SoK: Validating Bridges as a Scaling Solution for Blockchains

Patrick McCorry[1], Chris Buckland[1], Bennet Yee[2], Dawn Song[2]

[1] Infura
[2] Oasis Labs

Abstract. Off-chain protocols are a promising solution to the cryptocurrency scalability dilemma. It focuses on moving transactions from a blockchain network like Ethereum to another off-chain system while ensuring users can transact with assets that reside on the underlying blockchain. Several startups have collectively raised over $100m to implement off-chain systems which rely on a validating bridge smart contract to self-enforce the safety of user funds and liveness of transaction execution. It promises to offer a Coinbase-like experience as users caan transact on an off-chain system while still retaining the underlying blockchain's security for all processed transactions. Unfortunately, the literature for validating bridges is highly disparate across message boards, chat rooms and for-profit ventures that funds its rapid development. This Systematization of Knowledge focuses on presenting the emerging field in an accessible manner and to bring forth the immediate research problems that must be solved before we can extend Ethereum's security to new (and experimental) off-chain systems.

1 Introduction

Cryptocurrencies have risen to a market capitalisation of over $2 trillion in less than 12 years with Bitcoin and Ethereum making up more than 60% of the market. Yet, the scalability of both networks remains the same at around 15 transactions per second and it is not uncommon to witness users paying thousands of dollars in fees [49]. A promising approach for scaling blockchain networks, off-chain protocols, is a solution that can immediately be applied without changing the underlying blockchain protocol. To scale, it focuses on moving transactions from a blockchain network like Ethereum to another off-chain system while ensuring users can transact with assets that reside on the underlying blockchain.

Since 2015, state channel networks have been the focal point for off-chain protocol research and development. The most significant implementation is the lightning network with over 2,300 BTC locked (September 2021) and over the years research has focused on channel constructions [31,29,70,45,5,69,25] and the routing problem [7,35,64,55,88,54,68]. However, it has become apparent that state channel networks excel at routing value and instant synchronisation in an off-chain manner, but it is not an ideal platform for user interaction. Some reasons include requiring every user to open a new state channel via the underlying

blockchain, inbound capacity limits when receiving funds and a user's funds are at risk if they go offline for an extended period of time.

This paper expands upon the work of [47] which explored an emerging off-chain approach called commitchains which allows an operator to run an off-chain system while still retaining underlying blockchain's security. We focus on the cornerstone of a commitchain's security, the validating bridge contract, which is tasked with upholding the safety and liveness for the off-chain system especially if the operators turn out to be malicious. As we will see, this off-chain system can have its own set of operators, smart contract environment, blockchain scalability protocol and most importantly a widely replicated database that can be independently computed by any external party. The motivation for our SoK is to collect the rapidly emerging literature as it is still highly disparate across message boards, chat rooms, and for-profit ventures that fund its rapid development.[3] We focus on presenting the field of validating bridge research in an accessible manner and to bring forth the immediate research problems for the community to tackle. Our contributions are the following:

- We present an overview of the roles involved in validating bridge, the security goals to build a secure validating bridge, and the known solutions.
- We provide a discussion on the various issues that have arisen with the design of validating bridges over the years and present a list of relevant research problems for researchers to tackle,
- We evaluate the state-of-art implementations to present their design choices, financial cost of operation, and future direction.

2 Key concepts

2.1 Globally replicated database

Append-only log and database state All blockchain-based systems aim to globally replicate an append-only log to record the ground truth. Anyone with a copy of the blockchain can independently compute the network's database and validate its correctness. A block producer has the authority to propose a new block of transactions that can be appended to the log and it is up to the set of verifiers (users) to validate the block's correctness before accepting it into the canonical chain (blockchain).

Inside a block, each transaction decodes the sender's account, receiver's account, the value to be sent and a payload. This payload may contain a set of conditions that must be true for the overall transaction to succeed. It can be a script, bytecode to instantiate a smart contract (program) or instructions on how to execute a smart contract. To a certain extent, a transaction represents an atomic state update to the database that is re-executed by all verifiers.

Most databases in blockchain networks use a simple key-value method to store data and the database state can be encoded or represented in many ways.

[3] Several projects have collectively raised over $160m in the pursuit of deploying validating bridges and off-chain systems.

In Ethereum-like systems, the key is an address (public key or smart contract identifier), the value is the quantity of ether and optionally smart contract state (and bytecode).

2.2 Smart contracts and bridges

Trusted bridge contracts A bridge contract is responsible for accepting deposits from users and notifying an off-chain system to mint the same number of coins for the user. To keep it simple, this functionality can be summarised as *deposit*, *withdraw* and *update account balance*. The security of funds held by the bridge contract ultimately depends on the *update account balance* functionality and how the bridge is convinced about the user's new balance before a withdrawal transaction can be processed. We have witnessed three type of trust assumptions emerge for bridge contracts:

- *Single organisation.* One party has the authority to update the user's balance in the bridge contract. An example is a cryptocurrency exchange.
- *Multi-organisation.* A set of independent parties (K of N) can notify the bridge about a user's new balance. An example is the federation bridge used by Liquid [73].
- *Crypto-economic.* A dynamic set of parties, appointed by their weight in assets, can notify the bridge about a user's new balance. An example is the Polygon bridge that requires (2/3)+1 of staked assets to approve all notifications to the bridge contract [26].

In all cases, the bridge contract is only used to automate the withdrawal process for users and it cannot enforce, or even inspect, the off-chain system's integrity. The organisation is responsible for protecting deposits held by the bridge contract and if they are compromised for whatever reason, then the funds are at risk [30]. As we have witnessed over the years, custody of user funds in cryptocurrencies is increasingly becoming a liability for organisations [14,71].

We believe the above style of bridge contracts have emerged as protocols assume the other blockchain system is secure. If we can assume it is secure, then the goal is to minimise trust on how to relay information about the other blockchain's state to the bridge contract. As we will see with the introduction of a validating bridge contract, it changes the above assumption as it assumes the off-chain system may be compromised and its role is to protect the bridged assets (even if all operators are malicious).

3 Validating bridge overview

The validating bridge contract is tasked with protecting the integrity of the widely replicated database and liveness such that a proposed state update will eventually be applied to the database. Figure 1 presents an overview of a validating bridge and it has two components:

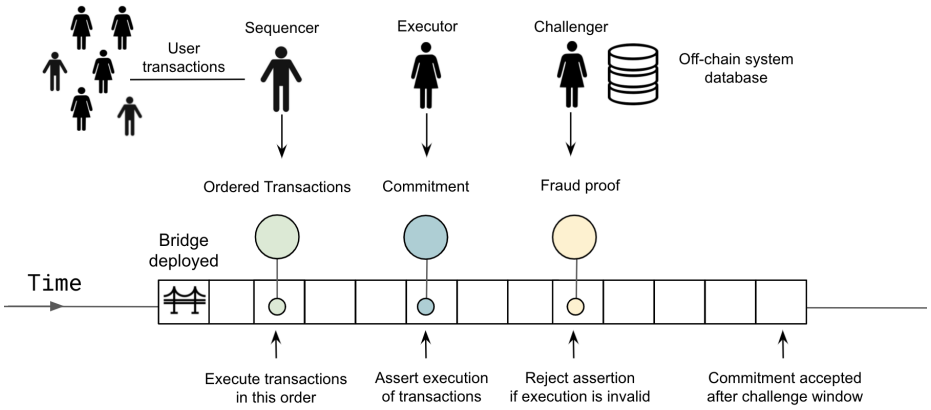

Fig. 1: An overview of a validating bridge and the agents that process transactions for the off-chain system's database (fraud-proof system).

- *Off-chain system* A system with its own appointment protocol, smart contract environment, and a widely replicated database that anyone can independently recompute.
- *Chain of commitments (commitchain)* Periodic commitments about the off-chain system's database state and the bridge must be convinced that all state updates to the database are valid before it can accept a new commitment.

An organisation is still responsible for optimistically running the off-chain system. They must periodically propose a new cryptographic commitment that asserts the new state of the database and be prepared to defend their assertion by providing evidence for why it is valid. The bridge contract can leverage the potential evidence as a basis for a proof of fraud (discrepancy detection) or a validity proof (discrepancy prevention) before accepting the commitment.

The validating bridge must be convinced the *commitchain is not compromised* and every state update applied to the database is valid before it permits any funds to be withdrawn. As such, an off-chain system that is built with a validating bridge is considered a scaling solution as users can transact on an off-chain system with assets that reside on the underlying blockchain while retaining the underlying blockchain's security.

3.1 Agents and protocol assumptions

We consider the agents, the protocol assumptions about the underlying blockchain which hosts the bridge contract and the adversary's power on the off-chain system.

Agents We assume there is an honest user who wants to transact on the off-chain system and they lack the computational resources to verify it in real-time. Figure 1 identifies three agents when assessing the security of a validating bridge:

- *Sequencer* One or more agents who propose an ordered list of transactions to the bridge contract. They are privy to the pending transactions that will soon be confirmed and are in a position to offer information about the pending state of the off-chain system's database (including account balances, smart contract state, etc).
- *Executor* One or more agents who propose the final execution result for a batch of transactions by posting state commitments to the bridge contract.
- *Challenger* One or more agents with the computational resources to validate commitments proposed by the executors. This requires the agent to process every state update to the widely replicated database and this role is only required if the validating bridge requires assistance.

Protocol assumptions We only consider protocol assumptions for the underlying blockchain that is hosting the bridge contract.

- *Blockchain neutrality* Block producers of the underlying blockchain are not colluding with the sequencers, executors or challengers.
- *Eventual delivery* An honest user's transaction will be mined in the underlying blockchain within N blocks if it pays an appropriate network fee.
- *Constrained smart contracts* A smart contract is considered a trusted third party with public state. It is immutable and it cannot be compromised as it shares the same security as the underlying blockchain. It has considerably less resources in terms of computation, bandwidth and storage than the off-chain system.

Threat model We assume an adversary can corrupt all sequencers and executors, but they cannot manipulate the challenger, bridge contract or honest user. The adversary can control the flow of messages on the off-chain system such that they can view, order and censor all messages. Due to the blockchain neutrality assumption, the sequencer cannot prevent the delivery of messages to the bridge contract. Finally, we assume the adversary cannot break standard cryptographic protocols or primitives.

3.2 Protocol goals

We outline goals that an off-chain system implemented with a validating bridge contract should try to achieve.

Operator goals It should be feasible for anyone to deploy a new off-chain instance with a validating bridge and we consider desirable goals for such an operator:

- *No collateral to operate* Sequencers and executors do not necessarily have to lock in any collateral to run the off-chain system or to onboard new users.

– *Operational cost efficiency* It should be cost-effective for the bridge contract to verify the integrity of an off-chain system.
– *Unrestricted experimentation* The off-chain system's protocol can experiment with new blockchain protocols, smart contract environments, and governance models. It should not be restricted due to the underlying blockchain's capability.
– *Proof of reserves* It is publicly verifiable that the off-chain system is fully collateralized (or running a fractional reserve).

User-experience goals The off-chain system should offer a Coinbase-like experience while allowing users to enforce self-custody of their funds:

– *No on-chain registration* A new user can receive funds and interact with smart contracts in the off-chain system without pre-registering on the underlying blockchain.
– *No transfer limit* There is no restriction, except for a user's balance, in terms of the total coins that they can send and receive.
– *Transaction post-condition's enforced* A user's transaction will only be executed if a defined post-condition is satisfied.
– *Validate pending database state* A user is assured about the pending database state for the off-chain system before authorising their transaction.

Security properties We outline what it means for a bridge contract to verify the integrity of an off-chain system. It can be broken into three goals:

– *Data availability problem* The bridge contract can verify the data is publicly available such that anyone can recompute the database independently.
– *State integrity problem* The bridge contract is convinced that all transactions executed in the commitment are valid and the commitment represents a valid new state of the database.
– *Censorship resistance* The bridge contract can enforce the *eventual ordering and execution* of transactions to ensure users can always unwind their positions and withdraw their funds.

3.3 Protocol overview

This overview presents the common features and protocol steps that all validating bridges (and off-chain systems) share. It includes the chain of state commitments, deposits, transacting on the network, how the bridge contract finalises transactions and finally how a user can withdraw their coins.

Periodic state commitment The state commitment is a cryptographic commitment and it may have several sub-commitments:

– *Off-chain system's database state.* All account balances and smart contract state.

– *List of transactions.* An ordered list of transactions that is applied to the off-chain system's database.
– *Intermediary state transitions.* For each transaction, it includes the pre-state of the database before it is executed, a set of instructions for executing the state transition and the post-state of the database after it is executed.

