<https://etherworld.co/2022/12/13/transient-storage-for-beginners/>

# Transient Storage for Beginners

EIP-1153: Need, Effects, Pros & Cons, Future Plans

[](https://etherworld.co/author/yash-kamal-chaturvedi/)

Published 13 December 20226 min read

By [Yash Kamal Chaturvedi](https://etherworld.co/author/yash-kamal-chaturvedi/)



## Table of Contents

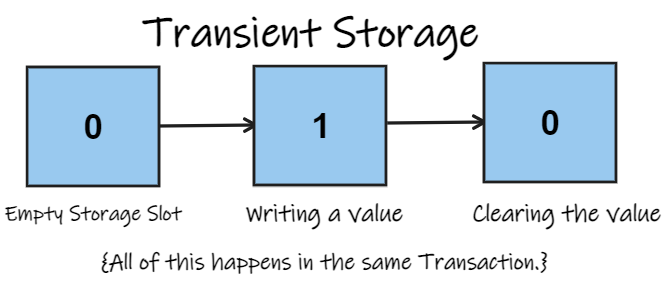
* [What is Transient Storage?](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section1)
* [Use Cases](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section2)
* [Why do we need EIP-1153?](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section3)
  + [Problem](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section31)
  + [Solution](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section32)
* [Effects of EIP-1153](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section4)
* [Pros & Cons of EIP-1153](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section5)
* [Future Plans](https://etherworld.co/2022/12/13/transient-storage-for-beginners/#section6)

"Transient Storage is already conceptually in the EVM. It is more than a gas optimization. It makes feasible new patterns for smart contracts that are much easier to work with, and many future protocols will adopt these patterns."

In the latest [Ethereum All Core Devs Meeting #151](https://youtu.be/y8jZ11_CQGo), Core Developers have decided to omit [EIP-1153: Transient Storage Opcodes](https://eips.ethereum.org/EIPS/eip-1153) from Shanghai Upgrade. EIP-1153 was pushed by Uniswap and Optimism Teams, but developers decided to push Withdrawals with EOF EIPs in the Shanghai upgrade. In this article, we will get an overview of Transient Storage and why it is important.

# What is Transient Storage?

Transient Storage = taking an empty storage slot, writing a value to it, and clearing it in the same transaction, so we don’t need to write anything to disk.

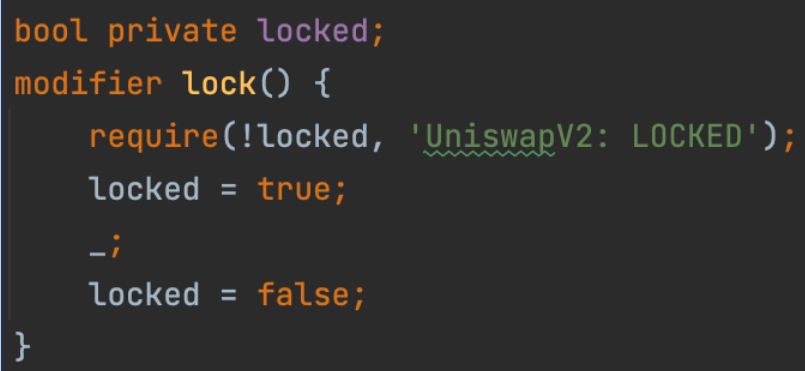


It is a new writable data location that is persistent for the duration of a transaction. Because the blockchain doesn’t have to store transient data after the transaction, nodes don’t have to use the disk, making it much less expensive than storage.

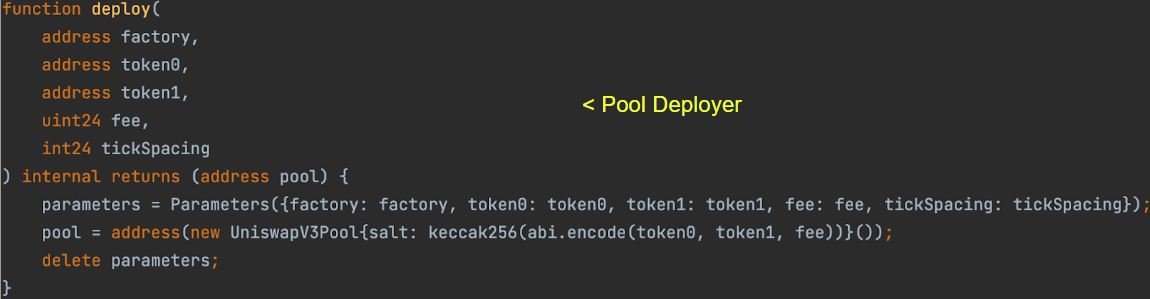
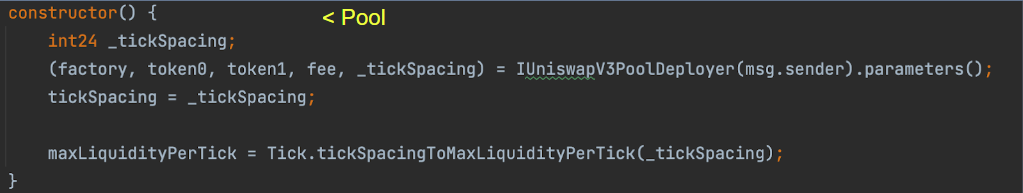
# Use Cases

