Notes: Outline and analysis of schema for scaled Kleros where courts all on same optimistic rollup

February 16, 2021

In these notes we will consider Ethereum scaling based on optimistic rollups. Notably, one expects in the long term to have many rollups on which different contracts are deployed. (Indeed, if one deployed all existing Ethereum contracts on the *same* rollup, this would recreate the scaling challenges already facing Ethereum.) This presents issues for composability between contracts on different rollups. `https://newsletter.banklesshq.com/p/when-defi-meets-rollup`

A fundamental choice to be made in designing rollup-friendly versions of the Kleros court contract is whether one should

- put different subcourts on the rollups where there are arbitrable applications that use them requiring cross-rollup communication between different court contracts when staking or resolving incentives involving appeals or
- put all courts on a single rollup, potentially requiring cross-rollup communication to arbitrable applications that are on different rollups.

Some of the broad tradeoffs that are relevant to this choice are discussed here: `https://github.com/kleros/research-docs/blob/master/research-notes/scalingcommunicationmodels.pdf`

In these notes we will focus on the second model, where all courts are on a single rollup. We will consider how one might facilitate required cross-rollup communication to arbitrable contracts on other rollups. We base our schema on that of `https://github.com/kleros/cross-chain-arbitration-proxy/#high-level-algorithm` Indeed, the models of cross-chain and cross-rollup communication have many similar issues related to asynchronicity. Cross-rollup communication has the advantage of being able to take advantage of the ability of both rollups to anchor themselves to the L1 consensus. However, the price of using this anchor is quite long finalizations delays, that indeed might be longer than one would even normally need to wait for cross-chain communication.

0.1 Actors and model

In the following schema, we consider an arbitrable contract on Rollup A, whereas the Kleros court contract is on Rollup B. A "plantiff" is a party that is interested in changing the current outcome of the arbitrable contract. A "defendant" is a party that is interested in preserving the current outcome of the arbitrable contract. To avoid waiting for the long finalization times of the rollups in order for a decision to be executed, we allow for a "reporter" to provide information to Rollup A on the result of the case as it has been finalized on Rollup B. If this report is challenged, the arbitrable application will wait for the finalization of the required information on Rollup B.

Note, that in our security model, the defendant is an actor on Rollup B that must be a node for Rollup A, namely follow the sequenced transactions on Rollup A and verify that they are valid. If he fails to do this, then a plaintiff's transaction may not finalize and the defendant risks paying arbitration fees without the possibility of winning a plaintiff's deposit. Similarly, the reporter should be a node for Rollup B. We assume that the L1 miners do not (successfully) censor actors; in particular the honest actor can place enqueue and fraud proof transactions when necessary.

In our security model, we will further assume that there is at least one honest party interested in the outcome of the dispute that is a node for both rollups and is willing to take on capital lockups and potentially pay gas fees to serve the roles of plaintiff, defendant, and/or reporter possibly using enqueue transactions, locking up the deposit to update the Optimism "state commitment chain", and issue fraud proofs on invalid "state commitment chain" updates if necessary.

0.2 Implementation details of Optimism and deadline management

We note a few details of Optimism, as currently implemented in their soft Mainnet launch, particularly with respect to the management of time on the rollup versus on L1 (at least as I understand this/having read their docs and partially looked over the code). See `https://community.optimism.io/compare/#behavioral-differences` and `https://github.com/ethereum-optimism/contracts-v2/blob/0ad4dcfdef11ef87e278a8159de8414c8e329ba1/contracts/optimistic-ethereum/OVM/chain/OVM_CanonicalTransactionChain.sol` and `https://research.paradigm.xyz/optimism#fnref:8`

- There is a list of transactions, in order, called the "canonical transaction chain" that is committed to on L1. There is a permissioned sequencer who can add batched entries to this list; however, a user can also issue an "enqueue" transaction on L1 such that their transaction is placed in the "canonical transaction chain". This provides a measure of censorship resistance on L2.
- Current state is defined by the status of the "state commitment chain".