Deposit funds To move funds onto the off-chain system, a user will deposit coins into the bridge contract on the underlying blockchain. The off-chain system will mint an equal number of coins for the user when the deposit transaction is processed by a future state commitment in the bridge contract.

Transacting on the network The sequencer is the portal to the off-chain system and they offer the following services to an honest user:

– *Fast-path transaction inclusion* They collect user-generated transactions that are destined for the off-chain system and propose the final order of execution for the bridge contract.
– *Information about the off-chain state* They are privy to the database's pending state and can offer up to date information about the state of smart contracts.

Decoupling transaction ordering and execution The bridge contract can separate the process of finalising a transaction into two parts:

– *Ordering of transactions.* Both the sequencer and user can submit transactions to the bridge contract which determines the final order of transactions for execution.
– *Execution of transactions.* The executor must submit a state commitment that represents the final execution result of the ordered transactions.

A bridge contract should only finalise the state commitment if it is convinced that all state transitions are valid (i.e., *state transition integrity problem*) and if the state updates are publicly available (i.e., *data availability problem*) to allow a challenger to independently compute the off-chain system's database. Of course, depending on the validating bridge contract, the ordering and execution of transactions can be a combined process or performed sequentially.

Withdraw funds An honest user can sign a withdrawal request transaction and its execution should eventually be processed by the bridge contract. The withdrawal request destroys coins on the off-chain system and inform the bridge contract about the user's entitlement to withdraw their coins back to the underlying blockchain. In most validating bridge contracts, the user has a *potential-claim* to coins held by the bridge contract and at withdrawal time the user may receive any set of the coins (i.e., coins have no serial number or unique features).

4 Sequencer and executor profile

We consider how sequencers and executors are appointed to their role, the transacting ordering protocol that governs which data is sent to the bridge contract, and how sequencers can provide assurance about the pending state of the off-chain system's database.

4.1 Rate-limiting and decision making

Purpose of rate limiting The concept of rate limiting is to restrict who can become a sequencer or an executor. In the case of a sequencer, it is a user-facing role as they collect user-generated transactions and order them for execution. Only agents who may offer an exceptional user experience should be selected to become sequencers and it should not consider the enforcement of censorship-resistance.[4] On the other hand, the executor focuses on eventual execution of transactions in the off-chain system. They are entrusted with a liveness property that impacts transactions waiting to migrate from this off-chain system to the underlying blockchain. The rate-limiting mechanism used to restrict who can become an executor should guarantee that at least one executor can self-appoint themselves to execute a significant portion of the pending transactions.

Appointment protocol The bridge contract needs to verify who is a member of the sequencer and the executor set. Assuming both agents are not pre-installed, then the bridge contract may implement an appointment protocol to allow new members to participate. Several appointment protocols may emerge and two include:

- *Delegated vote* A set of users can vote to appoint a member to become a sequencer (or executor) based on their weight in tokens (or another voting mechanism).
- *Weighted stake* Any user can stake funds in the bridge contract and self-appoint themselves to a role.

Transaction ordering protocol We need to consider how the set of sequencers reach a decision on how to order the pending list of transactions. A non-exhaustive list includes:

- *Single agent* A single sequencer decides the final ordering of transactions.
- *Round robin* The bridge contract provides a series of slots and each agent will decide the order of transactions in turn. A slot can be a fixed period of time and if the slot is missed, then it will move onto the next agent.
- *Majority vote* A threshold of agents must reach agreement about which data to send the bridge contract. It may be implemented as a PBFT-like protocol.

[4] Section 5.3 discusses how the bridge contract can self-enforce the inclusion of transactions for ordering. A sequencer cannot censor a transaction.

4.2 Information about the off-chain system's state

The off-chain system may have a public peer-to-peer network and gossip protocol for propagating new pending transactions. It does not necessarily inform a user about the next batch of transactions to be ordered by the sequencer for execution. As such, the sequencer is privy to the soon-to-be-ordered transactions and the pending state of the widely replicated database. The user can request this information before deciding to transact and the sequencer can offer assurance about how the user's transaction will be executed. This section focuses on the trust assumptions (and how to reduce trust) for when the user requests information from the sequencer about the off-chain system's state.

Trusted sequencer The sequencer is blindly trusted to provide assurances about transactions and off-chain state to the users. Users can only check if the sequencer was honest after the bridge contract has processed the transactions.

Transaction receipt The sequencer provides a signed receipt to the honest user with details about their transaction. It may, although not mandatory, include a signed promise to include the transaction, the transaction's position in the queue, and the transaction's expected execution. The user can verify whether the sequencer has adhered to the receipt by following the messages sent to the bridge contract. If the sequencer does not adhere the receipt's promise, then the user can send this receipt to the bridge contract and punish the sequencer (i.e., slash their stake).

Temporary fact-blocks The sequencer can publish a *fact block*, where a fact is the final execution for a batch of transactions. A chain of fact blocks can be sent to the bridge contract or broadcast across a gossip protocol (where anyone can then publish it on-chain). Fact blocks are considered temporary as the canonical chain and the final ordering of fact blocks is decided at a later stage by another process, but it provides some off-chain assurance about the final execution of a user's transaction [34].

Maximal Extractable Value (MEV) All transactions are ordered and eventually executed based on the total extractable value that profits the sequencer [74]. A simple example is a sandwich attack where the sequencer inserts a front-running and back-running transaction around a user's swap transaction [28]. This allows the sequencer to steal any positive slippage from the trade that would otherwise go to the user. Extractable value is possible as the sequencer has the authority and time to re-order all transactions to find the best combination that maximises their profit. Unlike the other approaches, the sequencer may not promise to eventually order a user's transaction for execution. As well, because the sequencer has not yet decided the final transaction ordering, they may not have information about the pending state of the database.

Fair-ordering protocols This assumes a set of sequencers cooperate to decide the ordering of transactions via a fair-ordering consensus protocol based on a criteria of *fairness*. The only proposed protocol for fair-ordering in a byzantine setting [52] defines fairness as ordering transactions based on the arrival timestamp of a transaction to the set of sequencers. To decide the final ordering of transactions, all sequencers participate in a majority vote protocol based on the transaction's timestamp. The goal is to prevent a threshold of sequencers from impacting the ordering of transactions for their own gain. Another approach is to pursue a commit-and-reveal approach where the user submits a commitment of their transaction and only reveals the transaction after it is ordered [86].

5 Bridge contract security goals and solutions

We outlined the security goals for the bridge contract in Section 3.2 which includes *data availability, state transition integrity and censorship resistance*. Together, the goals ensure an honest user only needs to trust the bridge contract to protect their funds against a malicious off-chain system. In this section, we explore potential solutions to each security goal in order to get as-close-as-possible to retaining the underlying blockchain's security

5.1 Data availability

It is up to the bridge contract to ensure the data is publicly available before a commitment is considered final. This data can be the list of state updates, a list of transactions, or simply the new state of the off-chain system's database. We'll simply call it data for the rest of this section. Outside of allowing a challenger to check the validity of all previous commitments, data availability is necessary to allow an honest user to verify when a transaction is finalised and to obtain evidence that can convince the bridge contract that a withdrawal transaction can be processed. So far, there are four solutions to the data availability problem:

Trusted party or committee A threshold of parties will sign a message to vouch that the data is publicly available. The signatures are verified by the bridge contract before the commitment is accepted. This approach was implemented in StarkEx [91] and Oasis [94].

On-chain data availability challenges Every commitment initiates a challenge process for a fixed period of time. An honest user has time to return online and verify the data is publicly available (and download it). If the data is not publicly available, then the user can challenge the sequencer via the bridge contract to reveal the requested data. The commitment is discarded by the bridge contract if the sequencer does not reveal the data by the challenge period's expiry time. This approach was proposed for NOCUST [56] and variants of Plasma [16].

Wait for data availability in a fixed-claim system The following solution only works in a *fixed-claim* system such that there is a one-to-one matching of coins held by the bridge contract. Every coin is associated with a single entry in the off-chain system's database (i.e., each coin has a unique serial number). Instead of checking the integrity of the entire database, the honest user only needs to check the integrity of state updates for the single entry in the database. As a result, the honest user only needs to obtain the history of updates for this database entry and check that it is indeed valid according to the historical on-chain commitments. From that point onwards, assuming the user was transferred the coin and they can verify its entire history, then the user does not need to consider the data for any future commitment.

To withdraw a coin, an honest user will provide evidence to the bridge contract that they are the assigned owner of the coin (and database entry). This initiates a withdrawal challenge process where the ownership claim can be tested by any other user on the network. To falsify a withdrawal, another user can prove the coin was already spent (i.e., the honest user has transferred it) or participate in an interactive-process to pinpoint an invalid transfer in the coin's history. This approach was proposed by Plasma Cash [58] and it solves the problem as an honest user is only required to fetch the coin's history up to the point in which they received it. They can ignore all future commitments in regards to checking for data availability.

Post to the layer-1 blockchain All data alongside the commitment is posted to the bridge contract (*rollup*). This approach was proposed by Barry Whitehat [103] and it has been adopted by most projects.

5.2 State transition integrity

The next step is to consider how the bridge contract can be convinced about the validity of an asserted commitment. Specifically, a commitment will assert the valid transition of the database based on the previously accepted commitment and the list of transactions to be executed. There are two approaches:

- *A fraud proof system.* The bridge contract initiates a challenge process for a commitment. This provides an opportunity for a challenger to submit evidence if there is an invalid state transition and for the bridge contract to reject the commitment. If there is no evidence of fraud by the time the challenge expires, then the bridge contract accepts the commitment as final.
- *A validity proof system.* The executor must present evidence alongside the commitment to prove that all state transitions are valid. There is no challenge period and the bridge contract can accept the commitment immediately.

Pinpointing and re-executing disputed state transition A fraud proof system, often called an Optimistic approach, is effectively a form of lazy evaluation. The challenger's task is to pinpoint the disputed state transition and for the bridge contract to check its validity by simply re-executing the state transition. This

allows the bridge contract to independently determine whether the asserted commitment should be rejected. There are currently two approaches for pinpointing a disputed state transition:

- *One-round fraud proof* The challenger has the list of intermediary state transitions for the asserted commitment and this is used to pinpoint the disputed state transition. The challenger submits the disputed intermediary state transitions alongside a list of inclusion proofs about the state transition's pre-state, the state transition to execute, and the executor's asserted post-state after it is executed. Given this information, the bridge contract can execute the state transition and compare the executed result with the asserted post-state. If there is a mismatch, then the commitment is considered invalid and it is rejected.
- *Multi-round fraud proof* The list of intermediary state transitions for an asserted commitment is not sent to the bridge contract and it is not trivial to pinpoint an individual state transition in a single-round. Instead, the challenger and executor must agree upon the total number of instructions executed within the commitment and then perform a binary search until they pinpoint the disputed instruction. Each step in the search involves one party claiming the new state after the instruction is executed and the other party must agree/disagree with it. If they agree, then all previous instructions must be valid, otherwise they continue to search until the single disputed instruction is identified.

Validity proof A validity proof is indisputable evidence that all state transitions are correct. The naive approach involves the verifier re-executing all transactions before accepting a new block (or state commitment) which is common for public blockchain networks. This is not applicable for a validating bridge contract as it is assumed to have significantly less computational resources than the off-chan system. Another approach, implemented using zero knowledge proofs, allows the prover to produce a proof of computational integrity (while hiding its inputs) that every state transition for a given commitment is well-formed. This proof can be verified with minimal computational or bandwidth resources (i.e., a few milliseconds). It can be verified by the bridge contract, or anyone, to confirm the commitment (and the off-chain computation) is indeed valid. On the other hand, the downside of a zero knowledge proof is the computational burden placed upon the prover, especially for generic computation, and it remains an active engineering task to alleviate [50,33].

5.3 Censorship resistance

If the off-chain system is offline, halts or the operators are malicious, then a censorship-resistance mechanism is essential to allow the user to still transact and eventually withdraw their funds from the bridge contract.

Forced transaction inclusion A cornerstone of censorship-resistance is the ability to bypass the sequencer with *forced transaction inclusion* such that the user can submit a transaction to the bridge contract and it will eventually be ordered for execution. We call it the *slow path* for transaction ordering and as such the sequencer's role has little impact on censorship-resistance. Instead, we need to consider the set of executors as they are entrusted with liveness of execution and the complexity of transactions on the off-chain system.

