Special thanks to Piper Merriam for helping to come up with this idea

This is a generalization of cross-shard locking schemes and similar techniques to enabling cross-shard activity to solve train-and-hotel problems. Philosophically speaking, “locking” a contract that exists on shard A effectively means freezing its state on shard A, saving the state into a receipt, importing the contract to shard B, performing some operation between that contract and some other object on shard B, then using another receipt to send the contract back to shard A where it then continues its existence.

We can simplify and generalize this by changing the mechanism from “locking” to “yanking”. We add an opcode to the EVM, YANK (one stack argument: target_shard

), which deletes the contract from the state and issues a receipt that contains the state of the contract and the target_shard

; this receipt can then be processed in the target_shard

to instantiate the same contract in that shard. Contracts are free to specify their own conditions for when they get yanked.

As an example, a hotel booking contract might work by having a function, reserve()

, that reserves a hotel room, instantiates a contract that represents the hotel room, and that contract then contains a move_to_shard(uint256 shard_id)

function which allows anyone to yank the contract to another shard. One can then solve the train-and-hotel problem by reserving the hotel room, yanking the contract to the same shard as the train booking contract, then atomically booking the hotel room and the train ticket on that shard. If desired, the hotel room contract’s book()

function could self-destruct the hotel room contract, and issue a receipt that can then be used to save a booking record in the main hotel booking contract. If a user disappears after yanking but before atomically booking, then anyone else who wants to reserve the hotel room can just use the same hotel room contract to do so, possibly yanking the hotel room contract back to the original shard if they wish to.

For yanking to be efficient, the yankee’s internal state must be small so that it can be encoded in a receipt, and the gas cost of yanking would need to be proportional to the yankee’s total size. In general, it’s a bad idea from a usability perspective for a contract that could be of interest to many users to be yankable, as yanking makes the contract unusable until it gets reinstantiated in the target_shard

. For these two reasons, the most likely workflow will be for contracts to have behavior similar to the hotel room above, where the contract separates out the state related to individual interactions so that it can be moved between shards separately.

Note that there is a nice parallel between cross-shard messages and yanking, and existing CALLs and CREATEs: CALL = synchronous intra-shard message passing, CREATE = synchronous intra-shard contract creation, CROSS_SHARD_CALL = asynchronous cross-shard message passing, YANK = asynchronous cross-shard contract creation. The YANK opcode does not necessarily need to both delete the existing contract and create a receipt to generate a copy on the new shard. Doing that could require calling both CROSS_SHARD_CREATE and self-destruct; that would make the symmetry complete, though a feature would be needed to allow creation of a contract in another shard with the same address as the original contract.