Token Transfer flow

In theprevious article we saw an example of token transfer between accounts found in different shards. That example was simplified and missed a few steps in the process. That was intentional, to keep the article and the video short yet explanatory, in order to give you a bigger picture for understanding concepts.

In this article we're going to have a look at the same data flow, but in detail and we will consider two additional scenarios:

- Token transfer between accounts found in different shards
- Token transfer between accounts found in the same shard

You may be asking what was missing in the previous explanation. The short answer is:Gas Refunds or simplyRefunds .

If you don't know whatGas is, pleaseread first the article about Gas from our docs.

As forRefunds , here's a quote from theGas article:

Attach extra gas; get refunded!

...

- If you attach more gas than needed, you'll get refunded

...

From NEAR Protocol docsGas. Attach extra gas; get refunded! What refunds mean in terms of data flow It means that literally every Transaction includes a refund. OK, this should be enough for introduction, now let's move on to examples.

Token transfer between accounts found in different shards

Basically it is an extension of the example from theNEAR Data Flow article.

Assume we have two accountsalice.near andbob.near . They belong to differentShards .alice.near sends a few tokens tobob.near .

ATransaction signed byalice.near is sent to the network. It is immediately executed.ExecutionOutcome is the output or result from converting the transaction into aReceipt .

During the above processalice.near , the sender, was charged a fee (gas). TheReceipt created as result of theTransaction follows these rules:

1. It will be executed not earlier than nextBlock
2. Itmust
3. be executed on the receiver'sShard

So, in our case the receiver isbob.near and that account belongs to a differentShard that's why theReceipt moves to the receiver's Shard and is put in the execution queue.

In our example the Receipt is executed in the very next Block.

Almost done. Remember the refund? So theExecutionOutcome for the Receipt will be another Receipt that is refunding the Gas to the sender.bob.near has received tokens fromalice.near . Now,alice.near becomes the receiver for a new (and last) Receipt (keep in mind the sender in this Receipt is alwayssystem).

Keep in mind rule #2: the Receipt must be executed on the receiver's Shard. So this Receipt moves to the Shard wherealice.near belongs to. And it is the last execution in this process.

This is it. Tokens has been transferred from the account on one Shard to the account on a different Shard, and the initial sender,alice.near , received a refund of Gas.

Token transfer between accounts found on the same shard

Let's have a look at the example where both accounts are on the sameShard . The process is the same as in the previous example, except there are no Receipts moving from one Shard to another.

ATransaction signed byalice.near is sent to the network. It is immediately executed.ExecutionOutcome is the result of converting the transaction into aReceipt .

The Receipt is already on the receiver's Shard, so it is put in the execution queue of the nextBlock . It is executed in the next Block, and theExecutionOutcome result is a new Receipt with the refund to the initial sender,alice.near . The Same rules

apply to this Receipt, it is put into the execution queue and executed in the next Block.

This is it. You may wonder why the process is overcomplicated for the same Shard case. The answer is:the same rules are always applied . Also, this mechanism allows to build the NEAR Protocol data flow by only one set of rules, no matter how many Shards exist. Also, we avoid a lot of "ifs" and we don't have to keep in mind different corner cases because the process always follows the same rules. Edit this page Last updatedonDec 5, 2022 byAnton Puhach Was this page helpful? Yes No

Previous NEAR Data Flow Next Tokens