Any user who has locked up a sufficient bond can make a transaction that updates the state commitment chain in accordance with how the ordered transactions of the canonical transaction chain should be processed. (This claim can eventually be challenged, triggering a process by which L1 determines the validity of these claims via fraud proofs.)

- The OVM timestamp and block number of an enqueue transaction are given by the timestamp and block number of the L1 block that includes this transaction. The OVM timestamp and block number of transactions that were issued on L2 are determined by the sequencer subject to certain rules.

- For example, the OVM timestamp of a transaction that is issued on L2 cannot be in the future with respect to the time stamp of the block in which it is sequenced on L1.

- Notably, there is a parameter FORCE_INCLUSION_PERIOD_SECONDS, which we will denote by F, such that transactions issued on L2 that have OVM timestamps dated more than F seconds in the past with respect to L1 time cannot be accepted in sequencing submissions. Thus, after an enqueue transaction is issued on L1, a sequencer could still place L2 transactions in the following F seconds before it by giving them timestamps in the past; however, after F seconds, this is no longer possible and the sequencer is obliged to place their transactions after the enqueue transaction in the canonical transaction chain.

We will denote OVM timestamps with lower-case letters, e.g. t_{TX}, and L1 timestamps with upper-case letters, e.g. T_{TX}. As OVM timestamps cannot be after the L1 timestamps of the blocks where they sequenced, for any given transaction we generally have $t_{TX} \leq T_{TX}$.

0.3 Deposits and incentivization of reporters

The participants in our system will, as usual, place deposits. For the purpose of the scheme in these preliminary notes, we break down these deposits into independently tunable parameters as follows:

- Plaintiff: PlaintiffStake + RepTip
- Defendant: ArbFees + DefStake + CrowdFundComp
- Reporter: RepStake.
- Report Challenger: RepChStake

Of particular note, the defendant, as the only participant on the rollup with the arbitrator contract, will pay the arbitration fees regardless of whether he wins or loses the case. In the event that he wins, he is compensated by receiving PlaintiffStake on Rollup A. Following a similar logic, we include a

new component of the defendant's deposit CrowdFundComp. This amount is used to incentivize crowdfunders that might pay the final round appeal fees of an alternative that then wins by default when the appeal fees of the other alternative(s) are not paid.

If a report is not disputed, then amount RepTip is given from the plaintiff's deposit to the (successful) reporter. This mechanism incentivizes third-party reporting, which allow for wait times for results to be available to the arbitrable contract to be minimal. This is an effect similar to that of liquidity providing services that allow users access to tokens on L1 or a different rollup in exchange for a fee. It might make sense to allow the plaintiff to choose this value herself at the time of making her objection. She could set Rep = 0 at the expense of risking waiting the full finalization period for the result to be available to the arbitrable contract. Note that this is not a viable strategy for a malicious plaintiff to delay a result being reported as if the plaintiff is ruled incorrect she must pay RepComp to the reporter, and an interested third-party can always act as the reporter for no reward. Note that if a report is unsuccessfully disputed, then the report is compensated for his increased period of capital lockup by receiving RepChStake from the report challenger[1].

In some sense, *ideally*, RepChStake would increase as a function of the timestamp of the report challenger's transaction so that it is generally the cost of appeal fees for an additional appeal on Rollup B. (This does not need to be exact, so rather than attempting to use cross-rollup communication to determine what stage of an appeal one might be in, the contract can just guess how long the rounds tend to take and apply a step function to the deposit size based on the internal clock of Rollup A.) Then the cost to cause a delay of some fixed time would be roughly the same whether doing so by forcing the arbitrable contract to wait for finalization of Rollup B or by triggering an additional appeal. However, this might require unacceptably capital lockup to challenge a report, so some cap on this might be necessary to not facilitate malicious reports to get through. Maybe just keep RepChStake constant.

