .

If there is a competition condition conflict, it indicates that the NFT Item of the FT1 Step3.3 operation is the same as the NFT Item of the FT2 Step3.3 operation.

Similar to the reason in 3.1.1.4, these two steps will not conflict.

3.1.1.9 FT1 Step 3.4 and FT2 Step 3.4 (TRC 404 Wallet)

There are five data involving competition conditions, jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance, Pending_transfer_nft_queue.

For jetton_balance , in Step 3.4 , jetton_balance may execute jetton_balance + 1 - pending_reduce_jetton_balance or set

jetton_balance=0 . From inside the contract, these are all atomic operations and there is no competition conflict.

For owned_nft_number, in Step 3.4 , owned_nft_number may increase by 1 or remain unchanged. From within the contract, these are all atomic operations and there is no competition conflict.

For owned_nft_dict, in Step 3.4, an NFT index will be added into owned_nft_dict or remain unchanged. If a race condition conflict occurs, it means that the NFT Item index to be added iNFT1 Step 3.4 is the same as the NFT Item index to be added iNFT2 Step 3.4. Similar to the reason in 3.1.1.4, these two steps will not conflict.

For pending_reduce_jetton_balance, in Step 3.4 , pending_reduce_jetton_balance may execute pending_reduce_jetton_balance - jetton_balance -1 or remain unchanged. From inside the contract, these are all atomic operations and there is no competition conflict.

For pending_transfer_nft_queue , in Step 3.4 , contract will try to delete an NFT item index from pending_transfer_nft_queue. Similar to the reason in 3.1.1.4, these two steps will not conflict.

3.1.1.10 FT1 Step 3.4 and FT2 Step 3.4.3 (TRC 404 Wallet)

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, in Step 3.4 , an NFT index will be added into owned_nft_dict or remain unchanged. In 3.4.3, an NFT index will be deleted from owned_nft_dict.

Similar to the reason in 3.1.1.4, these two steps will not conflict.

3.1.1.11 FT1 Step 3.4.2 and FT2 Step 3.4.2 (NFT Item)

There is one data involving a race condition, contract active status

For contract active status, the operation performed in Step 3.4.2 is burn NFT, so the contract active status will become non-active.

The reason why these two steps will not conflict with race conditions is similar to what is described in 3.1.1.4.

3.1.1.12 FT1 Step 3.4.3 and FT2 Step 3.4.3 (TRC 404 Wallet)

There is one data involved in a race condition, owned_nft_dict.

For owned_nft_dict, in 3.4.3, the contract will attempt to delete the NFT index from owned_nft_dict (the deletion may fail because the NFT index might not exist).

The reason why these two steps will not conflict with race conditions is similar to what is described in 3.1.1.4.

3.1.2 A transfer FT to B and B transfer FT to C competition analysis

Assume B starts with 0 FT and 0 NFT. FT1 :A transfer FT to B and FT2: B transfer FT to C. The analysis of possible competing nodes in these two transactions are shown in the following table:

	FT2(B transfer FT to C)	Step 1(Sender TRC404wallet)	Step 2.1(collection)	Step 2.2(NFT Item)	Step2.3(Sender TRC404wallet)	Step3.1(Receiver TRC404 wallet)	Step3.2(collection)	Step3.3(NFT Item)	Step3.4(Receiver TRC404 wallet)	Step3.4.1 (collectiton)	Step3.4.2(NFT Item)	Step3.4.3(Receiver TRC404 wallet)
2	FT1(A transfer FT to B)											
3	Step 1(Sender TRC404wallet)											
4	Step 2.1(collection)											
5	Step 2.2(NFT Item)											
6	Step 2.3(Sender TRC404wallet)											
7	Step 3.1(Receiver TRC404 wallet)	jetton_balance										
8	Step 3.2(collection)											
9	Step 3.3(NFT Item)			contract active status								
10	Step 3.4(Receiver TRC404 wallet)	jetton_balance, owned_nft_number, owned_nft_dict			owned_nft_dict							
11	Step 3.4.1(collectiton)											
12	Step 3.4.2(NFT Item)			contract active status								
13	Step 3.4.3(Receiver TRC404 wallet)	owned_nft_dict			owned_nft_dict							

Note: Similar to 3.1.1, data updates for the collection contract will no longer be analyzed, only potential conflicts between the TRC404 Wallet contract and the NFT Item contract will be analyzed.

