Mevolution

- Lest Research -

Summer, 2024

Abstract

Traditionally, Miner Extractable Value (MEV) has been viewed as a challenge, where miners or validators exploit arbitrage opportunities through creative block production techniques. The options have been to either combat MEV by limiting opportunities or embrace it by introducing fairness into block production politics. This paper explores a third approach: leveraging the concept of time in blockchain transactions. By transitioning from a sequential model to a flexible time model, we can engineer radical new transaction semantics. These new spatio-temporal dimensions of MEV inspired transactions framework opens up novel possibilities of building L1.5 technologies and hybrid ~~Đ~~apps by leveraging MEV-time Oracles.

1. Introduction: MEV and Its Evolution

Traditionally, MEV has been thought of as a problem dealing with miners/validators exploiting arbitrage opportunities by being creative in their block production mechanisms and techniques. Currently, there exists only two options broadly construed, either fight the MEV by creating ways to limit MEV opportunities or to embrace it and create new ways of bringing back fairness in the block production politics for it is but inevitable, the MEVolution is here. What is underlying these options is the framing of greed once available becomes irresistible. This makes it a design choice between greed versus fairness, how best to strike the delicate balance, and how to introduce other forces in this picture so as to enlighten the dark forest. These new forces come in the forms of

cryptoeconomic mechanisms that have been proposed, most of which, leveraging the PBS and Flashbots research.

There is another way of looking at it. This is the Question and the Concept of Time. Specifically, how it plays out in models of consensus, as is evident in the original name given by Nakamoto for blockchain: *time*chain. MEV has opened up the plasticity of time in Ethereum.

> In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed **time**stamp server to generate computational proof of the chronological order of transactions.
> *— Nakamoto S., Bitcoin: A Peer-to-Peer Electronic Cash System, 2008 [Highlight by the authors]*

The change in blockchain transactions' ordering and execution model from a sequential one to one that reflects a flexible time, as exhibited and exposed by MEV searchers, lets us imagine and engineer radical new transaction semantics. Specifically, transactions can now validate values provided to them by searchers (via transactions infrastructure), reverting when validation fails. Such programmable atomicity (or semi-atomicity and its many variants) exemplify the fact that blockchains are technologies of time.

For example, consider a "smart" transaction pattern where searchers provide valid oracle values to a transaction or else the transaction reverts. This can be used to implement parameterized function calls, allowing searchers to more directly solve and search/optimize transactions. It can also be used to provide signed data to transactions, including off-chain price information or additional third-party transaction validation.

For another example, consider a pattern where searchers must provide values *from the future*, under the logic that the transaction reverts if it finds a different value when it finally arrives at this future point of its execution. This pattern is leveraged by our transactions infrastructure (See §7) called the `callbreaker`, which has searchers front the return values of the `callobjects` (aka internal

transactions) that are executed by the breaker as a bundle. It can also be leveraged in the `time turner`, through which searchers front entire `callobjects` from the future.

Blockchain transactions exist in space and time. The former being the space in the block it is included in, of the chain that such a block is further part of, of the space of the validation logic that legitimizes it to be even considered by the validators/proposers/blockbuilders for inclusion in the current block. The temporalities of blockchain transactions can be expressed in the lifecycle of every transaction in terms of the following three modalities:

- **Past Tense**: Confirmed transactions permanently recorded on the blockchain. They remain in the past frozen, impossible to be tampered with as are all things past, as operating under the immutability logic of ever deepening cumulative confirmation by consensus.
- **Future Tense**: Pending/proposed transactions in the mempool awaiting inclusion in a block. All transactions and requests for transactions once left the wallet and user apps being broadcast node to node as they await to be picked up by searchers/builders/proposers to be worked. In the mempool they remain as potentialities for a future where they might be confirmed.
- **Present Tense**: The transitional time between past and present when transactions are being executed in an effort to confirm them onchain. This is the time when transactions are being picked up, searched/solved, validated and bundled/sorted into blocks by miners or validators/proposers.

As per Nakamoto consensus, the unwritten law of the present tense was that, as long as they are valid transactions, a miner should give them priority for inclusion based only on the amount of transaction fees they pay, informally prohibiting other attempts to interact with the transactions as amounting to tampering the sanctity of temporal purity. "Replace-by-fee" was a controversial Bitcoin soft protocol change, allowing transactions that are in the mempool or already in miners' blocks to be replaced by other mutually exclusive transactions that pay higher fees. That could lead to miners reading the contents of the transactions and selecting new transactions based on their preferred UTXO update or worse leading to censorship. Miners had to act in a passive way, never

actively interacting with transactions, just follow the fee based allocation of blockspace for valid transactions, adding no flavor of their own.

This assumption that miners were passive actors during transaction execution changed for the first time with MEV on Ethereum: ETH miners started interacting with the blockspace by manipulating the order of transactions, giving preferential inclusion to low latency traders who were competing for arbitrage and other trading opportunities by "tipping" or "bribing" miners directly for inclusion. With MEV, miners transitioned from passive to an active mode in the present tense modality of transaction lifecycle. MEV remains to be most radical (change) departure in the temporality of transaction lifecycle in the life of the "timechain" [Nakamoto]. Thus, we get two further delineations under the Present Tense modality of our three transaction temporalities:

- **Present Passive**: Transactions are executed without any significant third-party interpretation of their internal or implicit semantics beyond validity and transaction fees. This was the only mode of temporality as per Nakamoto Consensus.
- **Present Active**: Transactions are executed after an extensive search over different possible inclusions in different bundles with other third-party transactions, not simply based on their own independent validity and transaction fees paid. In the case of trades/swaps these bundles typically include the transactions of adversary traders, or of the miner/proposer themselves. This mode of temporality was made possible solely due to MEV.

	Past	Present	Future
Passive	Confirmed Tx	Nakamoto Miners / Proposers in PBS	Mempool Tx
Active	BTC Scripts	Validators carrying out MEV themselves	Searchers

Table 1

2.1. Current Attempts To Fix The Timechain

MEV represents a major disruption in the temporality of blockchain transactions. Analysts and engineers are trying to find solutions that will get rid of MEV or to wind back the clock on blockchain transaction semantics. This is focused around the framing that MEV resulted in the concentration of powers among and around validators since it made them become the only actors to benefit from the *dual* power of being able to be in the present tense *and also* in the active mode of interacting with transactions. To mitigate this in an effort to ensure fairness, Vitalik and Flashbots introduced a new set of actors, elaborating that of bundle searchers into "builders" along with turning validators into what they are called now "proposers" as is in the name: Proposer-Builder-Searcher.

In PBS, the powers of proposers (erstwhile validators) now supposed have been reset back being only in the present tense (so present *passive*), as was the case with Nakamoto consensus, by restricting proposers to only be able to engage via a block builder auction. PBS leaves the proposers with neither the power to build blocks nor to select/organize transactions, while the searchers/solvers are now wholly responsible with the power to actively interact with transactions. So leaving only builders and searchers as active participants but not present, as actively interacting with transactions in a pre-chain way.

MEV, as an intervention with respect to the temporality and the timing of blockchain transactions, has been largely ignored by the more prevalent rhetoric of MEV being about trading profits/losses, greed, and fairness. Even when the current dominant attempts at addressing the issues of MEV was about decoupling the transaction temporalities via PBS, this aspect around time still remains largely underdeveloped or rarely explicated in technical analytical terms.

Temporality (about states)	Timing (about relationships)
Past—Present—Future	Before—After (and Now, not so much as the midpoint insofar as it is the double negation: as neither before, nor after)

Temporality (about states)	Timing (about relationships)
Confirmed—BeingMined—Mempool	Order of Transactions
Present creates the Past/Future	Before-After creates the Now
Subjectivity (there is no present tense without a subject being present)	Objective (before-after relations are agnostic to subjective experience)
Immutability	Time Travel
Block (Re)-Ordering	Transaction (Re)-Ordering

Table 2

Please note that we have been talking about *two different notions as it regards to time*. The past, present and future tenses represent the modalities in the transaction lifecycle. This is not to be confused with the time travel reference made earlier when transactions and internal transactions make calls to the future (within the same block/bundle) for return values and calls to be returned, and this operates under the relationship of before-after (and now). So the two notions are: past-present-future, and before-after-now. The former speaks of temporalities, while the latter are notions of (time in terms) of timing, so we have *temporality versus timing*. This is akin to the Hellenistic Greek notions of Chronos versus Kairos.

(The differences between the notions of Temporality and Timing carries with it significant similarities with McTaggart's notion of A series versus B series as explicated in his 1908 paper, "The Unreality of Time," [source] in that only the *A series has tense* as in a universe with subjects experiencing the time, while the *B series lacks any sense* of tense as in a block universe.)

