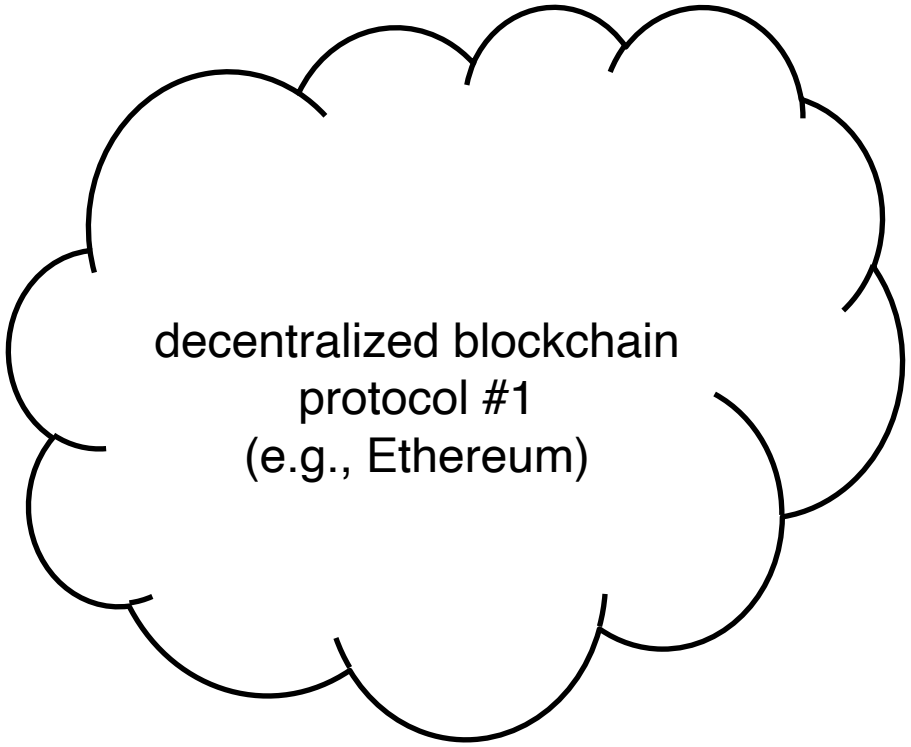
Lecture #19:
Bridges

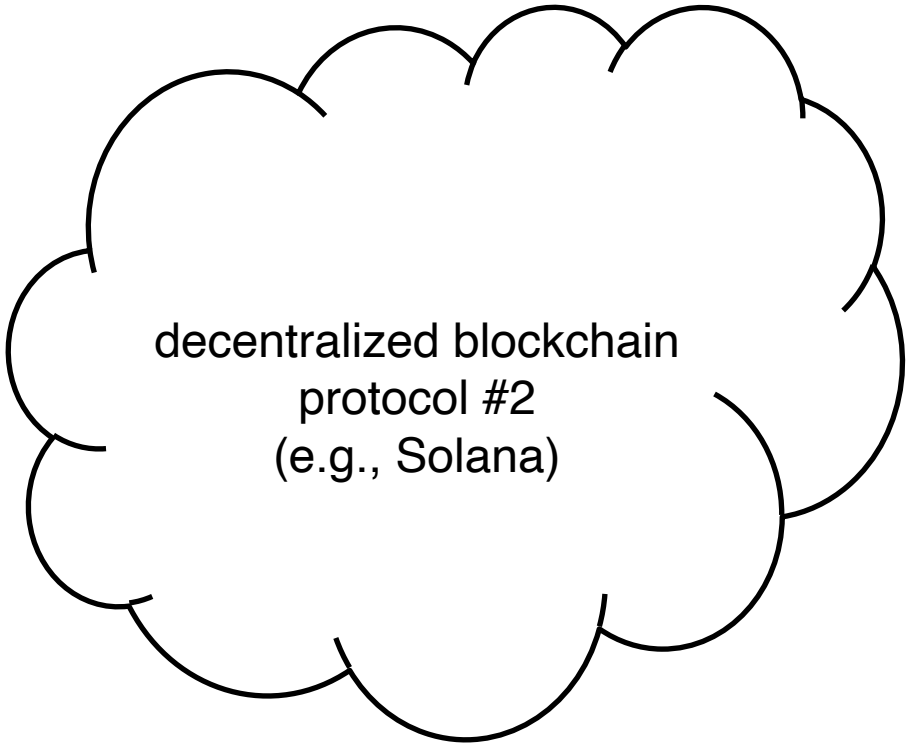
COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