0.4 Non-atomicity and raising a dispute on Rollup B

Remark 1. *Note that a common challenge in many scaling solutions is a lack of "atomicity". On L1 a transaction can be constructed so as to make two state changes either together on different contracts, or to back out throwing an error and making neither state change. However, as soon as these two state changes are no longer part of the same "consensus", because they belong to different chains, rollups, shards, etc, this is no longer possible. For Kleros, with its days-long decision times, this is generally not a huge issue. However, there is one*

[1] We have also considered having a required component of the defendant and plaintiff deposits RepComp that is given to a successful report only if that party loses. This can be problematic if there is no dispute, in which case there is no losing party, or even if there is a dispute but the defendant wins, as this would require further cross-rollup communication to allow a reporter to claim their reward on Rollup B based on reporting actions on Rollup A. Hence, the additional incentivization of reporters this choice would allow might not justify the increased cross-rollup complexity.

point on which it does seem to be important and around which one will have to adapt the architecture of Kleros contract. For a typical current L1 Kleros arbitrable contract, the following two operations are performed atomically:

- *calling the arbitrator contract and paying arbitration fees to trigger a dispute, and*
- *notifying the arbitrable contract that a dispute has been raised, in particular so that it takes on a pending state until it receives a response from the arbitrator, and no further transactions will be able to raise arbitration on this same case.*

In a non-atomic setting, these operations must be disconnected. In particular, this potentially raises the issue that one might accidentally double raise arbitration on a case, e.g. Bob and Charles could both try to create a dispute at the same time.

Then, a potential issue is how to make Rollup B *aware* of a case that can be potentially disputed coming from Rollup A, conveying such information as the question to be asked to jurors. Ideally, one would make the disputeID of a raised dispute available to the arbitrable contract so that it can easily identify whether a ruling that it is presented corresponds to its dispute. In `https://github.com/kleros/cross-chain-arbitration-proxy/#high-level-algorithm` this setup is done by having a back-and-forth between the proxies on the two chains. However, this does not necessarily seem viable in an optimistic rollup framework, where again cross-rollup communication requires long finalization times.

We propose the following approach:

- The information on the dispute can be provided by the defendant. Then the result of the dispute will only be recognized as valid by the arbitrable contract on Rollup A if (a hash of) this information matches the question, etc that the arbitrable contract expects.
- This information should include:
  - Any information required to raise the dispute, such as the question to be posed meta-evidence, etc.
  - The plaintiff's address[2]
  - The L1 timestamp at which the plaintiff's transaction is sequenced, T_1, see below.
  - The TX hash of the plaintiff's transaction.

[2]This is required so that if the plaintiff is ultimately judged correct, Rollup B knows to whom the defendant's deposit should be transferred. If the defendant attempts to manipulate this information it should invalidate the use of the dispute by the arbitrable application similarly to if he altered the question.

