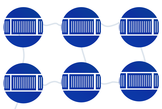
Cerberus - How Radix achieves infinite linear scalability while preserving atomic composability

Partial Ordering - Parallelization of Processing



If you remember how a blockchain works, such as Bitcoin, every node has to store a record of every transaction from the very first transaction. Then every new transaction, across the whole network, has to be processed by the entire network. The Bitcoin ledger is "globally ordered" (not to mention Ethereum and virtually every other blockchain in existence today).

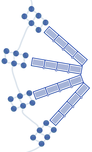
Bitcoin, Ethereum, and other blockchains do this by ordering the transactions within a block, and then ordering blocks in time - chaining each block to the next through hashes; all agreed by the network through consensus.

This severely constrains the number of transactions that can be processed. Imagine you wanted to buy a coffee, but you first had to wait for someone else's coffee on the other side of the world to be processed first! That's what a globally ordered ledger requires.

A more scalable solution would be to parallelize transaction processing. The blockchain shards we talked about in episode 3 are an example of parallel processing.

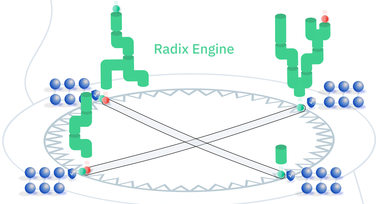
But on a blockchain, you still have to wait for other transactions as transactions are grouped into blocks. Additionally, as we described earlier, blockchain shards face difficulties communicating across shards - breaking "atomic composabilty".

To be truly scalable, what you want is for the network to process your transaction independently, irrespective of anybody else's transaction that is unrelated to yours. It's not waiting around for any other transaction so as to be ordered on a timeline - it just gets processed immediately.





But you also can't have no ordering whatsoever. There will be times when ordering is required because events are related and must execute in a certain order. Think back to the example of selling Token A to buy Token C, the swap from A → B logically has to happen prior to B → C.



So ordering restricts the throughput of a system; but you do need ordering sometimes.

Therefore ideally, you have a network that's intelligent enough to only order when it's absolutely necessary, so that it can process as many transactions as possible in parallel. We therefore want a network that has "partial ordering".

To make this work, the network needs a layer of logic that specifies dependencies - so that it knows when ordering is required or not required.

For Radix, that layer of logic is the Radix Engine. It tells Cerberus which substates are related and must be ordered, and which ones are not related and do not need to be ordered and can thus be safely parallelized.

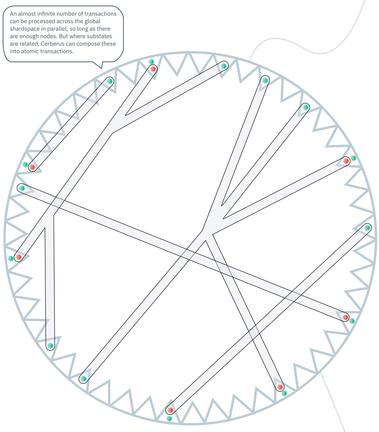
So how does the Radix Engine do this?

When a transaction is created, it must follow rules stipulated by the Radix Engine. Those rules stipulate that only required substates are used in a transaction, and that when new substates are created, that those are broken down into as many substates as possible, that still make logical sense. This is so that if future transactions need to use those substates, those transactions can be processed in parallel.

For example, the substate that represents Alice owning Token A and the substate that represents Bob owning Token B are two separate substates. This means that if Alice or Bob ever wanted to spend those tokens in future, those future transactions could be processed in parallel. It doesn't make logical sense to join them into one substate.

Because of the Radix Engine, nodes therefore only need to undertake consensus on the absolutely most minimal set of shards required.

If events are not related, they just get processed in parallel, across the entire shardspace, which is so large in size as to be practically infinite.



As an example, the five transactions above are submitted to the network around the same time.

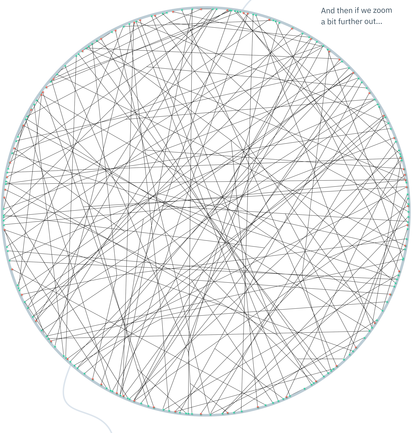
For each transaction, the Radix Engine has forced the transaction to specify the dependencies between substates. So within each transaction, events are ordered.

However, between them, the transactions are completely unrelated - think two different people buying coffees on opposite sides of the world. So at the transactional level, they can be processed safely in parallel.

As the shardspace effectively stretches out to infinity, to process more transactions, all you need to do is add more nodes so that you have more compute and network capability. This is what "linear scalability" means.

Additionally, if a single shard was included in two different transactions, those transactions would then be in conflict! The validator set serving that shard would see two different transactions that it was being asked to vote on, and would have to decide on only one of them. The other transaction would then be rejected.

And then if we zoom a bit further out...



...we can see there really isn't any limit to how many transactions can be processed across the practically infinite shardspace.

So let's now wrap up the final responsibility of a node - maintaining a record of transactions.

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

© Copyright Radix Tokens (Jersey) Limited

v1.0 - June 2021