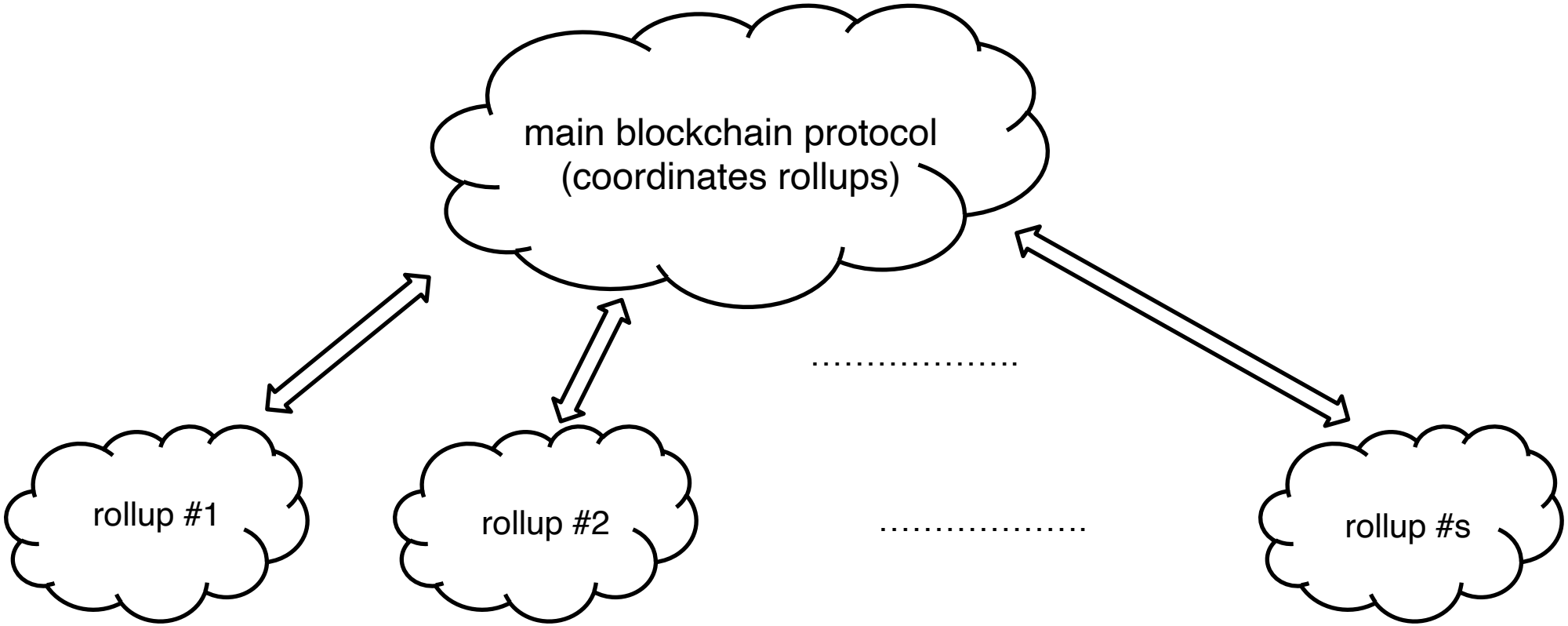
Tim Roughgarden

A Multi-Chain World





Scaling Execution via Rollups

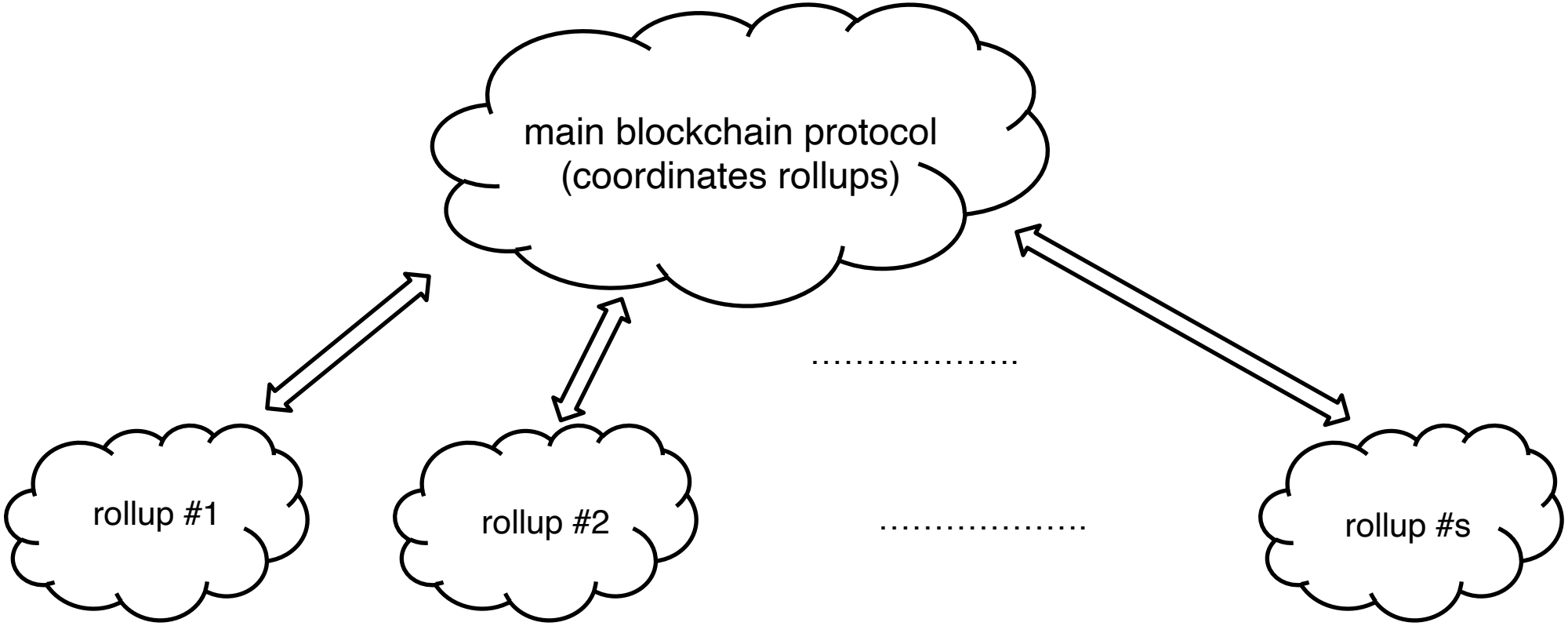


Goals for Lecture #19

1. Rollup bridges.
   - moving assets from an L1 to its rollups via “minting” and “burning"
2. General bridges and cross-chain messaging.
   - how can blockchain Y “know” that some tx executed on blockchain X?