3.1.2.1 FT1 Step 3.1 and FT2 Step 1 (TRC 404 Wallet)

There is one data involving a race condition, jetton_balance.
For jetton_balance, the operation performed in Step 3.1 is to directly attempt to add the amount included in the internal_transfer request to the contract, and then subtract the number of NFTs that need to be minted. In Step 1, the contract directly attempts to subtract the amount contained in the transfer request. If the result is less than 0, it exits directly and no further operations are performed.

The operations of Jetton_balance are all atomic operations, and there is no race condition problem.

3.1.2.2 FT1 Step 3.3 and FT2 Step 2.2 (NFT Item) *

There is one data involving a race condition, contract active status.

For contract active status , the operation performed in Step 3.3 is mint NFT, so the contract active status will become active. The operation performed in Step 2.2 is burn NFT, so the contract's active status will become non-active.

If a race condition conflict occurs, the NFT of the operation performed in Step 3.3 and the NFT of the operation performed in Step 2.2 are required to be the same NFT. But in Step 3.1 of FT1 , adding NFT index to owned_nft_dict is not written in Step 3.1 , but needs to wait until Step 3.2, Step 3.3, and Step 3.4 are executed before the corresponding NFT is added, then add jetton_balance and NFT item index to owned_nft_limit.

Therefore, for FT2 Step 2.2, if you want to make the NFT index of Step 2.2 burn the same as the NFT index of FT1 Step 3.3 mint, you must wait until FT1 Step 3.4 is completed. If FT1 Step 3.4 has already been executed, then when executing FT2 Step 2.2, even if it is the same NFT, it still operates normally and there will be no abnormal situation like attempting to burn the NFT contract before the NFT is successfully deployed. Therefore, there will be no conflicting competitive conditions between these two steps.

3.1.2.3 FT1 Step 3.4 and FT2 Step 1 (TRC 404 Wallet)*

There are three data involving race conditions, jetton_balance , owned_nft_number, owned_nft_dict.

For jetton_balance , in Step 3.4 , jetton_balance may execute jetton_balance + 1 - pending_reduce_jetton_balance or set jetton_balance =0 . In Step 1, the contract directly attempts to subtract the amount contained in the transfer request. If the result is less than 0, it exits directly and no further operations are performed. These are all atomic operations and there will be no competition issues.

For owned_nft_number , in Step 3.4 , owned_nft_number may increase by 1 or remain unchanged. In Step 1, owned_nft_number will be subtracted from the number of nft that needs to be burned. These are all atomic operations and there will be no race issues.

For owned_nft_dict, in Step 3.4 , an NFT index will be added into owned_nft_dict or remain unchanged. In Step 1, will try to find the NFT index with the normal status that needs to be burned and set the status of these NFT indexes to pending_delete.

The situation about owned_nft_dict is more complicated. Let’s analyze it in detail below:

If FT2 Step 1 is executed first and Step 3.4 is executed later, then the NFT index operated by Step 1 must be different from Step 3.4.

If Step 3.4 of FT2 is executed first and Step 1 is executed later, there may be situations where the NFT index to be set in Step 1 is the same as the NFT index added in Step 3.4. However, after Step 3.4 has been executed, all operations have been completed. Therefore, even if it is the same NFT, it is still normal to operate, and there will be no abnormal situations such as FT1 attempting to burn the NFT contract before the NFT that needs to be minted is successfully deployed. In summary, there will be no conflicting competitive conditions between these two steps.

3.1.2.4 FT1 Step 3.4 and FT2 Step 2.3 (TRC 404 Wallet)*

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, in Step 3.4, an NFT index will be added into owned_nft_dict or remain unchanged. In Step 2.3, the specified NFT item index will be deleted from owned_nft_dict.

The reason why these two steps will not conflict with race conditions is similar to what is described in 3.1.2.3 .

3.1.2.5 FT1 Step 3.4.2 and FT2 Step 2.2 (NFT Item) *

There is one data involving a race condition, contract active status

For contract active status, the operation performed in Step 3.4.2 is to burn NFT, so contract active status will become non-active. The operation performed in Step 2.2 is to burn NFT, so the contract's active status will become non-active.

If there is a conflict of competitive conditions, the NFT operated in Step 3.4.2 and the NFT operated in Step 2.2 need to be the same NFT. But if you enter step 3.4.2 of FT1, it means that in step 3.4, the contract did not add the NFT index to ownedunft_dict. Therefore, step 1 of FT2 cannot obtain the index of the NFT to be burned in step 3.4.2, that is, the NFT operated in step 3.4.2 and step 2.2 must be different, so there will be no competition condition conflict.