3. MEV-Time and Cross-Time Transactions

What we get with the Present Active temporality of a transaction lifecycle, is that finally EVM based infrastructures can now be in an interactive relationship with transactions as they are under the confirmation process. This opens up realtime operations for the first time in crypto, as realtime is the confluence of present and active.

However, to be precise, there was always a case for Active mode of transaction lifecycle in crypto but not in the full spectrum as MEV has opened up. This was the case, where for example, Bitcoin scripts once included in the block, will later on be offchain executed by whoever intends to claim the bitcoins associated with that script, then sending the result in a transaction for it to be included for those coins to be claimed/forwarded, making the active mode being possible *but* at a later time of the initial transaction script being included in the chain, as also its verification of having been executed at an earlier time than the inclusion in the chain. There is also a similar decoupling of active from any present modality for offchain computations that are verified later onchain, or with rollups. But the change that MEV brought was the coming together of the active and the present, which is referred to as realtime.

3.1. MEV-time > Blocktime

With MEV, we have realtime in terms of the temporalities of *transaction lifecycle* (Chronos), as we have time travel with respect to the (timing as now) before-after notion of *transaction timing* (Kairos). These two advancements in technologies of time as the timechain, when put together results in « hypertime = realtime + time travel ». We refer to this as MEV-time, or MEV*t*.

MEV represents an evolution in the way time operates in consensus protocols by evolving from blocktime to MEV-time. Even with the limitations of having to wait for blocktime for confirmation, interaction with the chain is now mid-blocktime as well, as a result of hypertime.

Searchers can interact with transactions during MEV-time by introducing values, updating oracle data, and performing operations within the transaction flow. This interaction breaks the traditional atomicity of transactions, allowing for more complex and adaptive behaviors. By leveraging MEV-time, transactions can become aware of multiple possible futures and adjust their execution based on the most favorable outcomes.

3.2. MEV-Time Oracles: Communicating Across Time

Traditionally, information travels in space, moving between servers, nodes, client endpoints, and so on. Oracles exist to facilitate such movement between blockchains and the outside world, since all information within the chain must be verifiable, oracles act as trusted portals for such movement in space. However, with time travel, transactions can refer to values or calls from the future for their own validation, This is information that is travelling no longer in space, as both sides of the movement remain within the chain (even with the same block at this development stage of our infrastructure). Rather the information is travelling across time: since cross-time transactions do not leave the chain, they come with the same degree of trust guarantees on both sides of source/destination. It is a purely onchain movement of information but with different time horizons. Thus, the terms of on versus off chain are no longer sufficient here, as they are spatial metaphors — on is the space inside the chain as compared to off as outside that space.

We need new terms to refer to movement of information across time but within the same chain. This will be the case of an oracle communicating not between the here and the there (being the case with traditional oracles), but between the now and the then. How about *nowchain* and *thenchain*? How about cross-time transactions instead of cross-chain transactions.

Since the information never leaves the chain, both sides of the here and the now follow the same logic of verification that all onchain information must abide by. Thus, this particular type of oracle is working across time, or if we call it, the MEV-time Oracle. It does not suffer from the challenges of the traditional oracle (i.e., the space oracle), that of the trust assumption. The information of the there can be verified onchain by the now. MEV-time Oracles are onchain oracles.

To sum it up, we can say: it is onchain oracles that act as portals for cross-time transactions by being a pathway for data to move between the thenchain and the nowchain since such data comes with the same verification logic on both sides of the movement.

4. Key Concepts

Key Concepts, or how to reframe existing blockchain concepts in horological ways so we can apply them to our analysis of MEV, leveraging which we then design a horological resolution to MEV.

4.1. Trust as Timing: Timechain design insights

The question of trust is always also one of time (or, *timing* to be precise): if I can go forward in time and always ensure that assertions are never broken, that such a time travel acts as a compensation for the need to trust. What precedes and remains inherent in the question of trust is the question about future. This comes mostly in the form of uncertainty and how we deal with it: what *will* happen in and as the future, what does the future hold for us in terms of the promises being made in the past as we ask this question the present.

Since it remains impossible to escape our present (see §4.3, “when we time travel, so does our present”), the question of trust can then be addressed when reframed in terms of our ability to travel to a transition that come after the current one so as to be able to report back to the current one. More specifically, this is time as

timing in terms of being the before-after relation, and not on temporality (that of the past-present-future relationship). Timing is also the question of when exactly does a transition happen or need to happen, for the need to fulfill an assertion, thus trust assumptions as timing assumptions/assertions.

If we had a crystal ball to look into transitions that come after the current one such that we could make assertions that will *always eventually* come true, we would indeed inhabit a trustless universe. This is echoed in the theme of *trustlessness* ["this is the first time we're trying a decentralized, non-trust-based system", Nakamoto] that Nakamoto inspired in all of us, as the rationale behind the invention of Bitcoin. Thus, the term *timechain* which Nakamoto used to refer to what we now call the blockchain seems in alignment to this logic of trust as a question of time. It is in this sense that we can speak of the concept of trustlessness in terms of timing under the logic of: *to always eventually come true. (In Lamport's framework of distributed systems, the* always-eventual aspect is the liveness guarantee, and the coming-true is the safety guarantee [source])

- **Trusted third parties** can then be framed as third parties that demand our time (as patience) in order for us to be able to infer whether they end up holding onto their promises (as an alternative to Szabo's framing of "Trusted Third Parties are Security Holes", source). Thus, when we deem a service to be untrustworthy, it is because over time it has kept failing at holding on to its guarantees. In both cases, the need to trust can then be translated as the need for the required time to pass, required for us to make the justifiable inference on trustworthiness or not. This makes the time travel argument: that if by some magic, we could peer into a later time from our vantage point in the here and now, that inference could be made immediately. It is in this way that time travel removes the need to trust — by replacing trust with facts from the future travelling back to past for a veracity check — making our relationship toward the service appear trustless. (more on this in §8.1)

- In fact the **replayability guarantee** (inherent in the definition of blockchains) states that if we go back in time and replay through all of the confirmed

transactions exactly in the order they appear on the blockchain (block by block), we will get the same resulting blockchain. Replayability is thus a rationale centered around the conception of the block production being akin to the ticking of a clock: as long as we bracket out the Leader Selection and the Transaction Selection Logic, by just considering the confirmed transactions as they appear on the block, each time we roll back the clock, we get the exact same chain. This is a clock that shows the time of the day in terms of the current block height, while also essentially leaving a trace (of provenance) of all of the time it had shown/ticked (in terms of the confirmed or past blocks — the "past tense" from §2). This is the time that is always and ultimately linear, in that no forks are allowed, as all attempted forks ultimately and always get collapsed into the singular/canonical chain (along with the *uncles* as the traces of those forks that lost, which is the case with some consensus protocols). Replaying is, thus, guaranteed to always and forever produce the same exact blocks all over again every single time. In the rhetoric of crypto, this is framed as trustlessness. This means creating a system where guarantees are so fully enforced that our reliance on trust is unnecessary for it to function as promised. This is largely made possible by the replayability guarantee. Essentially, it mirrors a clockwork universe through consensus mechanisms.

- A similar argument can be made about **verifiable computation**, such that we can have guarantees of what would happen if the computation was run without actually having to run it. When a **ZK Rollup** produces the proof to be anchored in the chain, verifying the proof onchain implies that the offchain computation must have been run with the results as implicated in the onchain proof, all of this, without the chain needing to actually re-run the offchain computation in an onchain way. If an L1 had to run the original computation that would defeat the main point of offchain and expensive computation with cheap onchain verifiability. It is in this way that we can say blockchains also represent technologies of time compression: time seems to move faster in the L2 than in the L1 provided that the former can be

compressed enough to be *squeezed* in between the moments/ticks that make up L1.

4.2. Time Travel Works On Timing and Never on Temporality

When we mention "time travel," it is meant *only* in the way that time could be represented in the before-after mode (i.e., timing), and *not* as the mode of temporality (past-present-future). From a computational perspective, this means that the *time* element of smart transactions is to be modelled as a data structure of transitions/events such that they are ordered in a before-after relationship. This translates to a ordered series of ‘when’ each smart transaction exists.

This reliance on timing is due to the fact that even with going back or forward in time, one (the user, the transactor) can only and always remain in one's present. Where one *is* in time, that is the present moment. There is no present without there being a subject present. This is not a question of experience for even in sleep, we are where are in time, we do not vanish from the timeline just because we are asleep or absent minded. By presence, we mean physical presence, psychological presence is besides the point here. The present, by being so, remains inescapable for subjectivity or subjecthood. Where the subject, is when the present is.

