Plasma: Scalable Autonomous Smart Contracts

Joseph Poon
joseph@lightning.network

Vitalik Buterin
vitalik@ethereum.org

August 11, 2017
WORKING DRAFT
https://plasma.io/

Abstract

Plasma is a proposed framework for incentivized and enforced execution of smart contracts which is scalable to a significant amount of state updates per second (potentially billions) enabling the blockchain to be able to represent a significant amount of decentralized financial applications worldwide. These smart contracts are incentivized to continue operation autonomously via network transaction fees, which is ultimately reliant upon the underlying blockchain (e.g. Ethereum) to enforce transactional state transitions.

We propose a method for decentralized autonomous applications to scale to process not only financial activity, but also construct economic incentives for globally persistent data services, which may produce an alternative to centralized server farms.

Plasma is composed of two key parts of the design: Reframing all blockchain computation into a set of MapReduce functions, and an optional method to do Proof-of-Stake token bonding on top of existing blockchains with the understanding that the Nakamoto Consensus incentives discourage block withholding.

This construction is achieved by composing smart contracts on the main blockchain using fraud proofs whereby state transitions can be enforced on a parent blockchain. We compose blockchains into a tree hierarchy, and treat each as an individual branch blockchain with enforced blockchain history and MapReducible computation committed into merkle proofs. By framing one's ledger entry into a child blockchain which is enforced by the parent chain, one can enable incredible scale with minimized trust (presuming root blockchain availability and correctness).

The greatest complexity around global enforcement of non-global data revolves around data availability and block withholding attacks, Plasma has mitigations for this issue by allowing for exiting faulty chains while also creating mechanisms to incentivize and enforce continued correct execution of data.

As only merkleized commitments are broadcast periodically to the root blockchain (i.e. Ethereum) during non-faulty states, this can allow for incredibly scalable, low cost transactions and computation. Plasma enables persistently operating decentralized applications at high scale.

1 Scalable Multi-Party Computation

With blockchains, the solution for enforcing correctness has generally been having every participant validate the chain themselves. To accept a new block requires one to fully validate the block to ensure correctness. Many efforts to scale blockchain transactional capacity (e.g. Lightning Network[1]) requires using time commitments to build a fidelity bond, (an assert/challenge agreement) so that the asserted data must be subject to a dispute period for participants on the blockchain to enforce the state. This assert/challenge construction allows one to assert a particular state is correct, and if the value is incorrect, then a dispute period exists where another observer can provide a proof challenging that assertion before a certain agreed time. In the event of fraudulent or faulty behavior, the blockchain can then penalize the faulty actor. This creates a mechanism for participants to be encouraged to enforce if-and-only-if the incorrect state is asserted. By having this assert/challenge-proof construction, interested participants can be able to assert ground truths to non-interested participants on the root blockchain (e.g. Ethereum[2][3]).

This structure can be used not only for payments, but extended to computation itself so that the blockchain is the adjudication layer for contracts. However, the presumption would be that all parties are participants in validating the computation. In Lightning Network, for example, the construction makes it so that one can establish commitments to computing contract state (e.g. with pre-signed tree of multisignature transactions of conditional state).

These constructions allow for highly powerful computation at scale, however there are some issues which require the summation of a lot of external state (i.e. summation of entire systems/markets, computation of a large amount of shared/incomplete data, large number of contributors). This form of commitment to multiparty off-chain state ("state channels"[4]) requires participants to fully validate the computation, or else there are significant amount of trust established in the computation itself, even in single-round games. Additionally, there is usually a presumption of "rounds" whereby the execution path must be completely unrolled before contract initiation, which gives participants the opportunity to exit and force expensive computation on-chain (as it is not possible to prove which party is halting).

Instead, we seek to design a system whereby computation can occur off-blockchain but ultimately enforcible on-chain which is scalable to billions of computations per second with minimal on-chain updates. These state updates occur across an autonomous set of proof-of-stake validators who are incentivized towards correct behavior enforced by fraud proofs, which allow for computation to occur without a single actor being able to easily halt the computation service. This needs to be able to minimize issues around the data availability problem (i.e. block withholding), minimizing the state updates in the root blockchain necessary in the event of byzantine actors to prevent risk-discounted transaction fees on the root chain, and a mechanism to enforce state changes.

Similar to the Lightning Network, Plasma is a series of contracts which runs on top of an existing blockchain to ensure enforcement while ensuring that one is able to hold funds

in a contract state with net settlement/withdrawal at a later date.

2 Plasma

Plasma is a way to do scalable computation on the blockchain with the structure of creating economic incentives to autonomously and persistently operate the chain without active state transition management by the contract creator. The nodes themselves are incentivized to operate the chain.

Additionally, significant scalability is achieved by minimizing the funds represented in a spend from a contract to a single bit in a bitmap, so that one transaction and signature represents a payment coalesced with many participants. We combine this with a MapReduce[5] framework to be able to construct scalable computation enforced by bonded smart contracts.

This construction allows one to be able to have externalized parties hold funds and compute contracts on one's behalf similar to a miner, but Plasma instead runs on top of an existing blockchain so that one does not need to create transactions on the underlying chain for every state update (including adding new users' ledger entries), with minimal data on-chain for coalesced state updates.

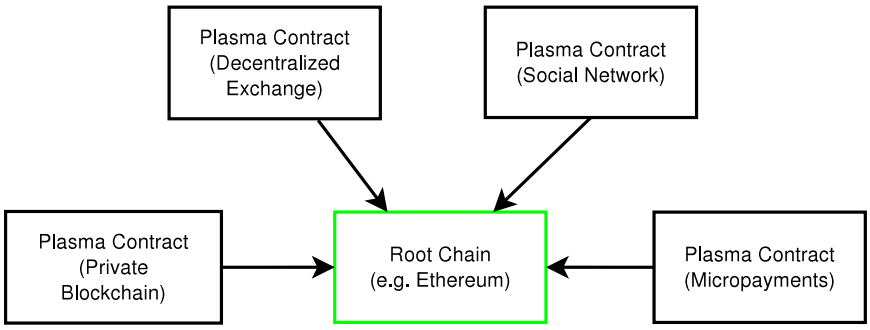

Figure 1: Anyone can create a custom Plasma chain for smart contract scalability for many different use cases. Plasma is a series of smart contracts which allows for many blockchains within a root blockchain. The root blockchain enforces the state in the Plasma chain. The root chain is the enforcer of all computation globally, but is only computed and penalized if there is proof of fraud. Many Plasma blockchains can co-exist with their own business logic and smart contract terms. In Ethereum, Plasma would be composed of EVM smart contracts running directly on Ethereum, but only processing tiny commitments which can represent an incredibly large amount of computation and financial ledger entries in non-Byzantine cases.

Plasma is composed of five key components: An incentive layer for persistently computing contracts in an economically efficient manner, structure for arranging child chains in a tree format to maximize low-cost efficiency and net-settlement of transactions, a MapReduce computing framework for constructing fraud proofs of state transitions within these nested chains to be compatible with the tree structure while reframing the state transitions to be highly scalable, a consensus mechanism which is dependent upon the root blockchain which attempts to replicate the results of the Nakamoto[6] consensus incentives, and a bitmap-UTXO commitment structure for ensuring accurate state transitions off the root

blockchain while minimizing mass-exit costs. Allowing for exits in data unavailability or other Byzantine behavior is one of the key design points in Plasma's operation.

2.1 The Plasma Blockchain, or Externalized Multiparty Channels

like raiden

We propose a method whereby multiparty off-chain channels can hold state on behalf of others. We call this framework a Plasma blockchain. For funds held in the Plasma chain, this allows for deposit and withdrawal of funds into the Plasma chain, with state transitions enforced by fraud proofs. This allows for enforcible state and fungibility since one is able to deposit and withdraw, with accounting of the Plasma block matching the funds held in the root chain (Plasma is not designed to be compatible with fractional reserve banking designs).

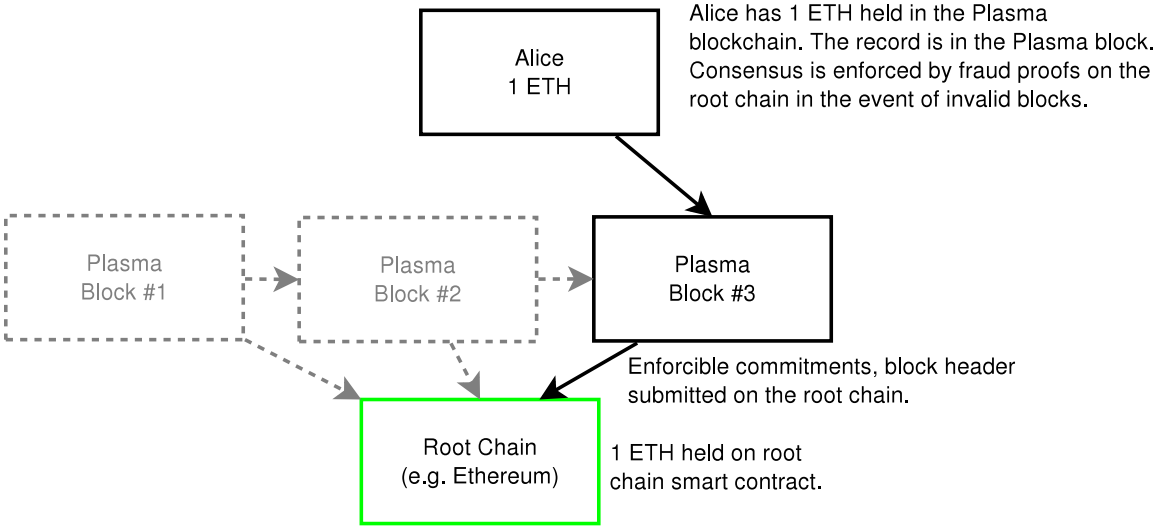

Figure 2: Plasma blockchains are a chain within a blockchain. The system is enforced by bonded fraud proofs. The Plasma blockchain does not disclose the contents of the blockchain on the root chain (e.g. Ethereum). Instead, blockheader hashes are submitted on the root chain and if there is proof of fraud submitted on the root chain, then the block is rolled back and the block creator is penalized. This is very efficient, as many state updates are represented by a single hash (plus some small associated data). This update can represent balances which are not represented on the root blockchain (Alice does not have her ledger balance on the root chain, her ledger is on the Plasma chain, and the balance in the root chain is representing a smart contract enforcing the Plasma chain itself). Gray items are old blocks, black is the most recent block that has been propagated and committed on the root chain.

Incredibly high amount of transactions can be committed on this Plasma chain with minimal data hitting the root blockchain. Any participant can transfer funds to anyone, including transfers to participants not in the existing set of participants. These transfers can pay into and withdraw (with some time delay and proofs) funds in the root blockchain's native coin(s)/token(s).

Plasma allows one (or a network of participants in a proof-of-stake network) to be able to manage a blockchain without a full persistent record of the ledger on the root blockchain

and without giving custodial trust to the 3rd party or parties. In the worst case, funds are locked up and time-value is lost with mass-exits on the blockchain.

We construct a series of fraud proofs as smart contracts[7] on the root blockchain which enforce state in this channel so that attempts at fraud or non-Byzantine behavior can be slashed.

These fraud proofs enforce an interactive protocol of fund withdrawals. Similar to the Lightning Network, when withdrawing funds, the withdrawal requires time to exit. We construct an interactive game whereby the exiting party attests to a bitmap of participants' ledger outputs arranged in an UTXO model which requests a withdrawal. Anyone on the network can submit an alternate *bonded* proof which attests whether any funds have already been spent. In the event this is incorrect, anyone on the network can attest to fraudulent behavior and slash the bonds to roll back the attestation. After sufficient time, the second *bonded* round allows for the withdrawal to occur, which is a bond on state *before* a committed timestamp. This allows for a withdrawal en masse so that a faulty Plasma chain can be rapidly exited. In coordinated mass withdrawal events, a participant may be able to exit with less than 2-bits of block space consumed on the parent blockchain (i.e. root Ethereum on-chain in worst case scenarios).

In the event of a block withholding attack, participants can rapidly and cheaply do a mass-exit, with substantial cost savings versus other previous off-chain proposals. Additionally, this does not place any trust in a coalition of validator nodes (Sidechain Functionaries, Fishermen).

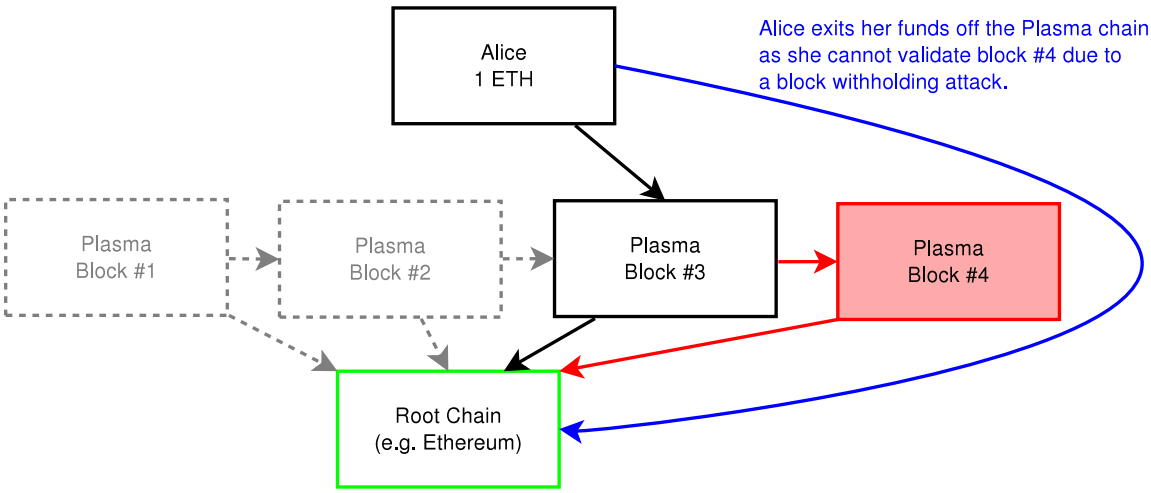

Figure 3: Exit of funds in the event of block withholding. The red block (Block #4) is a block which is withheld and committed on the root chain, but Alice is not able to retrieve Plasma block #4. She exits by broadcasting a proof of funds on the root blockchain and her withdrawal is processed after a delay to allow for disputes.

