

# Token Transfer flow

In the [previous 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 simply Refunds .

If you don't know what Gas is, please [read first the article about Gas](#) from our docs.

As for Refunds , here's a quote from the [Gas](#) article:

Attach extra gas; get refunded!

...

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

...

From NEAR Protocol docs [Gas. 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 the [NEAR Data Flow](#) article.

Assume we have two accounts `alice.near` and `bob.near` . They belong to different [Shards](#) . `alice.near` sends a few tokens to `bob.near` .

A [Transaction](#) signed by `alice.near` is sent to the network. It is immediately executed [ExecutionOutcome](#) is the output or result from converting the transaction into a [Receipt](#) .

During the above process `alice.near` , the sender, was charged a fee (gas). The [Receipt](#) created as result of the [Transaction](#) follows these rules:

1. It will be executed not earlier than next [Block](#)
2. It must
3. be executed on the receiver's [Shard](#)

So, in our case the receiver is `bob.near` and that account belongs to a different [Shard](#) that's why the [Receipt](#) 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 the [ExecutionOutcome](#) for the Receipt will be another Receipt that is refunding the Gas to the sender. `bob.near` has received tokens from `alice.near` . Now, `alice.near` becomes the receiver for a new (and last) Receipt (keep in mind the sender in this Receipt is always system ).

Keep in mind rule #2: the Receipt must be executed on the receiver's Shard. So this Receipt moves to the Shard where `alice.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 same [Shard](#) . The process is the same as in the previous example, except there are no Receipts moving from one Shard to another.

A [Transaction](#) signed by `alice.near` is sent to the network. It is immediately executed [ExecutionOutcome](#) is the result of converting the transaction into a [Receipt](#) .

The Receipt is already on the receiver's Shard, so it is put in the execution queue of the next [Block](#) . It is executed in the next Block, and the [ExecutionOutcome](#) 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 updated on Dec 5, 2022 by Anton Puhach Was this page helpful?  
Yes No

[Previous NEAR Data Flow Next Tokens](#)