So **when we time travel, so does our present**. Travelling in time remains in terms of temporality. Rather, it is implemented by changing the index of the before-after series of happenings: transaction ordering in the current block being produced, where a transaction belongs in a bundle/block, from where in the current timeline is it expecting calls and `returnvalues` as a condition for its execution). This is an expression of timing and not of temporality. From a computational perspective, this translates to reshuffling the ordering of transitions that make up the transaction. What remains impossible via time travel in smart transaction is reordering of blocks (aka a blockchain reorganization or a

reorg), for that would have been an exercise in time travel with respect to transaction temporality.

(However, when we colloquially speak of going back to the past or the future, if we reframe this with our current understanding, we are referring to the past/future of someone else who is still in the timeline we left from, someone who was in the same present as me before I left for the past/future. In fact, this someone else usually happens to be not another person, rather a projection of ourselves, as if a part of us is still there in the timeline I used to live in prior to my time travel.)

4.3. Time Travel Does Not Break Immutability

No time travelling transaction can go back beyond the current block, that is, to one of the confirmed blocks. However, it can go forward to a block that is not been confirmed, be it the current one in the middle of it being produced, or one that is to come later, by having the transaction remain in an onchain waiting station ready to be picked up by a Solver based on timing assumptions being true. This is key to the rationale behind virtualizing the mempool, as is the case with our infrastructure (see §7). This entails holding the mempool onchain rather than the usual way of prechain mempool. This is carried out by storing the new/pending transitions onchain. While waiting to be picked up is one aspect of timing, the other one is also around locks and partial isolations for transactions to be able to manage their side-effects — this is the logic behind atomicity, and to be precise version of it, that of semi-atomicity. Atomicity remains fundamental as a property for the safety of a transaction execution, even in the case of time travelling transactions.

This means, with time travelling transactions, the infrastructure should be able to change (such as, when deciding to commit a transition or to rollback -and-retry) a before transition/event based on validating what comes after (`returnvalues` or

calls), but must at all costs be barred from being able to change the past. Since the past represents confirmed transactions, we would have to be able to reorder confirmed blocks — that is definitely not conceptually possible and remains technically impermissible by what we mean by and how we design our time travel infrastructure. It is rather searchers reshuffling the before-after relationships of transaction ordering towards optimizing the most MEV. In fact this is where we see the great confluence of the blockchain immutability regime and the replayability guarantee: the latter ensures that we can always go back to the past, while the former prevents us from being able to change the past when we are in it. And now with smart transactions, transactions can travel to before and after places in the timeline to *change* the ordering of transactions being currently solved.

This is how we resolve the paradox of immutability with respect to time travelling transactions, since we:

- can traverse back to past blocks
  - and replay the past (as replayability operates under *temporality*)
  - but never be able to change the past (as immutability operates under *temporality*)
- yet we can change before-after relationships between transactions (this is time travel as it operates under *timing*).

Thus, time travel happens with respect to timing (going back to a before or an after, searchers reshuffling the ordering of transactions), while immutability is a matter of temporality, in that the past/confirmed transaction can never changed (Refer to Table 2). Therefore, contrary to popular opinion, time travel does not break immutability.

Time Travel's reliance on the before-after model is in line with Leslie Lamport's 1978 inaugural work on distributed systems where he framed the question of coordination as one of being able to synchronize around the differing before-after sequences of the nodes of the network; this was his paper on logical clocks. This reliance on timing over temporality as a design criteria is also reflected in

Nakamoto's design of the "peer-to-peer distributed timestamp server" as stated in the Bitcoin Whitepaper.

> In this paper, we discuss the partial ordering defined by the "happened before" relation, and give a distributed algorithm for extending it to a consistent total ordering of all the events. This algorithm can provide a useful mechanism for implementing a distributed system. We illustrate its use with a simple method for solving synchronization problems. Unexpected, anomalous behavior can occur if the ordering obtained by this algorithm differs from that perceived by the user. This can be avoided by introducing real, physical clocks. We describe a simple method for synchronizing these clocks, and derive an upper bound on how far out of synchrony they can drift.
> *- Leslie Lamport, 1978, "Time, Clocks, and the Ordering of Events in a Distributed System"*

4.4. Cryptoeconomics of Time Travel

The logic behind the meme, "don't trust, verify", is one of collapsing the twin sides of a different kind of time travel that what we discussed so far. We refer to this as cryptoeconomic time travel (differentiating it from the time travel using MEV oriented transaction reshuffling, that we discussed so far).

1. **Trustlessness is 'virtual' time travel travel to the future.** To trust x — is to have the assurance that promises that x entails will be kept. But if we are able to visit the future, we would not need to trust, for we would just be in presence of the events happening or not, to verify if indeed it is trustworthy. Thus, being able to free oneself from the burden of trust is as-if one is travelling to the future, so no trust is needed, just direct evidence.

2. **Verification of a Proof is 'virtual' time travel to the past.** But me verifying a *proof of x* signifies that the x already exists/happened for the proof of it to be generated/there. To be able to verify means that the proof is already there, so the event for which the proof is a stand-in, as it already happened. It is as-if

what the proof claims of what happened (what happened, captured as x) is indeed true without me physically travelling to the past, but with the same effect if I did, which is virtual time travel to the past.

In both cases of time travel described above, the future and the past are actual, they are not virtual. However, the travel is not actual but virtual. No one (e.g., a computer agent, a verification engine, an user, fraud checker) is physically leaving the present moment, so as to vanish and reappear at a point in the past or the future. Rather, it a movement in time *by virtue of* the cryptoeconomic setup, by doing so results in the same effect of an actual time travel *but without* actually doing so. This element of 'acting as if,' 'by virtue of, ' 'acting so but without doing so' brings in the aspect of virtuality in terms of the time travel to actual past and actual future.

A major difference must be pointed out. Cryptoeconomic time travel happens on the temporality register while the transaction reordering one is one of timing. The former is possible in spite of the objections in §4.2 because it happens virtually and not actually, while the latter is not a travel to past or future, but to the before or after by way of reshuffling the before-after relationship in terms of transactions ordering.

5. Design Rationale

Design Rationale, or on what principles and guidelines are we to build a time machine on Ethereum.

5.1. TLDR

- **The Current Problem underlying MEV**: Discovery of the « Present *and* Active » mode of Consensus
- **Current Solution** (a la PBS): Decoupling the Present from the Active, wrt Proposers and Searchers bring responsible for each respectively
- **New Solution** (a la STXN): « Time Travel » — Instead of The Decoupling, transactions can validate across time, by leveraging Future Active modes of temporality. This is

MEV-time. Kairos (Timing) to rescue Chronos (Temporality).

5.2. Time Travel Infrastructure

The philosophy of smart transactions extends transaction semantics to improve the ability of transactions to validate their execution. The basic approach is to create a space and an interface where searchers and smart transactions can interact, also allowing smart transactions to interact with each other across time. Smart transactions infrastructure should be used by searchers to provide timely and reliable information to transactions, information based on which transactions can conditionally revert their own execution, and also those of all transactions bundled with them. This information might include, for example, future return values of future transactions, amounts of gas consumed, and oracle values.

Consider a transaction designed to swap tokens. As its first criteria, it wants to maintain a certain slippage protection: *the transaction will search far and wide to find enough hops between many other pairs to eventually settle such that it results in the satisfaction of the acceptable slippage range, failing which, the transaction will revert itself.* As its second criteria, if the tokens to be swapped operate under the conditions that adjust the execution based on realtime price data from an oracle, then if the price moves unfavorably, the transaction can automatically revert or adjust the swap parameters to minimize losses.

Thus, smart transactions infrastructure is to be designed in such a way that the transactions manage their own lifecycle with the aim of ultimately satisfying the constraints encoded in the transactions. This demands that transactions be able to operate in MEV-time allowing transactions to

- self-adjust in an interactive mode with the Searchers (as these transactions are waiting to be confirmed for the current/next block)
- decide whether to execute or revert by verifying truth conditions

The infrastructure should allow for transactions to be able to inspect neighboring transactions within the same timeframe, or on the eventual outcomes from the future (as long as within the MEV-time horizon, which is blocktime in most cases), or on the information from the outside world (Oracles that run on MEV-time instead once every block). “Smart” means being — able to adjust its execution path, manage its lifecycle, — powered by MEV-time interactivity. Thus, we need to design the underlying transaction infrastructure in such a way that it facilitates interactivity.