3.1.2.6 FT1 Step 3.4.3 and FT2 Step 1 (TRC 404 wallet) *

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, in Step 3.4.3, an attempt will be made to delete a specified NFT index from the owned_nft_dict (the deletion may fail because the NFT index does not exist). In Step 1, will try to find the NFT index with the normal status that needs to be burned and set the status of these NFT indexes to pending_delete.

The reason why these two steps will not conflict with race conditions is similar to what is described in 3.1.2.5.

3.1.2.7 FT1 Step 3.4.3 and FT2 Step 2.3 (TRC 404 wallet)*

There is one data involving a race condition, owned_nft_dict.
For owned_nft_dict, in Step 3.4.3, an attempt will be made to delete a specified NFT index in owned_nft_dict (the deletion may fail because the NFT index does not exist). In Step 2.3, an attempt will be made to delete a specified NFT index from owned_nft_dict.

The reason why these two steps will not conflict with race conditions is similar to 3.1.2.5.

3.2 Transfer NFTs Competitive Analysis

3.2.1 A transfer NFT to B and A transfer NFT to C competition analysis

Because the entry contract of Transfer NFT is NFT Item, when NFT Item receives the transfer_nft_item message, it will first check whether the sender is the owner of the NFT. If not, it will exit and will not continue processing. Therefore, for A transfer NFT to B, and A transfer NFT to C, regardless of which transaction is executed first, only one transaction can successfully change the owner of the NFT, and the other transaction will definitely fail and then exit. So there will be no conflict of competitive conditions between these two types of transactions.

3.2.2 A transfer NFT to B and B transfer NFT to C competition analysis

NFT1: A transfer NFT to B, NFT2: B transfer FT to C. The analysis of possible competing nodes in these two transactions are shown in the following table:

	NFT2(B transfer FT to C)	Step 1(NFT Item)	Step 2(collection)	Step3(Sender(B) TRC404wallet)	Step4(Receiver(C) TRC404 wallet)	Step4.1 (collectiton)	Step4.2(NFT Item)	Step4.3(Receiver(C) TRC404 wallet)
2	NFT1(A transfer FT to B)							
3	Step 1(NFT Item)	owner					contract active status	
4	Step 2(collection)							
5	Step 3(Sender （A） TRC404wallet)							
6	Step 4(Receiver （B） TRC404 wallet)			jetton_balance, owned_nft_number owned_nft_dict				
7	Step 4.1(collectiton)							
8	Step 4.2(NFT Item)	contract active status					contract active status	
9	Step 4.3(Receiver(B) TRC404 wallet)			jetton_balance, owned_nft_number owned_nft_dict				

Note: Similar to 3.1.1, data updates for the collection contract will no longer be analyzed, only potential conflicts between the TRC404 Wallet contract and the NFT Item contract will be analyzed.

3.2.2.1 NFT1 Step 1 and NFT2 Step 1 (NFT Item)

There is one data involving a race condition, owner
For the owner, in Step 1, the contract will transfer ownership from the sender to the receiver.
Therefore, if NFT1 is executed first and NFT2 is executed later, both transactions will be executed successfully in Step 1, and there will be no race condition conflict. If NFT2 is executed first and NFT1 is executed later, NFT2 will fail because the owner of NFT is still A at this time, not B. In summary, there is no race condition conflict.

3.2.2.2 NFT1 Step 1 and NFT2 Step 4.2 (NFT Item)

There is one data involving a race condition, contract_active_status.
For contract_active_status, Step 1 will only change the owner, but not the contract active status. Step 4.2 will burn NFT, so contract_active_status will become non-active.
The reason why these two steps will not conflict with race conditions is similar to 3.2.1.1.

3.2.2.3 NFT1 Step 4 and NFT2 Step 3 (TRC404 wallet)* **

There are five data involving race conditions, jetton_balance,owned_nft_number,owned_nft_dict,pending_jetton_balance , and pending_transfer_nft_queue.

Consider the following two scenarios:
The first case: NFT1 Step 4 is executed first, and NFT2 Step 3 is executed later. At this time, because Step 4 has added the NFT index to B's owned_nft_dict, and B's jetton_balance has + 1, owned_nft_number has + 1, NFT2 Step 3 can successfully delete this NFT index from owned_nft_dict, and jetton_balance - 1 and owned_nft_number - 1 can be executed successfully.

