Suppose that Alice, Bob and Charlie are all users of the sharded Ethereum Serenity blockchain, and all three are on different shards. Each shard has an internal block time of ~6 seconds, but communication between shards takes ~6 minutes because crosslinks between shards only happen once per cycle. Suppose that Alice holds 5 ETH.

Suppose Alice wants to send Bob a payment of 5 ETH. Alice sends a transaction which destroys the 5 ETH on shard A, and creates a receipt (see here for how receipts work) that Bob can then use to claim the 5 ETH on shard B. In order for it to actually be possible for Bob to claim the receipt, shard B must be aware of the state of shard A up until that point, which requires waiting 6 minutes for a crosslink. However, note that Bob gets confirmation of ownership

within 6 seconds

; that is, as soon as Alice's receipt is in the chain on shard A, it becomes clear to Bob that he will be able to claim the funds and (aside from failures of the chain) nothing can happen that can stop him. Hence, after 6 seconds Bob's wallet can show the funds as belonging to Bob.

Now, suppose that as soon as Bob receives the funds, Bob wants to send 5 ETH to Charlie. Unfortunately, it will take 6 minutes for Bob to actually get the funds and be able to make a receipt.

It turns out that by making a change to Bob's account contract, we can get around this problem. We add to Bob's account state a queue of "encumberments", empty by default. An encumberment is a pair (recipient, amount)

. While the encumberment queue of an account is nonempty, Bob cannot send any transactions. However, once funds are available to pay for it, anyone can send a transaction that generates a receipt transferring the amount specified in the first encumberment in the queue from the account to the recipient (this transaction is a simple "poke"; it need not even have a signature attached), and removes this encumberment from the queue.

Now, Bob can

immediately perform an operation that will transfer the funds to Charlie

. Bob creates an encumberment on his account which transfers 5 ETH to Charlie on shard C. Actually transferring the funds will take 12 minutes: 6 minutes for the ETH to reach shard B, at which point the encumberment can be processed and generate a receipt, and then another 6 minutes for the ETH to reach shard C.

However, within 6 seconds, Charlie's wallet software can reason as follows: in shard A, a receipt has been included in the chain, which will transfer 5 ETH to Bob on shard B. In the state of shard B, Bob's account now has an encumberment, the first encumberment in the queue, which will transfer the 5 ETH to Charlie as soon as the funds are available. It is guaranteed that Bob's account can do nothing until the encumberment is removed, and it is guaranteed that in 6 minutes it will be possible (if Bob has not done so already) for Charlie to send a transaction into shard B which will include the receipt from shard A and add the 5 ETH to the account, which will then consume the encumberment and generate a receipt sending the 5 ETH shard C. 6 minutes after that, Charlie will be able to claim the receipt in shard C to get the funds. Hence, Charlie's wallet software can show the funds as belonging to Charlie as soon as Alice and Bob's original transactions get into their respective shard chains

, and in fact, we can extend the encumberment chain further, so Charlie's funds are immediately "spendable" within the same system.

If Bob is receiving and sending funds to and from multiple participants, the logic becomes more complex. The wallet software can generate a directed graph where each node is an account, and an outgoing edge gets added under the following conditions:

- If there is a receipt going from A to B of N coins, add an edge from A to B with weight N.
- If there is an encumberment in B going to C of N coins, and this encumberment is in a position where there are M coins in front of it in the queue, then if the incoming edges from A to B have total weight >= M+N, add an edge from B to C of weight N.

The wallet software then adds to the local balance the total weight of incoming edges. Note that the wallet software does not need to scan the whole chain; a payer can send the payee the subset of nodes that prove that an edge incoming to the payee exists.