Similar to how closing out Lightning is an interactive mechanism between two participants to enable enforcible infinite payments between themselves, this allows for an interac-

tive mechanism between n participants. The primary difference is that not all participants need to be online to update state, and the participants do not need a record of entry on the root blockchain to enable their participation – one can place funds on Plasma without direct interaction on-chain, with minimal data to confirm transactions when constructing these Plasma chains in a tree format.

2.2 Enforcible Blockchains in Blockchains

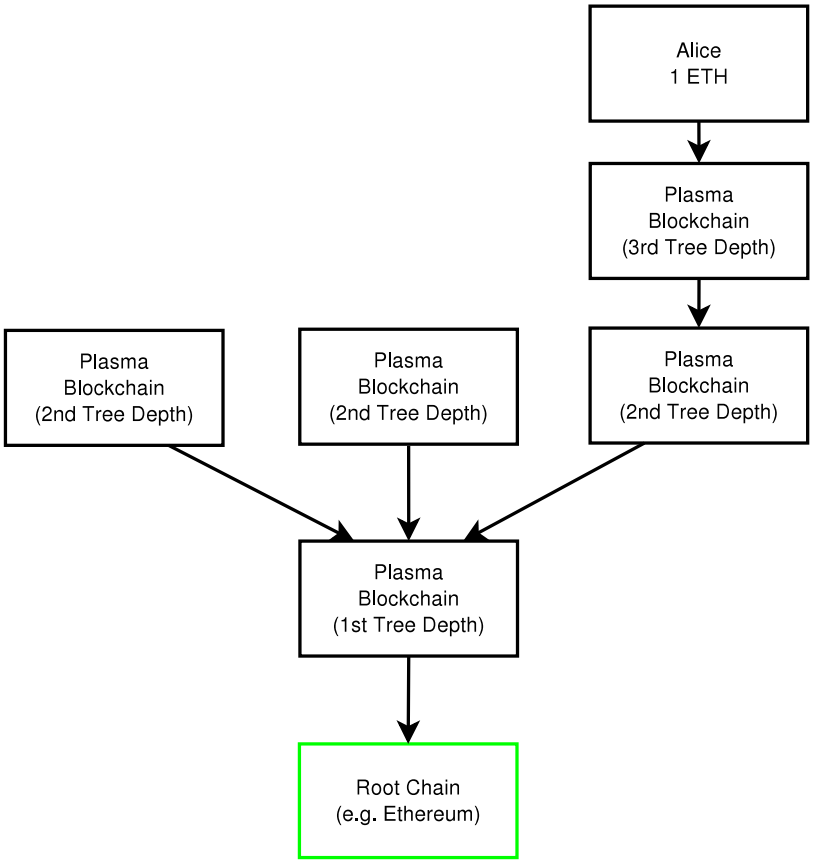

Figure 4: Plasma composes blockchains in a tree. Block commitments flow down and exits can be submitted to any parent chain, ultimately being committed to the root blockchain.

We construct a mechanism similar to the court system. If Lightning Network uses an adjudication layer for payments which is ultimately enforcible on the root blockchain, we create a system of higher and lower courts to maximize availability and minimize costs in non-Byzantine states. If a chain is Byzantine, it has the option of going to any of its parents (including the root blockchain) to continue operation or exit with the current committed state. Instead of enforcement of an incrementing nonce state (via revocations), we construct a system of fraud proofs to enforce balances and state transitions of these chain hierarchies.

In effect, we are able to create state transitions which are only periodically committed

to parent chains (which then flows to the root blockchain). This allows for incredible scale of computation and account state, as we are able to only submit raw data to parent (or root) chain in Byzantine conditions. Recovery from partially Byzantine conditions are cost-minimized since one can go to a parent Plasma chain to enforce state.

This child blockchain runs on top of a root blockchain (e.g. Ethereum) and from the root blockchain's perspective, is only seeing periodic commitments with the tokens bonded in the contract for enforcement of the Proof-of-Stake consensus rules and business logic of the blockchain.

This has significant advantages in maximizing block availability and minimizing stake for validation of one's coins. However, since not all data is being propagated to all parties (only those who wish to validate a particular state), parties are responsible for monitoring the particular chain they are interested in periodically to penalize fraud, as well as personally exiting the chain rapidly in the event of block withholding attacks.

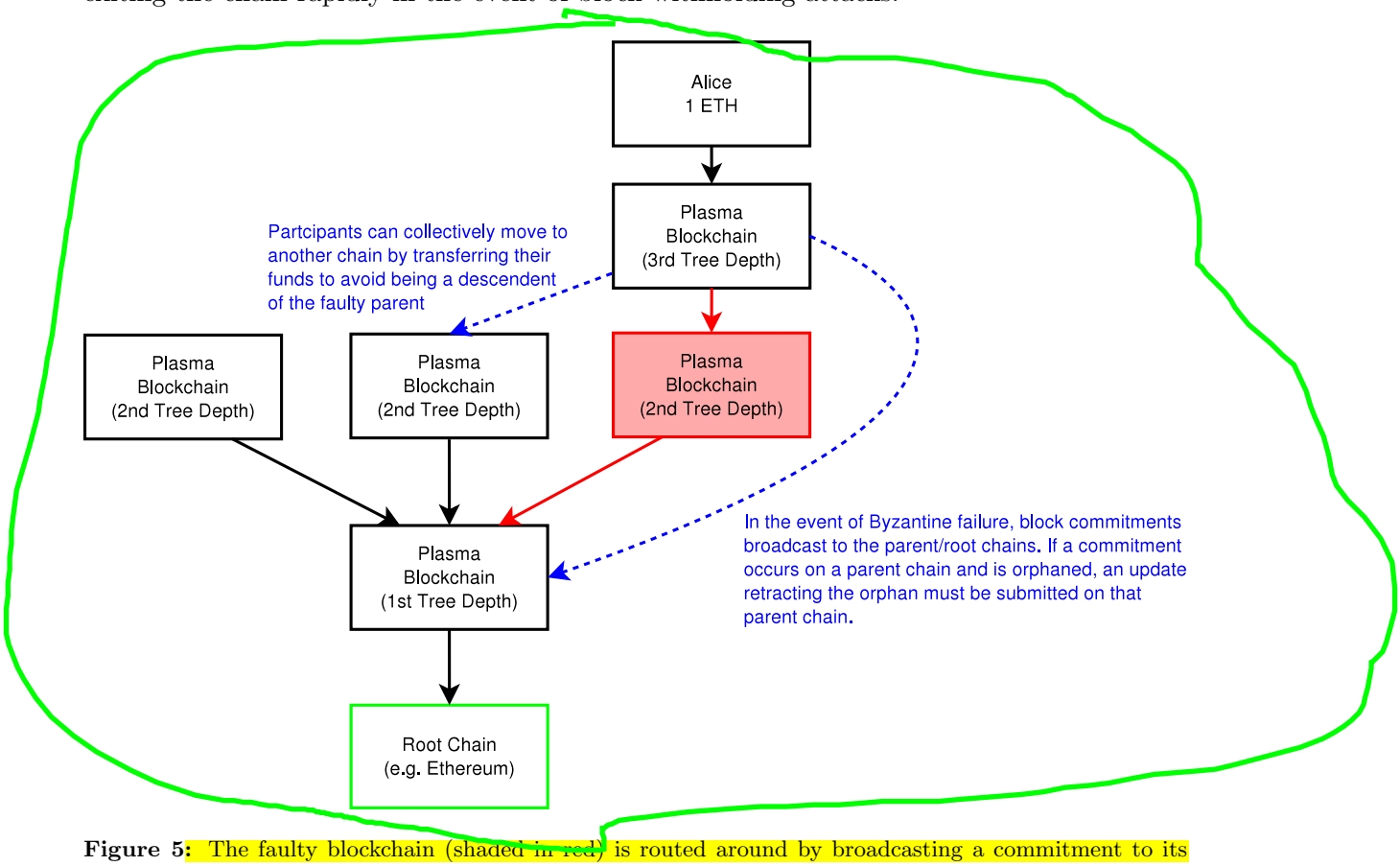

Figure 5: The faulty blockchain (shaded in red) is routed around by broadcasting a commitment to its parent Plasma/root chain(right dotted blue line). Participants in the 3rd depth Plasma chain do a mass migration to another chain together (left blue dotted line) after some period of time.

This construction, in non-Byzantine environments, coalesces the tree of blockchain states and update all child Plasma chains. An entire set of updates across all chains can be attested to in a 32-byte hash with a signature.

2.3 Plasma Proof-of-Stake

While it's fairly interesting to be able to hold fund on behalf of others with a single validator, we propose a method whereby a single party can enforce state with a set of validators, often in a proof-of-stake framework requiring either ETH bonding, or bonding in a token (e.g. ERC-20).

The consensus mechanism for this proof of stake system, is again, enforced in an on-blockchain smart contract.

We attempt to replicate the incentives around the Nakamoto Consensus, but using Proof-of-Stake bonds. We believe that one of the more useful incentive mechanisms constructed as a result of the Nakamoto mechanism is that there is incredible incentive to minimize block withholding attacks. This is since leaders are only probabilistically elected. Leaders are probabilistically known over time (in the original implementation it was 6 confirmations). When one finds a block, one is fairly sure they are likely the leader, but they are not yet certain if they are the leader. To ensure that they are the leader, they propagate their blocks to all participants on the network to maximize their odds. We believe this is a significant if not the key contribution with the Nakamoto mechanism and attempt to replicate this incentive.

Proof-of-stake coalitions face this issue since it's possible if one does straight leader election, block withholding attacks by majority cartels (also generalized as the "data availability problem") become magnified.

We can mitigate this in Plasma Proof-of-Stake by allowing stakeholders to publish on the root blockchain or parent Plasma chain which contains a committed hash of their new block. Validators only build upon blocks which they have fully validated. They can build upon blocks in parallel (to encourage maximum information sharing). We create incentives for validators to represent the past 100 blocks to match the current staker ratio (i.e. if one stakes 3 percent of the coins, they should be 3 percent of the past 100 blocks), by rewarding more transaction fees to be paid out to accurate representation. Excess fees (due to suboptimal behavior by stakers) goes to a pool to pay out fees in the future. A commitment exists in every block which includes data from the past 100 blocks (with a nonce). The correct chain tip is the chain with summed weight of the highest fees. After a period of time, the blocks are finalized.

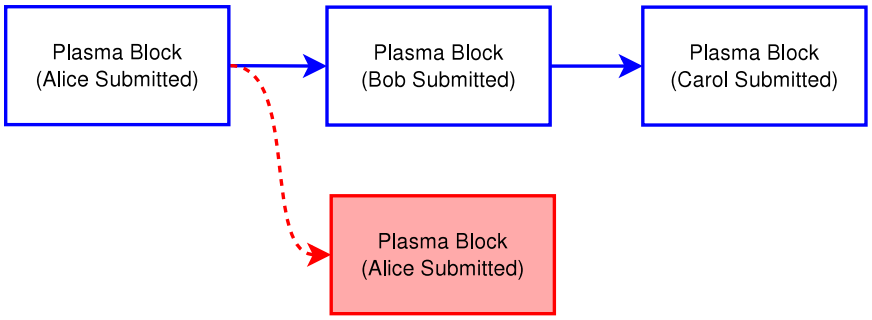

Figure 6: Presume that Alice, Bob, and Carol are 3 validators with equal amount of weight. They collectively are incentivized to build a round-robin structure for maximum returns. These commitments are submitted to the parent/root chain. Chain tip is contingent on maximum weight score by correct distribution of blocks over n periods (blue is the current candidate chain tip, red is an orphan). Suboptimal chain tips has any excess fees go into a pool for future validators with correctness above some threshold (e.g. 90%). After some n periods it's presumed that the blue chain tip is finalized.

This encourages participants to participate and replicates the 51% attack assumptions in the Nakamoto consensus. In the event a chain is attacked via block withholding or other Byzantine behavior, the non-Byzantine participants conduct a mass compact withdrawal on the parent/root blockchain. If bonds for the highest parent Plasma chain are in the form of tokens, it is likely the value of the token will significantly devalue as a result of the mass exit.

2.4 Blockchains as MapReduce

blockchain : git :: Plasma : Hadoop (MapReduce)

By constructing computation in a MapReduce format, it is also easy to design computation and state transitions in a hierarchical tree.

MapReduce gives a framework for high scale computation across thousands of nodes. The blockchain faces similar issues in meeting computational scale, but has additional requirements in generating proofs of computation.