The second case: NFT2 Step 3 is executed first, and NFT1 Step 4 is executed later. At this time, because Step 4 has not yet added NFT index to B's owned_nft_dict , and B's jetton_balance has not yet + 1 , nor has owned_nft_number +1, so:
(1) NFT2 Step 3 executes deleting this NFT from B's owned_nft_dict index will fail, so transfer is required The index is put into pending_transfer_nft_queue.
(2) In addition, because B's jetton_balance is 0 at this time, and then - 1 will be negative, so contract will let pending_jetton_balance +1 and jetton_balance still be 0.
(3) For owned_nft_number , owned_nft_number is 0 at this time, and owned_nft_number is of type int, which can store negative values, so owned_nft_number -1 is equal to - 1.

After NFT2 Step 3 is executed, NFT1 Step 4 will be executed:
（1）jetton_balance +1 - pending_jetton_balance ,the result is 0, as expected.
（2） owned_nft_number +1 , the result is 0, which is also in line with expectations.
（3） Remove transfer required from pending_transfer_nft_queue NFT index.
Overall, both situations met expectations and there were no abnormal results caused by competitive conditions

(Note: Why do we need to add the pending_jetton_balance and pending_transfer_nft_queue attributes?
Answer： It's mainly to be compatible with the following situations: B has 0 FT and 0 NFT, and when A transfers NFT to B, B transfers NFT to C, both transactions need to be executed successfully. Because in the getgems.io NFT platform, if you want to put your NFT on sale,actually, you need to execute A transfer NFT to B and B transfer NFT to C situation. Only when both transactions can be executed successfully can be compatible with getgems.io and other similar NFT marketplace platform.)

3.2.2.4 NFT1 Step 4.2 and NFT2 Step 1 (TRC404 wallet)* **

There is one data involving a race condition, c contract active status

Consider the following two scenarios:
The first case: NFT1 Step 4.2 is executed first, and NFT2 Step 1 is executed later. Because after executing NFT1 Step 4.2 , the NFT will be burned, so NFT2 Step 1 will definitely fail to execute because the status of the NFT is already non-active.
The second case: NFT2 Step 1 is executed first, and NFT1 Step 4.2 is executed later. After NFT1 Step 4.2, this NFT Index will be added to B's wallet contract to owned_nft_dict and set the status of NFT index to pending_delete. So, when NFT2 Tx is executed to Step 3, if the status of this NFT index is found to be pending_delete, which will directly terminate the subsequent steps. In other words, in the second case, NFT2 will also fail to execute.
In summary, both situations are in line with expectations (an NFT that will be burned should not be successfully transferred to other users), and there will be no exceptions due to conflicting race conditions.

3.2.2.5 NFT1 Step 4.2 and NFT2 Step 4.2 (TRC404 wallet)

There is one data involving a race condition, c contract active status

These two steps will not occur at the same time, because as long as any transaction enters Step 4.2, the other transaction Step 3 will definitely fail, so there will be no race condition conflict.

3.2.2.6 NFT1 Step 4.3 and NFT2 Step 3 (TRC404 wallet)

There is one data involved in a race condition, owned_nft_dict

Consider the following two situations:
In the first case, NFT1 Step 4.3 is executed first, and NFT2 Step 3 is executed later. If NFT1 Step 4.3 has been executed, it means that the NFT has been burned, so NFT2 will have failed in Step 1 and cannot enter Step 3.

In the second case: NFT2 Step 3 is executed first, and NFT1 Step 4.3 is executed after. NFT2 can enter Step 3, which means that Step 1 of NFT2 must be executed before Step 4.2 of NFT1, otherwise, NFT2 unable to enter Step 3 because the NFT has been burned.
For the second case, there are two sub-cases:
In the first sub-case, when NFT2 executes Step 3, NFT1 has already executed Step 4. In this case, NFT1 Step 4 will add this NFT index to owned_nft_dict and set the index status to pending_delete, so NFT2 will definitely fail to execute Step 3.
In the second sub-case, when NFT2 executes Step 3, NFT1 has not yet executed Step 4. At this time, Step 3 of NFT2 will add the NFT index into pending_transfer_nft_queue, let pending_jetton_balance + 1, owned_nft_number - 1, and send addOneFtAndNFT message to C's TRC 404 wallet contract. At this time, if NFT1 executes Step 4, the NFT index will be deleted from the pending_transfer_nft_queue, and let pending_jetton_balance = 0, owned_nft_number + 1, and will not enter NFT1 Steps 4.1, 4.2 and 4.3.
In summary, all situations are in line with expectations, and no exceptions will occur due to conflicting race conditions.