5.3. Key Design Objectives:

1. **Model Time as Timing (before-after relations) and not as Temporalities (Past-Present-Future).**
    - **Principle**: When a smart transaction exists is a question of where it is in the sequence of before-after relationships of transitions/events.
    - **Implementation**: The time data structure must be able to unequivocally establish the before-after sequence of transitions/events that make up the list of all smart transactions on the system.
    - **Example**: The ordering of transactions in a bundle/block already reflects their internal before-after relationships with respect to each other; this is also the index for each transition, as also the ever-increasing nonce for every attempt at sending transactions (the increasing aspect is what makes the after be at least one more than the before).
2. **Extend Transaction Semantics Over Transaction Execution**
    - **Principle**: The extension of transaction semantics is a fundamental principle that allows transactions to express conditions and preferences beyond the linear execution of code.
    - **Implementation**: Leveraging MEV search, transactions can interact dynamically with the context, making real-time adjustments based on the latest available I/O and state changes.
    - **Example**: A transaction can be designed to execute only if the real-time price data from an oracle falls within a specified range, ensuring optimal conditions are met before execution.

3. **Empowering Transactions to Trust-but-Verify**

- **Principle**: Ensures that if any transaction in a bundle is executed without invalidation, all other transactions in the bundle are also executed successfully.
- **Implementation**: Include verification steps within transactions to check the validity of their conditions before execution. This minimizes the risk of fraudulent or erroneous executions.
- **Example**: A batch of transactions includes multiple token swaps. If one swap fails the verification, the transaction reverts, but the other valid swaps proceed, ensuring partial execution instead of complete failure.

4. **Enabling Context Awareness in Transactions**

- **Spatial Awareness**: Awareness of the current state of the blockchain, including the mempool and blockspace context. This involves understanding the status and details of pending transactions and the state of the world outside the blockchain.
  - **Implementation**: Transactions are designed to adapt based on realtime data from the blockchain and external sources, adjusting their behavior accordingly.
  - **Example**: A transaction adjusts its gas price based on the current congestion in the mempool, ensuring timely execution without overpaying.
- **Temporal Awareness**: Awareness of past, present, and future states, allowing transactions to validate values from the future or recontextualize present execution based on past states.
  - **Implementation**: Transactions consider temporal factors, scheduling execution based on anticipated future events or validating conditions from past states.
  - **Example**: A transaction can be scheduled to execute at a specific future block height or upon receiving a particular oracle update.
- **Always Be Retro-fitting Prophecies To Remain Fulfilled**: Transactions should MEV-timely verify their conditions and adapt to ensure they meet their intended outcomes, even as the blockchain state evolves.
  - **Implementation**: Embed adaptive logic within transactions to MEV-timely check and adjust conditions to align with desired outcomes.
  - **Example**: A transaction designed to trigger an automated market maker (AMM) adjustment MEV-timely checks oracle data and adjusts parameters to ensure market alignment.

5. **Turning Transactions into (Virtual) Programs or Autonomous Entities / Blurring the Boundaries Between Transactions and Contracts**

    - **Principle:** Transactions are no longer static instructions but dynamic entities that can adapt to the blockchain's state (current and future state) and external information.
    - **Implementation:** Transactions incorporate conditional logic and verification steps, enabling them to function autonomously and interact with multiple contracts and data sources.
    - **Example:** A transaction can include logic to check multiple conditions, interact with various smart contracts, and trigger additional actions based on outcomes, similar to a smart contract's functionality.

6. Introducing Smart Transactions

Smart Transactions are **Self-Adjusting** and **Context-Aware:**

- They self-adjust their performance based on assertions and performance guarantees to be strictly enforced.
- They adapt to their environment, reacting to various conditions and contexts to optimize execution.
- They will be more than just programmable; they will "know" their surroundings and adjust accordingly.
- They are supported by an underlying layer of third party searchers that provide compute and storage services (leveraging legacy Web services frameworks) — both reducing the liability for service provider (as they are not real providers but extract value through smart transaction interactions with external third parties specific to domains), but also and more importantly, provide services that can be leveraged through API connections between itself and the Solvers (creating links to offchain services for hybrid apps in the future).

Smart Transactions **Time Travel in Blockchain:** The concept of allowing transactions to borrow from future states to optimize current outcomes. The lifecycle of a transaction can now include temporal claims, such as specific ordering requirements or constraints. This requires robust validation mechanisms to ensure that future states can be reliably accessed and utilized (See § 7.2).

Example Scenarios:

- **Flash Loans**: A user borrows funds, executes a trade, and repays the loan within a single transaction, leveraging future profits for current operations.
- **Zero Capital Trading**: Transactions utilize future gains to execute trades without needing initial capital.

Some of the **Key Features** are:

- **Scheduled Transactions**: Plan and execute transactions in the future, considering dependencies on other events or on the outcomes of previous transactions.
- **Just-In-Time Liquidity**: Access funds precisely when needed, reducing the need for large reserves. Dynamic provision of liquidity using smart contracts that awaken based on predefined conditions. Solvers provide liquidity by preempting race conditions and ensuring advantageous transaction placements.
- **Automated Operations**: Implement schedulers for periodic payments, contract renewals, and other time-dependent actions.
- **Asynchronous Execution**: Handle values not yet computed at the time of transaction execution, interacting intelligently with anticipated future states.
- **Oracle Integration**: Access to execution traces and MEV-time oracles for atomic pre-confirmations, enhancing transaction security.

Compatibility: Designed to work with the existing Ethereum Virtual Machine (EVM) and infrastructure, including Flashbots bundle standards.

7. The Smart Transaction Infrastructure

Alice wants to swap tokens, ensure minimal slippage and fast turnaround. Alice initiated her transaction by sending her request to the `Laminator`. The `Laminator` stored it onchain, acting as a virtual mempool. Its job was to provide spatial awareness. It understood the current state of the blockchain, including the status of all pending transactions. The transaction is now ready to be picked up by the Solver.

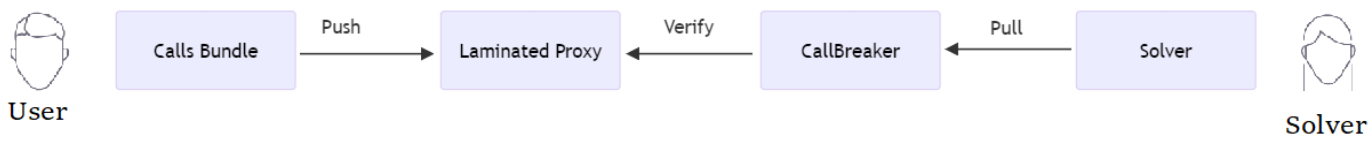

Fig 1. Smart Transaction flow, simplified

With respect to the Laminator's virtual mempool, Alice's transaction gets broken down by the wallet (or the equivalent frontend application used) into smaller individual transitions that make up the entire transaction. These are known as `CallObjects`, or *internal transactions*. Each `CallObject` represents a specific part/transition of the transaction, such as checking the token price, verifying account balances, or executing the swap itself.

The `CallBreaker` interacted with the `CallObjects`, verifying `the truth conditions` for each one. For instance, before executing the token swap, the `CallBreaker` would verify that « the token price was within the acceptable range specified by Alice » (being one of `the truth conditions`). If any condition was not met, the `CallBreaker` would halt the process, preventing any unintended or malicious actions.

7.1. Avoiding The Paradoxes of Time Travel

Any infrastructure that leverages time travel must have the capacity to handle paradoxes around changing events/transitions that further invalidate other such events/transitions which could then lead to invalidating the call from which the time travel was initiated. This is another form of the classic Grandfather Paradox. In order to avoid this, we use Deutsch's solution around "causal consistency check" ["Closed Timelike Curves Make Quantum and Classical Computing Equivalent", source]. This is the logic: upon return from the after/before transition, enforcing a causal consistency check as a default for any commitment on the call execution to be considered final, failing which, causes a rollback followed by trying again with a different timeline such that eventually no such causal consistency checks fail.

Transactions with Temporal Awareness: $T_i = (C_i, t_i)$

- C_i: Conditions.

- t_i: Logical times.

Oracle Function: $O(t) = D$

Example: Time Traveling Token Swap Transaction with Causal Consistency Check

1. **Initial Transaction Definition:**
   *U*ser *init*iates the *T*ransaction.
   $U \rightarrow T_{\text{init}}$

   Alice defines the transaction to swap 100 *X* tokens with a maximum acceptable price of 50 USDC per token at a future date.

   $T_{\text{init}} = \text{SWAP 100 X tokens in the future at } t_{future}, \text{Max Price} = 50 \text{ USDC}$

2. **Storage and Spatial Awareness:** *L*aminator (virtual mempool) stores the transaction onchain, providing spatial (à la blockspace) awareness:
   $L(T_{\text{init}}) \rightarrow T_{\text{store}}$
   Stored transaction with conditions: $T_{\text{store}} = T_i, C_i$

3. ***S*olver and Sequential Execution:** Solver pulls the transaction sequentially for execution (could be at a later block based on temporal conditions: t_{future}).
   $S(T_{\text{store}}) \rightarrow T_{\text{exec}}$

4. ***E*xecution and *V*erification:** Transaction is executed and verified to check if conditions are met.
   $E(T_{\text{exec}})$
   $V(T_{\text{exec}}) \rightarrow \{\text{True}, \text{False}\}$

5. **Interaction with Time:**

   i. **Check Past Conditions:** Oracle provides historical data: $O(t_{\text{past}}) = 45$ USDC
   Condition met as $45 \leq 50$.
   ii. **Check Future Conditions:** Oracle provides future data: $O(t_{\text{future}}) = 48$ USDC
   Condition met as $48 \leq 50$.

