Block numbers and time

Arbitrum chains and their parent chains With the release of Arbitrum Orbit, Arbitrum chains can now be L2s that settle to Ethereum (or one of their testnets), or L3s that settle to one of the Arbitrum L2 chains. For simplicity, in this page we speak in terms of Arbitrum One (L2) and Ethereum (L1), but the same logic can be applied to any chain and its parent chain. As in Ethereum, Arbitrum clients submit transactions, and the system executes those transactions at some later time. In Arbitrum, clients submit transactions by posting messages to the Ethereum chain, eitherthrough the sequencer or via the chain'sdelayed inbox .

Once in the chain's core inbox contract, transactions are processed in order. Generally, some time will elapse between the time when a message is put into the inbox (and timestamped) and the time when the contract processes the message and carries out the transaction requested by the message.

In this page we describe what does this mechanism mean for the block numbers and the time assumptions of the transactions submitted to Arbitrum.

Block numbers: Arbitrum vs. Ethereum

Arbitrum blocks are assigned their own L2 block numbers, distinct from Ethereum's block numbers.

A single Ethereum block could include multiple Arbitrum blocks within it; however, an Arbitrum block cannot span across multiple Ethereum blocks. Thus, any given Arbitrum transaction is associated with exactly one Ethereum block and one Arbitrum block.

Ethereum block numbers within Arbitrum

Accessing block numbers within an Arbitrum smart contract (i.e.,block.number in Solidity) will return a valueclose to (but not necessarily exactly) the L1 block number at which the sequencer received the transaction.

// some Arbitrum contract: block . number // => returns L1 block number ("ish") As a general rule, any timing assumptions a contract makes about block numbers and timestamps should be considered generally reliable in the longer term (i.e., on the order of at least several hours) but unreliable in the shorter term (minutes). (It so happens these are generally the same assumptions one should operate under when using block numbers directly on Ethereum!)

Arbitrum block numbers

Arbitrum blocks have their own block numbers, starting at 0 at the Arbitrum genesis block and updating sequentially.

ArbOS and the sequencer are responsible for delineating when one Arbitrum block ends and the next one begins. However, block creation depends entirely on chain usage, meaning that blocks are only produced when there are transactions to sequence. In active chains, one can expect to see Arbitrum blocks produced at a relatively steady rate. In more quiet chains, block production might be sporadic depending on the rate at which transactions are received.

A client that queries an Arbitrum node's RPC interface (for, e.g., transaction receipts) will receive the transaction's Arbitrum block number as the standard block number field. The L1 block number will also be included in the addedl1BlockNumber field .

const txnReceipt =

await arbitrumProvider . getTransactionReceipt ('0x...') ; /**txnReceipt.blockNumber => Arbitrum block number txnReceipt.l1BlockNumber => L1 block number ("ish")* / The Arbitrum block number can also be retrieved within an Arbitrum contract viaArbSys precompile:

ArbSys (100) . arbBlockNumber ( )

// returns Arbitrum block number

Example

Wall Clock time 12:00 am 12:00:15 am 12:00:30 am 12:00:45 am 12:01 am 12:01:15 am L1block.number 1000 1001 1002 1003 1004 1005 L2block.number * 1000 1000 1000 1000 1004 1004 Arbitrum Block number (from RPCs)* *370000 370005 370006 370008 370012 370015* L2block.number : updated to sync with L1block.number approximately every minute. Thus, over time, it will, like the L1block.number , average to ~12 seconds per block.

**Arbitrum block number from RPCs: note that this can be updated multiple times per L1 block (this lets the sequencer give sub-L1-block-time transaction receipts.)

Case study: the Multicall contract

The Multicall contract offers a great case study for the differences between L1 and L2 block numbers.

Thecanonical implementation of Multicall returns the value ofblock.number . If attempting to use out-of-the-box, some applications might face unintended behaviour.

You can find a version of the adapted Multicall2 deployed on Arbitrum One at0x7eCfBaa8742fDf5756DAC92fbc8b90a19b8815bF .

By default thegetBlockNumber ,tryBlockAndAggregate , andaggregate functions return the L2 block number. This allows you to use this value to compare your state against the tip of the chain.

ThegetL1BlockNumber function can be queried if applications need to surface the L1 block number.

Block timestamps: Arbitrum vs. Ethereum

Block timestamps on Arbitrum are not linked to the timestamp of the L1 block. They are updated every L2 block based on the sequencer's clock. These timestamps must follow these two rules:

1. Must be always equal or greater than the previous L2 block timestamp
2. Must fall within the established boundaries (24 hours earlier than the current time or 1 hour in the future). More on this below.

Furthermore, for transactions that are force-included from L1 (bypassing the sequencer), the block timestamp will be equal to either the L1 timestamp when the transaction was put in the delayed inbox on L1 (not when it was force-included), or the L2 timestamp of the previous L2 block, whichever of the two timestamps is greater.

Timestamp boundaries of the sequencer

As mentioned, block timestamps are usually set based on the sequencer's clock. Because there's a possibility that the sequencer fails to post batches on the parent chain (for example, Ethereum) for a period of time, it should have the ability to slightly adjust the timestamp of the block to account for those delays and prevent any potential reorganisations of the chain. To limit the degree to which the sequencer can adjust timestamps, some boundaries are set, currently to 24 hours earlier than the current time, and 1 hour in the future. Edit this page Last updatedonMar 22, 2024 Previous Differences between Arbitrum and Ethereum: Overview Next RPC methods