3.3 transfer FT and Transfer NFT Competitive Analysis

3.3.1 A transfer FT to B and A transfer NFT to B competition analysis

FT1: A transfer FT to B , NFT2:A transfer NFT to B.The analysis of possible competing nodes in these two transactions is shown in the following table:

	NFT2(A transfer NFT to B)		Step 1(NFT Item)	Step 2 (collection)	Step3(Sender(A) TRC404wallet)	Step4(Receiver(B) TRC404 wallet)	Step4.1 (collectiton)	Step4.2(NFT Item)	Step4.3(Receiver(B) TRC404 wallet)
2	Involved data:		owner	/	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance, Pending_transfer_nft_queue	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue	current_total_supply	contract active status	owned_nft_dict
3	FT1(A transfer FT to B)	Involed data							
4	Step 1(Sender(A) TRC404wallet)	jetton_balance, owned_nft_number, owned_nft_dict			jetton_balance owned_nft_number owned_nft_dict				
5	Step 2.1(collection)	current_total_supply							
6	Step 2.2(NFT Item)	contract active status	contract active status					contract active status	
7	Step 2.3(Sender(A) TRC404wallet)	owned_nft_dict			owned_nft_dict				
8	Step 3.1(Receiver(B) TRC404 wallet)	jetton_balance				jetton_balance			
9	Step 3.2(collection)	`current_total_supply,` `next_item_index`							
10	Step 3.3(NFT Item)	owner contract active status	contract active status					contract active status	
11	Step 3.4(Receiver(B) TRC404 wallet)	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue				jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue			owned_nft_dict,
12	Step 3.4.1(collectiton)	current_total_supply							
13	Step 3.4.2(NFT Item)	contract active status	contract active status					contract active status	
14	Step 3.4.3(Receiver(B) TRC404 wallet)	owned_nft_dict				owned_nft_dict			owned_nft_dict

Note: Similar to 3.1.1, data updates for the collection contract will no longer be analyzed, only potential conflicts between the TRC404 Wallet contract and the NFT Item contract will be analyzed.

3.3.1.1 FT1 Step 1 and NFT2 Step 3 (TRC404 wallet)

There are three data involving race conditions, jetton_balance, owned_nft_number, owned_nft_dict.

For jetton_balance and owned_nft_number, FT1 Step 1 and NFT2 Step 3 will perform subtraction operations. Because they are all atomic operations, no conflicts will occur.

For owned_nft_dict, the operation performed by FT1 Step 1 is to set the NFT index status to pending_delete. In NFT2 Step 3, the operation performed is to delete the NFT index from owned_nft_dict or add NFT index into pending_transfer_nft_queue, or leave it unchanged, exit directly, and send a message to notify that the NFT has been burned.

Consider the following two situations: In the first case, NFT2 Step 3 is executed first, and FT1 Step 1 is executed later, because NFT2 Step 3 will delete the NFT index, so the NFT index of FT1 Step 1 operation will not be the same as the NFT index in the FT2 Step 3 operation.

In the second case, FT1 Step 1 is executed first, and NFT2 Step 3 is executed later. Because FT1 Step 1 will set the index status to pending_delete, when NFT2 Step 3 is executed, the contract will exit because the NFT index of the operation has been changed to pending_delete, which is in line with normal expectations (NFT that has been burned cannot be transferred to others.)

In summary, all situations are in line with expectations, and no exceptions will occur due to conflicting race conditions.

3.3.1.2 FT1 Step 2.2 and NFT2 Step 1 (NFT Item)

There is one data involving a race condition, contract active status.

Consider the following two situations: In the first case, FT1 Step 2.2 is executed first, and NFT2 Step 1 is executed later. If FT1 Step 2.2 is executed first, that means NFT has been burned, so NFT2 Step 1 will definitely fail to execute as expected.

