Here is a scheme for improving the user-experience of contracts getting potentially deleted through storage rent. The general approach is as follows:

- Given any contract, you can easily determine the contract's time-to-live (denoted in block numbers for simplicity). The TTL can be extended by adding more ETH to the contract.
- If the current block number exceeds a contract's TTL, anyone can "poke" the contract to put it to sleep.
- If a contract is asleep, it can be re-woken by submitting (i) a Merkle proof which proves the contract's state at the time of deletion, and (ii) a set of Merkle proofs for each block (or possibly series of blocks) that proves that the contract was not woken already (preventing a scenario where a contract dies at time T1, is resurrected at time T2, dies at time T3, and then is resurrected again but using the Merkle proof from T1 instead of the correct T3)

This can actually all be done fairly safely as a layer-2 system with no base-layer protocol changes by taking advantage of the contract creation address mechanism. The address of a contract created with CREATE is sha3(creator + code + salt)[12:]

, so if we create a specialized factory for creating contracts then one can easily verify that a contract was created by that factory, and so if the contract is put to sleep only that same factory can re-create ("wake") the contract at that address. The factory can have two types of contract creation:

- createNew(code, _salt)

: calls CREATE with salt = sha3(_salt, block.number)

- wake(address, proof)

: checks the Merkle proof and wakes the contract if the proof is valid

The dependence on block number of createNew

ensures that it cannot be used to create contracts at addresses that already have code, and wake

requires a Merkle proof to establish that it's actually waking a contract correctly. If we establish that a contract, if it is created at all, must stay awake for one entire week (that is, require a minimum TTL of 1 week when a contract's state is modified), then the Merkle proof would simply be one Merkle branch proving non-existence of the contract per week. If the block headers contain a skip list where each block points to the block one week earlier, then the proof size would be ~1000 bytes per week, or ~1 MB per 20 years. If a Merkle proof is larger than the maximum size of a block, it can be submitted across multiple transactions in multiple blocks.

The user experience would be as follows. Cross-contract calls to empty addresses could throw by default; this would create a security model similar to that created by access lists and the historical call stack depth limit where contracts need to assume that any contracts they call are not necessarily available. If a user wants to perform a call, and they determine that the call they want to make depends on a contract that is now asleep, they would need to find the historical data, likely from some kind of second-layer market or asking an archive node, to construct the Merkle proofs, and then they would send multiple transactions; the first transaction(s) would be the Merkle proof(s) to perform any needed reawakening(s), and the last transaction would be the user's intended operation.

Keeping contracts in the state would be viewed as paying for the convenience of having them be accessible at a relatively low gas cost without needing to provide Merkle proofs and benefiting from the censorship resistance of arbitrary dynamic state access.