3. Externally validated bridges.
   - rely on third parties to attest to what’s happened on blockchain X
4. “Trustless” bridges.
   - blockchain Y runs (as a smart contract) light client for blockchain X

Rollup Bridges

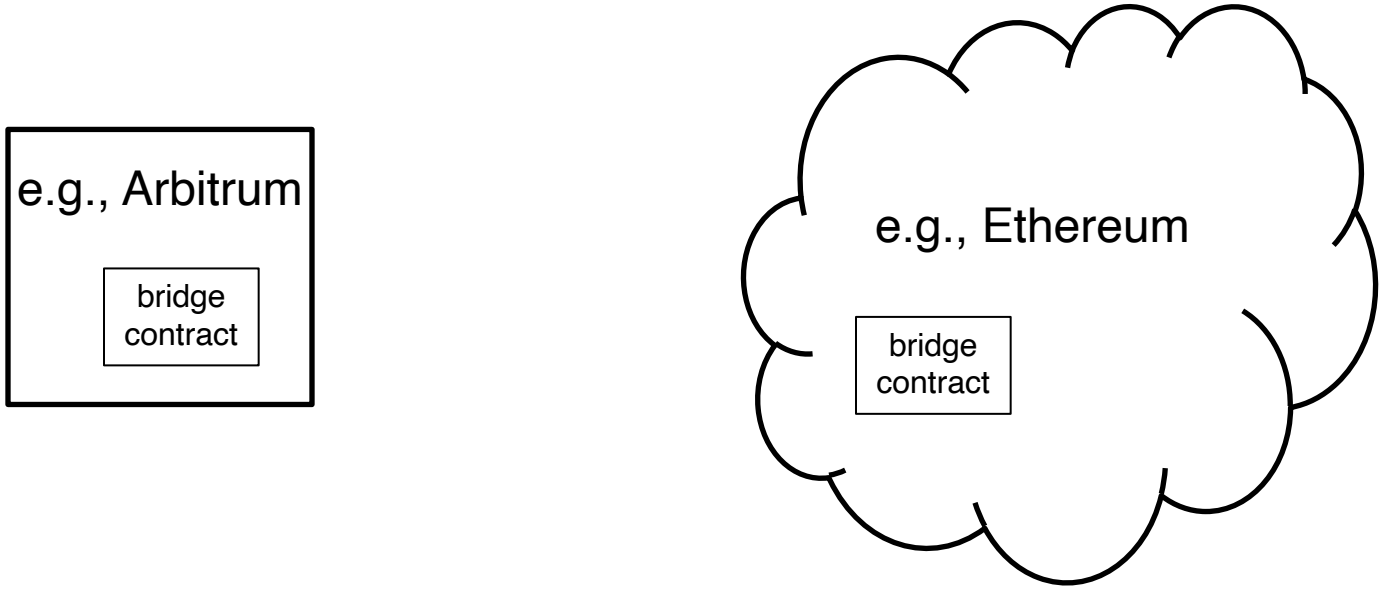
Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.