In the second case, NFT2 Step 1 is executed first, and FT1 Step 2.2 is executed later. FT1 can enter Step 2.2, which means that FT1 has already changed the NFT index status to pending_delete in Step 1, so even if NFT2 can complete Step 1, it will still fail when NFT2 Step 3 is executed, for the same reason as 3.3.1.1.

In summary, all situations are in line with expectations, and no exceptions will occur due to conflicting race conditions.

3.3.1.3 FT1 Step 2.2 and NFT2 Step 4.2 (NFT Item)

There is one data involving a race condition, contract active status.
These two steps will not conflict for the same reason as 3.3.1.2 .

3.3.1.4 FT1 Step 2.3 and NFT2 Step 3 (TRC404 wallet)

There is one data involved in a race condition, owned_nft_dict.
These two steps will not conflict, the reason is similar to 3.3.1.2.

3.3.1.5 FT1 Step 3.1 and NFT2 Step 4 (TRC404 wallet)

There is one data involving a race condition, jetton_balance.
For jetton_balance, FT1 Step 3.1 and NFT2 Step 4 perform addition operations. Because they are all atomic operations, no race condition conflicts will occur.

3.3.1.6 FT1 Step 3.3 and NFT2 Step 1 (NFT Item)

There is one data involving a race condition, contract active status.
The new NFT minted for B in FT1 Step 3.3 will not be the same NFT operated in NFT2 Step 1 for A, so there will be no conflict.

3.3.1.7 FT1 Step 3.3 and NFT2 Step 4.2 (NFT Item)

There is one data involving a race condition, contract active status.
These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.1.8 FT1 Step 3.4 and NFT2 Step 4 (TRC404 wallet)

There are five data involving competition conditions, jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance, pending_transfer_nft_queue.

For jetton_balance, owned_nft_number, pending_reduce_jetton_balance, both steps perform addition or subtraction operations, and there will be no conflict.

For owned_nft_dict and pending_transfer_nft_queue, similar to 3.3.1.6 , the operations must not be the same NFT. index, so there will be no conflict.

3.3.1.9 FT1 Step 3.4 and NFT2 Step 4.3 (TRC404 wallet)

There is one data involving race conditions, owned_nft_dict.
These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.1.10 FT1 Step 3.4.2 and NFT2 Step 1 (NFT Item)

There is one data involving a race condition, contract active status.
These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.1.11 FT1 Step 3.4.2 and NFT2 Step 4.2 (NFT Item)

There is data involving a race condition, contract active status.
These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.1.12 FT1 Step 3.4.3 and NFT2 Step 4 (TRC404 wallet)

There is a data involved race condition, owned_nft_dict .

These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.1.13 FT1 Step 3.4.3 and NFT2 Step 4.3 (NFT Item)

There is a data involving race condition, owned_nft_dict.
These two steps will not conflict, the reason is similar to 3.3.1.6 .

3.3.2 A transfer FT to B and B transfer NFT to C competition analysis

FT1: A transfer n(n>=1) FT to B, NFT2: B transfer NFT to C. The analysis of possible competing nodes in these two transactions is shown in the following table:

	NFT2(B transfer NFT to C)		Step 1(NFT Item)	Step 2 (collection)	Step3(Sender(B) TRC404wallet)	Step4(Receiver(C) TRC404 wallet)	Step4.1 (collectiton)	Step4.2(NFT Item)	Step4.3(Receiver(C) TRC404 wallet)
2	Involved data:		owner	/	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance, Pending_transfer_nft_queue	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue	current_total_supply	contract active status	owned_nft_dict
3	FT1(A transfer FT to B)	Involed data							
4	Step 1(Sender(A) TRC404wallet)	jetton_balance, owned_nft_number, owned_nft_dict							
5	Step 2.1(collection)	current_total_supply							
6	Step 2.2(NFT Item)	contract active status							
7	Step 2.3(Sender(A) TRC404wallet)	owned_nft_dict							
8	Step 3.1(Receiver(B) TRC404 wallet)	jetton_balance			jetton_balance				
9	Step 3.2(collection)	`current_total_supply,` `next_item_index`							
10	Step 3.3(NFT Item)	owner contract active status	owner contract active status					contract active status	
11	Step 3.4(Receiver(B) TRC404 wallet)	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue			jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue				owned_nft_dict
12	Step 3.4.1(collectiton)	current_total_supply							
13	Step 3.4.2(NFT Item)	contract active status	contract active status					contract active status	
14	Step 3.4.3(Receiver(B) TRC404 wallet)	owned_nft_dict			owned_nft_dict				owned_nft_dict

Note: Similar to 3.1.1, data updates for the collection contract will no longer be analyzed, only potential conflicts between the TRC404 Wallet contract and the NFT Item contract will be analyzed.