Past and future conditions are *V*erified using MEV-Time Oracles.

$V(C_{\text{past}}, t_{\text{past}}) \rightarrow O(t_{\text{past}}) = D_{\text{past}}$

$V(C_{\text{future}}, t_{\text{future}}) \rightarrow O(t_{\text{future}}) = D_{\text{future}}$

6. **Causal Consistency Check:** Ensure the transaction does not create paradoxes: by ensuring that the future condition does not invalidate the past condition.

   $V(T_{\text{exec}}) = \text{True iff } T_{\text{future}} \text{ maintains } C_{\text{past}}$

   This involves finding a fixed point where the state remains consistent over time.

   Find x such that $f(x) = x$

7. **Transaction Completion:** If all conditions are met and causal consistency constraint is maintained, the transaction is considered completed (i.e., ready to be submitted for inclusion in the block, or an equivalent mechanism depending on the specifics of the confirmation logic of the chain). Otherwise, rollback and try again.

   $T_{\text{exec}} \rightarrow T_{\text{complete}}$

7.2. The Smart Transaction Lifecycle

As the `LaminatedProxy` held the transaction onchain, and the `CallBreaker` verified the conditions, the `CallObjects` interacted with each other across time. This dynamic interaction was orchestrated through the concept of MEV-time, allowing transactions to adapt and respond in hypertime (i.e., to both real-time data and future states as well).

The `CallObjects` would query oracles for the latest price data, check gas fees, and ensure that all parts of the transaction aligned perfectly. They communicated seamlessly, adjusting their execution paths based on the information they received.

Once all conditions were verified and optimal conditions were met, the `CallBreaker` would give the green light: the `Laminator` releases the

transactions from the virtual mempool, and the `CallObjects` execute their tasks. If the Solvers actually optimize the transaction's capital performance, Alice's token swap would be completed seamlessly, with minimal slippage and guaranteed timing.

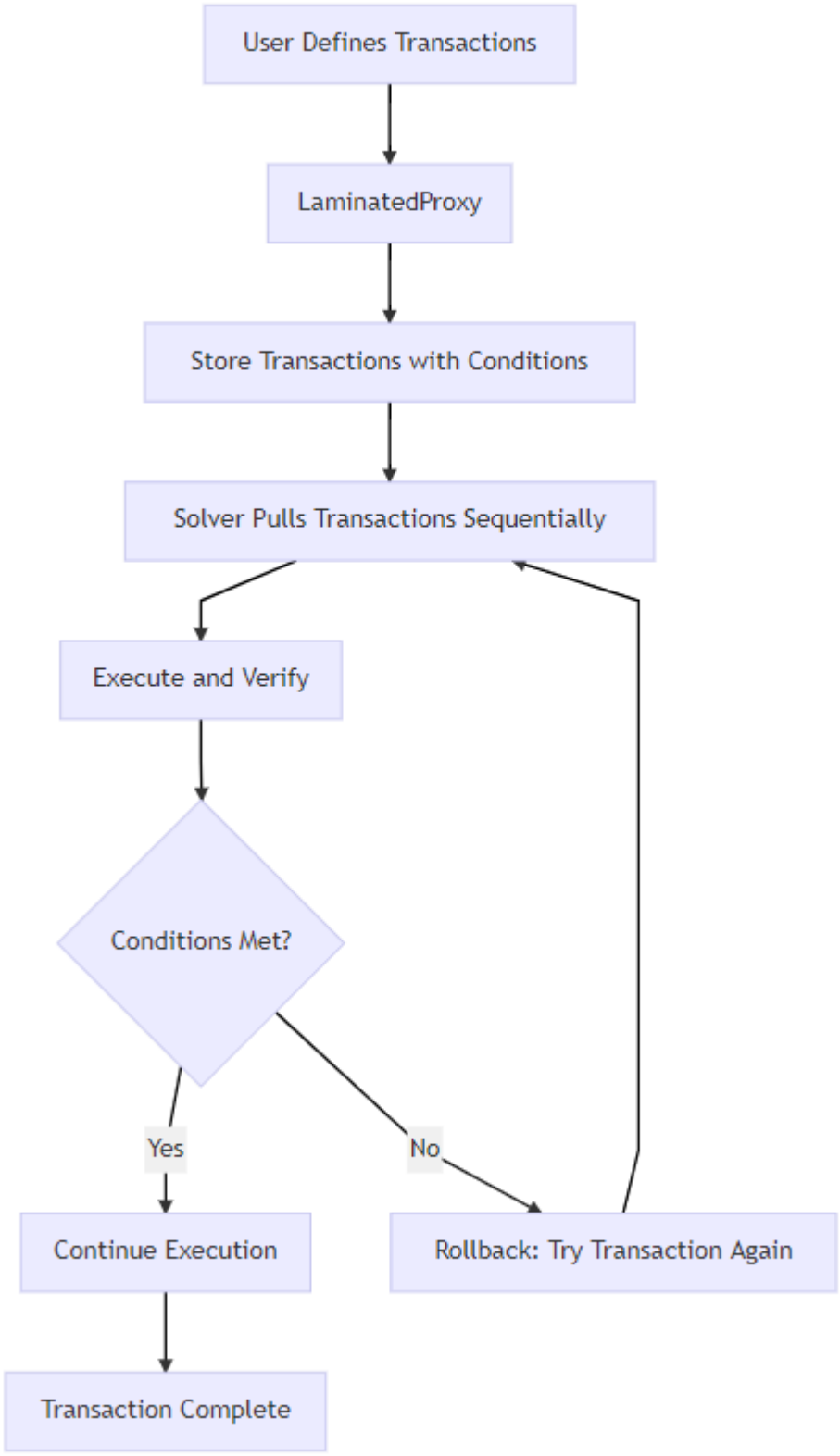

Fig 2. Smart Transaction lifecycle

7.2.1. Example Workflow for a Token Swap with Conditional Execution Alice wants to swap tokens under specific conditions. She pushes her transaction calls to the `LaminatedProxy`, which includes an assertion that the solver must call a function `checkBalance()` to verify the trade conditions. The solver, Bob, then executes these calls, ensuring that Alice's conditions are met before completing the trade.

1. **Transaction Queuing:** Users push a series of `CallObjects` (aka *internal transactions*) to the `LaminatedProxy` with specified conditions. For example, a user might queue a transaction to swap tokens only if certain price conditions are met.
2. **Transactions Stored Onchain:** The proxy stores the internal transactions associated with trade execution along with other internal transactions associated with conditions/assertions.
3. **Verification and Execution:** The `LaminatedProxy` holds these transactions until their specified conditions are satisfiable by the smart transactions solvers. The `CallBreaker` executes the calls and ensures their integrity by reverting and invalidating the bundle if 1) a call fails or if 2) a call doesn't return the expected return value. If the bundle is executed fully, then all included internal transactions' revert-conditions must have been satisfied.