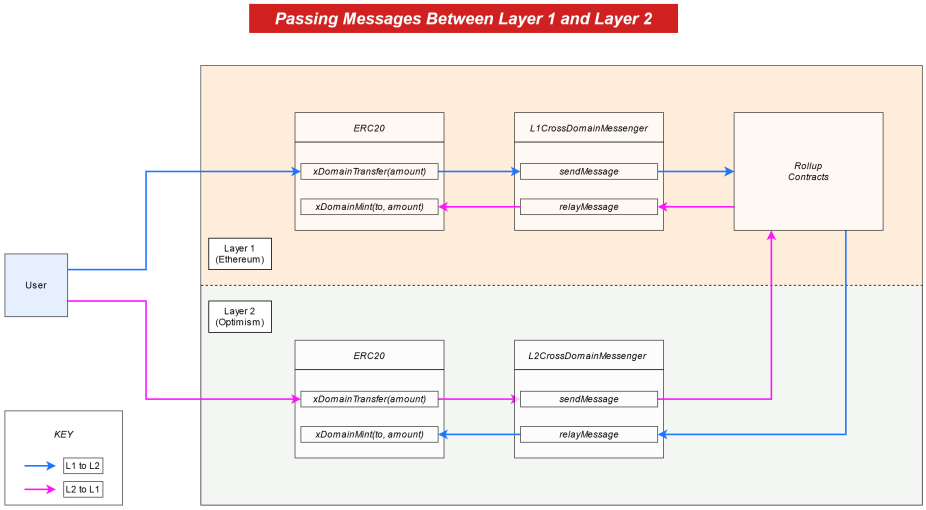

Figure 1: Model of L1-L2 communication in Optimism. From `https://github.com/ethereum-optimism/Buidler-Deposit-Withdraw-Example`

- Then one would further want some mechanism on Rollup B that will block additional disputes from being raised on the same case to avoid duplicate disputes. For instance, the hash of the information the defendant provides on the dispute can be used to index a hash table. Then if a transaction attempts to raise another dispute with matching information (particularly noting, barring hash collisions, that the plaintiff's transaction hash should be unique), this should give an error.

Then we consider the scheme in the tables below.

0.5 Properties of scheme

Then, we have:

Proposition 1. *Assume standard liveliness and consistency properties on L1. Then, even if Rollup A requires L1 finalization following a fraud proof during the course of the dispute, the court's decision is still executed by the arbitrable contract. Similarly, further suppose there is some mechanism to notify the contract on Rollup A of the existence of an ongoing fraud proof related to Rollup*

[3]Footnote to step 8 of protocol: It is not totally clear to me from Optimism documentation whether there are issues involved in giving Rollup A access to T_1 and T_2, which recall are L1 timestamps of Rollup A sequencing transactions. Note these values are only needed in situations where a challenged report forces the arbitrable contract to wait for finalized data from Rollup B. In the worst case could probably replace T_1 and T_2 with bounds involving OVM timestamps and F.

	Rollup A		Rollup B
1.	All of the normal interactions in an arbitrable contract that one might have prior to the step of asking for arbitration. For an element that can be objected to, there is some time limit t_1 (in terms of the clock of Rollup A) in which this objection can be made before it is finalized. (E.g. before an element is accepted onto a list, before an escrow pays out, etc.)		
2.	Before t_1, the plantiff objects to the pending result. She pays a deposit of PlaintiffStake + RepTip (on Rollup A) that she will lose if an eventual arbitration determines that the pending result was correct. Take T_1 to be the L1 timestamp of the block in which this transaction is sequenced (but not finalized) by the Rollup A sequencer.		
		3.	The defendant sends a transaction to the relayer on Rollup B along with a deposit of ArbFees + DefStake + CrowdFundComp. This transaction should include all of the information specified in Section 0.4 and result in an error of the hash table lookup step indicates that this is a duplicate dispute. Furthermore this transaction pays for the arbitration fees to raise a dispute that challenges the plaintiff's objection. Denote by K_1 the number of seconds that the defendant should be allowed to complete this transaction. The relayer should have logic that this transaction must be included by the sequencer of Rollup B (potentially via an enqueue transaction) with an OVM timestamp of at latest $T_1 + K_1$, otherwise it produces an error for exceeding the time limit to challenge.

Rollup A	Rollup B
	4. The court contract handles the dispute in a similar way to present courts, entirely on Rollup B. In particular, (crowdfunding of) appeals and evidence submission should take place entirely on Rollup B. 5. Once a decision is reached, incentives to participants on the arbitrator side (namely jurors and in this schema crowdfunders) are paid out. The outcome, the OVM timestamp t_2 at which this step is executed, and the address of the defendant should be written in state on Rollup B so that (after a finalization period) they would be available on Rollup A.
6. At any point after step 2, a reporter can place the deposit RepStake on the claim that either • There is not and will not be any defendant transaction to raise a dispute with a OVM timestamp bounded by $T_1 + K_1$ on Rollup B. (Particularly, the respondent can know this if the sequencer of rolllup B has made at least one sequence submission beyond timestamp T_1+K_1+F, and and no defendant has yet paid arbitration fees.) or • that arbitration has finalized and a given option has won. (The reporter should typically wait until the sequencer on Rollup B makes a transaction with timestamp greater than t_2+F so that he can sure that "earlier" transactions aren't added that will change the result.) In this case the reporter should also provide the address of the defendant. Suppose this transaction has OVM timestamp t_3 and is committed to by the Rollup A sequencer (but not finalized) in the block with L1 timestamp $T_2 \geq t_3$.	