Rollup Bridges

Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.

Canonical design: coupled “bridge contracts” on L1 and rollup.



Recall: Forced Transaction Inclusion

Requirement for “classic” rollups: forced tx inclusion via the L1.

Recall: Forced Transaction Inclusion

Requirement for “classic” rollups: forced tx inclusion via the L1.

- any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs
  - L1 tx records the specified rollup tx in queue in rollup’s L1 contract
  - next publication of rollup txs must “clear the queue” to be valid

Recall: Forced Transaction Inclusion

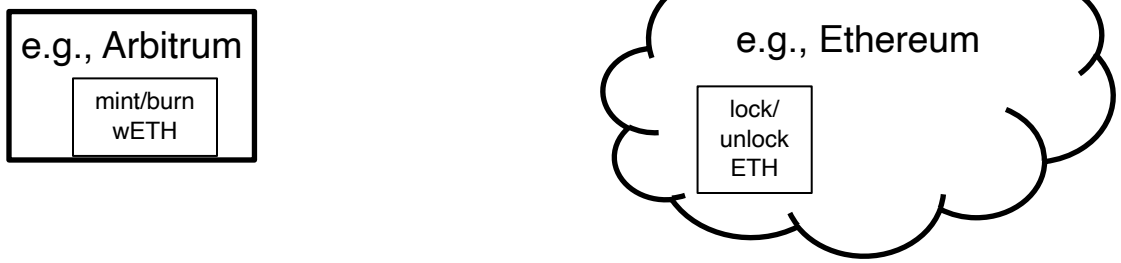
Requirement for “classic” rollups: forced tx inclusion via the L1.

- any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs
  - L1 tx records the specified rollup tx in queue in rollup’s L1 contract
  - next publication of rollup txs must “clear the queue” to be valid
- rollup liveness failure ➔ can use L1 for liveness until reboot completes
- rollup inherits the “censorship-resistance” of the L1

Rollup Bridges

Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.

Canonical design:

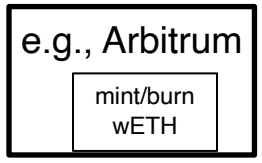


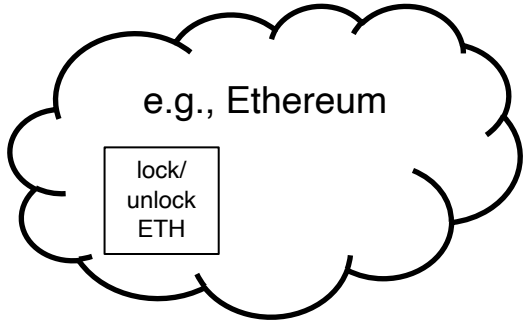
Rollup Bridges

Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.

Canonical design: coupled “bridge contracts” on L1 and rollup.

– L1 contract exports a “deposit” function, any L1 user can send ETH to it



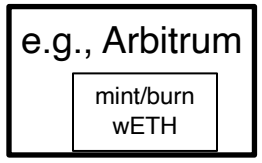


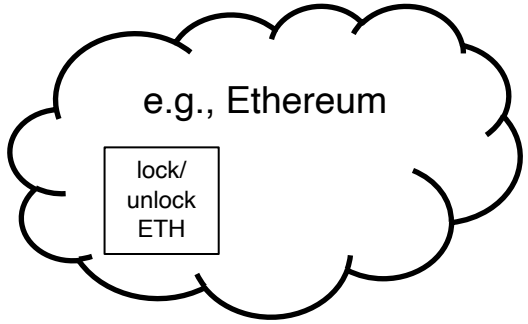
Rollup Bridges

Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.

Canonical design: coupled “bridge contracts” on L1 and rollup.