In this section, we will look at some use cases of transient storage.

* In the present scenario, developers get gas refunds due to EVM accounting rules when they clear the value of the slot in the same transaction they wrote.
* **Re-entrancy Locks**: These locks are required to prevent re-entrants, which can often cause a vulnerability. In the image given below from Uniswap V2, we can see a storage variable that is set to a non-zero value, such as true, and then developers do their contract logic, and then it is set back to false. It is the most common usage. Developers set a storage variable and use that variable to detect and prevent re-entrance. At the beginning and end of every transaction, locked is always false.



* **Reading constructor arguments from msg.sender**: Developers use this in Uniswap V3 to make it cheaper to compute a pool address on-chain. Here, developers use transient storage to store the constructor parameters of a pool so that the pool init code is always the same, rather than passing the constructor arguments as part of the init code. Developers did this so the init code hash would be a constant, not dependent on the pool tokens or fee, so developers could cheaply compute the pool address from the tokens and fee. In the image given below from Uniswap V3, Developers are setting the parameters and then deleting them.

* **Arbitrum uses it for L2-to-L1 outbox:** They use transient storage to represent the L2 state when executing an L2 message on L1. This means the arbitrum contract doesn’t have to assume any interface of the L1 contract, it can make any arbitrary call, but the L1 contract also has access to the L2 state from the call. Click here to learn more about [L2-to-L1 outbox](https://developer.arbitrum.io/arbos/l2-to-l1-messaging).

# Why do we need EIP-1153?

Now, the next question arises if transient storage is a exists, then why do we need EIP-1153.

## Problem

* **Capped Refunds:** Refunds are capped to 1/5th of transaction gas spent since [London](https://etherworld.co/2021/08/05/all-you-need-to-know-about-ethereums-london-upgrade/) fork. Also, the traditional method uses too much transient storage, and we end up paying 20k gas per slot for just 32 bytes of memory.
* **Storage Refunds are not given on revert:** Let's say a user sets a re-entrance lock, and then something happens that we need to revert in the transaction, then the user won't get the gas refund. Also, there is [EIP-3978: Gas Refunds on Reverts](https://eips.ethereum.org/EIPS/eip-3978) to fix this issue.
* **Reading from disk cannot be avoided:** The client has to read the original value of the unlocked slot from the disk every time the pool is used. In the contract, it’s obvious that the slot is always 0 at the beginning and end of a transaction. However, the node cannot know that the slot is always 0 at the beginning of a transaction.
* **Storage Refund logic is complicated:** It is difficult to implement correctly, and hard for solidity developers to understand and inspect.
* **Transient Storage Blocks Parallelism:** As a result, users can't run two transactions in parallel that touches the same contract because the client doesn't know that both transactions fit around parallel because they touch the same storage slot and they have to read from the slot.