Value-transfer's escape hatch This assumes the off-chain system only supports value-transfer and smart contracts do not have a balance. In this case, forced transaction inclusion is sufficient to enforce censorship-resistance as the bridge contract can order the user's transaction for execution and freeze the entire off-chain system if it is not executed in a timely manner. If the system is frozen, then it can activate an escape hatch and allow all users to simply withdraw their balance via an on-chain exodus.

Smart contract functionality and enforced liveness This assumes a smart contract in the off-chain system can have a positive balance (i.e., hold funds on behalf of users). In this case, the user may need to perform several transactions to unwind their position in a smart contract before they can issue a withdrawal request. If we tried to use the escape hatch mechanism, then the user's funds may still be locked in the smart contract and the only solution is to withdraw the entire smart contract (and its state) to the underlying blockchain. This is not a feasible option as the smart contract may be larger than what can be instantiated on the underlying blockchain.

Another approach is to rely on the bridge contract to ensure that all transactions will eventually execute. Some mechanisms that can be implemented include:

- *Permissionless set of executors.* An honest party should have the ability to self-appoint themselves to become an executor.
- *Opportunity to post commitment.* The bridge contract's fork-choice rule for deciding which commitment to accept should not discriminate against the honest party.
- *Minimum computation requirement.* The bridge contract should require every state commitment to process a minimum number (or percentage) of transactions.
- *Staked executors.* Rate-limit who can become an executor and to cover the challenger's cost for undoing malicious behaviour by the executor.

Overall, the censorship-resistance property for an off-chain system with smart contract functionality is not as well studied compared to the data availability or state integrity problem. As discussed in Appendix A, both Optimism and Arbitrum use a combination of the above to achieve censorship-resistance, but the analysis of how well it achieves this goal is design-specific. As well, there are potential denial of service attacks which can only be alleviated by slashing as a

deterrent. An outstanding research task is to define the individual security goals that can be combined to achieve censorship resistance and to model the bridge contract's capabilities for upholding it.

6 Discussion

6.1 Evolution of the bridge contract

Data availability challenge and the rise of rollups Plasma [82] was the initial approach for building a validating bridge contract. Its goal was to move bandwidth, computation and storage from Ethereum to a Plasma network. Initial designs proposed the challenge process as a solution to the data availability problem and a fraud proof system for the state transition integrity problem. While several projects attempted to implement Plasma including OMG [53], Plasma Group [46] and Loom [22], it appears that a practical implementation failed to materialize. This led to a growing consensus that a challenge process for data availability hindered the implementation of Plasma and thus it was not a practical solution [84]. The reasoning is twofold:

- *Fisherman's dilemma* It is indistinguishable whether the sequencer withheld data or if the user issued an unnecessary challenge. The sequencer can keep data private and force the user to issue a challenge for data availability. Since blame cannot be ascribed, the user must cover the cost for the sequencer's malicious behaviour.
- *Process challenge limitations* The bridge contract lacks the resources to process all data availability challenges in a timely manner and as such users may fail to force the sequencer to reveal their data. This can result in a mass-exit as all users attempt to exit the off-chain system.

After the release of Barry Whitehat's zkrollup [103,17], most implementations have pursued publishing the transaction data onto Ethereum as the preferred solution to the data availability problem. While it requires stronger assumptions than outlined in Section 3.1 (i.e., the bridge contract now requires similar same bandwidth as the off-chain system), it is considered the most workable solution in the immediate term as bandwidth on Ethereum is considered the most abundant resource compared to computation and storage [2].

Research question: Can we remove the data availability risk without posting all data to underlying blockchain? Can the design of Plasma Cash assist with this endeavour?

Mass exit A mass-exit (also known as mass-withdrawals) occurs when there is a possibility that users will lose access to coins held by the bridge contract [82] and it arises because a security property of the validating bridge was broken. There are two type of risks that can lead to a mass-exit:

- *Potential-claim risks* This assumes an off-chain system where users have a potential claim to coins in the bridge contract. If the state transition integrity

property is broken, then the adversary may steal some coins from the bridge contract and the remaining assets can no longer cover its liabilities (i.e., a fractional reserve). Not all withdrawals can be honoured by the bridge contract and as a result users will race to withdraw their coins before others.
– *Data availability risks* It may become foreseeable that users will loose access to the off-chain system's database due to the data availability property being broken. This can prevent users withdrawing their funds via the bridge contract as they cannot prove ownership of their coins. Users will race to withdraw their funds before the situation arises when access to the data is lost.

Off-chain systems that post all data to the validating bridge contract (rollups) remove the data availability risk. A fraud proof system can only alleviate, but not remove the potential-claim risk. This is because the bridge contract can accept an invalid state transition if a challenger fails to submit a proof of fraud during the challenge process. On the other hand, Plasma Cash (i.e., a fixed claim system as discussed in Section 5.1) is the only fraud-proof system that can alleviate the potential-claim risk as a failed challenge process only impacts the user associated with the stolen coins. There is no potential-claim risk in a validity proof system.

Research question: A measurement study on the impact of a mass-exit on the underlying blockchain and whether the closely related concept of an on-chain exodus remains a viable option as the off-chain system's database grows in size.

Data availability via committees While Ethereum has reduced the cost of posting *calldata* to the blockchain [2], it remains the dominant cost to operate a rollup. This has led to some projects introducing a stronger trust assumption to keep data off-chain via a data availability committee including StarkEx (Validum) [91] and ZkSync (zkPorter) [60]. If the committee does not reveal the data, then it impacts the safety guarantees for a fraud-proof system as an honest party lacks the data to challenge an asserted commitment. On the other hand, if the data is accidentally lost, then it can impact the liveness of a validity-proof system as the executors cannot produce new proofs.[5] Be that as it may, the ongoing work for Ethereum is to better utilize the network's bandwidth as rollups offer a potential solution for sharded execution. This has led to a roll-up centric roadmap [19] with proposals to change the underlying blockchain in favour of it becoming a data availability layer [1,32,15] with minimal computation.

Research question: Protocols for securely sharding data availability [21,1].

One-round vs multi-round fraud proof The core difference in the fraud proof implementations appears to be the the *granularity of state transitions* which impacts whether it is cost-effective to post the list of intermediary state transition commitments to the underlying blockchain.

In Optimism, the state transition is an entire transaction and as discussed in Section B.2 it costs approximately 1,112 gas per transaction to post the intermediary state commitment. All intermediary state commitments are sent to the

[5] Historically, ripple lost the first 32k ledgers due to bug in the Ripple servers.[89].

bridge contract and this allows a challenger to single-handedly pinpoint which state transition should be executed by the bridge contract. No cooperation with the executor is necessary and a valid commitment cannot be evaluated as invalid by a fraud proof. A commitment including its dependents can be discarded if it is proven to be invalid (alongside slashing the executor who proposed it).

In Arbitrum, every intermediary commitment covers a single instruction (opcode). It is not cost-effective (and arguably infeasible) to post all intermediary state transition commitments to the underlying blockchain. This necessitates the multi-round bi-section protocol as the challenger must cooperate with the executor to pinpoint the disputed state transition. After all, the challenger cannot independently compute every intermediary state commitment if the asserted state commitment is indeed invalid. It is possible for the bridge contract to be convinced that a valid asserted state commitment is invalid and there are two examples to illustrate it. First, an executor may fail to defend a commitment because they do not finish the bi-sectional protocol. Second, if the adversary is both the challenger and executor, then they can post a fake intermediary state commitment during the bi-sectional protocol and the bridge contract will evaluate it as invalid. This is because the bi-sectional protocol can identify that one party is malicious, but not that the other party is also malicious. As such, a commitment is not rejected in Arbitrum and instead the challenger is turned into a zombie. A commitment is only finalised if all non-zombie executors have staked on it (and the challenge process has expired).

Research question: Can we design a single-round fraud proof system that evaluates state transitions for instructions (opcodes)?

Transactions or state updates for scalability Bandwidth is a bottleneck for off-chain systems that post data to the validating bridge contract as the underlying blockchain's bandwidth is shared amongst all projects who will compete for the same scarce resource. The type of data used by the verifier to compute the widely replicated database (Section B.1) will impact the bandwidth requirements and it appears that validity proof systems are superior in terms of reducing their on-chain footprint (alongside computation). This is because the operators can aggregate multiple user-generated transactions into a single update and the verifier does not need to re-execute every transaction before applying a state update to the database. Given there are real-world implementations for both systems and historical transactions on Ethereum, it is now feasible to produce an objective measurement on the potential savings.

Research question: A measurement study to investigate the potential savings in terms of bandwidth and computation of a validity system compared to re-executing every transaction to compute the widely replicated database (fraud proof system).

6.2 Sequencer and executor incentives

Network fee for transactions on the off-chain system The user is expected to pick a network fee that will entice a sequencer to include their transaction in the

fast-path. This network fee combines the financial cost for consuming resources on the off-chain system and the fee for consuming resources on the underlying blockchain. Plus it must take into account the risk of sudden congestion and volatility of the underlying blockchain fee market as it may no longer be economically viable to finalise it.

Fundamentally, this problem assumes the fee market relies upon a first-price auction and the user's proposed network fee only represents a single bid in the auction. Based on our code inspection in Appendix A, Arbitrum is the only system to replace the first-price auction with a fee model similar to EIP-1559 [100]. The user sets a max fee they are willing to pay, but all transactions pay a single clearing price (base-fee). Separating the max fee and clearing price can provide sufficient spread to accommodate for sudden rise in the underlying blockchain fee.

Research question: Fee models (or protocols) to accomodate underlying blockchain's network fee volatility due to sudden congestion.

Sharing collected fees amongst the agents Assuming the sequencer and executor are separate roles, then the protocol should fairly split the collected fees amongst them. This may lead to the validating bridge collecting network fees into a pool of funds and only pay out based on certain criteria such as proof of participation.

One issue that arises is the on-chain bounty problem. If multiple executors submit a commitment to the bridge contract at the same time, then the bridge contract needs to decide whether to reward the first executor or to split the reward evenly. The former is not necessarily fair to other executors due to the financial cost to participate and the latter can dilute the reward such that it is no longer profitable to participate. This is a race-condition issue and should be considered when deciding how to fairly share the reward.

A related problem is the ability for a sequencer to earn rewards via MEV and this is not shared amongst the other parties as it cannot be detected by the validating bridge. This undermines the network's fee model as the sequencer can earn significantly greater rewards compared to executors. Some research has focused on splitting the sequencer's role into a set of block proposers who compete in an auction to pay a block builder to include their ordered list of transactions [18,40]. It is expected that this approach may encourage proposers to split the MEV profit with the executors.

Research question: A proposal for how to fairly distribute the network fee amongst the agents in the off-chain system. Some directions of research includes creating a market amongst the participants to pay upfront to earn rewards, to introduce a network fee pool to split rewards, and to consider how to resolve the on-chain bounty issue.

Verifier's dilemma The security of a fraud proof system relies on challengers validating every commitment and posting a proof of fraud if applicable. A dilemma arises as challengers may stop checking for fraud and this may happen for several reasons. For example, challengers may assume no one will commit fraud,

they may get lazy as they assume other challengers are checking commitments or it is simply not cost-effective as the on-going cost to check is greater than the probability they'll collect a bounty [65,37]. This raises the question whether we can verify that challengers are continuously checking the integrity of new commitments and what is the incentive for a challenger to continue checking if they cannot collect a bounty because there is no instance of fraud.

Related work recommends incorporating a canary protocol [96,51,36] that will sporadically issue invalid state transitions to test whether challengers respond in a timely manner. Another approach, as implemented in Arbitrum, is to combine the roles of challenger and executor. All executors must stake on an existing asserted commitment before they are allowed to propose a new one. This act of staking will put their funds at risk and they must be prepared to defend any challenges from other executors. As such, it is expected the executor will check if the commitment is valid before moving their fund onto it.

We should consider if it is necessary to solve the verifier's dilemma in an incentive-compatible manner and whether it is better to assume at least one challenger will behave altruistically absent of an explicit reward [57]. This type of behaviour can be seen in practice as users verify networks like Bitcoin and Ethereum because they have a vested interest in its success. For example, they may validate to protect their funds or it is simply a business model that is offered as a service like Infura.

Research question: A study on financial incentives for the verifier's dilemma and whether it is necessary to design an incentive-compatible protocol for challengers.

