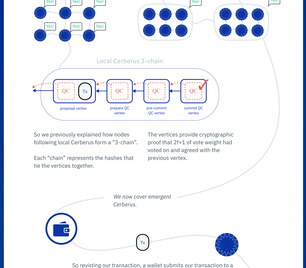
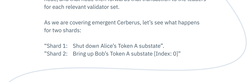
RADIX

Cerberus - How Radix achieves infinite linear scalability while preserving atomic composability

Consensus - Emergent Cerberus

So far we've covered local Cerberus - how nodes come to consensus within a single shard. But as you know, all transactions are cross-shard. There must therefore be a way for validator sets to come to consensus with other validator sets, across multiple different shards.





So we previously explained how nodes following local Cerberus form a "3-chain".

Each "chain" represents the hashes that tie the vertices together.

The vertices provide cryptographic proof that 2f+1 of vote weight had voted on and agreed with the previous vertex.

We now cover emergent Cerberus.

So revisiting our transaction, a wallet submits our transaction to a node, and that node then forwards that transaction to the leaders for each relevant validator set.

As we are covering emergent Cerberus, let's see what happens for two shards:

"Shard 1: Shut down Alice's Token A substate".
"Shard 2: Bring up Bob's Token A substate [Index: 0]"

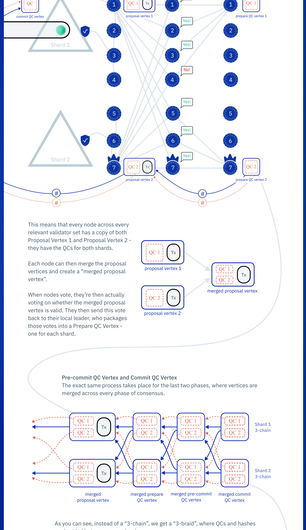
Proposal Vertex

So in the example below, we show the same four nodes for Shard 1 (nodes 1-4); and three nodes for Shard 2 (nodes 5-7). The leaders of each respective shard are nodes 1 and 7.

The key difference between local and emergent Cerberus, is that in emergent Cerberus, nodes "braid" consensus across all relevant shards.

In this example, the hashes for the proposal vertices for both Shard 1 and Shard 2 (QC vertex 1 and proposal vertex 1) derive from the previous transaction.

This is because in our example, we are shutting down substate representing Token A in Shard 1, and bringing up substate representing Token A in Shard 2. The input substate for both shards is thus the same.



1. Broadcast
So for the broadcast step, instead of broadcasting the transaction to only nodes within its validator set, each leader broadcasts the transaction to all nodes across all relevant validator sets.

2. Vote
Then, when nodes vote, they are voting on whether the entire transaction:

(i) sent by both leaders is identical
(ii) is valid as it relates to only their own shard.

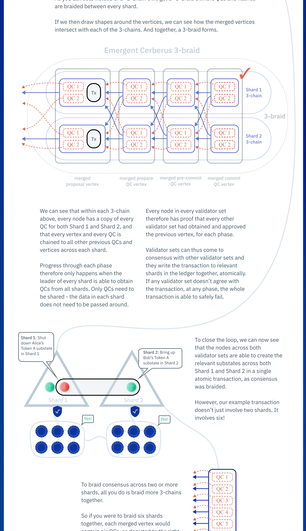
This is because nodes only have data for their own shard, and so they send their vote back to only the leader for their shard.

3. Proposal Vertices
The leaders of each validator set then collect the votes for their own respective shards, and create two separate proposal vertices - Proposal Vertex 1 for Shard 1, and Proposal Vertex 2 for Shard 2. While the transaction is the same in these vertices, the QCs are different, as they represent the votes for different shards.

These vertices both tie back to the same commit QC vertex from the previous transaction, as these are the shut down and bring up substates for the transfer of ownership for Token A.

Prepare QC Vertex

For the next phase, the process repeats: the leader of each validator set now broadcasts their proposal vertex to all nodes across all validator sets.



This means that every node across every relevant validator set has a copy of both Proposal Vertex 1 and Proposal Vertex 2 - they have the QCs for both shards.

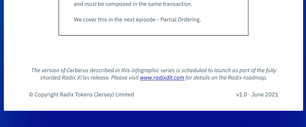
Each node can then merge the proposal vertices and create a "merged proposal vertex".

When nodes vote, they're then actually voting on whether the merged proposal vertex is valid. They then send this vote back to their local leader, who packages those votes into a Prepare QC Vertex - one for each shard.



Pre-commit QC Vertex and Commit QC Vertex

The exact same process takes place for the last two phases, where vertices are merged across every phase of consensus.



As you can see, instead of a "3-chain", we get a "3-braid", where QCs and hashes are braided between every shard.

If we then draw shapes around the vertices, we can see how the merged vertices intersect with each of the 3-chains. And together, a 3-braid forms.

Emergent Cerberus 3-braid



We can see that within each 3-chain above, every node has a copy of every QC for both Shard 1 and Shard 2, and that every vertex and every QC is chained to all other previous QCs and vertices across each shard.

Progress through each phase therefore only happens when the leader of every shard is able to obtain QCs from all shards. Only QCs need to be shared - the data in each shard does not need to be passed around.

Every node in every validator set therefore has proof that every other validator set had obtained and approved the previous vertex, for each phase.

Validator sets can thus come to consensus with other validator sets and they write the transaction to relevant shards in the ledger together, atomically. If any validator set doesn't agree with the transaction, at any phase, the whole transaction is able to safely fail.

To close the loop, we can now see that the nodes across both validator sets are able to create the relevant substates across both Shard 1 and Shard 2 in a single atomic transaction, as consensus was braided.

However, our example transaction doesn't just involve two shards. It involves six!

To braid consensus across two or more shards, all you do is braid more 3-chains together.

So if you were to braid six shards together, each merged vertex would contain six QCs, as depicted to the right.

The cross-shard braiding of consensus in emergent Cerberus is perhaps its single most important innovation. It's what allows transactions to be composed across any number of shards across the entire shardspace, containing any combination of substates, with validator sets committing or rejecting all the substates within those transactions atomically - either all together, or not at all.

Given that all transactions are composed and settled across shards, so long as there are enough shards and nodes to serve them, there will never be a limit to how many transactions the network can process in parallel. The lack of composability that results from sharding blockchains is just not a thing in Radix, as the network has an almost unlimited number of atomically composable shards right from the very start. Radix therefore scales linearly with every additional node added to the network, without ever adding any friction.

Moreover, one of the key concepts that allows this to happen is that of parallel processing. In a future where Radix has scaled to fulfill the DeFi needs of billions of people, the vast majority of transactions will involve completely unrelated substates, shards, and validator sets. Those transactions will be processed entirely in parallel across the vastness of the shardspace without ever being "aware" that the other transactions exist.

But not everything can be processed in parallel. There must still be some logic that tells Cerberus when substates are related and *must* be composed in the same transaction.

We cover this in the next episode - Partial Ordering.

The version of Cerberus described in this infographic series is scheduled to launch as part of the fully sharded Radix 木Ian release. Please visit www.radixdlt.com for details on the Radix roadmap.

© Copyright Radix Tokens (Jersey) Limited