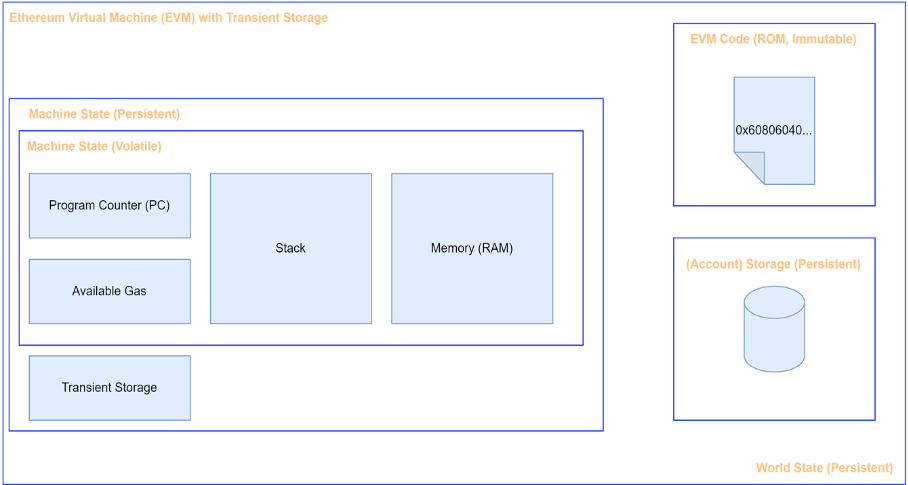
## Solution

EIP-1153 introduces transient storage to Ethereum, which allows storage to be used only for the duration of a transaction. This EIP was proposed by [Alexey Akhunov](https://twitter.com/realledgerwatch) and [Moody Salem](https://twitter.com/sendmoodz) on 15th June 2018.

In addition, it introduces two new opcodes that work the same as storage, except they can have much lower gas prices and no complex refund logic because they do not require reading or writing to the state trie, which are the most expensive operations a node can do.

Transient storage is accessible to smart contracts via two new opcodes, i.e., TLOAD and TSTORE, where “T” stands for “transient". Transient storage is also cheaper since it never requires disk access. In addition, it enables cleaner smart contract designs, saves tons of gas, and simplifies the EVM design.

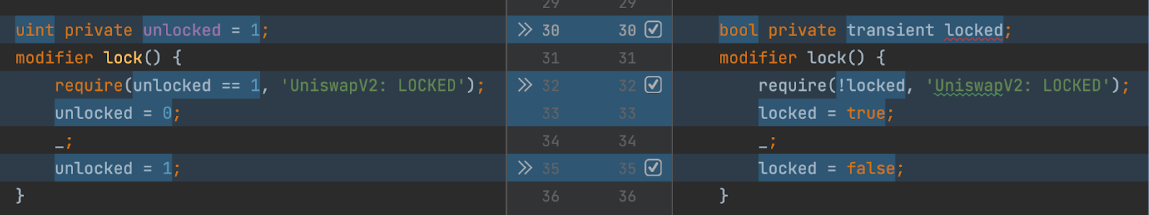
Transient storage is closer to the machine state than the persistent storage region.



Source: [Pascal Marco Caversaccio](https://twitter.com/pcaversaccio)

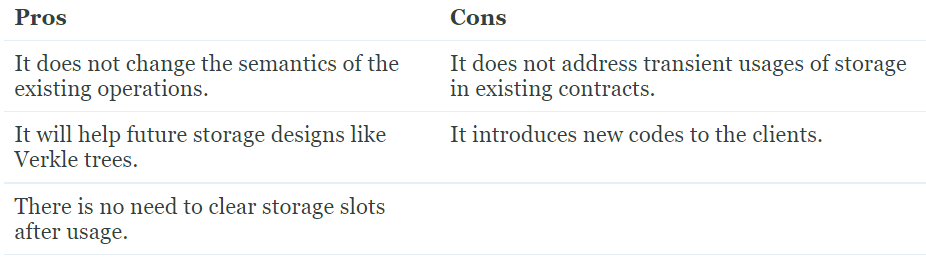
# Effects of EIP-1153

* All the above use cases will be more efficient by using TLOAD and TSTORE, with 0 mental overhead to implement.
* Uniswap V2 could have saved several percentage points in gas on every operation using this EIP-1153. If everyone did this, it would instantly scale the network up a few percent.
* Developers can save a high amount of gas with one simple trick.



* It will help smart contract developers to design new patterns and improve their development experience.
* Transient storage is helpful for any case where a contract needs to store data in one call frame, for use in another call frame during the same transaction.
* Many smart contracts do not use re-entrancy guards due to the gas overhead. With transient storage, developers will protect many more contracts from reentrancy attacks.
* It also enables new smart contract patterns that enable more simple and auditable code.
* In Solidity, a qualifier transient can be introduced to implement EIP-1153.
* Since the addressing scheme of TSTORE and TLOAD is same as for SSTORE and SLOAD, code generation routines that exist for storage variables can be easily generalized to support transient storage.

# Pros & Cons of EIP-1153



# Future Plans

Transient Storage is already implemented in 3/5 EL clients with pending PRs on 2/5. As per [ACD Call #151](https://youtu.be/y8jZ11_CQGo), EIP-4844 is considered the next upgrade's main feature after the Shanghai upgrade, with EIP-1153 and EIP-2537, are considered for inclusion. Developers have also built a comprehensive [test](https://github.com/ethereum/tests/pull/1091) suite. According to [Hayden Adams](https://twitter.com/haydenzadams), developers have completed all the possible work to support this EIP.