3.3.2.1 FT1 Step 3.1 and NFT2 Step 3 (TRC404 wallet)

There is one data involving a race condition, jetton_balance.
These two steps will not conflict, the reason is similar to 3.2.2.3 .

3.3.2.2 FT1 Step 3.3 and NFT2 Step 1 (NFT Item)

There are two data involving race conditions, owner and contract active status.
Consider the following two situations: In the first case, FT1 Step 3.3 is executed first, and NFT2 Step 1 is executed later. If FT1 Step 3.3 is executed first, the NFT has been minted, and the owner is set to B, so NFT2 Step 1 will be executed successfully. It's in line with expectations. In the second case, NFT2 Step 1 is executed first, and FT1 Step 3.3 is executed later. At this time, because NFT has not been minted yet, NFT2 Step 1 will definitely fail, which is also in line with expectations.

3.3.2.3 FT1 Step 3.3 and NFT2 Step 4.2 (NFT Item)

There are two data involving race conditions, owner and contract active status.
Consider the following two situations: In the first case, FT1 Step 3.3 is executed first, and NFT2 Step 4.2 is executed later. If FT1 Step 3.3 is executed first, the NFT has been minted, and the owner is set to B, so NFT2 Step 4.2 will be executed successfully, which is in line with expectations. In the second case, NFT2 Step 4.2 is executed first, and FT1 Step 3.3 is executed later. Because NFT has not been minted yet, NFT2 Step 4.2 will definitely fail to execute, which is in line with expectations.

3.3.2.4 FT1 Step 3.4 and NFT2 Step 3 (TRC404 wallet)

There are five data involving race conditions, jetton_balance,owned_nft_number,owned_nft_dict,pending_jetton_balance , and pending_transfer_nft_queue.

These two steps will not conflict, the reason is similar to 3.2.2.3.

3.3.2.5 FT1 Step 3.4 and NFT2 Step 4.3 (TRC404 wallet)

There is one data involving a race condition, owned_nft_dict
These two steps will not conflict, the reason is similar to 3.2.2.3 .

3.3.2.6 FT1 Step 3.4.2 and NFT2 Step 1 (NFT Item)

There is one data involving a race condition, contract active status
Consider the following two situations: In the first case, FT1 Step 3.4.2 is executed first, and NFT2 Step 1 is executed later, because FT1 Step 3.4.2 will burn NFT, so NFT2 Step1 will definitely fail to execute, as expected. In the second case, FT2 Step 1 is executed first, and FT1 Step 3.4.2 is executed later. FT1 can enter Step 3.4.2, which means that the status of the NFT index has been set to pending_delete in Step 3.4, so even if NFT2 Step 1 is executed successfully, it will definitely fail when NFT2 Step 3 is executed, as expected.

3.3.2.7 FT1 Step 3.4.2 and NFT2 Step 4.2 (NFT I tem) *

There is one data involving a race condition, contract active status.
These two steps will not occur at the same time, because if FT1 enters Step 3.4.2, NFT2 will definitely fail when it reaches Step 3. If NFT2 can execute Step 4.2, the NFT1 will exit after executing Step 3.4, and the subsequent Step 3.4.2 will not be executed. In summary, there will be no race condition conflict between these two steps.

3.3.2.8 FT1 Step 3.4.2 and NFT2 Step 3 (TRC404 wallet) *

There is one data involving a race condition, owned_nft_dict.
These two steps will not occur at the same time, the reason is similar to 3.3.2.7 .

3.3.2.9 FT1 Step 3.4.2 and NFT2 Step 4.3 (TRC404 wallet) *

There is one data involving a race condition, owned_nft_dict.
These two steps will not occur at the same time, the reason is similar to 3.3.2.7.