	Rollup A		Rollup B
7.	Prior to a deadline of OVM timestamp $t_3 + K_2$ for some K_2, a report can be challenged also requiring placing a deposit of RepChStake. Then the arbitrable contract remains unresolved until the following step when finalized information from Rollup B is available.		
8.	A challenged report is handled as follows. Note that, if necessary, when receiving communication from Rollup B, the arbitrable contract could verify that information corresponds to the right dispute by checking that (some hash of) the question statement, etc, match the dispute on the arbitrable contract. • If the challenge is to the claim that there is no dispute due to a lack of defendant paying fees, – A timer of K_3 seconds begins in terms of the clock of Rollup A. K_3 should be longer than the finalization time of Rollup B. – If the contract receives a message attesting to finalized state of a dispute being raised (without it necessarily needing to have been resolved) on Rollup B with OVM timestamp at most $T_1 + K_1$, then the report is invalidated.	8.	As necessary, messages indicating finalized appeals of the case are sent from Rollup B to Rollup A. When available a message attesting to a finalized result (as well as the timestamp t_2 and the defendant's address) is sent from Rollup B to Rollup A.

Rollup A	Rollup B
• – If the OVM timestamp $T_1 + K_1 + K_3$ passes on Rollup A without such a message arriving, then the report is considered valid, in the sense that an execution transaction on Rollup A with a timestamp greater than $T_1 + K_1 + K_3$ can then resolve incentives and the winner in the arbitrable contract. (It is possible until $T_1 + K_1 + K_3 + F$ that some transaction with an earlier timestamp will be sequenced; however then this execution would just throw an error.) • If the contract has not resolved as per above, then the arbitrable contract waits to receive a finalized result. If the finalized result or the defendant's address do not match the claimed result in the report, the report is invalidated. • The contract checks whether $t_2 \leq T_2$ or not. If $t_2 > T_2$ then the report is invalidated as it was made too early[3].	
9. Incentives are paid out based on the outcome as in Section 0.3. In particular, if the defendant wins, he is compensated for the fees that he paid on Rollup B by receiving the plaintiff's deposit on Rollup A.	9. Incentives are paid out based on the outcome as in Section 0.3. In particular, if the plaintiff wins, she is compensated for the risk she took on her deposit on Rollup A by receiving the defendant's deposit on Rollup B minus arbitration fees.

B so that the Rollup A contract waits longer. Then, if Rollup B requires L1 finalization following a fraud proof during the course of the dispute, the court's decision is still executed by the arbitrable contract.

Proof. (Sketch of proof) If Rollup A requires a fraud proof, the valid transactions will eventually finalized (potentially on L1). At this point the finalized transaction from Rollup B would be available. Similarly, based on the assumption in Section 0.1 of an honest actor, even if Rollup B requires a fraud proof, it too will eventually finalize to correct information. Then as long as the contract on Rollup A does not time out before this result is provided, the ruling will still be executed.

□

Proposition 2. *Suppose*

$$K_3 \geq F + \begin{array}{c}\textit{rollup } B\\ \textit{finalization delay}\end{array} + \begin{array}{c}\textit{longest timestamp delay to include}\\ \textit{enqueue transaction in a block}\end{array}.$$

Then, rollups A and B are coherent on whether or not the defendant's transaction is before the time limit. Namely, one can convince the contract in step 8 that there is no defendant's transaction in step 3 if and only if there is no dispute.