- L1 contract exports a “deposit” function, any L1 user can send ETH to it
- deposit generates rollup tx to mint equal amt of “wrapped ETH (wETH)”
  - e.g., could add rollup tx to “forced inclusion” list in rollup’s L1 contract



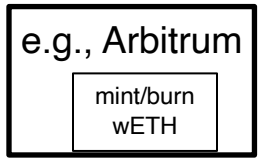


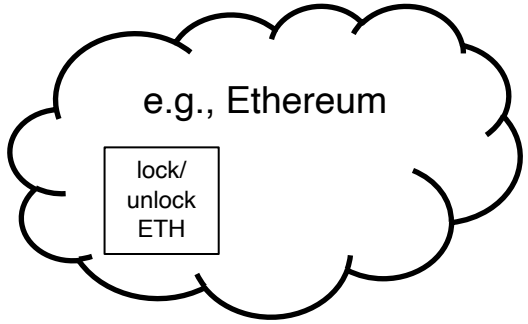
Rollup Bridges

Running example: L1 = Ethereum, user wants to moving native currency ETH from L1 to a (optimistic or validity) rollup.

Canonical design: coupled "bridge contracts" on L1 and rollup.

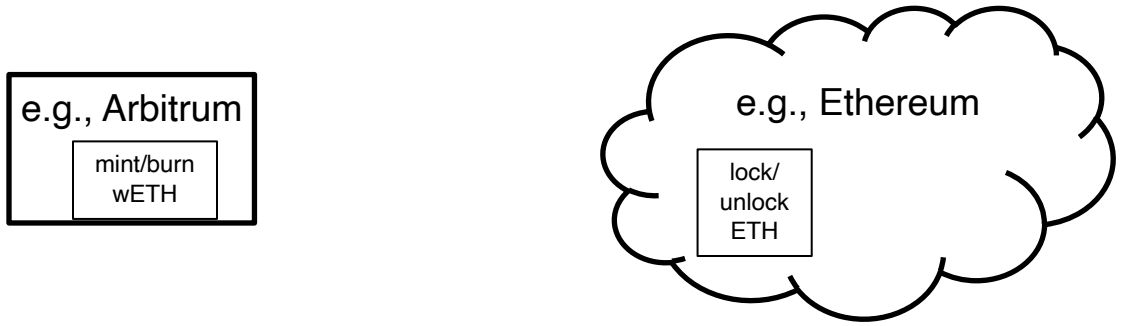
- L1 contract exports a "deposit" function, any L1 user can send ETH to it
- deposit generates rollup tx to mint equal amt of "wrapped ETH (wETH)"
  - e.g., could add rollup tx to "forced inclusion" list in rollup's L1 contract
- one rollup tx executed, new wETH transferred to user's (rollup) account





Rollup Bridges (con’d)

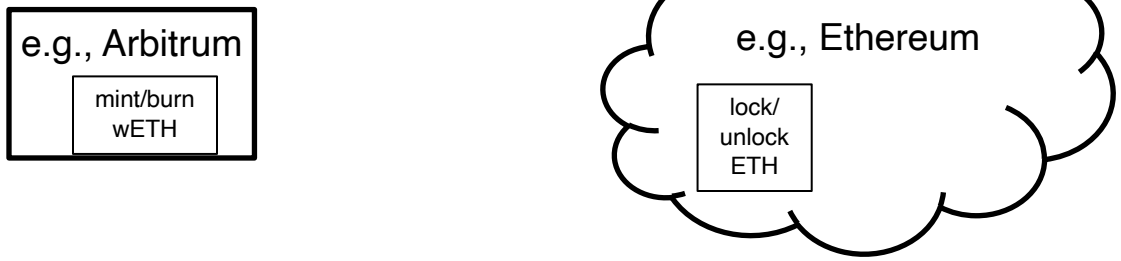
Canonical design (con’d): to “withdraw” ETH from rollup:



Rollup Bridges (con'd)

Canonical design (con'd): to "withdraw" ETH from rollup:

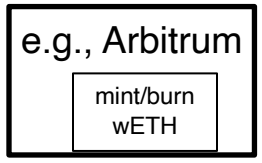
- user sends x wETH to bridge contract on rollup, which is burned
  - note: if necessary, user can submit tx directly to L1 to force inclusion

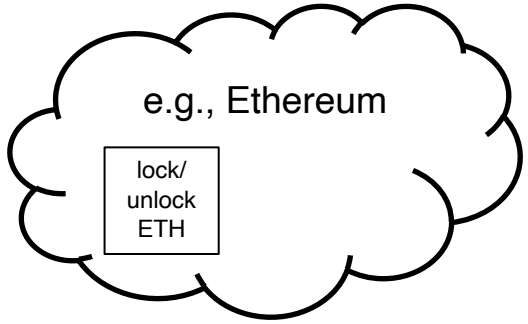


Rollup Bridges (con'd)