Charging for on-chain transaction inclusion We consider how a fee can be charged on the off-chain system for user-generated transactions which are ordered for execution via forced transaction inclusion (slow path). ZkSync and StarkEx do not charge a fee as the user can only request a full withdrawal of their funds. However, if an off-chain system supports smart contracts and arbitrary execution, then there is a potential denial of service attack. Transactions ordered via forced inclusion may pay an unfair fee (significantly less) compared to transactions included by the sequencer as it bypasses the off-chain system's fee auction mechanism. If the slow path allows a sender to pay significantly less, then they can consume all available resources and delay the execution of other transactions.

One solution deployed by Arbitrum is to change the fee model such that there is a single clearing price paid by all transactions and any transactions sent via the slow-path will pay the same fee. There are two potential issues with this approach. First, it is important the sequencer cannot tamper with fee charged by the validating bridge contract as it may allow them to censor transactions by increasing the clearing fee to a value that is unaffordable. Second, a user (or a contract) may not have a balance on the off-chain system, but to support composability of contracts they may want to issue a transaction destined for the off-chain system via the bridge contract. Another solution implemented by Optimism, albeit hacky in nature, is for the transaction sender to perform a

useless, but expensive computation on the underlying blockchain as a way to charge a fee.

Research question: A protocol to charge an appropriate network fee for on-chain transaction inclusion that alleviates denial of service attacks, but upholds censorship-resistance.

Is MEV a business model or an attack? Several retail-focused financial products such as Robinhood and eToro do not offer an explicit fee for their users. They make money by extracting value from their user's transactions. For example, this can be taking the spread of a trade or payment for order flow to market makers. This is a form of MEV as they impact the execution of a user's transaction to maximise their profit.

The same extractable fee model can be applied to the off-chain system. It is an optimisation problem for the sequencer (or a set of searchers [39]) to evaluate a batch of transactions. In March 2021, Ethermine were making approximately $1m a day from MEV bundles sent by searchers on the network and since then over 85% of block producers on Ethereum are now extracting value with the assistance of Flashbots [24]. To reduce the impact of MEV, a user can leverage smart contracts to limit the total extractable value for their transaction. For example, the user can set a post-condition that must be satisfied in order for the transaction to succeed (and execute). This post-condition can limit the price movement on a trade or simply require the user to receive a minimum number of coins. The sequencer must abide by the user-defined limits to avoid failing the transaction and reaping no rewards.

The counter-argument is that MEV should be considered harmful and exploitative [38]. Users may pay significantly more in fees than is necessary to include their transaction in the off-chain system. If a transaction has an expiry time, then the sequencer is incentivised to withhold it up to the expiry time as opposed to processing it immediately if there is a chance the sequencer will extract more value at a later time. In the end, there is a centralisation risk if a small set of sequencers (or searchers) earn excess profits and out-compete other off-chain systems.

Research question: An evaluation of the risks MEV poses to the off-chain system's stability and a comparison of the solutions to reduce it including the option to *smooth* the MEV reward amongst the block producers.

Upgradability and community control All project's have implemented an upgrade path for their bridge contracts. There are several reasons to include admin-like functionality such as the ability to implement new features (including gas-efficiency savings), temporarily incorporate training wheels to prevent a zero-day exploit [75], and to protect against future hard-forks on the underlying blockchain that may impact the bridge contract [11]. Upgradability has led to governance issues on who has the authority to propose (and confirm) an upgrade. While the trust assumptions outlined in Section 2.2 remain relevant for deciding which organisation can propose an upgrade, we suspect some projects

will eventually issue a token and transfer governance control to the token holders. MakerDAO is a recent example on transferring power to the community and away from the core team who deployed the system [67]. To reduce trust in the upgrade process, we recommend a time-delay for locking in a new proposed upgrade and only activating it after a fixed time period. This can provide time for the community to audit the upgrade and withdraw their funds if any of the security goals for the off-chain system are compromised.

Research question: A measurement study on the success of token-holders to control the governance process of a smart contract-based project.

6.3 Cross-chain interoperability and asynchrony

One success behind smart contract platforms like Ethereum is the centralisation of collateral and the composability of smart contracts. It is not uncommon for a transaction to include a list of internal transactions that interact with different smart contract applications (*the DeFi lego blocks*) and for the entire transaction to revert if a single interaction fails. For example, a user may take a flash loan, use the loan to maximise an arbitrage opportunity for a swap and the transaction fails when repaying the loan as the arbitrage moment was lost. Moving transaction processing onto multiple off-chain systems threatens this atomic composability as it fragments smart contract applications and collateral. It is the same train-and-hotel problem faced by sharding [20] as booking a hotel should only be successful if the train is booked (and vice versa). Our aim is to restore composability for smart contracts that are instantiated on their own off-chain system.[6] or deployed on one (or more) off-chain systems[7] Protocols that focus on cross-chain communication [104] can help re-introduce atomic composability which can be broken into message delivery and value transfer.

Asynchronous message delivery A message can be the state of a smart contract at a certain time, an executable transaction, or simply transfer value. It is considered asynchronous as message delivery takes place over a period of time and the delivery should be persistent (i.e., it is only delivered once to the receiving contract). This is important when a smart contract will only execute a function if it can verify whether an event, state, or transaction, has occurred on another off-chain system. Some approaches include building a light-client smart contract that can verify proofs which assert the inclusion of a transaction, event, or state on the off-chain system like BTC Relay or the Rainbow bridge [83]. On the other hand, trust can be distributed amongst a set of oracles who assert that an event has occurred (and a fraud-proof system can be incorporated to penalise dishonesty [23]). Finally, with the use of conditional transfers (hashed time-locked contracts) it may be possible to transfer the message off-chain amongst the parties who are also facilitating the transfer of value.

[6] Reddit plans to deploy their own instance of Arbitrum [72].
[7] Sushiswap is deployed on at least five blockchain networks [48].

Research question: Repayment protocols synchronise value transfer across two or more off-chain systems. Is it possible to synchronise time-dependent function calls and events?

Liquidity providers The destination smart contract may not be concerned about the authenticity of a message and it will execute a function as long as the sender has sufficient funds. This has led to liquidity providers who will provide the user with coins on the destination off-chain system. The most significant issue is capacity as the liquidity provider cannot lend more coins than they own. There are two approaches for alleviating this issue. First, the sender can split the payment across multiple liquidity providers in an atomic manner [79,7]. Second, users may invest in the receiver's pool of funds and earn a yield from the fees collected. To the best of our knowledge, there is no trust-minimized protocol for the latter approach. We'll cover two popular approaches which includes repayment protocols and burn/mint liquidity.

Research question: A protocol to allow users to invest in a pool of funds for cross-chain transfers, but it minimises trust in the liquidity provider.

Repayment protocols A repayment protocol is when a liquidity provider sends the user funds on the destined off-chain system if, and only if, they are eventually repaid the funds. There are two types of repayment protocols:

- *Immediate repayment* The liquidity provider optimistically has immediate access to the repaid funds after paying the user.
- *Eventual repayment* The liquidity provider must wait a period of time until they have access to the repaid funds.

Atomic swaps represent an immediate repayment protocol. Both parties can exchange coins on different off-chain systems. Neither party has to trust each other and its atomicity can be enforced with the use of conditional transfers to ensure both transfers will succeed. Swapping is not restricted to a single asset type and the liquidity provider can absorb the exchange rate risk. This can introduce a free American call option to lock-in an exchange rate [87]. On the flip side, it is possible to extend the swap beyond two parties with state channel networks. The sender can find a path that connects them with the destination and this may involve multiple hops across several blockchain networks (including off-chain systems). In fact, it is feasible for Bitcoin's lightning network to facilitate cross-chain swaps between off-chain systems that are anchored onto Ethereum.

An eventual repayment protocol includes the credit-based MakerDAO [66] proposal to alleviate the long withdrawal times for fraud proof-based validating bridges. At a high level, an oracle opens a new debt vault on the underlying blockchain that mints new coins (DAI). This vault will require the collateral that props up the DAI to be locked in by an expiry time. The user must submit a withdrawal transaction on the off-chain system that repays the vault and the oracle will wait for this transaction to be ordered for execution by the bridge

contract before sending new DAI to the user. While the oracle only has to trust the off-chain system will eventually execute the withdrawal transaction, it is not a fair-exchange protocol as the user cannot force the oracle to issue the coins.

Another example of an eventual repayment protocol includes the debit-based approach proposed by the Hop Exchange [102]. It relies on set of bonders to facilitate cross-chain transfers. There is a special smart contract on the destination chain that tracks payments by a unique identifier. A bonder will send the user their requested funds on the destination chain (with the required unique identifier) after the user has sent them funds on the source system. The user's transfer should include the same unique identifier and it will migrate via the underlying blockchain to the destination chain. When the funds arrive, the smart contract will verify whether the bonder has indeed paid funds to the user using the unique identifier. If so, it will pay the bonder, otherwise it will refund the user. As well, HOP protocol has set up automated market-maker liuqidity pools [101] on multiple blockchain networks to establish an exchange rate between an asset with real-value (ETH) and a synthetic token (hETH). The goal is to allow the synthetic token to represent liquidity and to create a financial incentive for arbitragers to keep funds well-balanced across networks.

Research question: Repayment protocols that can synchronise a path of events across two or more off-chain systems. This may offer a synchronous-like experience.

Burn and mint liquidity The liquidity provider can mint tokens on one off-chain system and burn tokens on another off-chain system. This approach is only viable if the token can be pegged to another asset that is valuable. Tether, a stablecoin provider, already offers the stablecoin (USDT) on several networks and mints/burns coins to blockchains where it is required [97]. We foresee that an automated market maker's liquidity pool can be used to ascribe real value to a synthetic token. Assuming the synthetic token can be ascribed value, then an external authority can burn (and mint) tokens across networks as a way to transfer liquidity. The issue to mitigate is that the authority can mint an unlimited number of tokens and potentially drain the liquidity pool. To the best of our knowledge, this remains an outstanding research problem.

Research question: A trust-minimised burn and mint liquidity protocol that protects users who provide collateral for the liquidity pools.

6.4 Comparing to related scalability solutions

State channels A channel allows a fixed set of parties to lock up funds, transact by unanimously authorising the new state of the channel, and eventually redeem the final agreed state on the underlying blockchain. If we consider the two party case for a payment channel, then the respective state is the balance of both parties and there is a fixed number of coins locked in the channel. To represent a payment, both parties adjust the balance of each party and then mutually sign it (alongside a mechanism to uniquely identify it as the latest agreed state).

If one party is offline (or uncooperative), then the counter-party can initiate a challenge process to submit a pre-authorised state and eventually withdraw their funds. The immediate issues with payment channels include:

- *In-bound and out-bound capacity issues.* Each party can only receive up to or send the total coins locked in the channel.
- *On-chain transaction to join.* Every party requires an on-chain transaction to open a channel and subsequent channels to increase their inbound/outbound capacity limits.
- *Hot wallet risk.* The signing key must be online for each party to sign new payments.
- *Online assumption.* Each party must return online periodically to detect if a challenge process was initiated and ensure the latest authorised state is submitted as evidence.
- *Lack of open-access smart contracts.* Channels are constrained to a set of parties and there is no protocol to allow external parties (outside the channel) to participate in a smart contract.

There are additional issues that arise if we try to replicate an off-chain system (similar to a validating bridge). The operator must allocate their coins to new customer channels and this represents an opportunity cost if the customer then decides not to transact. While payment channels reduce trust in the operator such that the customer can redeem their current balance at any time, the collateral requirements and quantity of on-chain transactions to maintain the system result in practical limitations. In our view, payment channels excel as a solution for interoperability (a repayment protocol) as opposed to building operator-run off-chain systems.

Channel factories The concept of a channel factory aims overcome the issue of requiring subsequent transactions to increase channel capacity [12,80]. It requires N parties to lock funds into a base-layer channel. They can unanimously agree to spawn new channels on top of the base-layer and periodically refresh the channels with different capacities and channel partners. The issue with the factory approach is the need for all N parties to unanimously agree. If one party is offline, then the factory must be closed and re-opened. It does not solve hot wallet risk, the online assumption, or requiring an on-chain transaction to join the system.

Sidechains The original sidechain paper proposed how to build a bridge contract that locks bitcoins in Bitcoin and unlocks the bitcoins on another off-chain system (as well as supporting its return). The construction relies on simplified payment verification proofs (SPV) to facilitate a two-way peg as each system must prove to the other system that the bitcoins are locked and ready to be withdrawn on the other side. Unlike a validating bridge, the sidechain approach assumes the off-chain system safety (integrity of its ledger) and liveness (continuous progress of confirmed transactions) is outside the scope of the bridge contract. As such, the sidechain's bridge contract can verify the state of the off-chain system, but it cannot verify its integrity or force its progression.