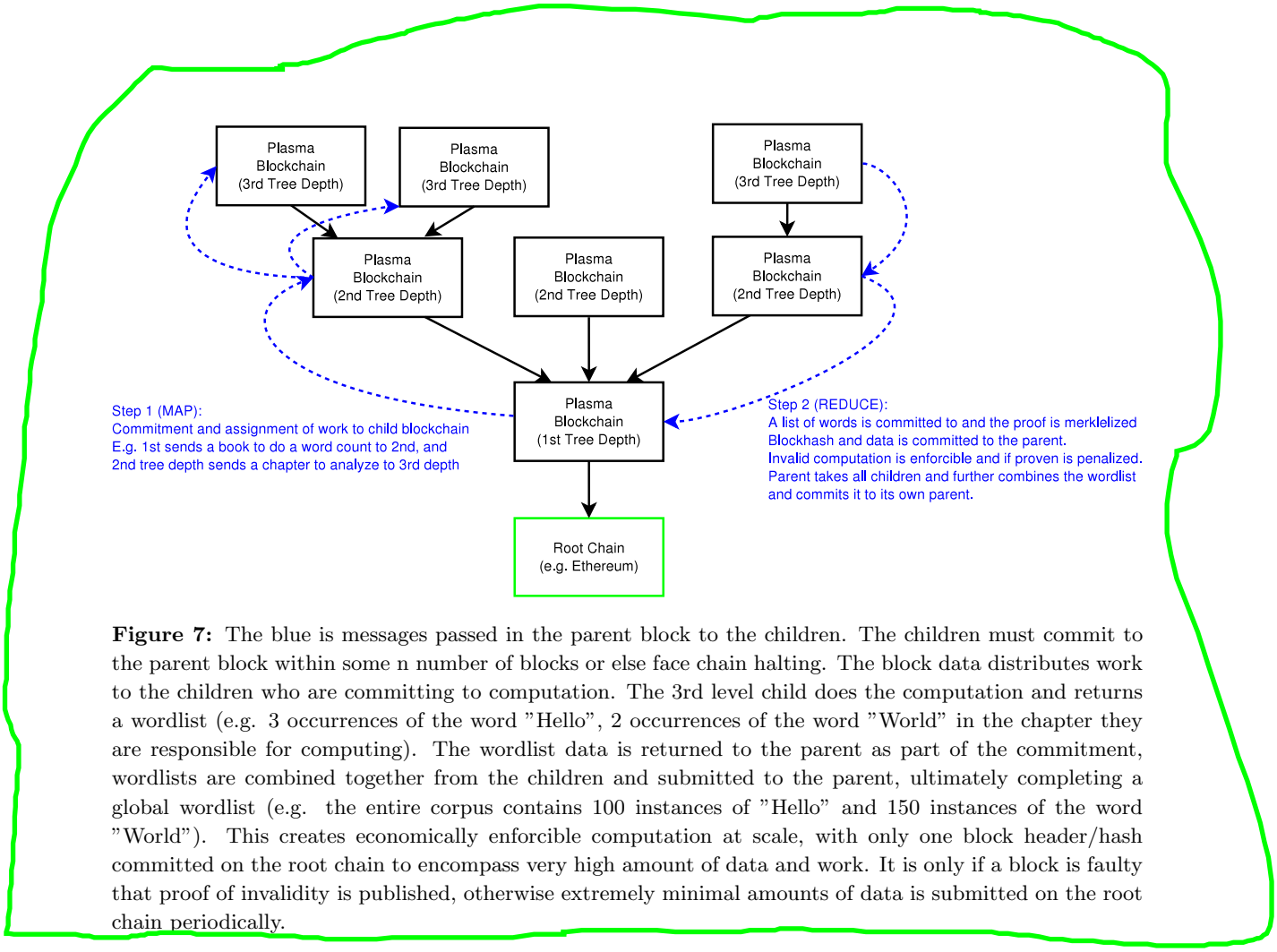

Figure 7: The blue is messages passed in the parent block to the children. The children must commit to the parent block within some n number of blocks or else face chain halting. The block data distributes work to the children who are committing to computation. The 3rd level child does the computation and returns a wordlist (e.g. 3 occurrences of the word "Hello", 2 occurrences of the word "World" in the chapter they are responsible for computing). The wordlist data is returned to the parent as part of the commitment, wordlists are combined together from the children and submitted to the parent, ultimately completing a global wordlist (e.g. the entire corpus contains 100 instances of "Hello" and 150 instances of the word "World"). This creates economically enforcible computation at scale, with only one block header/hash committed on the root chain to encompass very high amount of data and work. It is only if a block is faulty that proof of invalidity is published, otherwise extremely minimal amounts of data is submitted on the root chain periodically.

We propose a method whereby the map phase includes commitments on data to computation as input, and in the reduce step includes a merkleized proof of state transition when returning the result. The merkleized state transition is enforced via fraud proofs constructed on the root blockchain. It is also possible to construct a zk-SNARKs proof of state transitions. For some computational constructions, a bitmap on state transitions may also be necessary in the reduce step (therefore more than one bit can be used per UTXO/account for these use cases).

Our construction enables incredible high-scale computation, with time or speed tradeoffs. These tradeoffs produce a network where nodes assert computation and participants are responsible for verifying them. This does not produce a system whereby one can completely outsource computation without trust, it enables the ability to compress computation into bonded proofs. These bonded proofs encourage participants to only attest to things honestly. This again, follows the narrative in the Lightning Network, whereby if a tree falls in the forest and nobody listens to it, it presumes that it doesn't matter whether it makes a sound or not. Similarly, if no one is watching/enforcing the computation, it's presumed to be correct, or it simply doesn't matter what the result may be. Computation can be watched by any participant in open networks, but participants who hold balances and/or

require correct computation will periodically watch the chain to ensure correctness. The scaling benefit comes from removing the requirement to watch the chains one is not economically impacted by, one should watch the chains where one wishes to enforce correct behavior. Behavior on other Plasma chains can be netted together as part of the reduce step so that the computation which affects one is expressed in a minimal state. For example, for a decentralized exchange, one doesn't care about which counterparties put in what order, they only need to see one coalesced orderbook, so one only needs to observe all other chains as a single counterparty, whereas one's own chain is fully validated to enforce transactions and order fills to the correct person (including oneself). Another example is one can construct a BBS on a tree of Plasma chains, and one doesn't need to receive updates on the topics one doesn't care about.

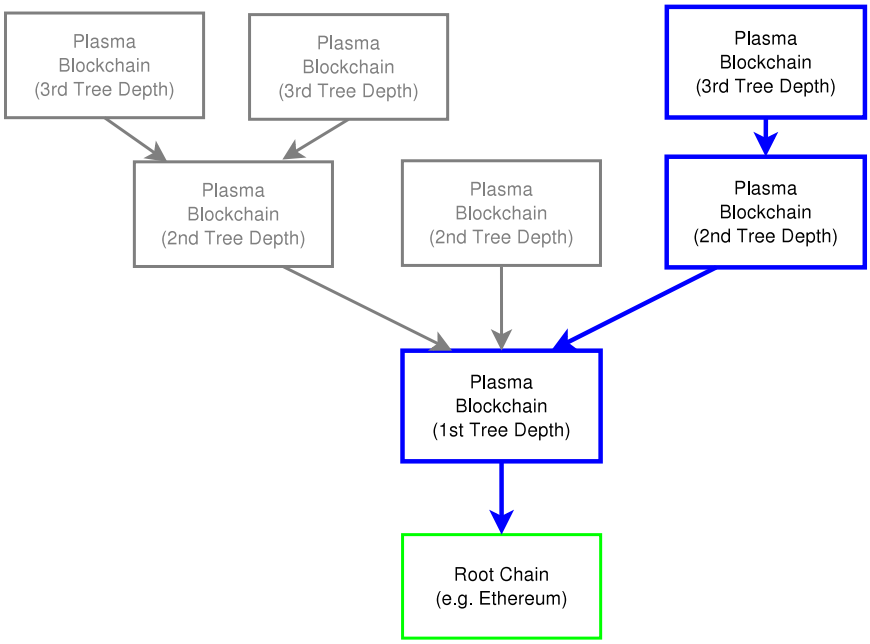

Figure 8: One only needs to watch the data which one wants to enforce. If economic activity or computation occurs on other Plasma chains which is not necessary to enforce (gray), one can treat all other chains as a single counterparty. E.g. in a Plasma decentralized exchange, one just needs to watch the chains which affects one's commitments (bolded blue).

2.5 A Description of Economic Incentives around Persistent Decentralized Autonomous Blockchains

We propose a structure whereby one can create economic incentives to persistently keep a child blockchain running. For state which does not require significant complexity or reliance in state transitions, the native token (e.g. ETH for Ethereum) can be used for bonding of state. However, for complex contracts, there may be significant incentives to continue operating the chain due to incentives around ensuring liveness and order fairness of the system[8][9].

Every Plasma chain is represented by a set of contracts. These contracts enforce the consensus rules of the chain, and fraud causes significant penalties to be applied if the fraud proof can be produced.

However, to incentivize avoidance of Byzantine states, especially around correctness and liveness, it may be ideal to create a token per contract. This token represents the network effects in operating the contract, and creates an incentive to maximize security of this contract. As the Plasma chain requires the token to secure the network in a Proof-of-Stake structure, stakers are disincentivized against Byzantine behaviors or faults as that would cause a loss in value of the token. The role of the token is to ensure there is costs localized to the validators if they act faulty via value declines in the token.

With simple contracts and business logic such as a basic contract account holding funds on behalf of its users, an Ethereum bond can represent a stake in the Plasma chain.

The stakes who put up bonds (whether it be a token or ETH) have incentives to continue operating the network as they receive transaction fees for operating the network. These transaction fees are then paid out to the stakers of the network which encourages non-Byzantine behavior, and creates long-term value for the token.

Since the stakers have an incentive to continue operating this network to collect transaction fees, they will persistently run the chain and are bound by the fraud proofs defined in the contracts in the root blockchain.

3 Design Stack and Smart Contracts

Historically, many people believe that the blockchain is best applied towards transactional payments as a gross settlement system. However, it's understood that a gross settlement system has difficulty scaling. Net settled designs such as the Lightning Network, a payment channel network, changes the structure to allow for nearly unlimited payments between participants. Transactional capacity is increased dramatically as channels are net-settled on the blockchain. Payments can be routed across a network of these channels.

This structure additionally allows for effectively instantaneous payments. This is instrumental for not only payments which require a high degree of time sensitivity, but also for contracts as well.

Plasma is not designed to reach assured finality rapidly, even though transactions are confirmed in the child chains rapidly, it requires it to be finalized on the underlying root blockchain. Channels are necessary to be able to have rapid local finality of payments and contracts (enforcible on-chain).

In smart contracts, there is an issue of the "free option problem" whereby the receiver (second or last signer) of a smart contract offer is needed to sign and broadcast the contract in order to enforce it – during that time the receiver of the contract may treat it as a free option and refuse to sign the contract if the activity does not interest them. This is exacerbated as smart contracts are most effective when dealing with counterparties who are untrusted (as that creates minimization in counterparty risk and thereby information

costs).

Plasma does not solve this problem on its own, as there are no guarantees of atomicity with the first and second signature step for interactive protocols in blockchains.

With Lightning (including Lightning on top of Plasma), it's possible to do incredibly rapid updates with reasonable sense of localized finality. Instead of having a single payment which gives optionality to the last party, a payment can instead be split into many small payments. This minimizes the free option to the amount per split fraction. Since the second party of the smart contract only has the free option on the amount in the split fraction, the value of the free option is minimized.

Within the above use cases, it's possible that Lightning may be a primary interface layer for rapid financial payments/contracts on top of Plasma, as Plasma allows for ledger updates with minimal root chain state commitments.

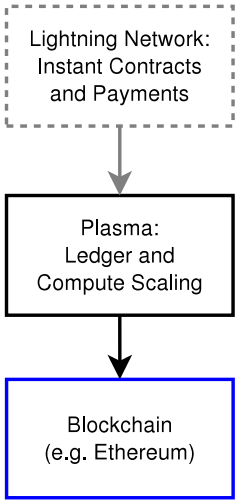

Figure 9: At the root is the blockchain, which is the adjudication layer for contracts and payments. The contracts themselves are located on the root blockchain. The Plasma chain contains the current ledger state which can be settled and redeemed on the root blockchain. Fraud proofs exist to allow for funds to be redeemed. Plasma represents a nested set of Plasma chains to create venues to withdraw funds in a scalable way with minimal blockchain transactions. On top is the Lightning Network, which allows for instantaneous payments across Plasma and Block Chains.

3.1 The Most Significant Problem in Sharding is Information

With sharded data sets, there is a significant risk of individual shards to refuse to disclose information. It would thereby be impossible to produce fraud proofs.

We attempt to resolve this using 3 strategies:

1. A new Proof-of-Stake mechanism which encourages block propagation. The underlying mechanism does not entirely rely upon correct functionality of incentives. However, this should significantly decrease faulty behavior.

2. Significant withdrawal delays which allow for accurate withdrawal proofs. Individuals don't need to watch the blockchain that frequently and fraud on higher plasma chains can be prevented on the root blockchain by any honest actor on the same plasma chain as the user. In the event of block withholding, plasma chains can immediately lock up funds via a proof, preventing an attacker from submitting fraudulent withdrawal proofs. In the event the attacker attempts to withdraw funds above their limit and more funds are locked, the attacking plasma chain loses their deposit.

3. Creating child chains whereby transactions can be propagated in any parent chain. For this reason, participants on networks will desire to submit transactions to deep child chains. This creates economic efficiency for smaller balances which do not have the economic ability to pay high transaction fees on the root blockchain and therefore moving funds can be achieved with many small balances. People are therefore encouraged to create deeply nested child chains which represent significant value. Note that there is some presumption about reputation around chain selection for individuals holding very small balances which cannot be on the root blockchain transaction fees, however is mitigated by having deeply nested chains. This security model is the key novelty of plasma chains.

4 Related Work

Some related projects propose a merkle tree with reduction steps as proof of computation, however this proposal is primarily around data availability and encouraging cost minimization around fraud proofs, with a protocol to manage these via an economically incentivized persistent sharded group of chains.

Other related projects propose a system of child blockchains, but have significant differences in approach.

Plasma uses a merkleized proof to enforce child chains.

4.1 TrueBit

Plasma shares a great degree of similarity in the reliance of fraud proofs as TrueBit[10]. The fraud proof construction is similar to TrueBit, and nearly all the work by TrueBit can directly apply to Plasma, especially the work around merkleized proofs of state transitions.

TrueBit design allows for compact proofs to be created to submit to the Ethereum blockchain, which is necessary for Plasma, so nearly all of the heavy lifting done by the TrueBit paper and team is directly applicable in this design. The use of the Verification Game, which generates merklized proofs provides increased benefits with reducing the scale of computation. Similar assumptions as TrueBit applies, namely that computational state must be computable and broadcastable online (large pieces of data must be split in multiple rounds), data availability problems needs to be mitigated, failure must be disclosed. We attempt to mitigate these problems, especially the latter two.

The primary aspect which Plasma attempts to build upon TrueBit is the notion of multiparty participants which need to compute on shared state. For example, a set of participants only care about a subset of the data and computation and needs to only compute aspects related to themselves (e.g. BBS or exchange). We also attempt to mitigate the issue of enforcement of computational rounds via off-chain venues of enforcement.

4.2 Blockchain Sharding

Current work on blockchain sharding[11] uses similar techniques and goals, e.g. Ethereum Sharding proposal. This construction may be compatible as a higher layer. If the root blockchain is sharded, then the Plasma chain can run on top of this for greater scalability and other benefits. This can also be a testbed for different sharding techniques as there are no consensus changes necessary in Ethereum and other rich stateful blockchains to begin basic operation.