Canonical design (con'd): to "withdraw" ETH from rollup:

- user sends x wETH to bridge contract on rollup, which is burned
  - note: if necessary, user can submit tx directly to L1 to force inclusion
- once rollup tx executed and corresponding rollup state root finalized on L1, user submits L1 tx (to rollup's L1 bridge contract) to claim x ETH

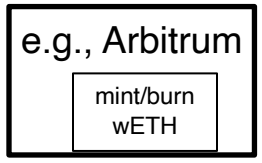


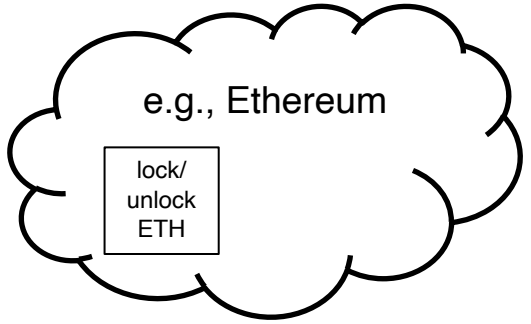


Rollup Bridges (con'd)

Canonical design (con'd): to "withdraw" ETH from rollup:

- user sends x wETH to bridge contract on rollup, which is burned
  - note: if necessary, user can submit tx directly to L1 to force inclusion
- once rollup tx executed and corresponding rollup state root finalized on L1, user submits L1 tx (to rollup's L1 bridge contract) to claim x ETH
  - validity rollups ➔ as soon as requisite SNARK proof posted to L1

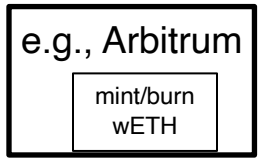


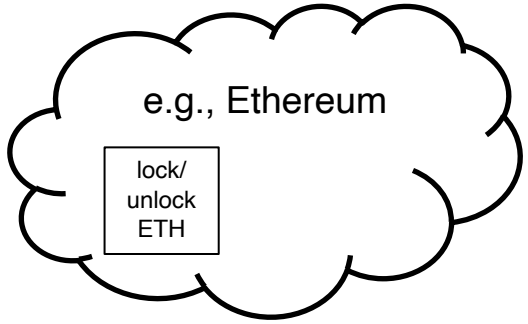


Rollup Bridges (con'd)

Canonical design (con'd): to "withdraw" ETH from rollup:

- user sends x wETH to bridge contract on rollup, which is burned
  - note: if necessary, user can submit tx directly to L1 to force inclusion
- once rollup tx executed and corresponding rollup state root finalized on L1, user submits L1 tx (to rollup's L1 bridge contract) to claim x ETH
  - validity rollups ➔ as soon as requisite SNARK proof posted to L1
  - optimistic rollups ➔ after 7 days, assuming no disputes





Rollup Bridges: Assumptions

Key property: any user can convert ETH (on L1) to wETH (on rollup) and back through submission of suitable L1 and rollup txs.

Rollup Bridges: Assumptions

Key property: any user can convert ETH (on L1) to wETH (on rollup) and back through submission of suitable L1 and rollup txs.

- assumes no bugs in bridge contracts
  - e.g., can't mint unless there's been a corresponding deposit

Rollup Bridges: Assumptions

Key property: any user can convert ETH (on L1) to wETH (on rollup) and back through submission of suitable L1 and rollup txs.

- assumes no bugs in bridge contracts
  - e.g., can't mint unless there's been a corresponding deposit
- assumes consistency and liveness of L1
  - else could freeze/revert rollup updates