Proof. Using Proposition 1, we can assume that the sequencers of Rollups A and B submit valid transactions and that there are no frauds proofs on L1 during the relevant period. If the transaction in step 3 has a timestamp of at most $T_1 + K_1$, then it will be sequenced in a block whose L1 timestamp is at most $T_1 + K_1 + F$. So this is finalized at latest at

$$T_1 + K_1 + F + \text{finalization delay}.$$

Then, if necessary, one can issue an enqueue transaction constructing the required message to attest on Rollup A to a dispute having been raised by

$$T_1 + K_1 + F + \begin{array}{c}\text{rollup B}\\ \text{finalization delay}\end{array} + \begin{array}{c}\text{longest timestamp delay to include}\\ \text{enqueue transaction in block}\end{array}.$$

On the other hand, if the transaction has an OVM timestamp on Rollup B that is beyond the time limit of $T_1 + K_1$ so that no dispute is raised, then this transaction will also be accepted in step 8.

□

Proposition 3. *Suppose*

$$K_3 \geq F + \begin{array}{c}\textit{rollup } B\\ \textit{finalization delay}\end{array} + \begin{array}{c}\textit{longest timestamp delay to include}\\ \textit{enqueue transaction in a block}\end{array}.$$

Then the defendant has at least K_1 blocks to pay arbitration fees after the prosecution transaction in step 2 becomes publicly available. In particular, if the defendant includes his transaction in step 3, any reporter who claims that he did not submit arbitration fees on time in step 6 can be successfully challenged.

Proof. The plaintiff's transaction is guaranteed to become publicly available (as calldata) at T_1. Then, if necessary, the defendant can place an enqueue transaction so that the dispute is raised before $T_1 + K_1$ on Rollup B. Then by Proposition 2, any claims on Rollup A that the defendant did not pay fees in time can be successfully challenged.

□

Proposition 4. *In the timestamp check in step 8, if $t_2 > T_2$ then the report was made before the transaction executing the result of the case on Rollup B was sequenced. On the other hand, if the reporter waits until he is confident that a sequencing transaction by the Rollup B sequencer with OVM timestamp greater than $t_2 + F$ will be included in the ultimate L1 chain, then $t_2 \leq T_2$, and the report will not be ruled invalid based on its timestamp.*

Proof. We consider the perspective of time as divided into the L1 timestamps of blocks that are ultimately included in the main chain. Using Proposition 1, we can assume that the sequencers of Rollups A and B submit valid transactions and that there are no frauds proofs on L1 during the relevant period. Then if $t_2 > T_2$, we have

$$\begin{array}{c}\text{L1 timestamp of block} \\ \text{with transaction sequencing execution} \\ \text{of dispute result from step 5}\end{array} \geq t_2 > T_2 \geq \begin{array}{c}\text{L1 timestamp of first} \\ \text{block at or after report}\end{array}.$$

On the other hand, if the reporter waits until he sees a sequencing transaction from Rollup B with OVM timestamp greater than $t_2 + F$ in a block which he is confident will ultimately be included in the main chain, then he knows that no other transactions will be able to included by the Rollup B sequencer with OVM timestamps less than or equal to t_2. Then,

$$t_2 \leq \begin{array}{c}\text{L1 timestamp of block} \\ \text{with transaction sequencing execution} \\ \text{of dispute result from step 5}\end{array} \leq \begin{array}{c}\text{L1 timestamp of block with} \\ \text{sequencing transaction with OVM} \\ \text{timestamp greater than } t_2 + F\end{array} \leq T_2.$$

□

Note that a disadvantage of this scheme is that the plaintiff and defendant of a case do not generally receive their rewards on the same rollup as they place their deposit in. So to the degree that users are more active in one of these roles than the other, they will need to periodically send (wrapped) ETH between the two rollups.