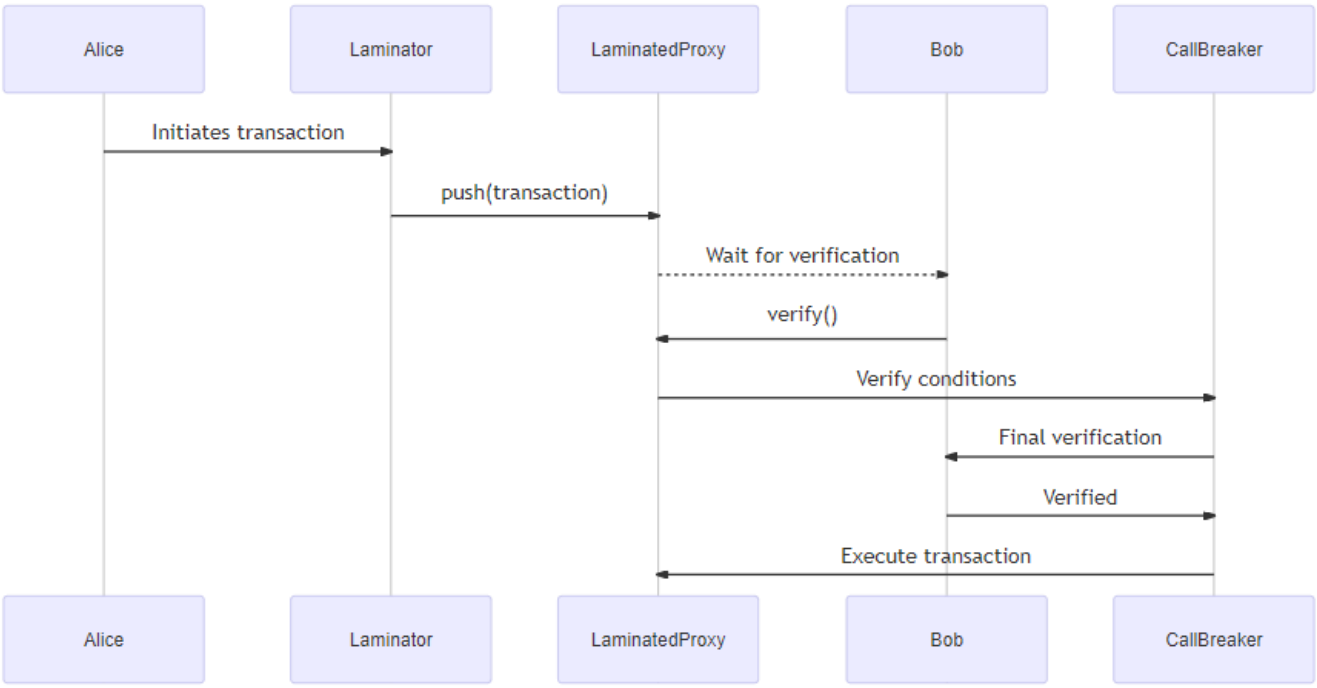

Fig 3. Alice swaps tokens with Bob as Solver

Diagram Description:

1. **Alice**: Initiates a transaction.
2. **`Laminator`**: Manages transactions and uses `push()` to forward the transaction to the `LaminatedProxy`.
3. **`LaminatedProxy`**: Acts as an intermediary, handling calls from users and directing them appropriately.
4. **Bob**: Checks conditions and states within the system.
5. **`CallBreaker`**: Verifies conditions and safeguards the integrity of the transaction flow. Interacts with Bob's verification step as a final check and authorization step before the transaction is executed.

Flow:

- Alice's call to the `LaminatedProxy` includes a transfer of 10 Token A.
- The transaction asserts that Bob will make a future call to `checkBalance()`.
- Bob pulls calls from the `LaminatedProxy` with `verify()` to ensure the conditions are met.
- The `CallBreaker` interacts with Bob's verification as a final check before the transaction is executed.

7.3. The `Time Turner` Mechanism

A mechanism that allows for checking and acting upon conditions in both the past and the future. Enables interaction with future states for temporal flexibility by embedding entire calls from the future. Basically, the setup ensures that smart transactions consider both historical and future states before execution, providing a robust mechanism for **temporally aware condition-based transaction management.**

This is the `Time Turner` high-level flow:

1. **EVM Execution Direction**:
    i. **Check if in the past**: A step to verify past conditions.
        a. The `Time Turner` checks if the condition is met in the past.
    ii. **First make a call to the past, then make call to the future**: A sequential step where past conditions are validated before future conditions.

a. If the condition in the past is met, it proceeds to execute actions related to the past.

b. The sequential call step first makes a call to the past and then to the future.

iii. **Check if in the future**: A step to verify future conditions.

a. Finally, the future condition is checked before executing future-related actions.

2. **Smart Transaction Execution Direction**:
   - This subgraph represents the ordered execution of transactions within the smart contract system.
   - Each step (marked with numbers 1 to 5) represents a direction in the transaction execution sequence/flow.

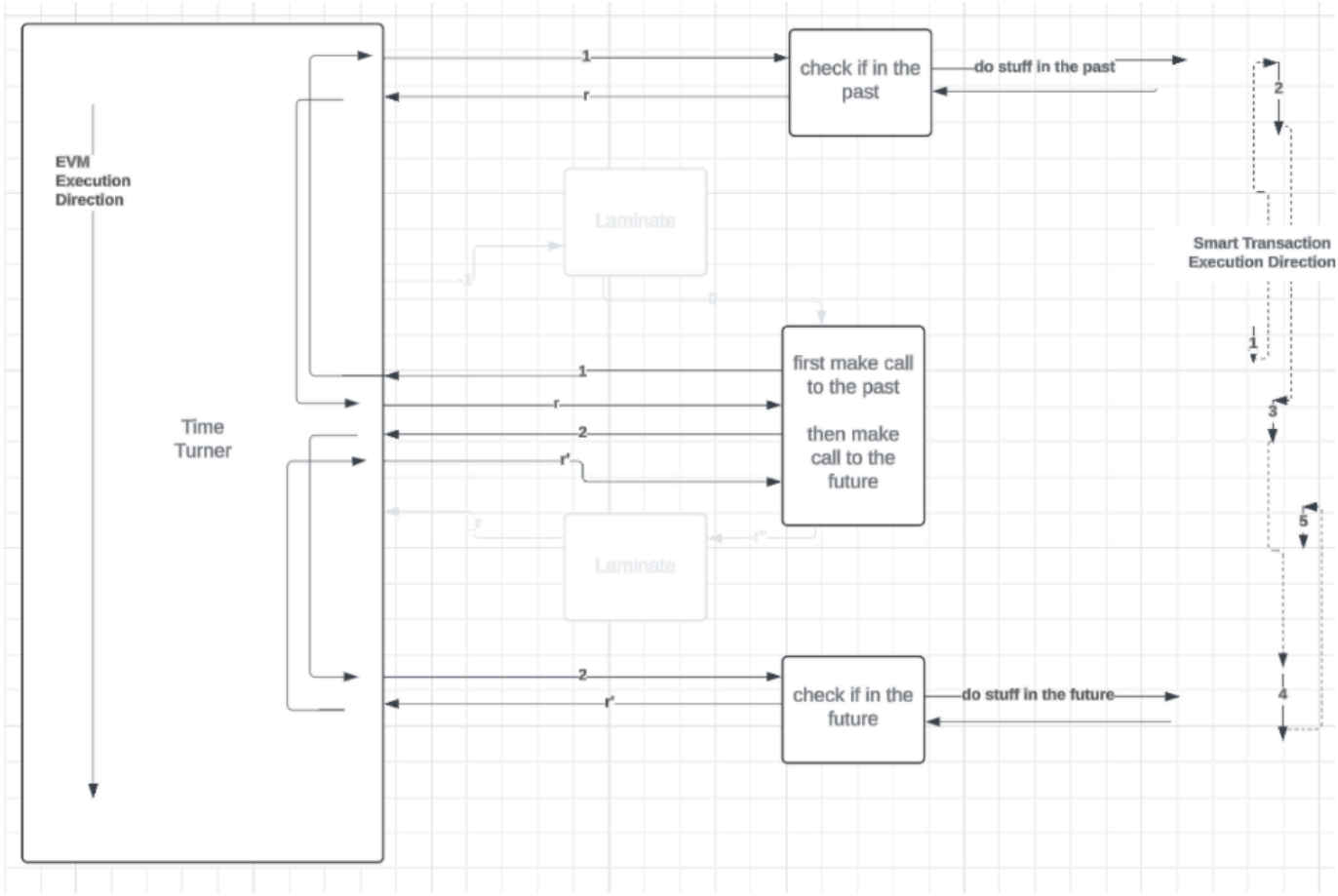

Fig 4. `Time Turner` *High-level flow example*

7.3.1. Detailed Breakdown of The `Time Turner` Lifecycle

1. Initial State (Push to Past)
2. Sequential Calls
3. Future State (Push to Future)

1. **Initial State (Push to Past)**
    - **User Action:** The user initiates a transaction sequence and pushes it to the `LaminatedProxy`.
    - **Condition Check:** This sequence includes a condition to be checked in the past facilitated by the `Time Turner`.
    - **Steps:**
        a. Push the first transaction to check if the condition in the past is met.
        b. If the condition is met, proceed to « do stuff in the past ».