Rollup Bridges: Assumptions

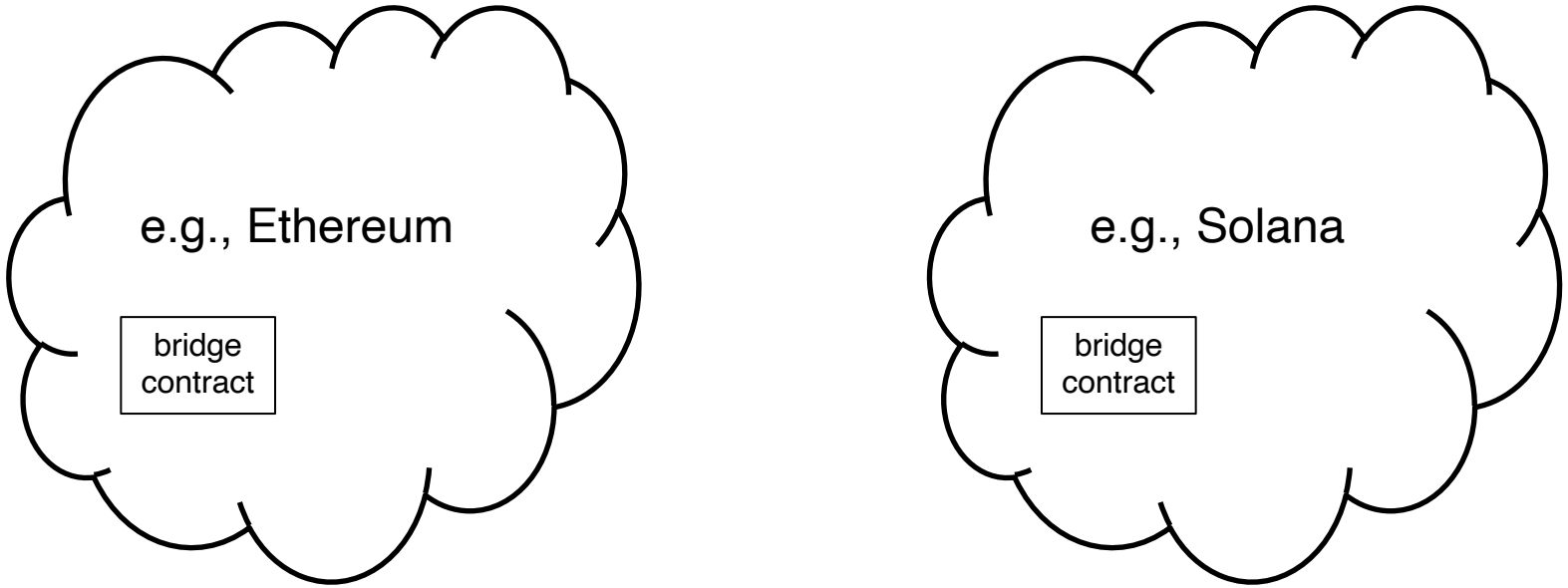
Key property: any user can convert ETH (on L1) to wETH (on rollup) and back through submission of suitable L1 and rollup txs.

- assumes no bugs in bridge contracts
  - e.g., can't mint unless there's been a corresponding deposit
- assumes consistency and liveness of L1
  - else could freeze/revert rollup updates
- assumes liveness of rollup sequencer
  - i.e., eventually posts new state commitments accepted by the L1
  - forced inclusion mechanism ensures withdraws get processed
  - escape hatch: liveness failure ➔ let anyone take over as sequencer

General Bridges

Scenario: two independent blockchain protocols, X & Y.

- e.g., want to transfer assets from one to the other



General Bridges

Scenario: two independent blockchain protocols, X & Y.

- validators of X unaware of Y and vice versa
  - cf., rollup sequencer, very aware of underlying L1
  - bridge contracts generally deployed by third party using only the application layers of X and Y

General Bridges

Scenario: two independent blockchain protocols, X & Y.

- validators of X unaware of Y and vice versa
- no DA: X's txs not posted to Y, Y's txs not posted to X
  - cf., rollup txs posted to underlying L1
  - X, Y each do their own DA (e.g., anyone can run full node for X or Y)

General Bridges

Scenario: two independent blockchain protocols, X & Y.

- validators of X unaware of Y and vice versa
- no DA: X's txs not posted to Y, Y's txs not posted to X
- even if commitments to X's state posted to Y and vice versa, neither blockchain can block the other's commitments
  - cf., L1's ability to block invalid rollup state commitments
    - if rollup fails to execute all the txs in the forced inclusion list
    - if rollup tries to commit a safety violation (i.e., steal funds)
  - though could perhaps at least recognize invalid commitments (more later)

General Bridges

Scenario: two independent blockchain protocols, X & Y.