Remark 2. *Note that arbitration fees can change between the time when a plaintiff makes a transaction on Rollup A and when a defendant makes a transaction on Rollup B. In order to "notify" the plaintiff on Rollup A that she must adapt to a fee change would require another round of cross-rollup communication. While conceivably possible (such as by including additional reporting possibilities in step 6), for the moment this schema assumes that the defendant*

will adapt to changes in arbitration fees, paying higher fees if required or potentially benefiting from lower fees if fees decrease. The plaintiff and defendant deposits should be set so that parties have an interest in participating even with some limited variation in fees.

Remark 3. *Note that there might be dApps that remain on L1 that would want to obtain rulings from the Kleros court. The above scheme works largely unchanged in this case, with the OVM timestamps on Rollup A being replaced with L1 timestamps; in particular $t_3 = T_2$. Unfortunately, there do not seem to be many points in the scheme where we could take advantage of the dApp being on L1 to obtain simplifications. This is because, as a defendant can always provide the information about a dispute on Rollup B himself, the important transfers of information between the court and the arbitrable application are all in the direction of going from the court to the arbitrable application. One might have hoped to recombine steps 2 and 3, so that a single party could issue an L1 transaction that atomically pauses the arbitrable application and issues an enqueue transaction to raise the dispute; this would have allowed more flexibility in the plaintiff/defendant model presented above in these cases. However, even here, the arbitrable application cannot know that the part of the transaction raising the dispute has gone through successfully. Particularly, if there is a fee change on the court, a dispute may not be raised due to inadequate fees being paid even as the arbitrable application would be paused expecting a dispute to be pending.*

As a parameter change on the court seems to be the only reason that such a transaction would fail, it might be conceivable to require some kind of L1 transaction to trigger each court parameter update. This would give a marker visible to applications on L1 that would allow them to know that there had not been a parameter change in a given period of time. Then the arbitrable application could trust that its transactions raise expected disputes. This could allow for models, as we currently have with Realitio, where a single party could execute steps 2 and 3 atomically.

Remark 4. *In the above scheme, we have assumed that both Rollup A and Rollup B have the structure of Optimism. It is, of course, possible that a dApp using Kleros would be deployed on some other optimistic rollup. This should not pose significant challenges; our assumptions on Rollup A were fairly general. The main points that might cause issues would be around how this rollup handles timestamps.*

It is somewhat less clear currently how an arbitrable application on a zk-rollup might obtain a ruling from Kleros. This might depend somewhat on the exact structure of that zk-rollup. At worst, it would seem possible for decisions to be provided on L1 by a reporter as described above, possibly in some kind of batched, Merkelized form, so that the applications on the zk-rollup would have access to them following a reporting challenge period.

0.6 Possible approaches to relax plaintiff/defendant structure

In this section, we consider how one might relax the plaintiff/defendant structure required by the above schema, at least in some special cases. While the structure of having two conflicting parties taking opposite sides of a dispute, and ultimately directly or indirectly covering the arbitration fees when their side loses, is frequent in applications of Kleros, there exist some situations where it is not appropriate. Particularly, in the current integration with Reailitio, disputes are raised by a single party that pays all arbitration fees. Typically this party is incentivized by an interest in an external application of an oracle result, or by the returns from having previously posted bonds on Realitio.

Some possible approaches for how the Kleros/Realitio integration could be maintained in a future scaled version are the following:

- A simple approach would be to just convince Realitio to implement itself on the same optimistic rollup as Kleros. Then, when Realitio decisions need to be used on L1 or on other rollups, it seems straightforward to relay them using a similar kind of reporting system as described above for Kleros decisions

- Recall that the main reason that the plaintiff/defendant structure is required in the above schema is that it allows the plaintiff to pause further transactions on the arbitrable application and the defendant to raise a dispute in the court in two different transactions, while minimizing griefs that could come from pausing an arbitrable application improperly, etc. One possibility would be to allow one to make a transaction on Rollup A that blocks further actions (bonds, withdrawals, etc) and, instead of directly triggering a dispute as in the status quo instead offers a "bounty" that can be claimed by a party that triggers a dispute on Rollup B. See the schema below. This approach has the disadvantage that two large deposits (currently equivalent to the arbitration fees of 500 jurors in the general court for Realitio cases, though this could be reduced in subsequent releases that have an appeal structure) are required instead of one. However, as the party that actually triggers the dispute is guaranteed to be paid back out of the "bounty" this is mostly a liquidity issue.