4.3 Federated Sidechains

Plasma is not a federated sidechain[12] as it does not rely upon the federation for honest activity, nor does it rely entirely upon trusted actors to enforce state inside the chain. Plasma also externalizes ledger state to another blockchain allowing the use of the same coin/token, however it does enforcible verification if the fraud proof is available. Plasma does not rely upon a strong federation of actors, which requires significant underwriting risk on the correctness of these actors, for this reason it is not a Federated-Pegged Sidechain.

Drivechains[13] shares similarity to federated sidechains, except the validators are an unknown, changing set of participants (miners), with greater decentralization.

4.4 Merge-Mined Blockchain

Examples include Namecoin, which create concurrent blocks with the parent blockchain[14]. This presumes full validation of the blockchain, thus does not provide scalability benefits. Extension blocks are an example of a merge-mined chain which allows for funds to move between the primary blockchain and the merge-mined chain (with an enforcement mechanism of the full set of miners as a consensus rule on the root chain). Merge-mined chains permit for new consensus rules and election of users to validate only the chains they care about, but the miners/validators must validate everything. The goal of Plasma is to ensure that only users and miners need to validate the chains relevant to themselves.

4.5 Treechains

Treechains[15] proposes tree structures blockchains which are validated in the child blocks using Proof of Work. The root chain has the summed proof of work of all child blockchains. Lower down the stack has higher security, but higher along the stack may or may not depending on the level of validation and work. While treechain's topology is in a tree

structure, its structure is dependent upon mining security being summed via the branches. The security model has lower security on the leaves, as it is secured by Proof-of-Work. Plasma is the reverse with the mining done with full security only on the root, with the security and proofs flowing from the root. Similar work is in building proofs of blocks seen in a tree formation.

4.6 zk-SNARKs and zk-STARKs

Non-interactive proofs of computation allow for one to have significant benefits in scalable computation[16]. zk-SNARKs/STARKs and other forms of non-interactive compact proofs is complimentary to Plasma. A proof can be provided along with the result of the merklized computation. Additionally, benefits exist to reduce systemic attacks when holding small balances on a child plasma chain. There has already been research in SNARKs for MapReduce functionality[17] and we hope this leverages on that research and Plasma extends it by making the proofs orderable and enforcible within a set of blockchains.

Further benefits include proofs of computation which allow for faster syncing and verifying of chains themselves. Note that zk-SNARKs does not solve the issue around data availability, just reduces the amount of data requirements and computation. This is especially useful as a replacement or complement for any assert/challenge time-based mechanisms. zk-SNARKs can be helpful as defense in depth. If the last line of defense is using the blockchain without fancy cryptography, the second line of defense can be zk-SNARKs, and the first line of defense is trusted computing hardware.

Withdrawals from Plasma chains could be secured by zk-SNARKs which gives the benefit of optionally not requiring the bitmap, which may allow for very small balances to be transferred.

4.7 Cosmos/Tendermint

Cosmos[18] arranges blockchains in a Cosmos "Hub" and has child blockchains "Zones" validated over a proof of stake system. Significant similarity with the construction of child blockchains exist, however Plasma is reliant upon construction fraud proofs to enforce state in child chains and is genericized to be applicable to many chains. The proof of stake construction for Cosmos presumes a 2/3 honest majority of validators, including validators of its Cosmos Zone.

4.8 Polkadot

Polkadot[19] also constructs a structure for a hierarchy of blockchains. There is some similarity with the design of Polkadot. Instead of a structure with "fishermen" validators ensuring block accuracy, we construct a series of child blockchains which enforce state via merkle proofs. The Polkadot construction is reliant on the child blockchains ("parachains") state and information availability being enforced by the fishermen.

4.9 Lumino

Lumino[20] is a design for an EVM contract with compressed updates on the blockchain. This allows participants to only update minimal committed state. Plasma's output management design takes things further with only a single bit denoting a particular output. This allows for rapid, low cost coordination around mass withdrawal in the event of child Plasma chain failure.

5 Multiparty Off-Chain State

The goal is to construct a method whereby participants can hold funds in the native underlying coin/token of the blockchain, without significant on-chain state. Plasma begins to blur the line between on-chain and off-chain (e.g. are shards on-chain or off-chain?).

There are two common issues in efforts to establish off-blockchain multiparty channels. The first is the need to do synchronized state update amongst all participants when there needs to be an update on the system (or otherwise make tradeoffs on availability of global state updates) and therefore must be online. The second is that adding and removing participants in the channel require a large on-blockchain update, enumerating all participants which are added and removed.

It would instead be preferable to construct a mechanism whereby many participants can be added and removed without significant root chain state updates and internal state updates are possible without all parties participating, they only need to participate if their balances are being adjusted or if Byzantine behavior is detected.

The general construction is a child blockchain which allows for holding balances represented in a smart contract on the root blockchain (e.g. Ethereum). The balances of the smart contract are represented and allocated to the balances of finalized blocks in the child Plasma blockchain. This allows one to hold the native coin in the child blockchain with full representation of balances on the root blockchain, allowing for withdrawals after a dispute mediation period.

In order to achieve this, we construct a UTXO (Unspent Transaction Output) model for the ledger. While this is not an explicit requirement, it is easier to reason about with rapid withdrawals. The rationale for a UTXO model is that it is easy to compactly represent whether a particular state has been spent or not. This can be represented within a trie for merkleized proofs, and as a bitmap for a compact representation parsable by others. In other words, the smart contracts are held in accounts on the root chain, but the Plasma chain maintains a UTXO set of balances for the allocation of the balances held in the root chain account. For child chains which do not have significant requirements around state transitions, it is possible to use an account model for more complex or frequent state transitions, however there is more reliance on block space availability in the parent blockchain(s).

For now, one can presume a single block leader which selects a block of the child Plasma

chain. It is possible to construct this as a Proof of Stake set or a named preset n-of-m validators, however in these examples, we are using a single named validator for simplicity. The role of the validator is to propose blocks which serve a role of ordering transactions. The validator/proposer is restricted by the fraud proofs constructed in the root blockchain contract. If they propagate a block with an invalid state transition, any other participant who receives the block can submit a merkleized fraud proof on the parent blockchain and the invalid block is rolled back with a slashed penalty.

The blocks are propagated to the participants who wish to observe the blocks, including participants who hold balances or want to observe/enforce computing on the individual Plasma chain.

While there is minimal complexity in maintaining deposits of off-chain state, state transitions and withdrawals create greater complexity.

5.1 Fraud Proofs

All states within this child blockchain is enforced via fraud proofs which allows for any party to enforce invalid blocks, presuming block data availability.

However, the greatest difficulty in this construction is that there are no explicit guarantees around data/block availability.

At the root blockchain (e.g. Ethereum), there are a collection of fraud proofs which ensure that all state transitions are valid when block data is available. For complex computation, the state transitions must be merkleized for effective verification.

Additionally, state transitions can also be enforced via zk-SNARKs/STARKs which ensure that improper exits are not possible. A zk-SNARKs constructions may need recursive SNARKs for maximum efficacy, and therefore may require further research on the possibilities. However, the system is designed to work without SNARKs.

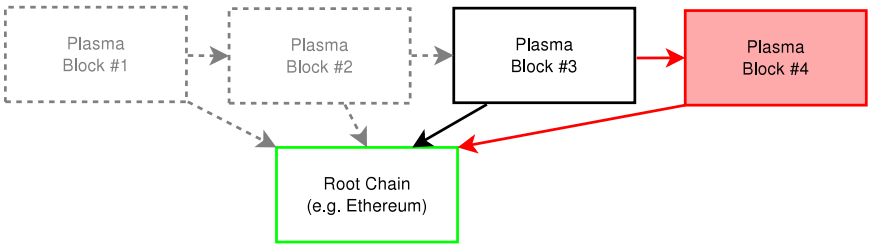

Figure 10: Everyone has block data for blocks 1-4. Block 4's committed state transition is provably fraudulent via merkleized commitments in Block 4 and data from the prior block.

The fraud proofs ensure that all state transitions are validated. Example fraud proofs are proof of transaction spendability (funds are available in the current UTXO), proof of state transition (including checking the signature for the ability an output can be spent, proof of inclusion/exclusion across blocks, and deposit/withdrawal proofs. Some more complex proofs require an interactive game. The general construction would be to take a functional

approach towards block verification. If one programmed this consensus mechanism in Solidity, there would be an additional input per function of merkle proof of block being verified and the output would return whether the verification is valid. One then simply replicates the consensus verification code to process it in compact merkleized proof form (so that one does not need to process the entire block to generate fraud proofs).

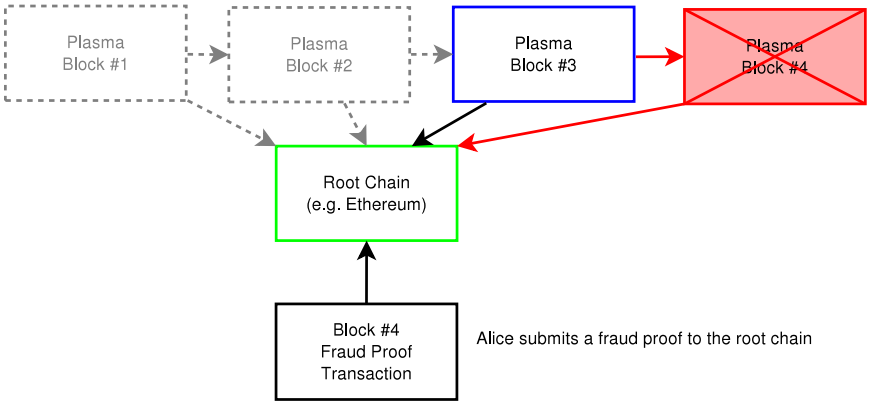

Figure 11: Alice has a copy of all block data, so she submits a proof of fraud on the root chain. Block 4 gets invalidated and rolled back. The submitter of Block 4 gets penalized by losing a bond held in the smart contract. The current block is now Block 3 (blue). After some set amount of time blocks are finalized and no fraud proofs can be submitted. One should only be building upon blocks which will not be proven fraudulent by fully validating blocks.

In order for this construction to have minimal proofs, though, all blocks must provide a commitment to a merkleized trie of the current state, a trie of outputs spent, a merkle tree of transactions, and a reference to the prior state being modified.

The fraud proofs ensure that a coalition of participants are not able to create fraudulent blocks without getting penalized. In the event a fraudulent block is detected and proven on the root blockchain (or parent Plasma chains), the invalid block is rolled back. This encourages individual participants to have incentives against Byzantine behavior, which solves the state transition vulnerability in federated-peg Bitcoin Sidechains.

The result is highly-scalable state transitions are capable in the Plasma blockchain while ensuring that observers who have access to block data are able to prove (and therefore discourage) invalid state transition. In other words, payments can occur in this chain with only periodic commitments on the root chain.

5.2 Deposits

Deposits from the root chain are sent directly into the master contract. The contract(s) are responsible for tracking current state commitments, penalization of invalid commitments using fraud proofs, and processing withdrawals. As the child Plasma blockchain is a full validator of the root blockchain, the incoming transaction must be processed using a two-phase lock-in.

Deposits must include the destination chain blockhash to specify the destination child chain and is achieved using a multi-step process to ensure coins are not unrecoverable.

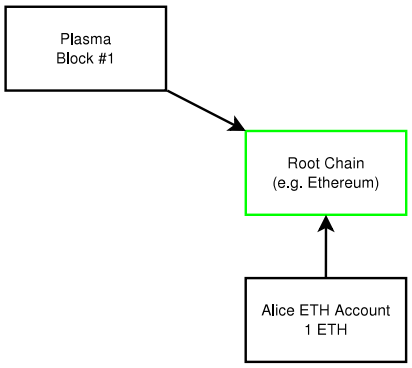

Figure 12: Alice has an ETH account with 1 ETH. She wants to send it into the Plasma blockchain. She sends it into the Plasma contract.

1. The coins/tokens (e.g. ETH or ERC-20 token) are sent into the Plasma contract on the root blockchain. The coins are recoverable within some set time period for a challenge/response.

2. The Plasma blockchain includes an incoming transaction proof. At this point, the Plasma blockchain is committing to the fact that the transaction is incoming and will be spendable in the event of either a lock-in transaction or spend initiated by the depositor. When this is included, the blockchain is committing to the fact that it will honor a withdrawal request. However, there is no confirmation yet that the depositor has sufficient information to generate a fraud proof, so there is not yet a commitment from the depositor. This block includes the addition in the state tree, bitmap, and transaction tree, so that there is a compact proof of correct inclusion.

3. Depositor signs a transaction on the child Plasma blockchain, activating the transaction, which includes a commitment that they have seen the block with the chain's commitment in Phase 2. The role of this phase is the depositor is attesting to the fact that they have sufficient information to withdraw funds.

After this process, the chain has committed to the fact that they will handle these coins and gave allocation so that withdrawals can be compactly proven. With the third phase, the user is attesting to the fact that they can withdraw.

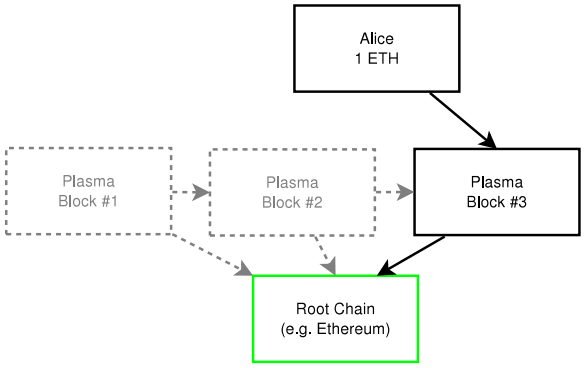

Figure 13: Alice now has 1 ETH in the Plasma block. She has committed that she has seen the funds and is now locked in. The funds are held in a smart contract on the root chain, but the ledger record is in this particular Plasma blockchain (hence state transitions, i.e. sending funds to others or smart contracts) can occur without significant root blockchain expense.

In the event that the depositor has not gone through with Phase 3, then the depositor can attempt a withdrawal on the root blockchain. The depositor submits an unconfirmed withdrawal request, and must wait some extra long time period for anyone on the network to produce a fraud proof that the depositor has signed off and locked in the funds in the Plasma blockchain. If there is no proof, then the depositor may withdraw their unconfirmed funds. This withdrawal requires a sizable root-chain bond to ensure non-Byzantine behavior.