	Arbitrum	Oasis	Optimism	StarkEx	ZkSync
			Operator Goals		
Collateral to operate	Staked Executors and Challengers	Staked Executors	Staked Executor	None (permissioned)	None (permissioned)
Operational cost	Fixed priced auction	First priced auction	First priced auction	First priced auction	First priced auction
Experimentation	Solidity opcodes	Modules	Solidity opcodes	Zk-friendly	Zk-friendly
Proof of reserves	Yes	Yes	Yes	Yes	Yes
			User experience goals		
Functionality	EVM	EVM & Rust	EVM	Misc	Misc
On-chain registration	No	No	No	Withdrawal	No
Transfer limits	None	None	None	None	None
Enforce transaction	Failed tx	Valid tx	Failed tx	Valid tx only	Valid tx only
Pending database state	Trusted	Trusted	Trusted	Trusted	Trusted
			Security Goals		
Data Availability	On-chain	Committee	On-chain	Committee	On-chain
State Transition	Multi fraud proof	Single fraud proof	Single fraud proof	Validity proof	Validity proof
Sequencer censorship	Forced inclusion	Rotating leader	Not implemented	Forced inclusion	Forced inclusion
Execution liveness	Staked executor	Staked leaders	Staked executor	On-chain exodus	On-chain exodus

Table 1: An overview of the leading validating bridge projects (more information in Appendix A)

7 Achieving the protocol goals

Table 1 highlights how each project has satisfied the protocol goals defined in 3.2. It is based on our code inspection in Appendix A and in the following we explore the results in more detail.

7.1 Operator goals

No collateral to operate The collateral requirements are not required to facilitate transfers, but to rate-limit who can become a sequencer or executor. All projects do not require the sequencer to lock funds as it is not necessary in the name of censorship-resistance.

An executor (and challenger) may be required to open a collateralized position in both a fraud-proof system and a validity-proof system. In the former, the stake should act a deterrent to prevent an executor posting an invalid commitment (Arbitrum and Optimism). As well, it should deter challengers from initiating a challenge on a valid commitment with the goal to delay its confirmation (Arbitrum).

In the both a fraud-proof and validity system, the stake should allow one honest party to self-appoint themselves as an executor and to post a new commitment that executes the majority of transactions. This is the case for all projects except for StarkEx and ZkSync as they are permissioned systems, but it does not hamper censorship-resistance as they do not support smart contracts.

Operational cost efficiency The fees charged on the off-chain system should cover the financial cost of interacting with the bridge contract should be amortized amongst all users. As mentioned throughout the discussion, it remains an open-problem to fairly reward each set of agents in a decentralized setting and all projects except for Arbitrum rely on a first-priced auction.

Unrestricted experimentation As explored in Appendix C, all projects have implemented a virtual machine that extends beyond the capability of the EVM. Arbitrum and Optimism can extend their functionality by implementing new operation codes as solidity smart contracts. In ZkSync and Starkware, the EVM is only concerned with verifying a validity proof and it is not aware of the virtual machine details.

Proof of reserves The difficulty with previous attempts at proof of reserves [27] was the complexity to implement it for private (and custodial) databases. This is a default property for all validating bridges as the off-chain database is public. Any user can verify the funds held by the bridge contract, the recorded account balances in the off-chain system's database and that all historical state transitions are valid.

7.2 User experience goals

Functionality Both Arbitrum and Optimism offer an EVM-compatible smart contract environment and the subtle aspects of their implementations are discussed in Appendix C. Oasis is a dedicated platform for supporting the EVM and Rust smart contracts. Both ZkSync and StarkEx offer *Misc* functionality which we summarise to include transfers (and swaps). They have plans to support turing-complete smart contract environments as discussed Appendix C.

No on-chain registration The goal of is achieved for Arbitrum, Optimism and ZkSync. While it should be removed in the foreseeable future, the current version of StarkEx does require the user to register a STARK-friendly key with the bridge contract before interacting with the off-chain system.

No transfer limits Unlike a state channel network, the operator does not need to lock collateral with their users to facilitate payments. There are no inbound or outbound capacity issues for any project and a user can send their entire balance in a single transaction.

Enforce a transaction's post-condition In Arbitrum and Optimism, a transaction will fail if the post-conditions are not adhered too (similar to Ethereum). In ZkSync and StarkEx, a sequencer can only include a transaction for ordering if its execution will succeed. It remains an outstanding problem for a zk-friendly virtual machine to process failed transactions.

Validate the pending database state All projects only allow the user to validate the pending database state when the final ordering of transactions is sent to the bridge contract. The sequencer is fully trusted to provide an accurate view of the pending database state. As mentioned in Section 4.2, there are potential solutions that can be adopted to reduce trust in the sequencer.

7.3 Security goals

In Table 1, we have split up the censorship-resistance property into *sequencer censorship* and *execution liveness*. It is based on a code inspection which is further explored in Appendix A.

Solving the data availability problem All projects except for Oasis and StarkEx post the data to it by the underlying blockchain[8]. In Oasis and StarkEx, it relies on a data availability committee. As discussed earlier, there is work towards a hybrid approach of allowing users to decide on-the-fly whether to post data on-chain or to keep it with the data availability committee [81].

[8] StarkEx can support posting data on-chain and this is available in the dYdX deployment. However, the Immutable deployment we studied only relies on a committee.

Solving the state transition integrity problem All projects implement a fraud proof or validity proof system. Optimism plans to change their one-round fraud proof to a multi-round fraud proof due to practicality issues impacting the maximum size of a transaction and to avoid publishing the intermediary state transition commitments as this can reduce the on-going operational cost [76].

Preventing sequencer censorship All projects except for Oasis have implemented forced transaction inclusion via the bridge contract. We highlight that Optimism's implementation remains incomplete at the time of assessment (September 2021). In the case of Oasis, it relies on a round-robin leadership protocol that eventually picks a single party to order the transactions for execution. Both ZkSync and StarkEx cannot censor withdrawals, but the sequencer can prevent the execution of a transfer (or trade) on the off-chain system.

Enforcing execution liveness All projects except for StarkEx and ZkSync rely on the bridge contract allowing an honest executor to self-appoint themselves and enforce the execution of a pending transaction. Only Arbitrum enforces a minimum number of transactions to be processed. StarkEx and ZkSync do not support smart contracts to hold user funds which allows them to implement the on-chain exodus approach outlined in Section 5.3.

8 Conclusion

Our SoK has turned the spotlight on the validating bridge which is the cornerstone for extending the underlying blockchain's security to off-chain systems. An organisation is still trusted to offer the fast-path for updating the widely replicated database, but a permissionless set of executors can enforce the eventual execution of all transactions. In fact, we expect designs to emerge that will constrain the sequencers such that they are *not even trusted with transaction ordering* and they will simply accept inbound transactions, follow a deterministic ordering protocol, and then publish the final order for execution.

The motivation for organisations to adopt a validating bridge is threefold. First, there is no restriction on the type of financial services that can be offered by an organisation to its users. For example, StarkEx powers an NFT (Immutable) and derivative exchange (dYdX), Reddit plans to launch its own Arbitrum instance for community points [72] and Arbitrum One already hosts over 80 projects (September 2021). Second, organisations no longer have the liability of holding custody of their user's funds. This can reduce the impact of regulatory pressure as their service is only trusted with ordering transactions and not with protecting the user's funds from adversarial actors. Third, with the rise of validity proofs, we expect the average financial cost for leveraging the underlying blockchain's security to reduce as the transaction throughput on the off-chain system increases. Thus, we can scaling transaction throughput without sacrificing the underlying blockchain's security.

Finally, as explored in Appendix A, off-chain systems built with validating bridges are not restricted by the underlying blockchain. They can experiment

with different smart contract languages, virtual machines, and transaction ordering protocols. In a way, it is making the original vision for sidechains as a platform for experimentation a reality [6]. It is our hope that this SoK and the research questions highlighted will help towards building bridges for a multi-chain world.

References

1. Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *Int. Conf. Financial Cryptogr. Data Secur*, page 25, 2021.
2. Tom Brand Louis Guthmann Avihu Levy Alexey Akhunov, Eli Ben Sasson. Eip-2028: Transaction data gas cost reduction. Accessed 08/09/2021, `https://eips.ethereum.org/EIPS/eip-2028`.
3. Zachary Williamson Antonio Salazar Cardozo. Eip-1108: Reduce alt bn128 precompile gas costs. Accessed 21/08/2021, `https://eips.ethereum.org/EIPS/eip-1108`.
4. Arbitrum. Onestepproof. Accessed 08/09/2021, `https://github.com/OffchainLabs/arbitrum/blob/master/packages/arb-bridge-eth/contracts/arch/OneStepProof.sol`.
5. Georgia Avarikioti, Eleftherios Kokoris Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous payment channels. *arXiv preprint arXiv:1905.11360*, 2019.
6. Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 72, 2014.
7. Vivek Bagaria, Joachim Neu, and David Tse. Boomerang: Redundancy improves latency and throughput in payment-channel networks. In *International Conference on Financial Cryptography and Data Security*, pages 304–324. Springer, 2020.
8. Ohad Barta. Oracle price feed on starkex. Accessed 08/09/2021, `https://medium.com/starkware/oracle-price-feed-on-starkex-2b15a3ca122`.
9. Eli Ben-Sasson. One stark proof dealing simultaneously with @deversifi, @immutable and @sorarehq. Accessed 08/09/2021, `https://twitter.com/EliBenSasson/status/1437127156790927360`.
10. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.
11. Jeff Benson. Istanbul hard fork may lead to broken ethereum contracts. Accessed 08/09/2021, `https://decrypt.co/9013/istanbul-hard-fork-may-lead-to-broken-ethereum-contracts`.
12. Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society Open Science*, 5(8):180089, 2018.
13. Vitalik Buterin. Big integer modular exponentiation. Accessed 21/08/2021, `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-198.md`.

14. Vitalik Buterin. Control as liability. Accessed 08/09/2018, https://vitalik.ca/general/2019/05/09/control_as_liability.html.
15. Vitalik Buterin. An explanation of the sharding + das proposal. Accessed 08/09/2021, https://hackmd.io/@vbuterin/sharding_proposal.
16. Vitalik Buterin. Minimal viable plasma. Accessed 21/08/2021, https://ethresear.ch/t/minimal-viable-plasma/426.
17. Vitalik Buterin. On-chain scaling to potentially 500 tx/sec through mass tx validation. Accessed on 21/08/2021, urlhttps://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477.
18. Vitalik Buterin. Proposer/block builder separation-friendly fee market designs. Accessed 21/08/2021, https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725.
19. Vitalik Buterin. A rollup-centric ethereum roadmap. Accessed 08/09/2021, https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698.
20. Vitalik Buterin. What is the train-and-hotel problem? Accessed 08/09/2021, https://vitalik.ca/general/2017/12/31/sharding_faq.html#what-is-the-train-and-hotel-problem.
21. Vitalik Buterin. Why sharding is great: demystifying the technical properties. Accessed 21/08/2021, https://vitalik.ca/general/2021/04/07/sharding.html.
22. Matthew Campbell. Plasma on loom network dappchains: Scalable dapps with ethereum-secured assets. Accessed on 21/08/2021, urlhttps://medium.com/loom-network/loom-network-plasma-5e86caaadef2.
23. Celo. Optimistic interchain communication. Accessed 08/09/2018, https://github.com/celo-org/optics-monorepo.
24. Alex Obadia Christine Kim, Ben Edgington. An unlikely but effective solution to lowering fees on ethereum. Accessed 08/09/2021, https://www.coindesk.com/podcasts/mapping-out-eth-2-0/an-unlikely-but-effective-solution-to-lowering-fees-on-ethereum/.
25. Tom Close. Nitro protocol. *IACR Cryptol. ePrint Arch.*, 2019:219, 2019.
26. Mattison Asher Coogan Brennan. Analyzing polygon's proof of stake network. Accessed 01/09/2021, https://consensys.net/blog/blockchain-explained/analyzing-polygons-proof-of-stake-network/.
27. Gaby G Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 720–731, 2015.
28. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*, 2019.
29. Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. *White paper: https://blockstream. com/eltoo.pdf*, 2018.
30. Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and mtgox. In *European Symposium on Research in Computer Security*, pages 313–326. Springer, 2014.
31. Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

