

Lifecycle of a Transaction

Transactions are constructed by users to express the intent of performing actions in the network. Once in the network, transactions are converted into Receipts, which are messages exchanged between network nodes.

In this page we will explore the lifecycle of a transaction, from its creation to its final status.

Recommended Reading To dig deeper into transaction routing we recommend to read the [nearcore documentation](#)

Receipts & Finality

Lets walk through the lifecycle of a complex transaction, and see how it is processed by the network using blocks as time units.

Block #1: The Transaction Arrives

After a transaction arrives, the network takes one block to validate it and transform it into a single Receipt that contains all the [actions](#) to be executed.

While creating the Receipt, the signer gets NEAR deducted from its balance to pay for the gas and any attached NEAR.

If the signer and receiver coincide - e.g. the signer is adding a Key - the Receipt is immediately processed in this first block and the transaction is considered final.

Block #2: The Receipt is Processed

If the signer and receiver differs - e.g. the signer transfers NEAR to the receiver - the Receipt is processed in a second block.

During this process a Function Call could span across-contract calls, creating one or multiple new Receipts.

Block #3...: Function Calls

Each Receipt created from the function call takes an additional block to be processed. Notice that, if those Receipts are Function Calls they could spawn new Receipts and so on.

Final Block: Gas is Refunded

A final Receipt is processed in a new block, refunding any extra gas paid by the user.

info A transaction is considered final when all its receipts are processed. tip Most transactions will just spawn a receipt to process the actions, and a receipt to refund the gas, being final in 1-3 blocks (~1-3 seconds):

- [One block](#)
 - if the signer and receiver coincide - e.g. when adding a key
- [Three blocks](#)
 - if the signer and receiver differ, since the first block creates the Receipt
 - , and the last reimburses gas

Function calls might take longer, as they can spawn multiple receipts. Network congestion can also increase the time to process a receipt, and thus a transaction.

Transaction Status

As the Receipts of a Transaction are processed they get an status:

- Success
 - : the actions on the receipt were executed successfully
- Failed
 - : an action on the receipt failed
- Unknown
 - : the receipt is not known by the network

If an action in a Receipt fails, all the actions in that Receipt are rollback. Notice that we are talking about the Receipt status,

and not the Transaction status.

The status of a transaction is determined by its first receipt, i.e. the one that contains all its actions. If any of the actions in the first receipt fail, the transaction would be marked as failed.

Notice that, it could happen that a transaction is marked as successful, but some of its receipt fails. This happens when a `FunctionCall` successfully spawns a new receipt, but the consequent function call fails. In this case, the transaction is marked as successful because the original function call was successful.

See the examples below for more details.

Status Examples

Example: Transaction with Transfer

1. bob.near
2. creates a transaction to transfer 10 NEAR to alice.near
3. The transaction is converted into a receipt
4. The conversion fails because bob.near
5. does not have enough balance
6. The transaction is marked as failed

Example: Deploying a Contract

1. bob.near
2. creates a transaction to: * create the account contract.bob.near
3.
 - transfer 5 NEAR to contract.bob.near
4.
 - deploy a contract in contract.bob.near
5. The transaction is transformed into one receipt
6. The account is created, the money transfer and the contract deployed
7. The transaction is marked as successful ✓

Example: Deploying a Contract Fails

1. bob.near
2. creates a transaction to: * create the account contract.bob.near
3.
 - transfer 5 NEAR to contract.bob.near
4.
 - deploy a contract in contract.bob.near
5. The transaction is transformed into one receipt
6. The account is created, but the transfer fails because bob.near
7. does not have enough balance
8. The whole process is reverted (i.e. no account is created)
9. The transaction is marked as successful

Example: Calling a Function

1. bob.near
2. creates a transaction to call the function cross-call
3. in contract.near
4. The transaction is transformed into one receipt
5. The function cross-call
6. creates a promise to call the function external-call
7. in external.near
8. The function finishes correctly and the transaction is marked as successful ✓
9. A new receipt is created to call the function external-call
10. in external.near
11. The function external-call
12. fails
13. The original transaction is still marked as successful ✓ because the first receipt was successful tip You can check the status of a transactions using the [NearBlocks explorer](#) [Edit this page](#) Last updated on Mar 25, 2024 by gagdiez Was this page helpful? Yes No