5.3 Mass Withdrawals and Bitmapped State

The primary concern around this system is around the inability to verify state.

To be able to conduct maximum compaction of state transaction, outputs may be optionally represented in a bitmap. This is necessary for withdrawal proofs which may be too expensive to conduct on the root chain. The goal of this construction allows for holding small balances on the Plasma chain. These balances are held in full reserve on the contract on the root blockchain, but the full ledger is not on the blockchain. The primary attack which needs to be mitigated is withheld invalid blocks (with commitments to the root chain). In the event the system observes invalid state transitions, the participants conduct a mass exit of the transactions.

With a bitmap construction, a withdrawal includes a bitmap of signed transactions which wish to exit. A game/protocol is constructed which is enforced by a smart contract to ensure correct information. The bitmap ensures that everyone is able to reason about what outputs are being spent.

As this is a bitmap, it necessitates that the state be represented in an unspent transaction output data structure (UTXO) for maximum efficacy of small-value balances. Spentness can be compactly proven, and a large set of state transitions can be cleanly enforced. After a predefined settlement time period, the bits can be reused.

There is a gradient for expensive with high assurance, to cheap with low assurance:

1. Ledger state on root blockchain
2. Ledger state on Plasma, economically viable to enforce a single transaction on-chain
3. Ledger state on Plasma, economically viable to enforce using bitmap (1-2 bit cost)
4. Ledger state on Plasma, not economically viable to enforce using bitmap on root blockchain. The 1-2 bit mass withdrawal cost too high.

For those holding balances which can be enforced on the root chain, the requirement to conduct a UTXO bitmapped format is not necessary. However, for those holding balances and enforcement is only viable if the 1-2 bit transaction gas/fee on the root blockchain is sufficiently low.

For the fourth type (1-2 bit on-chain cost too high for mass withdrawals), the system is still designed to be resilient (albeit with some assumption that named entities will be reliable). Later sections in this paper describe a hierarchical blockchain structure to create many venues where one can economically conduct a mass withdrawal. Additionally, if the total value of transactions in the fourth category is significantly below the token value, then it can be game theoretically too expensive to attack those balances as the tokenholders will suffer reputational damage.

5.4 State Transitions

By default, state transitions in the Plasma chain run in a similar multi-phase process as the deposit. This is to ensure that users have information available to provide state transitions. However, unlike with the deposit construction, once a transaction is signed and included in a block, there is a commitment to participate. For this reason, state transitions should include a signature, state updates (e.g. destination, amount, token, and any other associated state data), as well as some kind of TTL for expiration and a commitment to a particular block. This TTL, while not required, should be below the time to construct exit proofs to ensure that adversarial exit conditions are known. Pre-signed transactions, of course, should not contain a TTL. Weak liveness assumptions are presumed with this construction, as there are already liveness presumptions with withdrawals with regards to deep reorgs. The commitment to a block is a commitment by the spender that the entity broadcasting the transaction in the Plasma chain has observed the chain up to that point and is able to enforce proofs and must be after the block in which the output being spent has occurred.

The multi-phase commitment occurs as follows for fast finalization:

1. Alice wants to spend her output in the Plasma chain to Bob in the same Plasma chain (without the full transaction record being submitted on the blockchain). She creates a transaction which spends one of her outputs in the Plasma chain, signs it, and broadcasts the transaction.

2. The transaction is included in a block by validator(s) of the Plasma chain. The header is included as part of a block in the parent Plasma chain or root blockchain, ultimately being committed and sealed in the root blockchain

3. Alice and Bob observes the transaction and signs an acknowledgement that he has seen the transaction and block. This acknowledgement gets signed and included in another Plasma block.

For slow finalization, only the first step needs to occur.

After the acknowledgement occurs, the transaction is considered finalized. The reason the third step exists is to ensure that block availability is ensured with the participants (Alice and Bob). This third step is not required, but there will be significant delays in finality. The rationale is that a transaction should not be viewed as finalized until the block validity and information availability can be proven by all parties relevant to the transaction.

In the event blocks are withheld after step 1, Alice is unclear whether her transaction has been spent. If a transaction has been included in a block (whether it is withheld or not), it is treated as unconfirmed if step 3 has not completed. Therefore, Alice is able to still do a withdrawal of those funds if she has not signed off on the commit provided her withdrawal message on the root/parent blockchain occurs before the block is finalized. Alice cannot withdraw funds after block finalization, and blocks are presumed to be sent to Bob. If blocks are being withheld before finalization (between step 1 and 2) and Alice and/or Bob observes this, then Alice may withdraw her unfinalized funds. If blocks are being withheld after step 2 but before step 3, then it is presumed that Bob has sufficient information to withdraw funds, but since neither Alice nor Bob have fully committed to the payment, then it is not treated as complete, depending on information availability either party may theoretically be able to claim the funds. If both parties sign off on Step 3, then it is presumed that it is truly finalized. Pay-to-contract-hash[21] enforcement occurs after this step has been completed, specifically when the signatures are provably observed on-chain. In the event one party refuses to sign or blocks are being withheld, it is conditional upon a redemption proof. As all states are eventually committed to the chain via merkle proofs, there is less of a reliance on pay-to-contract-hash as payments are provable and enforcible after finalization.

Note that Step 3 can be conditional upon a smart contract instead of a signature by both parties, i.e. state is conditional upon an HTLC release of a preimage. This allows for mutli-chain or multi-transaction atomicity. Contract creation complexity may be increased, and writing higher level languages/tooling around this may be needed if these features are desired.

5.5 Periodic Commitments to the Root Chain

The Plasma chain must be able to create ordering of the blockchain. In a Plasma chain, there is ordering within blocks, but the blocks are not attested to and ordered themselves

on its own. As a result, it's necessary to create a commitment on the root blockchain. The Plasma chain publishes its block header on the root chain and its header is enforced by the fraud proofs. In the event a fraudulent header is published with data availability for others', any other participant can publish a fraud proof and the commitment and block is rolled back, with penalties to the publisher.

These commitments allow for true ordering without equivocation later in time. If equivocation is attempted, then there is sufficient proof of fraud and can be penalized. Blocks after a certain time become finalized, and as a result cannot be reordered provided the root blockchain also reaches sufficient finality.

5.6 Withdrawals

Plasma allows one to deposit funds of the native coin and tokens (i.e. ETH and ERC-20 tokens) off the root blockchain. It additionally allows for state transitions within the Plasma blockchain whose state is enforced by the root blockchain provided there is information availability. In the event of information availability failures, there is a need to do a mass exit on this Plasma chain. Finally, it's also possible to do a simple withdrawal of funds held in the Plasma chain.

However, in normal operation, one can do a simple withdrawal.

5.6.1 Simple Withdrawal

For a simple withdrawal, one is only allowed to withdraw funds which have been committed in the root blockchain and ultimately finalized in the Plasma chain.

We have described a design for deposits, compactly representing ledger state, and state transitions. Up until this phase, other than fraud proofs, there haven't been any publishing of the current Plasma chain ledger state on the root blockchain. With withdrawals, though, there needs to be a specific proof that the funds are held in the Plasma chain and they are current.

Withdrawals are the most critical component, as this ensures the fungibility of coins between the root blockchain and the child Plasma chains. If one is able to deposit funds onto the Plasma chain, do state transitions (i.e. transfer coins to other parties), and those parties capable of withdrawing funds, then the value should closely map to the value of coins on the root chain. In some cases, funds on Plasma can be more useful, as it has greater transaction capacity, while the security is ultimately dependent upon the root chain.

For a simple withdrawal, all funds require a large bond and all withdrawal requests must include a large bond as a fraud proof. If current block data is available, then it is possible for a third-party to provide this proof at exceptionally low cost, as the third-party service can verify the Plasma blockchain live and ensure that the withdrawal proof is valid.

All participants of the Plasma chain MUST validate all parent Plasma chains and the root blockchain to ensure that there are no withdrawals in-progress for particular accounts/outputs when updating state. If a withdrawal is in progress, a subsequent block

cannot spend the coins/tokens, any byzantine behavior here violates consensus and is subject to fraud proof, penalization, and block reversal per the Plasma contract in the root blockchain.

A withdrawal occurs in the following steps:

1. A signed withdrawal transaction is submitted to the root blockchain or parent Plasma chain. The amount being withdrawn must be whole outputs (no partial withdrawals). Multiple outputs may be withdrawn, but they must all be within the same Plasma chain. The output bitmap position is disclosed as part of the withdrawal. An additional bond is placed as part of the withdrawal to penalize false withdrawal requests.

2. A predefined timeout period exists to allow for disputes. This is similar to the dispute period in Lightning Network. In this case, if anyone can prove an output has already been spent in the chain being withdrawn to (in many cases, the root blockchain), then the withdrawal is cancelled and the bonded withdrawal request is lost. Anyone observing the chain can dispute this. If the fraud proof of spent outputs is provided, then the bond is lost and the withdrawal is cancelled.

3. A second time delay exists to wait for timeouts of any other withdrawal requests with a LOWER block confirmation height. This is to force ordered withdrawal in a particular Plasma chain or root chain.

4. If the agreed dispute time period defined in the Plasma smart contract has elapsed and no fraud proofs are provided on the root or parent chain, then it is presumed that the withdrawal is correct and the withdrawer will be able to redeem their funds on the root/parent chain. Withdrawals are processed in the order of old to new in terms of the UTXO/account age.

Note that it is possible, provided that it is economically viable, to do a withdrawal in the event of a block withholding attack in the Plasma chain.

The fraud proof only requires that anyone on the network prove a duplicate signature spending from the same output, which can be compactly proven. For Lightning and other state channels, an additional requirement must prove a higher nonce as well. For channels, if a lower nonce withdrawal is attempted, then the funds remain in the Plasma chain, available for withdrawal with a correct signature. Other constructions are also possible, but design may need to be front-loaded as part of the creation of the smart contract fraud proofs for the Plasma chain.

As normal withdrawals are a slow, expensive process, it is likely that they will be coalesced into a single withdrawal or others are willing to swap coins for other chains using Lightning or an atomic swap[22].

5.6.2 Fast Withdrawal

A fast withdrawal is the same construction as the simple withdrawal, but the funds are being sent to a contract which operates an atomic swap. The funds being swapped are for funds on the root/parent chain with low timelock for funds with high timelock exiting the Plasma chain.

A fast withdrawal is not instant. However, it significantly reduces the time to withdrawal down to the time for transaction finality provided that the Plasma chain is not Byzantine (including conducting block withholding). For this reason, a fast withdrawal swap is not possible during block withholding attacks, and a slow mass withdrawal request would instead be necessary.

A fast withdrawal occurs in the following steps:

1. Alice wants to withdraw funds to the root blockchain but doesn't want to wait. She's willing to pay the time-value for that convenience. Larry (the Liquidity Provider) is willing to provide this as a service. Alice and Larry coordinate to do a withdrawal to the root blockchain. The Plasma blockchain is presumed to be non-Byzantine.

2. Funds are locked to a contract on a particular output in the Plasma chain. This occurs in a manner similar to a normal transfer, in that both parties broadcast a transaction, and then later commit that they have seen the transaction in a Plasma block. The terms of this contract is that if a contract is broadcast on the root blockchain and has been finalized, then the payment will go through in the Plasma chain. If no transaction proof can be provided, then Alice can redeem the funds. It is also possible construct this as an HTLC by having Alice generate a preimage, and only release it after she deems it acceptable and that the funds are transferred.

3. After the above Plasma block has finalized and Larry is confident he can redeem funds in the event the contract conditions are met, Larry creates an on-chain contract which enables payment to Alice for the specified amount (the amount he will receive less the fee he charges for this service)

In our example, Larry the Liquidity Provider must be live and have fully validated the Plasma blockchain before accepting this swap. If Larry is unable to fully validate the Plasma chain (or is unfamiliar with the smart contract fraud proofs defined in the root chain), he should not conduct withdrawals. If Larry does not want funds in this chain and would instead prefer funds on the root blockchain, Larry can initiate a withdrawal after this completes, or conduct the atomic swap as part of a withdrawal itself.

In many cases, it may be more cost effective to do transfers between Plasma blockchains net-settled with liquidity providers. Transfers can occur between Plasma chains via Lightning or atomic swaps which allows for rapid finality.

As this is an atomic cross-chain swap, Alice and Larry are not giving custodial trust of funds to each other. Alice has her funds on the root/parent chain, and Larry will

be able to have full access to the root/parent chain at a later time. Provided low-cost block availability and finalized non-Byzantine behavior of the root blockchain, Larry can be reasonably confident that he will receive his funds even if Larry does not trust the Plasma blockchain itself.

5.7 Adversarial Mass Withdrawal

While an adversarial mass withdrawal transaction is within the framework of Plasma, this is not required for the protocol, its design is primarily for economic robustness of state (low gas/fees) in the event of block withholding. If one wishes to use account state inside the Plasma chain, then one can rely upon other designs as well such as a hierarchy of payments. Additionally, note a UTXO model is used here, but this system only works well if the root chain uses an account model. Further, if mass withdrawals are not a necessary or desired feature, it is possible to use an account model for holding funds in the Plasma chain and only allow for simple withdrawals (with an incrementing sequence number).

As the primary consideration of the design of Plasma is in relation to block withholding attacks to prevent fraud proofs (and other implications of a lack of data availability), there needs to be mitigations for detected data unavailability. When block unavailability is detected by users on a Plasma chain, it is imperative that participants exit the chain by a particular date. In the event the chain is not exited in time, the result is similar to not disputing an incorrect withdrawal in Lightning. This mechanism is the key to correct operation of the Plasma blockchain. Plasma relies upon the fact that if users detect Byzantine behavior via block withholding, the user is responsible for exiting the Plasma blockchain. The rationale here is that it is impossible to detect on the root blockchain whether a block is being withheld (either the user can assert that they never received the blocks, or the Plasma chain can assert that the user refuses to recognize that the block is available and is lying). As a result, the cost for asserted block unavailability has traditionally been presumed to disclose the current state on-chain (which is what Lightning does). However, for large blocks and state transitions, this can be incredibly expensive, and Plasma does not use this construction as it's unclear who is responsible for paying these costs. Instead, Plasma presumes that if a user believes that the Plasma blockchain is adversarially withholding blocks and may impact the ability to enforce state transitions in the future, then one simply should exit from this Plasma chain to another as quickly as possible.