32. Casey Detrio. Phase one and done: eth2 as a data availability engine. Accessed 08/09/2021, https://ethresear.ch/t/phase-one-and-done-eth2-as-a-data-availability-engine/5269.
33. Brecht Devos. Loopring's zksnark prover optimizations. Accessed 08/09/2021, https://medium.com/loopring-protocol/zksnark-prover-optimizations-3e9a3e5578c0.
34. Yael Doweck. Checkpoints for faster finality in starknet. Accessed 21/08/2021, https://ethresear.ch/t/checkpoints-for-faster-finality-in-starknet/9633.
35. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive, 2017: 635, 2017.
36. Ed Felton. Cheater checking: How attention challenges solve the verifier's dilemmas. Accessed 21/08/2021, https://medium.com/offchainlabs/cheater-checking-how-attention-challenges-solve-the-verifiers-dilemma-681a92d9948e.
37. Ed Felton. The cheater checking problem: Why the verifier's dilemma is harder than you thinkg. Accessed 21/08/2021, https://medium.com/offchainlabs/the-cheater-checking-problem-why-the-verifiers-dilemma-is-harder-than-you-think-9c7156505ca1.
38. Ed Felton. Mev auctions considered harmful. Accessed 21/08/2021, https://medium.com/offchainlabs/mev-auctions-considered-harmful-fa72f61a40ea.
39. Karl Floersch. Accessed 21/08/2021, https://ethresear.ch/t/mev-auction-auctioning-transaction-ordering-rights-as-a-solution-to-miner-extractable-value/6788.
40. Francesco. Committee-driven mev smoothing. Accessed 21/08/2021, https://ethresear.ch/t/committee-driven-mev-smoothing/10408.
41. Alex Gluchowski. Introducing zksync: the missing link to mass adoption of ethereum. Accessed 21/08/2021, https://medium.com/matter-labs/introducing-zk-sync-the-missing-link-to-mass-adoption-of-ethereum-14c9cea83f58.
42. Gnosis. Cowswap upgrades to final version by fully integrating with balancer v2. Accessed 08/09/2021, https://medium.com/@gnosisPM/cowswap-upgrades-to-final-version-by-fully-integrating-with-balancer-v2-21f4d635da1.
43. Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo–a turing-complete stark-friendly cpu architecture. 2021.
44. Lior Goldberg and Michael Riabzev. The fact registry. Accessed 21/08/2021, https://medium.com/starkware/the-fact-registry-a64aafb598b68.
45. Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489, 2017.
46. Plasma Group. Introducing plasma group. Accessed on 21/08/2021, urlhttps://medium.com/plasma-group/deployplasma-dd1cf0b2ab55.
47. Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020.
48. Samuel Haig. All you can eat: Sushiswap deploys contracts on five new networks. Accessed 08/09/2018, https://cointelegraph.com/news/all-you-can-eat-sushiswap-deploys-contracts-on-five-new-networks.
49. Colin Harper. Doge imitators help send ethereum transaction fees to all-time highs. Accessed 08/09/2021, https://www.coindesk.com/markets/2021/05/11/doge-imitators-help-send-ethereum-transaction-fees-to-all-time-highs/.

50. Kevin Lewi Harjasleen Malvai Irakliy Khaburzaniya, Kostas Chalkias. Open sourcing winterfell: A stark prover and verifier. Accessed 08/09/2018, https://engineering.fb.com/2021/08/04/open-source/winterfell/.
51. Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1353–1370, 2018.
52. Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Annual International Cryptology Conference*, pages 451–480. Springer, 2020.
53. Darpan Keswani. The difference between moreviable plasma and minimumviable plasma. Accessed on 21/08/2021, urlhttps://omg.network/more-viable-plasma-vs-minimum-viable-plasma/.
54. Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 31–40, 2019.
55. Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453, 2017.
56. Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. *Cryptology ePrint Archive, Report 2018/642*, 2018.
57. Georgios Konstantopoulos. How does optimism's rollup really work? Accessed 21/08/2021, https://research.paradigm.xyz/optimism.
58. Georgios Konstantopoulos. Plasma cash: towards more efficient plasma constructions. *arXiv preprint arXiv:1911.12095*, 2019.
59. Matter Labs. A test of zksync 1.0's exodus mode on ropsten. Accessed 21/08/2021, https://medium.com/matter-labs/a-test-of-zksync-1-0s-exodus-mode-on-ropsten-b1ece0499847.
60. Matter Labs. zkporter: a breakthrough in l2 scaling. Accessed 08/09/2021, https://medium.com/matter-labs/zkporter-a-breakthrough-in-l2-scaling-ed5e48842fbf.
61. Matter Labs. zksync 2.0: Hello ethereum! Accessed 08/09/2021, https://medium.com/matter-labs/zksync-2-0-hello-ethereum-ca48588de179.
62. Matter Labs. zksync 2.0 roadmap update: zkevm testnet in may, mainnet in august. Accessed 08/09/2021, https://medium.com/matter-labs/zksync-2-0-roadmap-update-zkevm-testnet-in-may-mainnet-in-august-379c66995021.
63. Matter Labs. zksync rollup protocol - data types. Accessed 08/09/2021, https://github.com/matter-labs/zksync/blob/3a1ac7cc5d93d5a742f2578ea893578bc98c8faa/docs/protocol.md#data-format.
64. Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using trusted execution environments. *arXiv preprint arXiv:1612.07766*, 2016.
65. Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 706–719, 2015.
66. Sam MacPherson. Announcing the optimism dai bridge with fast withdrawals. Accessed 08/09/2018, https://forum.makerdao.com/t/announcing-the-optimism-dai-bridge-with-fast-withdrawals/6938.

67. MakerDAO. The transfer of mkr token control to governance: The final step. Accessed 08/09/2021, https://blog.makerdao.com/the-transfer-of-mkr-token-control-to-governance-the-final-step/.
68. Patrick McCorry, Surya Bakshi, Iddo Bentov, Andrew Miller, and Sarah Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.
69. Patrick McCorry, Chris Buckland, Surya Bakshi, Karl Wüst, and Andrew Miller. You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*, pages 35–49. Springer, 2019.
70. Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *CoRR abs/1702.05812*, 2017.
71. Tyler Moore and Nicolas Christin. Beware the middleman: Empirical analysis of bitcoin-exchange risk. In *International Conference on Financial Cryptography and Data Security*, pages 25–33. Springer, 2013.
72. Saniya More. Reddit is scaling its two ethereum-based tokens using layer 2 solution arbitrum. Accessed 08/09/2018, https://www.theblockcrypto.com/post/112250/reddit-is-scaling-its-two-ethereum-based-tokens-using-layer-2-solution-arbitrum.
73. Samson Mow. Eight new members join the liquid federation. Accessed 01/09/2021, https://blockstream.com/2020/07/29/en-eight-new-members-join-the-liquid-federation/.
74. Charlie Noyes. Mev and me. Accessed 08/09/2018, https://research.paradigm.xyz/MEV.
75. Optimism. Mainnet soft launch! Accessed 08/09/2021, https://medium.com/ethereum-optimism/mainnet-soft-launch-7cacc0143cd5.
76. Optimism. Optimistic rollup overview. Accessed 08/09/2021, https://github.com/ethereum-optimism/optimistic-specs/blob/0e9673af0f2cafd89ac7d6c0e5d8bed7c67b74ca/overview.md.
77. Ethereum Optimism. Ovm 2.0 changeset. Accessed 21/09/2021, TheFutureofOptimisticEthereum.
78. Ethereum Optimism. Ovm deep dive. Accessed 21/08/2021, https://medium.com/ethereum-optimism/ovm-deep-dive-a300d1085f52.
79. Olaoluwa Osuntokun. [lightning-dev] amp: Atomic multi-path payments over lightning. Accessed 08/09/2018, https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.
80. Alejandro Ranchal Pedrosa, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Scalable lightning factories for bitcoin. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 302–309, 2019.
81. Polynya. Volitions: best of all worlds. Accessed 08/09/2021, https://polynya.medium.com/volitions-best-of-all-worlds-cfd313aec9a8.
82. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.
83. Near Protocol. The rainbow bridge is live on near protocol: Welcome to a new era of interoperability. Accessed 08/09/2021, https://medium.com/nearprotocol/the-rainbow-bridge-is-live-on-near-protocol-welcome-to-a-new-era-of-interoperability-a15747c04a
84. Ashwin Ramachandran. The life and death of plasma. Accessed on 21/08/2021, urlhttps://medium.com/dragonfly-research/the-life-and-death-of-plasma-b72c6a59c5ad.

85. Christian Reitwiessner. Eip-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt bn128. Accessed 21/08/2021, https://eips.ethereum.org/EIPS/eip-196.
86. Bailey Reutzel. Submarine sends: Ic3's plan to clamp down on ico cheats. Accessed on 21/08/2021, urlhttps://www.coindesk.com/submarine-sends-inside-ic3s-plan-to-clamp-down-on-ico-cheats.
87. Dan Robinson. Htlcs considered harmful. Accessed 08/09/2018, https://www.youtube.com/watch?v=qUAyW4pdooA&ab_channel=CyberInitiative.
88. Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.
89. David Schwartz. Why is ripple's effective genesis ledger at 32570? Accessed 21/08/2021, https://archive.is/zEfQN.
90. Matt Solomon. Porting solidity contracts to optimism: A guide using uniswap v2. Accessed 08/09/2021, https://hackmd.io/@scopelift/Hy853dTsP#OVM-vs-EVM-Compiling.
91. Starkware. Volition and the emerging data availability spectrum. Accessed 21/08/2021, https://medium.com/starkware/volition-and-the-emerging-data-availability-spectrum-87e8bfa09bb.
92. Arbitrum Team. Github. Accessed 21/08/2021, https://github.com/OffchainLabs/arbitrum/tree/3baeaa9500d884e2df146e55cb84d70ea14707c6.
93. Oasis Team. Github. Accessed 10-09-2021, https://github.com/oasisprotocol/oasis-core/commit/d1b69f5aac431eac05705cae0297936ad410a3eb.
94. Oasis Team. The oasis blockchain platform. https://oasisprotocol.org/papers.
95. Optimism Team. Github. Accessed 21/08/2021, https://github.com/ethereum-optimism/optimism/tree/10c75e80195e5deefc06e11468d964270dbc3a6f.
96. Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756*, 2019.
97. UToday. Tether just burned 1,000,000,000 usdt tokens. here's what happened. Accessed 08/09/2021, https://coinmarketcap.com/headlines/news/tether-just-burned-1000000000-usdt-tokens-heres-what-happened/.
98. Greg Vardy. Warp your way to starknet. Accessed 08/09/2021, https://medium.com/nethermind-eth/warp-your-way-to-starknet-ddd6856875e0.
99. Christian Reitwiessner Vitalik Buterin. Eip-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt bn128. Accessed 21/08/2021, https://eips.ethereum.org/EIPS/eip-197.
100. Rick Dudley Matthew Slipper Ian Norden Abdelhamid Bakhta Vitalik Buterin, Eric Conner. Eip-1559: Fee market change for eth 1.0 chain. Accessed 08/09/2021, https://eips.ethereum.org/EIPS/eip-1559.
101. Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021.
102. Chris Whinfrey. Hop: Send tokens across rollups. Accessed 08/09/2018, https://hop.exchange/whitepaper.pdf.
103. Barry Whitehat. Roll up. Accessed 21/08/2021, https://github.com/barryWhiteHat/roll_up.

104. Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: communication across distributed ledgers. 2019.
105. Zcash. What is jubjub? Accessed 21/08/2021, `https://z.cash/technology/jubjub/`.

A Code inspection of validating bridge projects

We provide an overview of validating bridge implementations deployed in practice which includes Abritrum, Optimism, Oasis, StarkEx and ZkSync. As a word of warning, all validating bridges and off-chain system designs are still evolving. Each project's implementation is a snapshot to evaluate its design, goals and future direction. In our assessment, we perform a code inspection of their bridge contracts to uncover how they solve the security goals for the data availability problem, the state integrity problem and censorship resistance.

A.1 Arbitrum

Arbitrum is an optimistic rollup that originates from academic research[51], although the implementation's protocol has evolved. Our code inspection is based on the following repository commitment[92]. We investigate the contracts:

- Inbox.sol Accepts transactions destined for execution on the off-chain system and orders them for execution.
- SequencerInbox.sol A single sequencer can send transactions for ordering via the fast-path.
- Rollup.sol[9] Co-ordinates all bridge components, manages the executors staking process and maintains the canonical chain of commitments.
- Node.sol Instantiates a single commitment.
- Challenge.sol Manages the multi-round fraud proof for a single commitment.
- Outbox.sol Processes transactions that migrate from the off-chain system to the underlying blockchain.