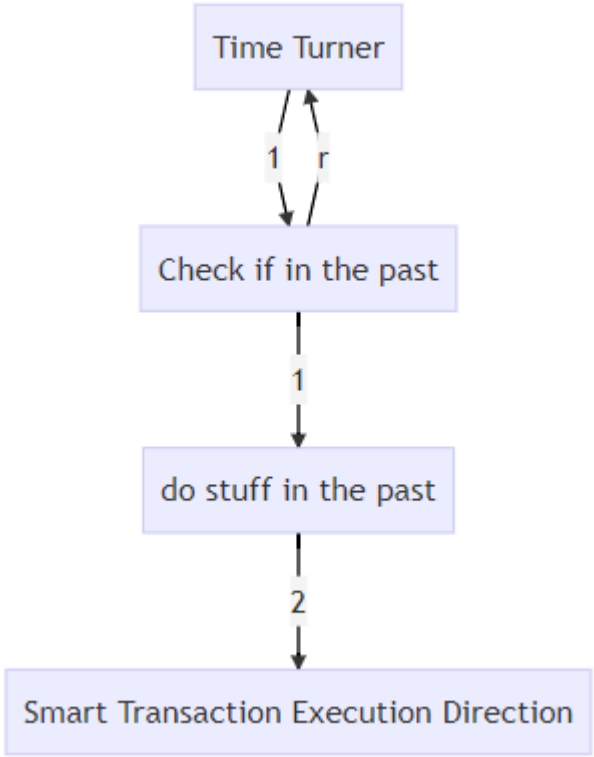

Fig 5. Pushing to Past: Initial State of `Time Turner`

2. **Sequential Calls**

    - **Validation:** After validating the past condition, the next step involves making a call to the past first, followed by a call to the future.
    - **Steps:**
        a. First make a call to the past using the `Time Turner` to validate conditions.
        b. Then make a call to the future to ensure future conditions are set up correctly.

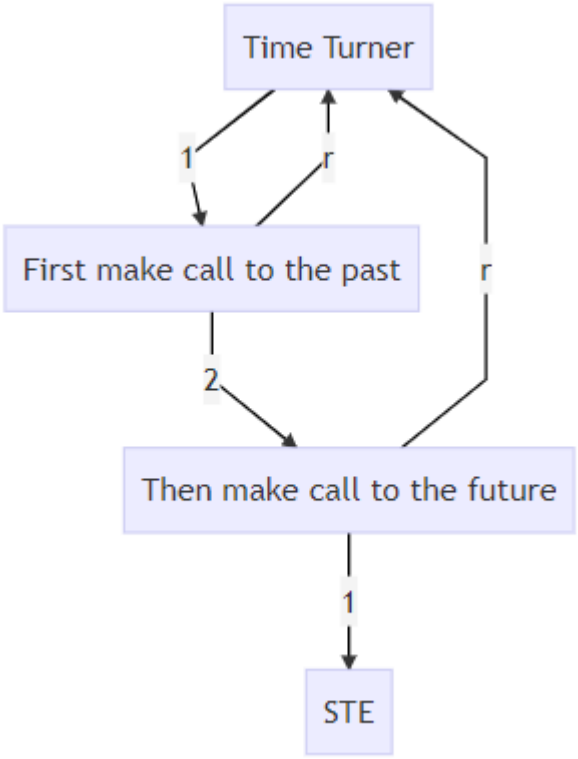

Fig 6. Sequential Calls in `Time Turner`

3. **Future State (Push to Future)**

- **Final Check:** The final sequence involves checking a condition in the future and executing actions based on this future state.
- **Steps:**
  a. Push the second transaction to check if the condition in the future is met.
  b. If the condition is met, proceed to « do stuff in the future ».

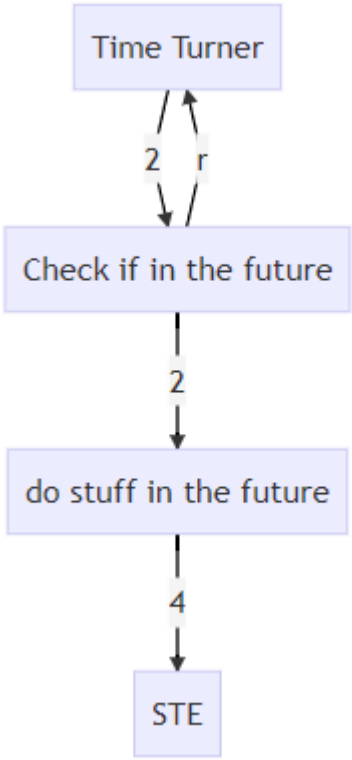

Fig 7. Push to Future in `Time Turner`

8. Beyond MEV: Smart ĐApps

The design of the smart transaction infrastructure must be aligned with respect to the applications that it will be catering to. Thus, a careful consideration of the application layer is essential for a design that is adaptable.

Currently, all of the innovations at addressing MEV, in spite of the high promises, ultimately almost always exclusively end up being deployed to build better and fancier services for swaps. It usually comes wrapped up with the framing of a DEX but at the base of it all, it is still a swap, it still deals with liquidity provision or order books or questions of matching efficiently counterparties. It is tragic and comic at the same time, that behind all of the sophistication of computer science and the complexities of cryptoeconomics, it is all at the service of: how to have slippage protection and low gas fees for swapping tokens. In fact, to look at the broader class of applications of and in crypto, swaps, staking, wallets, attestations, and the various variations of these and with each other, cover all of the use cases that are currently the most used. This is a stark contrast with today's dominant application usage patterns. This is where our approach differs:

1. A (smart) transaction is not always reducible to a swap. (For example, querying a database or uploading a file to cloud storage are transactional operations but not swaps)
2. Solvers must be solving real life problems (including but not limited to swaps)
3. MEV-time Oracles makes syncing I/O much more immediate than the previous limit of blocktime.

Solvers, optimizing for MEV, prioritize transactions that can be easily verified and yield high returns. Swaps, with their clear input-output relationships, are ideal candidates for this optimization. This dynamic has led to an ecosystem where most onchain operations are swaps, sidelining other potential uses of blockchain technology.

Unless a solver also can solve real world problems, like that of finding the most efficient route from Point A to Point B (such as for building a taxi hailing cab onchain), then solvers are only left with solving one task: (capital efficient) swaps. What prevents solvers to provide these other services is the trust assumption required behind these. But with swaps, we can establish objective functions to achieve capital efficiency in ways we can prove the veracity of the auction mechanism used, making it best suited to trust-minimized applications.

This is a careful design choice we make here:

1. Rather than look for a zero trust use case, we look for designing around applications that can be decomposed into its many aspects, some of which can be made trustless within acceptable levels. Each of these parts would act in the form of microservices based **third party searchers** to provide services.
2. MEV-time allows for anchoring trust points mid block instead of the previously possible one of the blocktime being the limit where anything that is beyond L1 (L2 and more) can be anchored, and the same for oracles for outside world communications. Instead with MEV-time anchoring of computations that can be in an interactive mode with the chain in almost realtime (à la just-in-time), or with MEV-time Oracles for outside world communications. We refer to this as **L1.5**
3. Smart ĐApps that are a hybrid leveraging the benefits of both crypto and Web by having the third party searchers sit in between the two worlds.

Microservices based virtual service provider that anchor its I/O (such as, by co-signing a hash of I/O data to have it MEV-timestamped) with respect to the chain

in MEV-time — that is the approach toward building hybrid apps.

8.1. Third Party Searchers

Third Party Searchers (TPS) bridge the gap between Web and crypto by offering services that enhance smart transactions. These services include HTTPS connections to send/fetch data on demand, off-chain services of web hosting, and transaction relaying. This is the aspect of the smart transaction infrastructure that leverages services across chains as also connecting beyond blockchains insofar as the trust assumptions — when leaving the home chain, especially when it relates to non-crypto service endpoints — can be justifiably compensated with respect to the capital benefits from such scale for all stakeholders involved. This would entail domain specific design and deployment of complex incentive schemes that can ensure acceptable levels of honesty across the stakeholders.

TPSs provide services via API connections, enabling smart transactions to leverage Web services for future hybrid applications. There could be multiple TPSs providing similar services in competition with each other, such that they are in a staked service layer marketplace, where if any provider gets caught providing faulty service could be punished. However, TPSs are not real providers but extract value through interactions with external third parties, reducing their liability.

They operate by making API endpoints where a HTTPS query is replied with a JSON object, such that the HTTPS Session and the crypto transaction lifecycle can be mutually incentivized. They are a bridge between Web and crypto, made possible largely due to the discovery of MEV-time, since it opened the possibility of I/O and onchain verifiability that does not have to wait for blocktime but can sync immediately between the worlds of crypto and Web. This opens up the possibility of building Hybrid ĐApps that leverage services between these worlds.

> *Ethereum [...] also opens the door to whole new kinds of applications that have never been seen before.*

> *— ethereum.org on the WayBackMachine from 2014.*

8.2. Hybrid (Ð)Apps need Hybrid Timelines

This question of hybrid applications is one of bridging the differences in trust assumptions between applications, especially, when it caters to decentralized applications on one side and centralized ones on the other, as in that case, the trust assumptions are not just different but incompatible, that is to say, there seems to be an intrinsic incompatibility of trust between these applications resulting from the differing network topologies (centralized/decentralized) of power and/as politics.