Hence, this is defined as an adversarial mass withdrawal insofar as if a block is unavailable, then it's presumed that the Plasma chain is adversarial or Byzantine. Mass exit ensures that the Plasma chain's Byzantine behavior does not impact one's funds beyond a significant time delay and halting the chain.

It is presumed that additional security mitigations using SNARKs will be used in the future, but the specific design remains an open question. This construction does not rely upon SNARKs for the withdrawal provided there is periodic liveness of the observers on the root blockchain, however by enforcing state transitions within the Plasma chain, the

ability for an attacking or Byzantine Plasma chain to be able to conduct adversarial block withholding to steal funds from those not periodically observing the Plasma chain can be minimized and enforced by the security properties of the SNARKs circuit. In that case, it would require a SNARKs proof to do state transitions and a SNARKs proof for a withdrawal for a gain of greater assurance of state transitions. However, the goal of Plasma is not to rely upon SNARKs for correct behavior of state transition provided that the user is observing the chain, the smart contracts encode the mechanisms correctly, and is able to withdraw on the root blockchain. Similar benefits can exist for the Lightning Network in ensuring correct current state by ensuring off-chain states are only possible in recursive SNARKs proofs committed by a 3rd party on chains which support smart contracts.

Plasma chains can be secured by defense in depth via the first line of defense with secure elements/hardware, the second line with SNARKs/STARKs, and the final line being the interactive game on-chain. The first line may fail, but the second line secures it using novel cryptography, with the final line being a public transparent interactive game. We propose Plasma as a system using the final line initially.

A mass withdrawal is achieved by creating an interactive game whereby exits occur in the following manner:

1. Alice coordinates with others on the Plasma chain to conduct a mass exit. Multiple mass exits may occur at once, but they should not have duplicate withdrawals. In the event they do, then the mass exits update their balances, they are processed in order, and the person making the duplicate is penalized. All parties should coordinate to send their funds directly into another Plasma chain.

2. Pat the exit processor is willing to organize this exit. Pat coordinates with the destination Plasma chain to send the funds and has committed to automatically recognize the funds as available in the new chain when the mass exit finalizes.

3. Pat verifies the Plasma chain up to the point of information availability. This point must be during the acceptable dispute and Plasma finalization period (separate from the root blockchain finalization) and is in accordance with the terms of the smart contract. Pat shows the pending destination ledger in the new Plasma chain to participants. Pat takes all the signatures from participants wishing to exit (including in our example, Alice). Pat verifies the blockchain that all participants have the right to exit for up until the highest point of data availability. Pat creates an exit transaction with a massive bond (as defined in the root blockchain smart contract). Pat may charge a fee for participants exiting.

4. Users sign off on the mass withdrawal again after downloading all signatures. This allows for the user to know that Pat will not be penalized, and now it is locked in. Users who have not submitted the second signature will not have their bit included.

5. Pat then observes whether there are any other exit transactions and removes duplicates if necessary, and signs the exit transaction and broadcasts on the root blockchain or

parent Plasma chain. In the event of duplicates, chain parents take precedence (up to the root blockchain as the highest precedence). Earlier transactions take higher precedence. Upon broadcast of the mass exit initiation transaction (MEIT), Pat is bonding an attestation that he has the following information correct: block validity, UTXO set at block height, non-finalization, merkleized mapping of bitmap to UTXO, amount committed (in a merkleized sum tree for rapid proofs), Alice and others' signatures are available if/when challenged. As part of the MEIT Pat publishes a full bitmap of the state being exited. This is so that other participants observing the root/parent chains are able to verify what is being exited and challenge if it looks incorrect. Finalization of the MEIT is a very long time, and may take many weeks, hence the MEIT is a last-resort transaction (future speedups may be possible with SNARKs).

6. If there are duplicate withdrawals, then Pat has the option to update the bitmap and balance being withdrawn during a small grace period.

7. Any participant on the network can challenge the data attested in the MEIT with a Disputed Mass-Exit Transaction (DMET). However, as Pat cannot know if a future block replaces outputs, Pat must not be penalized if a transaction has been spent on a future block (but a user can be). If a challenge is provided, then funds are locked up until the challenge game completes. These challenges must occur in an early grace period, and if a challenge is valid, then Pat must update the balance which is to be withdrawn.

8. If there are no challenges, then after the predefined finalization period for the MEIT proceeds and the users receive their funds.

The finalization window time for the Plasma chain is the time in which one must at least periodically watch the chain. After the finalization window, it is presumed that everyone has Plasma blockchain block data availability up to that window.

In effect, when the MEIT is created by Pat, Pat is attesting to correct records up to a particular Plasma block height as well as attesting to the fact that he has the signatures for withdrawal with each output. Pat is not penalized if an output has been double-spent after the attested period (as block withholding should not penalize Pat).

5.7.1 Mass Withdrawal Dispute: Incorrect Withdrawal Challenge

In the event a user such as Alice sees that Pat is attempting a mass withdrawal without her consent, she can invalidate the withdrawal by creating a challenge.

1. Alice sees that Pat has attempted a mass withdrawal of one of her outputs in the Plasma blockchain, as one of the bitmapped fields is enabled. Alice broadcasts a challenge with a massive bond. This bond is attesting to the fact that a challenge will not be produced. She broadcasts this on the blockchain.

2. If the challenge is not disputed after a set time period, Alice gets her bond refunded, and the entire MEIT gets cancelled. If the challenge is proven to be disputed as Pat or any other party produces a fraud proof on her Incorrect Withdrawal Challenge, then the MEIT remains in effect and her bond is slashed.

Participants are sure that signatures are available to prove as there is a second stage in the MEIT (step 4), so they have sufficient information to dispute a challenge if the challenge is fraudulent. Incentives are against producing fraudulent challenges, as they will be penalized provided block availability and non-censorship of the root chain.

5.7.2 Disputed Mass-Exit Transaction

In the event that an output has been spent from a Mass Exit Initiation Transaction in a later block, Pat may not know about this, so he should not be penalized, as one cannot prove block withholding.

There may be multiple disputes which dispute similar bitmapped sets, but they all must have a large bond attached.

Any participant can express a bitmap/range of spentness with a large bond. The large bond is an attestation that a coin has been spent in a later block, with a commitment to the block header.

However, this dispute cannot be proven compactly, so another iterated challenge is possible, to issue a Challenge on the Disputed Mass-Exit Transaction (CDMET).

The challenge on this dispute is as follows:

1. Alice notices that someone (i.e. the operator of the chain doing block withholding) attempts to dispute her mass withdrawal she is participating in. She submits a challenge on this dispute with a large bond attesting to the fact that the submitter of the dispute cannot produce a valid spend.

2. The submitter of the dispute must respond to the challenge within some time period. If the submitter cannot produce a proof of spend, essentially a signature of a later transaction, then Alice is vindicated, and the entire dispute gets cancelled (this is why duplicate disputes are accepted). If the submitter can prove that the coin has been spent, then Alice loses her bond and the dispute continues.

5.8 Recycling UTXOs

After a spent output has been finalized, it is possible to reuse UTXO bitmaps for compactness.

5.9 Summary

As a result of this mass withdrawal game, it's possible to do a mass withdrawal which consumes 1-2 bits of information per withdrawal in the most optimistic case for many

participants withdrawing.

Mass exits are something which is necessary in the event of block withholding. However, this may still be too costly. For this reason, we may need alternative strategies as well which don't rely on overloading the root chain.

This construction allows for many participants to hold their funds in a child blockchain, state invalidation occurs via fraud proofs if block information is available, state transitions can occur (i.e. payments), withdrawals are available, and mass exits (albeit with some delay) is possible in the event of block withholding.

6 Blockchains in Blockchains

As we have described, Plasma is at its core constructing a method to do scalable computing, however we need to contend with issues around block withholding to generate fraud proofs, as well as block space availability. The solution for block withholding in Plasma is to construct a system whereby one can do mass exits in the event of chain halt or Plasma block withholding.

However, a mass exit transaction on the blockchain can be very expensive, especially if the UTXO set is large and the bitmap needs to be published. Additionally, it may be desirable to just publish a single exit. Mass withdrawal transactions require a complex interactive game involving many participants. It should only be used as a last-resort.

Instead, we construct a system of higher and lower courts where particular venues can exist to prove state. One can view the root blockchain as the Supreme Court from which the power of all subordinate courts derive their power. It is the law of the root blockchain which allows for all lower courts to derive their judicial power. This allows for scalability in venues, it's only when the state of the lower courts is disputed or halted that one needs to move on to higher courts for a more represented venue. Broadcasting attestations of state in higher courts are always possible, but can be more expensive.

All state is merkleized and committed to the root blockchain. In the most optimistic case, block headers are published in the direct parent chain, and the parent's chain is published in its parent, and so on until it reaches the root chain. Inside the header is a merkleized commitment to the blocks one has seen in the parents.

Transactions can be submitted to the plasma chain and any parent plasma chain, as well as the root blockchain. The purpose of this is to ensure fungibility and censorship resistance. Specifically, in the event of halting and non-disclosure of block movement, one will still be able to withdraw funds.

When a block commitment is submitted, it must wait a certain amount of confirmations reflected in the root blockchain before it is approved. During this time, fraud proofs may be submitted to the root blockchain or any intermediary plasma chain (which is then committed to the root chain via a block root).

Each individual Plasma chain runs a state machine which bundles commitments into the Plasma block. The individual Plasma chain may or may not be able to introspect into

the details of the child Plasma chains. Instead, they have a running confirmed balance of the value of the plasma chain. When a child Plasma chain updates their state, they submit a hash of their plasma block header to any of the parent(s) plasma chain or root blockchain.

This means that a particular block state can be submitted to multiple parent chains. In the event of duplication, then it may not be faulty (but may be penalized under certain consensus rules depending upon the application). On the other hand, if there's equivocation in state, e.g. state committed to Parent1 differs from Parent2, then the bonders of the plasma chain may get their deposit slashed.

A new child state update may update using the following fields in their state update message: the fee being paid (and denomination), root blockhash being committed, previous blockhash, parent blockhash being committed to, proof of deposits, proofs of withdrawal.

Whatever the parent blockchain being committed to assumes that the child chain has seen everything up to that point, including recursively all parents above that. This is to commit to proofs that it has is not going to equivocate and double-spend transactions (and therefore exposing it to slashing in the event of equivocation).

In the event of equivocation, the parent chain state always takes precedence. Incentives are created to disclose equivocation by any party knowing them.

Deposits and withdrawals are possible on both parent chains as well as the root blockchain.

Withdrawals are also possible between plasma chains provided there is sufficient liquidity and another party is willing to take on the funds elsewhere. This can be done via a cross-chain atomic swap.

If one wishes to clear using the main blockchain, it could be possible to construct HTLCs between chains, which look like on-chain Lightning payments.

All fraud proofs must provide a merkleized proof of chain commitments. False proofs penalize the particular Plasma chain which is responsible for the fraudulent block.

The primary design complexity is a representation of transaction state broadcast across multiple parent chains in the interest of censorship resistance. Early iterations can presume that state transitions/transactions can only be conducted in the individual Plasma chain, and the only interaction with other chains is committed message passing to the parent/child and deposits/withdrawals. That way, the primary complexity is only proofs related to deposits and withdrawals.

Commitments to data are assumed as part of the proofs of inclusion.

6.1 Receiving Funds Inside a Chain

In this hierarchical framework of blockchains in blockchains, when one receives funds from another user, the process is as follows if Alice wants to send funds to Bob in a Plasma chain 3 levels deep:

1. Alice coordinates with Bob that she wants to send funds to Bob. Alice discloses to Bob the Plasma chain in which Bob will receive the funds. Bob decides whether

to accept the payment, specifically, Bob should ensure that the smart contract on the root blockchain is one in which he is willing to accept payment (smart contract code/mechanisms, as well as acceptable consensus exit delays, etc.)

2. If this is payment for some good, they pre-sign a statement defining the conditions of payment, in many cases it will be a proof of payment by block inclusion in the blockchain with sufficient maturity, however it can also be a pay-to-contract-hash in some circumstances. This is not on-chain but merely to attach the terms for settlement to prove to others.

3. Alice makes the payment inside the Plasma chain. The block gets signed off by the validators, and the commitment to the block header gets published into the parents' blocks. Merkleized commitments to child Plasma chains are included in every parent block and ultimately included in the root blockchain.

4. Bob fully syncs with the root blockchain, then validates the chain in which the funds are being received and any parent of that. Bob does not need to validate other Plasma chains which his funds are not a part of. Bob can fully validate that Alice has made the payment in the Plasma chain with sufficient maturity in the worst-case. However, if rapid finality is desired, Alice can sign off on the payment being fulfilled in the new block (see the previous statement on receiving payments inside a Plasma chain). If Alice is willing to sign off on the payment and Bob accepts it (as he is able to prove a withdrawal), then it is presumed that finality is reached. Bob is able to withdraw the funds from this Plasma chain.

The key aspect of this design is that one is wholly responsible for validating the child blockchains. If Bob does not validate the Plasma chain and all parents (ultimately the periodic commitment into the root chain is published), then it should not be treated as fulfilled. Similar to the construction in Lightning Network, Bob does not need to care what happens in the other Plasma blockchains. He only observes the correctness of the chains which matters to him. When he has the ability to use the coins, he then is confident he is able to spend them.

6.2 Receiving Funds from a Parent Chain

Receiving funds from a parent chain is similar to a deposit from the root blockchain, the only difference being the recipient needs to validate all parent Plasma chains (rather than just the Plasma chain itself). Deposits into a child Plasma chain is fast.

6.3 From a Tree to a Web

While the above description is about a single parent chain, it's possible for plasma chains to watch multiple root blockchains. This allows for one to update balances with child chains. Care must be given as failure on one parent may not be recognized by all participants at

once, and cascading systemic failures must be mitigated via time-delays and minimizing assumptions of cross-chain liquidity. The correct construction for this is an open problem.

6.4 Mitigating the Block Withholding Problem

By constructing many venues where one can broadcast a withdrawal transaction, there can now be many possible venues to exit from a chain which has halted or has blocks withheld. If a child chain fails, then an individual simple exit can be processed on the parent chains, even if transactions become expensive on the root chain.