3.3.3 A transfer NFT to B and B transfer FT to C competition analysis

NFT1: A transfer NFT to B and FT2:B transfer FT to C. The analysis of possible competing nodes in these two transactions is shown in the following table:

	FT2(B transfer FT to C)		Step 1(Sender(B) TRC404wallet)	Step 2.1 (collection)	Step 2.2(NFT Item)	Step2.3(Sender(B) TRC404wallet)	Step3.1(Receiver (C) TRC404 wallet)	Step3.2(collection)	Step3.3(NFT Item)	Step3.4(Receiver(C) TRC404 wallet)	Step3.4.1 (collectiton)	Step3.4.2(NFT Item)	Step3.4.3(Receiver(C) TRC404 wallet)
	Involved data:		jetton_balance, owned_nft_number owned_nft_dict	current_tot al_supply	contract active status	owned_nft_dict	jetton_balance	current_total_supply, next_item_index	owner contract active status	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue	current_total_supply	contract active status	owned_nft_dict
	NFT1(A transfer NFT to B)	Involved data											
	Step 1(NFT Item)	owner			contract active status								
	Step 2(collection)	/											
	Step 3(Sender (A) TRC404wallet)	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue											
	Step 4(Receiver (B) TRC404 wallet)	jetton_balance, owned_nft_number, owned_nft_dict, pending_reduce_jetton_balance Pending_transfer_nft_queue	jetton_balance, owned_nft_number owned_nft_dict			owned_nft_dict							
	Step 4.1(collectiton)	current_total_supply											
	Step 4.2(NFT Item)	contract active status			contract active status								
	Step 4.3(Receiver(B) TRC404 wallet)	owned_nft_dict	owned_nft_dict			owned_nft_dict							

Note: Similar to 3.1.1, data updates for the collection contract will no longer be analyzed, only potential conflicts between the TRC404 Wallet contract and the NFT Item contract will be analyzed.

3.3.3.1 NFT1 Step 1 and FT2 Step 2.2 (NFT Item)

There is one data involving a race condition, contract active status.

Consider the following two situations: In the first case, NFT1 Step 1 is executed first, and FT2 Step 2.2 is executed later. Because NFT1 Step 1 will change the owner of the NFT from A to B, and NFT1 Step 4 has not yet added this NFT index to the owned_nft_dict. So the NFT Item operated by FT2 Step 2.2 will not be the same as the NFT Item operated by NFT1 Step 1, that is, there will be no conflict.
In the second case, FT2 Step 2.2 is executed first, and NFT1 Step 1 is executed later. Similar to the first case, there will be no conflict.

3.3.3.2 NFT1 Step 4 and FT2 Step 1 (TRC 404 wallet)

There are three data involving race conditions, jetton_balance, owned_nft_number, owned_nft_dict.

For jetton_balance, and owned_nft_number, both steps perform addition or subtraction operations, and there will be no conflict.

For owned_nft_dict, there are two situations. Case 1. If NFT1 Step 1 is executed first and FT2 Step 1 is executed later, the two steps can operate on the same NFT index, at this time it means that NFT1 has completed the transaction, so it's a normal process to burn this NFT during transfer FT in FT2. Case 2, if FT2 Step 1 is executed first and NFT1 Step 1 is executed later, the NFT index of these two steps must be different and there will be no conflict.

3.3.3.3 NFT1 Step 4 and FT2 Step 2.3 (TRC 404 wallet)

There is one data involved in a race condition, owned_nft_dict.
These two steps will not occur at the same time, the reason is similar to 3.3.3.2.

3.3.3.4 NFT1 Step 4.2 and FT2 Step 2.2 (NFT Item)

There is one data involving a race condition, contract active status. These two steps will not occur at the same time, the reason is similar to 3.3.3.2 .

3.3.3.5 NFT1 Step 4.3 and FT2 Step 1 (TRC404 wallet)

There is one data involving a race condition, owned_nft_dict.

For owned_nft_dict, there are two situations. Case 1. If NFT1 Step 4.3 is executed first, FT2 Step 1 is executed later. If NFT1 Step 4.3 has been executed, it means that the NFT has been burned, so the NFT index operated in Step 1 of FT2 must be different from the NFT index operated in NFT1 Step 4.3. Case 2, if FT2 Step 1 is executed first and NFT1 Step 4.3 is executed later. Step 1 of FT2 will set the NFT index status to pending_delete, so NFT1 will terminate when it reaches Step 4 and will not enter NFT1 Step 4.3.

In summary, these two steps will not cause exceptions due to conflicting race conditions.

3.3.3.6 NFT1 Step 4.3 and FT2 Step 2.3 (TRC404 wallet)

There is one data involving a race condition, owned_nft_dict.
These two steps will not cause exceptions due to conflicting race conditions, and the reasons are similar to 3.3.3.5.