It is here that we see the relevance of the idea from a previous section §4.2: "the question of trust is always also one of timing." When we superimpose this idea onto the current one (that of hybrid applications as one of bridging the differences in trust assumptions of Web and crypto) into a single image, we realize that: the question of hybrid applications is one of time travelling but in a specific way; this is not travelling within the same timeline (for that would be the case between applications of compatible/interoperable trust topologies), rather that of time travelling between incompatible timelines, while needing to communicate between these disparate timelines.

Each application with its own timeline reflects the trust assumptions (the topologies that enable those). From a computational perspective, this translates to having not just a single sequence of before-after relations on which the multiple applications' transactions can then be mapped onto, instead multiple such sequences (at least one for each application), many of which do not even align their before-after markers (or index separators) with each other. That is, the transitions that make up the transactions in each application do not always align in terms of their before-after markers, making any attempt at time travel a considerable design issue.

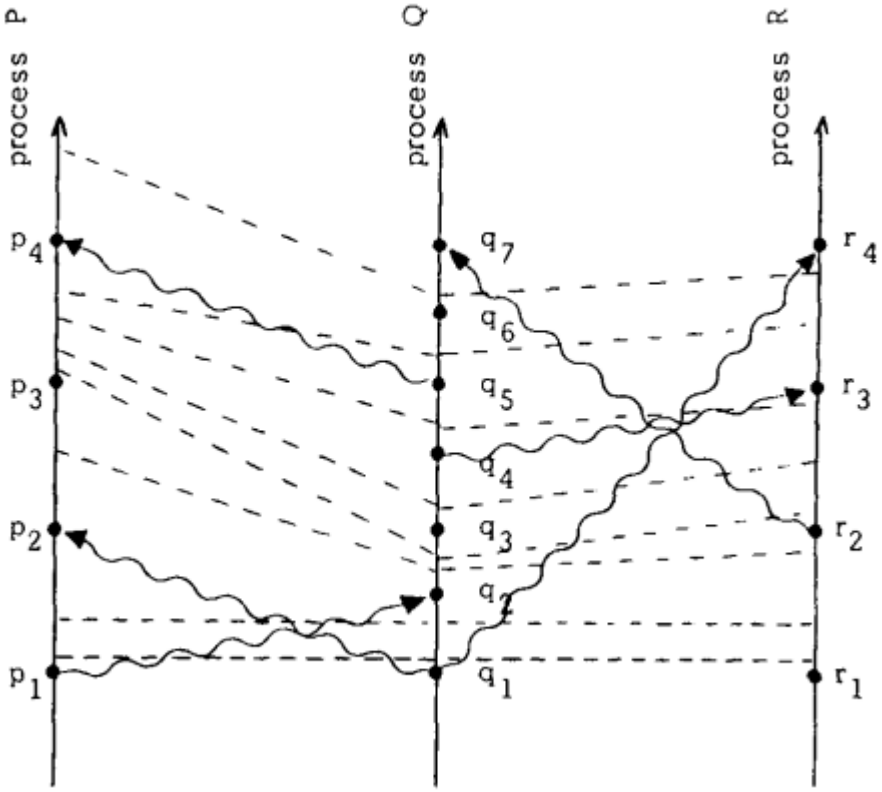

Fig 8. The Example of Three Timelines in Lamport's 1987 paper, a timeline for each process

This takes us back to Lamport's inaugural paper (that introduced the subject of distributed systems in computer science), as this mismatch was precisely the challenge that separated distributed systems (that Lamport wanted to study) from concurrent systems (that previously Dijkstra, Hoare and others studied). Just as Lamport went from improving the work of Dijkstra/Hoare on Concurrent Systems, what we have with MEV (and specifically, STXN in our case) is the challenge of going one step further from Lamport. We can do this by incorporating Nakamoto's improvements on time modeling: in addition to the before-after relationships for transaction ordering in Bitcoin (and later Ethereum) offers, the Proof-of-Work (and later all other consensus protocols in spirit, like that of Proof-of-Stake) makes the time plastic by mandating the sync block by block instead of the continuous model of Lamport and all traditional BFT ones as well, since the blocktime offers enough time to sync the gaps between the different before-after markers of the multiple timelines (as not all nodes in the network will be on the same timeline at all times). Our proposal here is to go one step further building on Nakamoto's sync-block-by-block-within-a-

before-after-sequence, by forcing the sync between the differing and incompatible timelines to happen in MEV-time instead of waiting on blocktime.

Basically, we have to leverage MEV-time Oracles to ensure time travel between systems that differ in their trust (and so timing) assumptions: MEV-time Oracles to bridge Web and crypto (like between Ethereum and other EVM chains). This is the design goal of Hybrid ĐApps as Hybrid Timechains since each app comes its own timeline: syncing up on the timing differences between the different trust topologies.

9. Challenges and Limitations

The first challenge of smart transactions is that they are non-trivial to solve, requiring that someone computationally and possibly interactively solve them, and then to submit their solution to the network for inclusion. This means that sophisticated execution capacity must be developed in order to facilitate smart transactions, which while presented here in the form of the solver network, this functionality can also conceivably be carried out by proposers, builders, or even smart transactors themselves. Organizing this search infrastructure to most efficiently meet the demands of smart transactions is expected to be an ongoing challenge.

A second major challenge facing smart transactions is the gas-efficiency of validation, another is the security engineering of validation. For example, a transaction might require that one of its components is included after expected adversarial transactions, however it may be prohibitively expensive to check every intermediate transaction. For another example, a transaction might want to validate counterfactual claims, but evaluating a lot of counterfactual traces might be prohibitively expensive. In the future, we can imagine transactions cheaply validating succinct zero knowledge proofs of properties that would be expensive to prove directly.

Another set of challenges is associated with transaction solutions being “stolen” from solvers by searchers, perhaps by block builders, or in coordination with

validators. This might violate transaction expectations, and it will reduce the profitability of smart transactions searchers/solvers. It is fundamental that only proposers themselves can protect transactions against the risk of proposers equivocating in order replay their execution, and therefore the separation of proposers from transaction execution introduced by PBS poses a theoretical limit on the achievable security of blockchain transactions.

Thus, the problem space of MEV, even and especially, as we reframe it as a question of time, reveals the inherent political nature of the problem: who decides what time it is now, under what regime do we organize transactions, which transactions are to be considered valid, under whose time are we operating? This is the politics of block production as it changed miners to validators and now proposers with searchers working to search the best bundles, along with the politics of trusted builders.

Hence, it is vital to acknowledge that we must struggle against the ever so pressing allure that a technical solution, no matter how multifaceted and scalable, will put an end to a sociopolitical crisis. This means that we must keep building the tech stack behind the smart transaction infrastructure with never losing in sight the awareness that key to the challenge we face is one of governance. Addressing MEV will also always remain a community effort beyond the efforts of a single protocol/team. This means what we need is a multidisciplinary effort that aids in the technical innovations that smart transactions bring forth, so we can foster effective mechanisms, ways, concepts, avenues, and other such socio-political streams of research and intervention.

10. Conclusion

Through a simple gesture of turning the question of MEV from a purely economic one to a horological one with chronophysical consequences, we offer alternative architectures in addressing the issues with MEV: the discovery of MEV-time has revealed new possibilities for near immediate I/O and even time travel.

Transactions ~~now~~ encompass both spatial and temporal dimensions, creating a verifiable "thenchain" alongside the already existing "nowchain." This temporal distinction allows for onchain oracles, termed MEV-time Oracles, without the same trust issues associated with traditional (i.e, offchain) oracles by keeping all data interactions within the same chain's timeline.

The spatial aspect is enabled with the virtualization of the mempools, by having transactions held onchain (à la `Laminator`) rather than in traditional prechain mempools. This introduces advanced lifecycle management for transactions, such as with the `Time Turner`, transitions (which comprise transactions) can be executed based on conditions that rely on future transitions, sending return values or entire calls across from the future to the past. Solving smart transactions is far from trivial, requiring a multidisciplinary approach that encompasses both technical and governance challenges.

Building on the differences in the modes of time — temporality (past-present-future) versus timing (before-after) — is essential for designing effective time travel dynamics. Emphasizing the timing modality ensures that transactions are executed in a logically coherent sequence, respecting the causal consistency relationships between transitions/events while being able to decide transitions based on future ones.

Smart transactions pay MEV and other application specific fees to searchers/solvers and to Third Party Searchers respectively, in exchange for the search results in terms of the quality of the results. It in this specific economic sense, that the infrastructure acts as a virtual matching engine: in the way a search engine does, but one with a time machine at its core, as the search is carried out combing through the different timelines to find the best possible execution of the transactions.

Smart Transactions builds on the *mevolution* of Nakamoto style Timechains to Time Machines (post MEV), all verifiable and onchain.