This allows one to have some measure of confidence in holding micropayment outputs on the Plasma chain, provided that they have assurance that one of the parent Plasma chains are operating correctly. This goal is the primary cause of this, as well as mitigating the impact of cascading failures.

If one has sufficiently large output balance held, they do not need to do significant underwriting if there isn't significant time-value, however, if one holds a single low-value output (where paying the transaction fee becomes too expensive), then one should have some measure of assurance that one of the parent Plasma chains have availability. If one wants greater assurance, one can run nested chains deeply with many independent parties running each Plasma chain at each level. There are some tradeoffs that exist by doing it this way, though, as if a particular Plasma chain starts becoming Byzantine, then everyone will need to do a mass-withdrawal to a new chain. If there is a parent which is not byzantine, though, it is possible to continue operating and facilitate rapid transitions to another chain if the parents refuse to process the Byzantine chain's commitments.

It's possible that services may arise whereby it doesn't do anything other than process transactions in the event of a child chain fails. The operator of this service doesn't need to do anything unless a child chain fails (to the point where they can be sufficiently passive that they could hypothetically turn off their servers until a failure occurs, block headers automatically skip them to be broadcast on a chain a level above the passive operator).

We expect that many of the withdrawals in parent chains will be simple withdrawals instead of mass withdrawals, as a parent chain can have incredibly high transaction volume (block size/gas limit).

6.5 Exiting

Mass exits are possible to a parent chain or the root chain. If the child chain begins to act Byzantine, it's presumed that any state may be invalid, similar to a Plasma chain without nested parent chains. Similarly, mass exits are a way to quickly exit from a Byzantine parent chain. It is possible to skip a parent chain (or the child chain itself) to its parent(s) or the root chain.

While it may seem as though there's some complexity in the design, the presumption is that if any chain is Byzantine, all its children must act. There are optimizations that

are possible so that exits are possible without coordination via a heartbeat (exits are by-default without a signature revoking it on the users' side and the Plasma chain itself making a commitment that it has received it, but may be a premature optimization).

The construction is basically the same as the simple exit or mass exit, however there are some minor changes in the design to support nested chains. Exits may be duplicated, but exits on the parent chain always take priority. If a parent chain begins to act Byzantine, then the exit can be committed on the root chain as well. It is the responsibility of the (perceived to be Byzantine Plasma chain) to reflect and update their state of the parent/root chain duplicated exit and revoke the duplicated exit in its own chain. However, if it does not do so, users' funds will be available on the root chain.

If a parent is Byzantine but the child in which one holds funds is operating correctly, it is possible to avoid doing a complex mass exit transaction. Participants find a new chain to send their funds to and do a simple exit whereby a liquidity provider receives funds in this child chain and the other users receive funds on the new chain (without the Byzantine parent). Commitments of the child chain blocks are published on the root chain or the higher-level parents (avoiding the Byzantine node). Users quickly have funds in the new chain and the liquidity provider exits their funds onto the root or highest parent later. The purpose of this is that new funds can be quickly allocated into the new chain and exits occur rapidly.

6.6 Scalability

This allows for UTXO bitmap scalability, in the event that a bitmap gets too large, one just needs to split the bitmap into multiple child chains. For child chains, it is presumed that it is represented as an account balance with block height nonces (and candidate chain tips) instead of an output. Similarly, for states which prefer to use accounts instead of UTXOs, then it is possible as well provided that one is willing to make the tradeoff of only supporting simple withdrawals.

The end result of this is a great deal of scalability for users. They only need to observe Plasma chains which their funds are held (as well as its parents). This effectively shards the dataset into validation which affects oneself.

7 Plasma Proof-of-Stake

We propose a simple proof-of-stake construction. This is likely not the optimal proof-of-stake construction, however this is illustrative of what is possible in Plasma chains.

Up until now we've assumed that the operator of a Plasma chain is a single entity who is responsible for signing off on the blocks. If they create an invalid block, anyone else who has the block data can generate a fraud proof and roll back the block with penalties to the operator. This proof is possible as the operator has signed off on the block with their signature. The publication of a merkleized commitment of the Plasma block in the root

chain (and as the highest parent Plasma block includes commitments to its children's state updates), hence state updates are ordered and bonded to correct behavior.

However, in many cases it's preferable to construct a proof-of-stake chain instead of a single-party proof-of-authority chain. This minimizes risks related to block withholding (it's possible to get the best of both words by nesting a chain into a single-party proof-of-authority as well as an open multi-party proof-of-stake chain). A tokenized proof-of-stake chain also gives incentives for tokenholders to operate correctly, as the value of the token declines from Byzantine behavior. More detail on the potential value of tokenization is provided in a later section.

Proof of Stake construction is easier to construct in Plasma, as it still relies upon the robustness of the underlying root blockchain. Problems related to withholding, finality, and other factors are pushed to the reliability of the root chain. Plasma can only at best be as secure as the root chain. If the root chain is running Proof of Work, then this is Proof of Stake on Proof of Work (Plasma on the root blockchain, respectively). If the root blockchain is Proof of Stake, this construction is Proof of Stake on Proof of Stake, however the Proof of Stake mechanisms may be simpler or different than the one running on the root blockchain.

7.1 Nakamoto Consensus Incentives

We attempt to replicate the primary incentives from the Nakamoto Consensus (Proof of Work mining). One of the most important incentive to replicate is that of encouraging block propagation to other miners.

Many existing proposed proof-of-stake mechanisms rely upon leader election, where at time t_0 a leader is elected and at time t_1, the leader has the right to produce a block. This does not replicate the Nakamoto Consensus incentives around block propagation. The Nakamoto Consensus does do leader election, but rather it does probabilistic leader election. If one finds a block, one believes it is likely that one is the leader, but one isn't completely sure. Someone else could have mined a block at the exact same moment. The best way to maximize odds that one is the leader is to broadcast that block as far and wide as quickly as possible so others can build on top of it. This creates incentives for information availability.

Plasma's Proof of Stake construction needs to do something similar.

We made tradeoffs in that we want to encourage everyone to propagate their blocks as far as wide as possible. There may be other constructions (especially ones in which construct heavy reliance on random selection and probabilistic leader election after-the-fact by allocating random scores to particular branches and determining the chain tip as the branch with the highest score).

7.2 Example Simple Proof of Stake Model

While this is a simple proposal to construct a Proof of Stake model, it is likely that it isn't anywhere near optimal. The goal is to construct something simple which Plasma can use.

Instead of creating enforcement mechanisms, the approach is to simply create incentives for proper coordination and correct behavior (block propagation).

Fees are allocated and distributed by the root contract and paid out periodically if desired, but the accounting is done inside the chain itself.

As part of the staking contract, stakers' funds allocated assign towards a delegated staker. The staker is responsible for acting on behalf of the user and the user gets penalized if the staker is faulty. Staking is committed for a specific time (e.g. 3 months). The minimum amount per staker is one percent of all tokens, with a maximum cap of five percent. If one wants to allocate more than five percent, then they should use multiple staking identities (the purpose is to maximize data distribution and minimize the efficacy of below 51% cartels).

Funds get allocated depending on whether the past 100 Plasma blocks are representative of all participants. For example, if someone is staked at 3 percent of the stakers, they should be 3 percent of the previous 100 blocks. If it is above that amount, the individual staker receives no additional reward for publishing extra block commitments. If the past 100 blocks, there is below 3, then the current block creator receives fewer rewards. Only one block can get allocated per block on the root chain.

This encourages all participants to coordinate and include everyone's blocks equally. The presumption is that one does not need to construct enforcement mechanisms, the participants will coordinate and ensure some kind of scheme (e.g. round robin) to ensure maximum rewards.

If they do not receive the maximum in transaction fees due to improper amount of blocks, the funds get allocated to a pool to pay out future blocks.

The result is economic encouragement that one includes participation from everyone.

However, this is not yet complete, as we are only encouraging accurate participation from stakers. In every block is a merkleized commitment to data in random parts of blocks from the past 100 blocks. This forces the staker to have the full block data and consequently forces a block creator to propagate it to all stakers.

The chain tip is determined by maximum reward, if there are parallel branches, then the one that wins is the one that has the maximum fee reward from maximum coordination.

This construction is not designed to stop 51% attacks, but instead is designed to encourage block propagation (as the threat is equal if one commits to withheld blocks). Additionally, this construction is reliant upon information availability and impartiality in block inclusion on the root chain; it is not possible to construct this type of Proof of Stake on the root chain due to assumptions on data availability and censorship incentives.

8 Economic Incentives

Within a Proof of Stake validation model, it's possible to construct incentives which align with correct operation of the contract terms. While fidelity bonds ensure accuracy towards the chain, we need to create further incentives around data availability and discourage

halting. By only allowing staking using a token specific to the Plasma chain, one is able to ensure that there is incentive to continue operating, as the value of the token is derived from the net present discounted value of all future returns from staking. Consequently, network failures reduce the value of the tokens being held and individual actors have heavy incentive to act in the best interest towards the network's continued operation.

The operators of the Plasma chains receive fees from broadcasting transactions on-chain. Computation may charge different fees for different operations, and fees may cascade down, especially for complex operations. While there is incentive for more transactions to be conducted in deep child chains, it's possible to create commitments for funds transferred or computation in the child chain, which propagates up. This allows parent chains to charge for computation in child chains, and if there is an incorrect commitment, the block data is invalid and unenforceable. This is not a necessity, and in many cases, one would prefer more computation on child chains with less necessity for distributing fees up to parent chains.

For systems upgrades, it's possible to upgrade the system by creating another contract which accepts the same token and announcing a transition period (or the community collectively decides in decentralized systems).

This may create systems which run themselves. Whereas one needed to pay and operate services on cloud computing to host sites which do data storage and computation, it is now possible to construct a set of smart contracts (with fraud proofs), a token, and with sufficient number of participants paying fees, that the system can operate on its own with a set of stakers who continue to correctly operate the network and computing infrastructure, truly making computing on the amorphous cloud.

8.1 Tokens vs. Coins and Economic Security

These fraud proofs and bonds ultimately held on the root blockchain can be the native token, e.g. Ether (ETH) for Ethereum, or it can be a separate token maintaining the consensus rules of the underlying blockchain.

It is ostensibly most simple to use the root blockchain's native token (e.g. ETH), however there are interesting economic security implications.

If the goal is to prevent chain halt and faulty behavior, depending on the blockchain application, there may be insufficient incentive to prevent faulty behavior if only ETH was used. The token's value will decline if the chain halts or is Byzantine. Additionally, the token value is approximately the NPV of future transaction fees which can make the token valuable. If one staked ETH, then one is staking the value derived from the time-value of the bond relative to the amount of fees gained. It's expected that this value being bonded would be much lower than the net present discounted value of the token. Additionally, chain halts and block withholding are difficult to prove and discourage and if one bonds ETH and gets the money back after the staking period, there is insufficient incentive to act in non-Byzantine ways, whereas with tokens, the value of the token would decline with widespread Byzantine behavior.

9 MapReduce for the Blockchain

Nearly anything which is computable on MapReduce should be computable on this chain as well. This requires a deep restructuring of the way we think about computation and programming on the blockchain. This is MapReduce but with fraud proofs. Each node represents a blockchain. This is highly compatible with the Plasma blockchain tree structure described in a previous section.

E.g. if one wants to do the standard word count, you can create a merkle tree of chains operating a reduce function. If there is proof of fraud, then the node that produced the fraud gets penalized. If you can produce a reduce function on summation, then you can also produce an average. E.g. average prices, etc. Map function is just sending computation to the individual chains and then committing to the results. Obviously, there is constraint around data throughput, which is why reduce fraud proofs are necessary. Arbitrary computation of all types is not possible, but it is possible to resolve many types of problem sets; usually memory constrained problem sets can be solved by first running sorting algorithms, which makes tradeoffs of inter-Plasma chain traffic.

If the nodes cannot produce the actual blocks to prove computation, then their results should be discarded and rolled back. Note that this does not guarantee computational scalability that MapReduce does (since you need to observe the chain to maintain consensus), however, it does give enforcement of activity and the ability to scale it up for the actors. As a result, the primary limitations are around the fact that parties which are affected by particular computation should observe that set of computation. If one only needs to observe some small part, then it should be fine, but if one needs to observe all computation then it does not offer scalability benefits (only benefits around scalable assurance). That said, many problems can be solved this way, e.g. decentralized exchange (your mapped set cares about your own trades, if everyone else's netted execution is enforcible you don't care about the details), etc.

The block format must be compatible with data which can be computable in a TrueBit construction. There are commitments to state (to be able to construct a UTXO/state trie which allows for proofs of state transition on inclusion/exclusion), account trie (for child chains and complex state transitions), commitment on fees (tree which commits to state transition on fees), merkleized transactions, commitment to data passed from parent/child blocks, commitment to seen parent/child blocks (to prevent reordering), as well as any business logic (e.g. word count example would have a merkleized sorted commitment to words and where it was seen). By constructing merkle commitments, one can create smart contracts provable on root or parent chains which are able to prove incorrect state transitions. There are some problem sets which may not be compatible with this format, but generic computation without significant memory requirements are possible. A mental framework for this is to treat the maximum memory size for computation as equivalent to the maximum of data allowed in a fraud proof.

A series of map and reduce functions allow for the blockchain to operate in such a way

whereby there is obligation to process data. This requires the parent and children to create obligation of processing. Children must include the parent passing in data else the chain will halt. The parent can enforce computation in children and if the children halts, enforcement of computation can occur by broadcasting the data in the parent chain and attesting to the proof there. A primary threat in TrueBit constructions is related to the issues around halting, so care should be constructed to allow for continued operation if the child chain halts, although this requires a great deal of complexity, especially over time (datasets can change and having time consistency is more difficult to reason about for some problems).

By constructing blockchain computation in a map and reduce framework with child chains, it's possible to take existing computer science research and directly apply it to distributed systems problem sets which exists for blockchains. It is possible construct Solidity contracts which produce many useful business applications in a scalable manner. One only needs to compute and verify activity relevant to oneself.

10 Example Applications

Decentralized Applications can be reframed as a MapReduce problem with economic incentives for correct activity bonded by a token.

10.1 Reddit Clone on the Blockchain

This is primarily about data storage (CRUD). Primarily computation and proofs are around access control, identity (votes and posts), and moderation. Many web applications are actually just doing CRUD on the back end.