Data Availability All transaction data (including the user's signature) is published to the bridge contract via Inbox.sol. The inbox associates every transaction with an Ethereum block height and this timestamp is used to order transactions for execution. The SequencerInbox.sol allows the sequencer to set a transaction's timestamp within a restricted range and this is used to facilitate the fast-path of transaction inclusion (i.e. up to 50 blocks in the past). This allows any user to verify the final ordering of transactions and the final execution is a deterministic process.

State transition integrity An executor creates a new unconfirmed commitment (Node.sol) and moves their stake onto it via Rollup.sol. All participants validating the chain can then view and verify this commitment. The commitment remains in an unconfirmed state for a long period (1 week) to give ample opportunity for anyone to verify the commitment and potentially challenge it. During this time all stakers who agree with the commitment must move their stake onto it via Rollup.sol. If a staker does not move their stake onto a new commitment,

[9] An additional contract RollupUser.sol extends the functionality of Rollup.sol via a delegatecall. We omit this detail.

they can be deregistered by any other staker. At the end of this period, and if all registered stakers have staked on it, then the commitment can be confirmed (finalised). If at any time prior to this a staker perceives a commitment to be invalid, they must post on-chain a rival sibling commitment, stake on it, then open a challenge between themselves and any of the stakers currently staked on the perceived invalid commitment.

Each challenge is contained within a new instance of a Challenge.sol which enforces the multi-round challenge rules as described in Section 5.2. Each party has a stop clock timer which is running during their turn, if they run out of time then they forfeit. The challenge process eventually ends with the execution of a single instruction to resolve the dispute, and the result is communicated to Rollup.sol. Half of the stake of the losing party is burnt, the other half is awarded to the winning party. The losing party is deregistered as a staker to satisfy the constraint that in order for a commitment to be confirmed it must be staked on by all registered stakers.

Censorship resistance To bypass the sequencer, any user can force inclusion of a transaction via Inbox.sol and the transaction will be ordered for execution after a fixed time period (i.e., after the sequencer's fast-path time period has expired). To guarantee liveness of execution, their implementation allows a user to become an executor by staking funds in Rollup.sol and a commitment must process a minimum number of transactions to ensure transactions are eventually executed. Note there is an approve-list for executors at the time of assessment.

A.2 Oasis

Oasis is a layer 1 blockchain with a validating bridge mechanism built-in its Proof-of-Stake consensus layer [94]. It uses a Tendermint-based consensus protocol for its consensus layer. Because it is not constrained by EVM compatibility limitations, instead of a bridge EVM contract Oasis uses a Go-based bridge module incorporated into the consensus layer [93]. The module serves as protocol endpoint for the discrepancy detection (DD)/discrepancy resolution (DR) protocols for a committee-based validating bridge design. Additional modules handle random beacon, the appointment protocol for committee election, etc. Oasis' validating bridge mechanism is designed to be general and can support different VMs (called ParaTimes) concurrently, such as EVM-compatible VM and Rust-based VM. The core of the implementation is in these files:

- go/api/commitment/pool.go Implements the consensus layer DD/DR logic. Compares fast-path DD results and triggers slow-path DR if a discrepancy is detected.
- go/consensus/tendermint/apps/roothash/roothash.go Handle DD/DR committee logic. Storage signature handling.
- go/roothash/api/commitment/executor.go Verify storage receipt signatures.

Data availability Transaction data and system state are stored via a committee of data availability providers. A threshold number of providers must sign the storage roots to declare the data is indeed available, and the consensus layer verifies signatures. In the future, this is planned to be extended to utilize erasure coding to improve replication efficiency and other data availability layers as they become available. Note that this separate storage system for data availability is a logical separation. In practice, this can be run in different ways. If a separate storage committee runs the storage system, then the data is effectively stored offchain. However, it can also be deployed in a way that the consensus nodes at the consensus layer runs the storage system, in this case the data is effectively stored onchain.

State transition integrity The DD protocol used in Oasis should be viewed as an error detection protocol and not an error correction protocol. The DD committee size is chosen so that the probability that all DD committee members are Byzantine is negligibly small. Thus, when Byzantine members report a divergent state root as the result of executing a batch of transactions, at least one honest member is available to detect the error—the determination of the correct state to accept is deferred to the DR protocol. This allows the DR protocol to use more resources than the DD protocol, e.g., involving many more nodes in a voting protocol, so that the adversary should be unable to overcome the resource limitations. One case of DR is to have the whole consensus committee to run the re-execution in the DR protocol. An honest-majority DR protocol is what is implemented in the code; of course, other DR schemes such as bisection could also be used. The DD/DR execution is non-interactive from the viewpoint of the consensus layer, since the bridge module does not forward the results to be committed until DD/DR has successfully determined a correct state root.

Censorship resistance Oasis uses a per-epoch random choice of staked nodes to act as executors and sequencers. Nodes serve in rotation as the sequencer, chooses transactions to include in a block, and determines transaction order within the block. Because the DD committee size is chosen so that not all members could be Byzantine, within each cycle of rotation there will be at least one fair sequencer.

A.3 Optimism

Optimism is an optimistic rollup with a single-round fraud proof. They have plans to move towards a multi-round fraud proof system due to issues with EVM-compatibility and constraints imposed on the size of a transaction [76]. Our code inspection is based on the single-round fraud proof system at commit[95]. We note it will soon become obsolete as they are moving towards a multi-round fraud-proof approach [76]. We investigate the contracts:

– OVM_LibraryManager Admin-only feature to install various configurations for the Optimism network including the sequencer.

– OVM_L1StandardBridge Processes deposits and withdrawals for ETH/ERC20 tokens.
– OVM_CanonicalTransactionChain.sol Orders transactions for execution from users and sequencers.
– OVM_StateCommitmentChain.sol Allows a bonded sequencer to assert a new commitment.
– OVM_BondManager Manages the locked stake on behalf of executors.
– OVM_FraudVerifier.sol Co-ordinates the fraud proof process for an asserted commitment.
– OVM_StateTransitioner.sol Takes as input a pre-state root and a transaction, and transitions to a post-state root.
– OVM_L1CrossDomainMessenger.sol Processes messages that migrate from the off-chain system to the underlying blockchain(and vice versa).

Data availability All transaction data (including the user's signature) is published to the bridge contract via OVM_CanonicalTransactionChain.sol. Any user can enqueue a transaction for ordering, but due to incomplete functionality, only the sequencer can force the final ordering of transactions.

State transition integrity The sequencer has priority for asserting a commitment, but an executor can proceed after a fixed deadline. The sender needs to submit a list of hashes (the state root of each transaction) to OVM_StateCommitmentChain and the contract will construct the commitment. There is a minimum of just one transaction that must be processed in a commitment.

The contract OVM_FraudVerifier.sol co-ordinates the fraud proof and a detailed explanation can be found here[57]. To assist with the fraud proof, some EVM operation codes (opcodes) have been re-implemented as smart contracts and as a result developers must re-compile their solidity project with the optimism compiler. The proof is non-interactive and the challenger can perform the entire proof of fraud single-handedly. Generally speaking, the challenger needs to submit a pre-state commitment to challenge a post-state commitment alongside the transaction that was applied to the pre-state root OVM_FraudVerifier.sol. This deploys OVM_StateTransitioner.sol which can be used by the challenger to set up the state and accounts required by the OVM_StateTransitioner.sol to execute the transaction. The post-state for a transaction is generated in OVM_StateTransitioner.sol and this is checked against the initial (post-state) commitment posted by the executor. If it is different, then fraud is confirmed and the commitment is rejected. As well, the executor who supplied the fraudulent state root is fully slashed. All pending commitments must form a canonical chain and rejecting an earlier commitment will reject all dependent commitments.

In the current implementation, this fraud proof mechanism has imposed computational limits on the size of a rollup transaction as it must be re-executable on Ethereum.

Censorship resistance A user can enqueue a transaction for ordering, but they cannot force the inclusion of a transaction for execution as the implementation

is not complete. In terms of execution liveness, there does not appear to be an approve-list for executors and anyone can stake funds via OVM_BondManager to become executor. An executor is only forced to execute at least one transaction which does not necessarily guarantee execution liveness. In the current implementation as a short-term measure, the sequencer has priority to post a commitment and it is possible for the sequencer to continuously publish commitments such that an honest executor cannot participate.

A.4 StarkEx

StarkExchange (StarkEx) has a validating bridge and relies on validity proofs. We inspected the code used by Immutable which is not a rollup and instead a committee is responsible for making the data publicly available. Furthermore, it supports trading assets, but not general smart contract functionality. Note the role of sequencer and executor is combined. We investigate the following contracts:

- Committee.sol Accepts a list of signatures from the data availability committee.
- Escapes.sol Facilitates an on-chain exodus if the vault is frozen.
- FullWithdrawals.sol Request withdrawal to be processed by sequencer.
- Operator.sol Installs (and removes) a single operator that is a hot wallet for the StarkEx service.
- Users.sol Associates a user's ethereum account with a stark public key.
- GpsStatementVerifier.sol Responsible for verifying a STARK proof
- UpdateState.sol Responsible for updating the state root (if the STARK proof is verified).
- FactRegistry.sol A *fact* is a statement that is considered true. For example, a contract will verify a STARK proof to consider a commitment as well-formed and the commitment can be stored in the fact registry. [44]

Data availability A committee of data availability providers are installed in Committee.sol. A fixed number of committee members must sign a message to attest that the data is indeed available and the signatures are checked by the contract. If the threshold of signatures is reached this result is stored as a data availability fact in a FactRegistry.sol for later use. Note this introduces a stronger trust assumption than outlined in Section 3.1 as the bridge contract cannot independently verify that the data is truly available.

State transition integrity Anyone can publish STARK proofs which are verified and, if valid, stored by the GpsStatementVerifier.sol as a fact. The sequencer may then publish a new commitment to UpdateState.sol which checks that a data availability fact and a proof verification fact have both been stored, and if so accepts the commitment as the new state root of the chain. When a new root is added to UpdateState.sol on-chain operations associated with the state update are also processed, such as withdrawals and deposits. Only the sequencer may

call UpdateState.sol, and they may only do so if the contract has not been frozen. Note, thanks to the validity proof, there is no requirement for a challenger to assist the bridge contract.

Censorship resistance A user can request to withdraw their funds via FullWithdrawals.sol and it must be processed in a future commitment by a fixed deadline. If the withdrawal is not processed by the deadline, then the user can freeze the entire system in Escapes.sol. This freezes the off-chain system's state for at least one week and all users can proceed to withdraw their funds via an on-chain exodus.

A.5 ZKSync 1.0

A validity proof rollup for value transfer of ETH, ERC20 tokens and NFTs. It is implemented using SNARKS and there is an upgrade coming that plans to be EVM-compatible [61]. We investigated the following contracts:

- ZkSync.sol Core functionality for processing deposits, withdrawals and new state commitments (including verifying the validity proof).
- AdditionalZkSync.sol Additional functionality related to freezing ZKSync and permitting an on-chain exodus.

Data availability All transaction data is publicly available on Ethereum and it is published in ZKSync.sol to create the next state commitment. This data is compressed which includes the from account id, to account id, token id, packed amount and fee. It does not include the signature and all identifiers represent an index in the database to reduce bytesize [63].

State transition integrity All transactions are batched into blocks and there are three steps to confirming a block. First, the sequencer is required to post the transaction data and ZkSync.sol will form a state commitment. Second, the sequencer is required to post a validity proof to prove that the state commitment is well-formed and valid. The sequencer is required to confirm the block that will execute all pending operations for the block such as withdrawal requests or changing the user's public key (i.e., users have a distinct public key that is separate to their Ethereum account). Because of the separate steps, it is possible for a sequencer to post a block of transaction data that cannot be validated. The sequencer can revert any pending blocks in order to publish another block that is valid. Note, thanks to the validity proof, there is no requirement for a challenger.

Censorship resistance A user can request a full withdrawal of their funds in ZkSync.sol and this sets a deadline for the transaction to be included in a future block. Assuming the sequencer has ignored the withdrawal request and they have not processed transactions in the on-chain priority queue, then the user can activate an on-chain exodus mode that freezes the network based on the last accepted commitment. All users can proceed to withdraw their funds using the functionality in AdditionalZkSync.sol. This mode was previously triggered on Ethereum's Ropsten test network [59].

B Data availability and gas costs

We consider the representation of data that is used to recompute the widely replicated database and afterwards we consider the gas costs for posting this data to Ethereum.