- For another approach, we can develop the ideas of Remark 3. Here a certain, limited number of L1 transactions that include enqueue elements towards both rollups are required. This has the advantage that the transaction can be

  - executed atomically
  - "seen" by both the the arbitrable contract and the court *almost* instantly. Indeed, the sequencers of the respective rollups can only continue to place other transactions before this transaction for F seconds after it is released on L1.

"Bounty on Rollup A" approach

Rollup A	Rollup B
1. Normal interactions in an arbitrable contract, in the case of Realitio this particularly includes posting of bonds.	
2. Alice places a large deposit is placed on Rollup A requesting dispute resolution. Further transactions, specifically posting of bonds, withdrawals, etc are blocked.	
	3. Another party (Bob) triggers dispute by paying all required arbitration fees. Duplicates of disputes are avoided using the hash lockup table approach of above, and giving an error when the hash of a dispute's Realitio question, etc matches that of another dispute. The court contract handles the dispute in a similar way to present courts, entirely on Rollup B. In particular, (crowdfunding of) appeals and evidence submission should take place entirely on Rollup B.
	4. Once a decision is reached, incentives to participants on the arbitrator side (namely jurors and in this schema crowdfunders) are paid out. The outcome, the OVM timestamp t_2 at which this step executed, and Bob's address should be written in state on Rollup B so that (after a finalization period) they would be available on Rollup A.
5. At any point, a reporter can place the deposit RepStake on claims regarding what the outcome of the dispute was. They must include Bob's address. This can be challenged as in the above schema, and the arbitrable contract resolves this as above.	
6. On proceeds as in the preceding schema. Ultimately, Alice's deposit is rewarded to Bob on Rollup A.	

A disadvantage of this approach compared to the status quo is that the rollups cannot time-effectively communicate back information to L1 on whether the enqueue transactions succeeded or reverted. Hence, the transactions must be designed so that they execute by construction, and there aren't circumstances when they would revert. Particularly, as described in Remark 3, L1 would need to be aware of the current court parameters so that sufficient fees are guaranteed to be paid. This constraint, as well as the possibility that further transactions would be sequenced on the rollups in the F second window which could in the case of Realitio lead to additional bonds being posted after the dispute is called but generally not to premature withdrawals, may be acceptable for some applications but not for others.

	L1	Rollup A	Rollup B
Remark 3 approach	2. Alice makes a transaction that includes enqueue calls to both rollups to pause further transactions on Rollup A and raise a dispute on Rollup B.	1. Normal interactions in an arbitrable contract, in the case of Realitio this particularly includes posting of bonds. . 3. The transaction to pause the arbitrable application is sequenced. Possibly further bond transactions were made in the F second delay in which the sequencer can back date transactions. 5. At any point, after step 2, a reporter can make a report of the outcome of the dispute to the arbitrable contract (which is otherwise paused by step 2). One can challenge and resolve these reports as in the proceeding schemes.	3. The transaction to raise the dispute is sequenced. The court contract handles the dispute in a similar way to present courts, entirely on Rollup B, including crowdfunding of appeals, evidence, etc. 4. Once a decision is reached, incentives to participants on the arbitrator side (namely jurors and in this schema crowdfunders) are paid out. The outcome and the OVM timestamp t_2 at which this step executed, should be written in state on Rollup B so that (after a finalization period) they would be available on Rollup A.