The root blockchain contains the smart contract consensus rules and fraud proofs. The topmost parent contains accounts of subreddits. Each subreddit is a Plasma blockchain child of the topmost parent. Within each subreddit is a Plasma chain of posts. That child chain of posts contains comments as well. Consensus mechanisms enforce access control. Randomized commitments to previous block data (with random nonces provided by the parent chain) are committed to in every block header. A reduce function periodically is computed for top posts and other statistics.

An individual user's computer downloads the data and software local to the machine formats the data. Submitting data requires paying transaction fees to incentivize inclusion of data, may have a fee for downloading old block data depending on availability.

To view a specific post, the user verifies the commitment on the root chain, then goes to the chaintip on the topmost parent (back n blocks to go back to finality time period, perhaps a week's worth), finds the state account trie for the relevant subreddit. Connects to a DHT network to discover nodes on the subreddit, downloads the chaintip (plus back n blocks to verify) of the subreddit and view list of posts, download the state trie as a light client and raw data for the relevant post with comments. Users only need to observe parts of the Plasma chains which are relevant only to themselves (download only the posts and

subreddits relevant to themselves).

This is a simple example of data storage with some computation on the blockchain. It is possible that validators fully validate all nodes, however, it's possible to shard it out. Once the sharding is too far, though, there are information availability concerns. One way to mitigate that is to give full control of the child chain to the subreddit owner.

10.2 Decentralized Exchange

A reddit clone on the blockchain, while makes obvious the implications for CRUD web applications, doesn't take significant advantage of MapReduce operations with the exception of site statistics.

A decentralized exchange shows that it is possible to trade low-latency for high computational capacity. As there are many states, it's possible outputs are defined in accounts instead of UTXO, or for each step in the state machine that a larger bitmap is used to represent each state instead of a single boolean value representing spentness in the bitmap.

Similar to the subreddit, there is a tree of child chains each representing a trading pair. Within each is a tree of chains to maximize scalability (for low activity pairs it may only be one Plasma chain, but for high-activity chains it may have much more children). Each of these chains have bonded activity and the amount which is able to transact per round is limited by the bonded amount.

The first step is to have balances in a child chain, so this is like a payments Plasma chain as a base.

Next, orders are issued directly into the child chain. As part of the commitment to the parent, all orders are coalesced into a merkleized commitment of a single order book represented as a single order by the chain itself. This step recursively reduces all children's order books into a single orderbook represented by that chain, until it reaches the highest Plasma chain parent. After orders are received, the order window closes and the trades execute in a batch fashion.

After this reduce step completes and is committed into the root blockchain, the individual chains are notified of their allocation, via a map step. The parent tells the child what allocation of their orders filled in a deterministic manner. Provided the child was able to see the other orders (which implies it is able to observe the parent chain during this step), they will be able to prove correct execution of allocation in this map step. After receiving the allocation, the map step continues recursively to that chain's children.

When this completes, a final reduce step is made by committing all fund updates into a block and submitting the block header to one's parent.

There are further optimizations by allowing for multiple MapReduce rounds in the event of significant price movements (allowing for high accuracy in pricing), however this general construction enables immensely high volume. It is theoretically possible to conduct all of the world's trading activity in this framework, with some tradeoffs in speed by turning it into a batch execution exchange fully committed to and bonded by the root blockchain.

This type of construction is useful for many types of financial activity and computation.

10.3 Decentralized Mail

To create D-Mail, it's possible to represent one's account in a Plasma chain and require payment to receive mail (insert message in the chain). Submissions are encrypted with one's public key. Enforcement is possible to ensure that unknown entities must pay, further optimizations are possible using zk-SNARKs. Parent chain contains a directory of chains and enforces payment. Simple design.

10.4 Decentralized CDN

It's possible to have a decentralized CDN. Similar construction as Ethereum sharding proposal. Treat each child blockchain as a shard. Have a randomness beacon (can be root blockhash or something else). Shuffle the data around between shards every n blocks. Parent chains are responsible for commitments to the shuffle. Others may lazily keep archives. Data loss is announced and those who keep archives are rewarded. Incentive to propagate, since one only gets rewarded if the other shard has the data. Robustness depends upon the requirement for the "length" of the flow; the more robust, the more shards need to have a copy at any one time. The key insight here is that storage is a function of bandwidth. The data is not treated as stationary data on disk. All data is actually in flow and motion to its next destination; very Taoist approach.

For downloading the data, one verifies the parent chain's shard and random beacon to know which shard contains the data, then connect to it via a DHT identifying the peers and download the data.

10.5 Private Chains

Participants are not obligated to disclose the data in the chain to others (although there is nothing stopping it from being public), and as a result if participants on the chain want to have some private blockchain network enforced by the root chain, they can do so. This is analogous to the intranet/internet separation. Transactions may occur on the local private chain, but also communicate and have financial activity bonded by the public chain.

11 Attacks, Risks, and Mitigations

11.1 Smart Contract Code

Writing good smart contract code is hard. The security is entirely reliant upon correct execution of the fraud proofs. It is possible that some fraud proofs may not be included and invalid state transitions may be valid in the root chain.

11.2 Closing Transaction on Main Chain Too Expensive

There's a risk where a transaction can be closed on the main chain, but it's not economically feasible to do so. This can create certain types of exit scams whereby one coordinates a large amount of small values to add up to a large value.

This can be mitigated by having an exit provision, sorting all transactions by the exit provision and allowing an exit from an entire chain at once after a certain dispute mediation period. This additionally allows for a 3rd party watcher to watch on one's behalf. However, this construction increases complexity significantly. These pre-signed coalesced transactions can propagate onto other parent networks depending on what part of the chain fails, or ultimately on the root network. However, this also relies upon named parties acting correctly, which is why individuals should coalesce all unspent payments to a single output or set of outputs whereby exiting transactions are economically feasible.

Additionally, one can leave micropayments on chains which are bonded by a highly valued token, and piggyback on that value (as there is sufficient disincentive from completely killing the Plasma chain).

If generalized recursive SNARKs/STARKs become feasible, it would be theoretically possible to ensure that the entity withdrawing doesn't have authority to do unauthorized exits, even with withheld blocks.

11.3 Finality

The dispute window for exit is essentially creating finality assumptions. If the underlying chain has some significant bonded costs to reorgs to force finality, then it can significantly reduce risks around deep chain reorgs creating a lack of synchronicity between chains. An example of mitigations is the planned Ethereum CASPER finality gadget.

11.4 Root Chain Lack of Capacity or Increasing Costs

Without mitigations in place, in the event that fees or gas becomes too expensive, it would not possible to exit the transaction within a set time period. E.g. if transaction fees / gas costs increase 50-fold or there's insufficient space to exit transactions and miners do not increase the capacity or gas limit.

There are several mitigations by extending the exit delay by pausing the exit counting mechanism which allows for exits in an orderly fashion. This can be done by pausing exits so long as there is an exit transaction which occurs in the past x blocks. This allows everyone to get out over time provided that there's at least one exit transaction recently. If an exit occurs before one's exit, then the counter is reset. The result would be that it is undetermined how long one must wait before receiving funds back, which increases the expense of liquidity providers rates. A simple mechanism would be to pause the clock on withdrawal times if average blockchain gas/fee costs are above a particular very high amount (gated by some reasonable upper bound time in which this paused state can persist).

Users holding funds should ensure that at least one parent Plasma blockchain has a high degree of confidence with information availability (ideally multiple independent parents).

11.5 Root Chain Censorship

The design assumes that over 51% of the root chain is honest. If participants on the root chain coordinate to attack the network by censoring blocks, then significant difficulties may arise from enforcement of exit transactions or state updates, potentially creating significant issues around loss of funds. Censorship is the primary factor of security and value-at-risk as well as finality tricks (e.g. CASPER finality gadget) may need to be constrained in future chains.

This can be mitigated by adding in zk-SNARKs/zk-SNARKs proofs of funds, but requires significant new engineering and research.

The use of a very large bond as part of every exit transaction encourages fraud proofs as the miners will likely be rewarded a significant majority of that fraud proof and censorship would be highly discouraged.

The security afforded to this network is a function of the honesty and correctness of one's parent plasma chains, the root blockchain, and the balance size.

Systemic limitations globally on the amount transferred (velocity limits on exits per block) and ensuring it is lower than a finality gadget could be a possible path of mitigation.

11.6 Chain halt

If the chain halts, after a set time period can have a pre-committed state transition offer. The consuming chain taking over the transactions can then broadcast an acceptance of that and the entire chain moves. This is only allowed after a chain (ignoring transactions broadcast on parents) does not move forward for a set time period. Fraud proofs can dispute the chain tip.

There may be greater incentive to chain halt for financial activity involving complex state transitions.

11.7 Inability to Change Consensus Rules

As the design is front-loaded it's not possible to change consensus rules without pre-programming in that ability. This can be mitigated by creating upgrade paths as part of the system (e.g. forced halt after a certain date). This inability can also have social implications around the inability to halt chains, as there is economic incentive for the tokenholders to continue operating the system, so may become difficult to halt the Plasma chain after it launches.

12 Future Research

There are additional areas of future research including benefits related to security around these chains. A current area of research is generalized recursive SNARKs/STARKs, which would significantly boost the security of exit transactions. It could still be desirable to have defense in depth, hence the last line of defense would be a straightforward decentralized exit mechanism allowing for disputed proofs, with front lines being novel cryptography and secure hardware elements. Further developments in novel uses for pairing cryptography or other forms of homomorphic encryption may also be helpful.

Greater specificity is required on the ability to watch multiple root chains at once while remaining synchronized (beyond simply just forcing hard synchronicity).

Further research is needed around finality and its interaction across multiple chains, as well as further minimization of blockchain exit risks (SNARKs/STARKs could help here).

13 Conclusion and Summary

Plasma is design with a primary focus on ensuring information availability (especially with regards to block withholding attacks) with data compression.

We propose a mechanism whereby one can submit enforcible commitments which allow one to hold funds in a blockchain whose state is enforced by the root blockchain.

This allows for significant computation and storage across a wide amorphous network of computers. Activity is bonded by economic actors enforcing the commitments, which is ultimately enforcible across all parent chains and enforcement flows down to the root blockchain who holds the smart contract enforcing truth. This construction allows for one to do state transitions which would not be otherwise cost-effective on the root chain.

It is possible from this construction for a blockchain to be processing commitments on nearly all financial computation worldwide (as long as it doesn't take too much working memory at one time). It is only if there is invalid computation that proofs are submitted and those commitments rolled back. One does not need to give custodial trust to the operators of the chain.

To reduce the incentives around chain halting and other Byzantine behavior, fees create incentive for the chain to continue operating. Halting is discouraged if the Plasma chain is bonded by activity in a token specific to this chain. If the chain halts, the value of the chain declines, creating significant economic incentive for continued operation.

This incentive and structure allows for one to create decentralized autonomous programs which continually operate funded by transaction fees. These decentralized autonomous applications can create true cloud computing, whereby the data is being processed and validated, but whose member set is ever changing and amorphous. Plasma can allow blockchains to scale up to serve generalized applications with the number of users without significant limitations. An application creator can just write the smart contract code, then submit the code to the blockchain and the incentives can persist to continue operating these contracts'

computation as long as people use the Plasma chain to pay its fees.

14 Acknowledgements

Many thanks go out to the TrueBit authors for a design and implementation of merkleized proofs, including Chrstian Reitwießner for review. Thanks to Vlad Zamfir for many areas of inspiration and his general framing of ideas helpful in formalizing these ideas. Thanks to Thomas Greco, Piotr Dobaczewski and Paweł Peregud for feedback and contributions.

TODO: Ask others for acknowledgements.

TODO: Improve bibliography and get more citations

TODO: Finish diagrams

References

[1] Joseph Poon and Tadge Dryja. Lightning Network. https://lightning.network/lightning-network-paper.pdf, Mar 2015.

[2] Ethereum. Ethereum. https://ethereum.org.

[3] Gavin Wood. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. http://gavwood.com/paper.pdf, Feb 2015.

[4] Raiden. Raiden Network. https://raiden.network/.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.

[6] Satoshi Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Oct 2008.

[7] Nick Szabo. Formalizing and Securing Relationships on Public Networks. http://szabo.best.vwh.net/formalize.html, Sep 1997.

[8] Fred Erhsam. Blockchain Tokens and the dawn of the Decentralized Business Model. https://blog.coinbase.com/app-coins-and-the-dawn-of-the-decentralized-business-model-8b8c951e734f.

[9] Naval Ravikant. The Bitcoin Model for Crowdfunding. https://startupboy.com/2014/03/09/the-bitcoin-model-for-crowdfunding/.

[10] Jason Teutsch and Christian Reitwiessner. A scalable verification solution for blockchains. https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf, Mar 2017.

[11] Vitalik Buterin. Ethereum Sharding FAQ. https://github.com/ethereum/wiki/wiki/Sharding-FAQ.

[12] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timn, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. https://blockstream.com/sidechains.pdf, Oct 2014.

[13] Paul Sztorc. Drivechain - The Simple Two Way Peg. http://www.truthcoin.info/blog/drivechain/.

[14] Bitcoin Wiki. Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification.

[15] Peter Todd. Tree Chains. https://github.com/petertodd/tree-chains-paper.

[16] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Mardas Virza. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. https://eprint.iacr.org/2013/879.pdf, May 2015.

[17] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster Computing in Zero Knowledge. https://eprint.iacr.org/2015/377.pdf, Apr 2015.

[18] Jae Kwon. Cosmos: A Network of Distributed Ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, Sep 2016.

[19] Gavin Wood. POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK. https://github.com/w3f/polkadot-white-paper/raw/master/PolkaDotPaper.pdf, Nov 2016.

[20] Sergio Demian Lerner. lumino transaction compression protocol (ltcp). https://uploads.strikinglycdn.com/files/9dcb08c5-f5a9-430e-b7ba-6c35550a4e67/LuminoTransactionCompressionProtocolLTCP.pdf, Feb 2017.

[21] Ilja Gerhardt and Timo Hanke. Homomorphic Payment Addresses and the Pay-to-Contract Protocol. http://arxiv.org/abs/1212.3257, Dec 2012.

[22] Tier Nolan. Re: Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949.