B.1 Data type

An ethos in the community is to scale the network's transaction throughput without increasing the resources required by the verifier (i.e., the cpu, storage and memory resources remain the same). This requires rollups, that post all data to the Ethereum network, to minimize the data's bytesize to avoid over-consuming the network's bandwidth. In regards to the data, there are two components that should be considered:

- *Validity.* Th evidence for why the data is correct and it should be processed.
- *Execution.* The data represents a set of instructions on how to update the off-chain system's database.

An example is a transaction with a function, arguments and a signature. The signature is *why the data is valid* as it confirms the user's identity and this can be used to verify if they have a sufficient balance to cover the transaction fee. The functions and arguments is the *execution* as it can be processed to update the off-chain system's database. As we will see next, how the bridge contract verifies a commitment's integrity will impact the type of data that is published:

Re-execution validity proof can only compress data In fraud proof systems, the bridge contract may re-execute a disputed state transition. To determine if it is valid, the bridge needs to verify why the state transition is valid (i.e., the user's signature) and its execution (i.e., the execution's asserted post-state is correct). It is possible to reduce the quantity of signatures posted by the sequencer via BLS signature aggregation [57]. This only represents a reduction in the data size and it does not remove the requirement to post why a user-generated transaction is valid. On the other hand, to reduce byte-size of data that represents the set of instructions to execute, so far it appears users may need to co-operate off-chain and batch their transactions into a single execution. A simple example is to combine several swaps into a single trade and some DeFi projects such as CowSwap [42] which already offers this service on Ethereum.

Zero knowledge proofs can aggregate state updates A zero knowledge proof can remove the need to post the validity aspect of the data (i.e, user signatures) as it is bundled into the proof. This is implemented in ZkSync as the sequencer only publishes the transfer details (and not the signature) [63]. Be that as it may, a zero knowledge proof can also replace the transaction data with a list of state updates (state diff) and aggregate the impact of multiple transactions into a single state update. As such, the details of every transaction can remain off-chain and this is implemented in StarkEx. For example, hundreds of oracle price updates can be reduced to a single state update [8].

Protocol	Deposit	Data availability	Confirm commitment	Withdraw
Arbitrum	96,041	5,155 per tx	923,000	162,392
Optimism	209,787	4,118 per tx	1,112 per tx	516,088
StarkEx	116,956	86,199*	3,243,418*	88,279
ZKSync	62,599	967 per tx	576,691	15,000

Table 2: Ethereum gas costs for operations performed by a validating bridge.

B.2 Gas costs for data

For Ethereum-based validating bridges, gas is the ultimate resource bottleneck and it is a common metric that can be used to measure their scaling efficiency. In the following, we evaluate the gas costs in Table 2 which highlights for each project the gas costs for depositing, withdrawing, data availability and confirming a new state commitment.

Deposits and Withdrawals A withdrawal in Optimism is expensive as a user provides an inclusion proof to demonstrate their entitlement before withdrawing coins. ZkSync is the cheapest as it amortizes the gas costs by processing all withdrawals in a single transaction. The remaining project's have gas costs that are expected for a deposit or withdrawal.

Data availability Since the sequencer is responsible for posting transaction data to the bridge contract, we have implemented a script[10] that computes the average gas used in Arbitrum, Optimism and ZkSync. The average gas for a transaction takes into account any user-generated transaction processed by the off-chain system and it does not differentiate whether it is a transfer, deployment, or withdrawal request. As we can see, Arbitrum and Optimism is roughly the same at 4-5k gas and the average ZkSync transaction is 850 gas as it does not support smart contract functionality.

Confirm commitment Arbitrum requires two transactions to assert the commitment (748k gas) and to confirm the commitment after the challenge period (175k gas). It is higher than necessary as every commitment is a new instantiation of a contract called Node.sol alongside storing a list of values. We have not taken into account the gas costs for executors to stake on a new commitment in Arbitrum. Optimism requires an intermediary state hash to be published for every transaction and this is used to compute the final commitment. There is no explicit cost to finalise a commitment as it is implicitly accepted by the bridge contract after the challenge period has expired. ZkSync verifies a single validity proof (PLONK) and this costs 576k gas. Overall, it appears that Arbitrum will eventually be cheaper than Optimism if a single commitment covers more than 930 transactions and surprisingly asserting a commitment in Arbitrum costs more than verifying a validity proof in ZkSync.

[10] Scripts can be found at https://github.com/fc22submission/commitchain-scripts

Special case: StarkEx We have implemented a script to compute the average gas cost for data availability and confirm commitment based on the deployment of Immutable.[11] However, we have added an asterisk in Table 2 as it is not fair to compare the gas costs with other projects. First, the immutable contracts rely on a data availability committee and the 86k gas reflects verifying the respective signatures. Second, there is another project called dYdX which does post all data to the underlying blockchain, but as discussed in Section B.1 the data represents state updates that can be applied to the database and not individual transactions. Third, several smart contracts are involved in verifying a STARK proof including FriStatementContract, MemoryPageFactRegistry, MerkleStatementContract and GpsStatementVerifier as there is no native support in the EVM. Within two weeks of our analysis (September 2021), all Starkware projects including Immutable upgraded to StarkEx V3 which allows a single prover to prove the computation for all projects together and thus amortize the cost [9]. Based on these findings, we have proposed a measurement study in Section 6.1 to evaluate the potential gas-savings of a validity proof system like StarkEx compared to the other projects.

C Virtual machines and EVM-compatible smart contracts

Virtual machine re-execution on Ethereum In a fraud-proof system, the virtual machine for the off-chain system must be designed such that the bridge contract can pinpoint and execute a disputed state transition. As discussed in Section A, Arbitrum pinpoints a single instruction for execution whereas Optimism executes an entire EVM-native transaction. The virtual machine for both projects must have a compatible implementation that is executable by the Ethereum Virtual Machine (EVM). Compatibility includes how to execute every operation code of the virtual machine and how to provide the bridge contract with a copy of the execution context (i.e., state) prior to the execution step.

In Arbitrum, the Arbitrum Virtual Machine (AVM) forms the basis for execution and proving fraud. It has at least two implementations: one that is run off chain by Arbitrum nodes for executing all transactions and one that is implemented in Solidity for the bridge contract. We are only concerned with the Solidity implementation. They have re-implemented every operation code to support a fully virtualised environment that manages memory and tracks resource usage when executing transactions [4]. This environment provides the internal context to prove the execution for a single step of the AVM. One advantage of defining a new virtual machine is that the VM context can be optimised for executing the proving step in the bridge contract. For example, the pinpointed state transition can be a single operation code as the bridge contract has access to the memory stack and execution trace, which is not typically accessible for the EVM. Another is that Arbitrum can offer new operation codes (and functionality) beyond what is supported by the EVM. An example of where they've

[11] An NFT project that is using StarkEx under the hood.

made use of this is to write new opcodes for all Arbitrum node operations that require proving, such as reading from the Inbox contract. This means that many operations of the Arbitrum node, not just transaction execution, can be proven correct by fraud proofs. The downside of the Arbitrum approach is maintaining duplicate implementations of the AVM as there is a danger the execution does not match up amongst them (i.e., including bug-for-bug). This can result in a slashing event by the bridge contract.

In Optimism, the Optimistic Virtual Machine (OVM) is a minimal adaptation of the EVM. The OVM tries to re-execute as much of the transaction as possible in the same environment (EVM) on both L1 and L2 networks. However since it's not possible for them to re-execute all opcodes due to the differences in state, they need to replace some opcodes with synthetic ones which mimic the behaviour of their EVM counterparts (OVM). One example includes block.number as the off-chain system and Ethereum will return different block heights [78]. This is because the global state must be synchronised in order for the transactions to yield the same responses. To resolve the issue with block.number, they have implemented a function call to one of Optimism's contracts (OVM_ExecutionManager.sol) which contains a pre-populated value for the block number. The same process is used for all opcodes that access global state. As well, other opcodes such as CALL, DELEGATECALL, etc are implemented as Solidity smart contracts as they need to invoke contracts deployed for Optimism (and not Ethereum). The benefit of the above approach is that most existing infrastructure can be re-used by the team and it is still possible to implement new operation codes as smart contracts. Because Optimism tries to execute as much as possible in the EVM, it cannot access the intermediary state of a transaction and this is why the fraud proof re-executes the entire transaction. Note, they are currently in the process of changing the OVM to remove the need for implementing a range of operation codes as smart contract function calls including the execution manager [77].

EVM-compatible smart contracts for fraud-proof systems Arbitrum have implemented a new language, Mini, which compiles to AVM code. To support EVM-compatible smart contracts, they have implemented an EVM to AVM compiler in Mini which allows the sequencer to accept a signed EVM transaction, transpile it to AVM code, and execute it. All steps are provable and executable by the bridge contract in the event of a dispute. The downside of Arbitrum's approach is the complexity of maintaining a provable transpiler and operating system on top of the EVM. Some existing tooling (such as the JSON RPC command debug_traceTransaction) may act in an unexpected way.

Optimism, at the time of writing (September 2021), requires all EVM smart contracts to be compiled using an adjusted Solidity compiler that generates OVM bytecode instead of EVM bytecode. Because some opcodes in the OVM bytecode are replaced by contract calls, transactions that run on Optimism have different (and greater) gas footprint than its corresponding EVM code. As a consequence, a transaction that can be executed within the Ethereum block limit may no longer be possible when it is executed by an OVM-compiled smart contract.

This impacts Optimism's single-round fraud proof system which requires all off-chain transactions to be fully executable by the bridge contract. To take this into account, all transactions running on Optimism have a lower gas limit than Ethereum. Another issue is that Ethereum has a 24KB max contract byte size and a smart contract that is transpiled to OVM bytecode may become too large for deployment. This may require pre-existing smart contracts to be split into sub contracts before deployment [90].

Verifying zero knowledge proofs in the EVM The Ethereum community have upgraded the EVM [85,99,13,3] to natively verify SNARKs [10] and this has substantially reduced the cost of verifying a proof.[12] SNARKs are form of *verifiable computing* that allows the off-chain system's executor to process all transactions and to provide a succinct proof that proves the computation was performed correctly. The bridge contract is only concerned with checking the succinct proof's validity and it will not learn anything about the virtual machine except that the computation was performed correctly.

The off-chain's virtual machine is never re-executed by the EVM and instead the goal is to build a virtual machine whose computation can be efficiently proven with a zero knowledge proof (*zk-friendly*). Most SNARKs represent computation as an arithmetic circuit (sometimes called constraints) over a prime finite field and not all computation can be efficiently represented as an arithmetic circuit. The issue often boils down to the fact that a computation may operate over a different field than the one supported by the SNARK and mixing different fields incurs additional computation. For example, sha256 and other symmetric cryptographic protocols are computed over a binary field (boolean operations) and this incurs an overhead of embedding the boolean field in the prime field [105]. At the same time, SNARKs are not turing-complete and how to represent unbounded loops requires special consideration such as using a recursive snark, a proof carrying data, or just bounding the total iterations and executing the loop's body an exact number of times.

EVM-compatibility for zk-friendly virtual machines ZkSync and Starkware have built CPU architectures, Zinc[41] and Cairo[43] respectively, that are zk-friendly in that they only offer operations that can be efficiently executed in their respective proving systems. ZincVM is similar to TinyRam which is a reduced instruction set computer and it is not turing-complete (no loops). Although this is being replaced with an upcoming release called zkEVM which is expected to be turing-complete [62]. On the other hand, Cairo is a turing-complete CPU architecture. As of writing, the languages have not gained significant traction in the Ethereum smart contract developer community. We suspect some potential reasons include the friction associated with switching to a new smart contract language, the desire to re-use tooling that is compatible with the EVM and smart contract teams may lack the human resources to re-write their smart contracts from scratch and get them audited just for one off-chain system.

[12] For example, [3] highlights the gas cost for AZTEC confidential transaction was reduced from 820k gas to 197k gas

This has led to ZkSync and Starkware working towards EVM-compatibility for their smart contract environment. However, the EVM architecture has specific opcodes which are not natively zk-friendly such as loops, cryptographic hash functions and composability amongst smart contracts. Both projects have implemented optimised circuits for specific operations to avoid the compiler introducing additional overhead. For example, ZincVM currently replaces keccak256 with a collision-resistant hash [61] and later re-introduce it as a precompile, whereas Cairo can natively support keccak256 although it will default to Pedersen hash when possible. The final goal is to allow smart contract developers to write code in Solidity and a transpiler will convert it into source code that is compatible with the zk-compiler. For example, Nethermind is working with the Starkware team to convert Solidity code into Cairo [98].