- validators of X unaware of Y and vice versa
- no DA: X's txs not posted to Y, Y's txs not posted to X
- even if commitments to X's state posted to Y and vice versa, neither blockchain can block the other's commitments

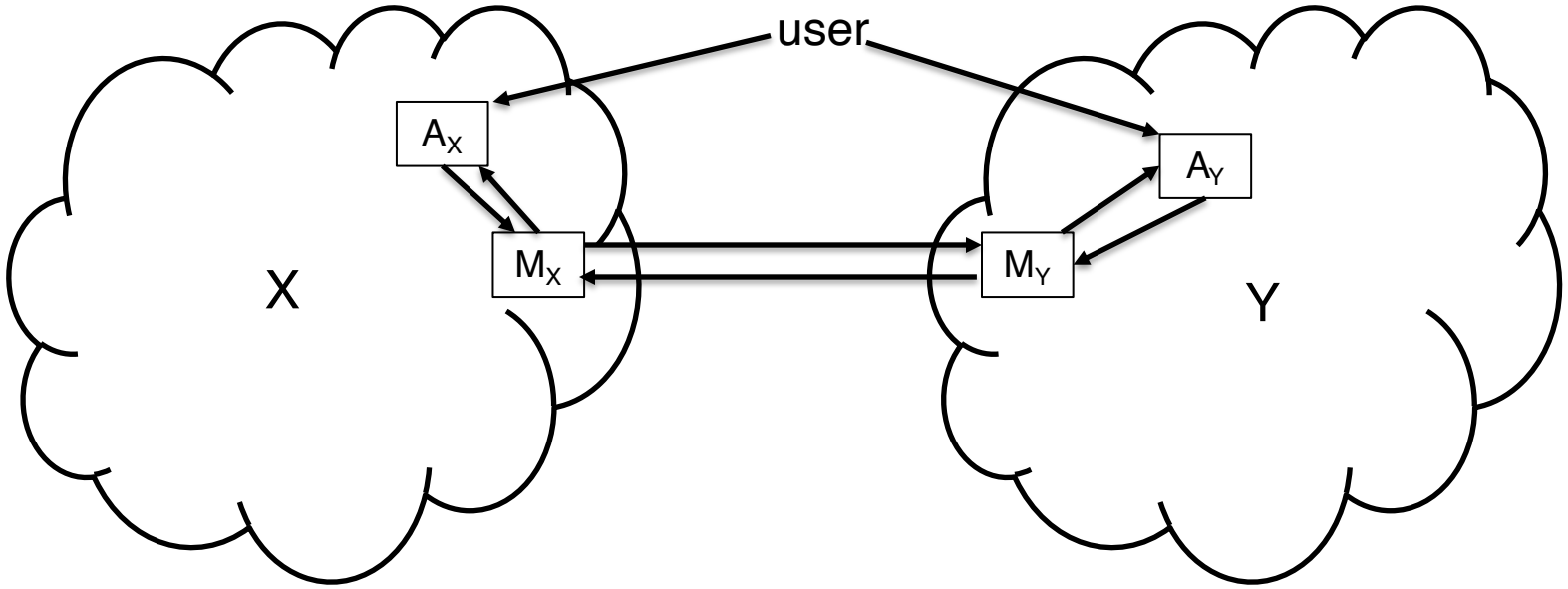
Summary: ground truth for a rollup's state controlled by underlying L1, ground truth for the state of independent blockchains X and Y is controlled by themselves (i.e., their validators).

  - most general bridges resort to additional trust assumptions

Cross-Chain Messaging System

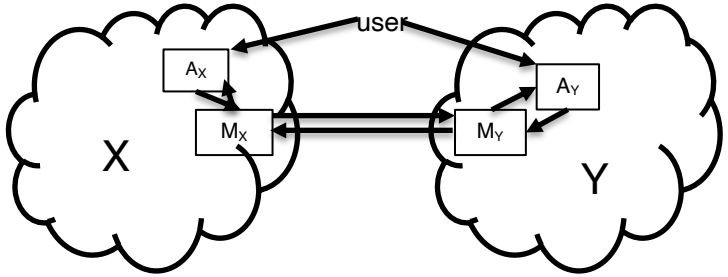
General solution: cross-chain messaging system.

- pair (M_x,M_y) of coupled contracts that allow other contracts (e.g., bridge contracts) on X and Y to “send messages” to each other



Cross-Chain Messaging System

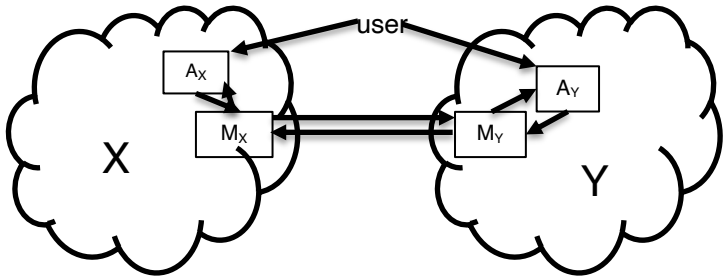
Ideal communication flow: (X → Y)



Cross-Chain Messaging System

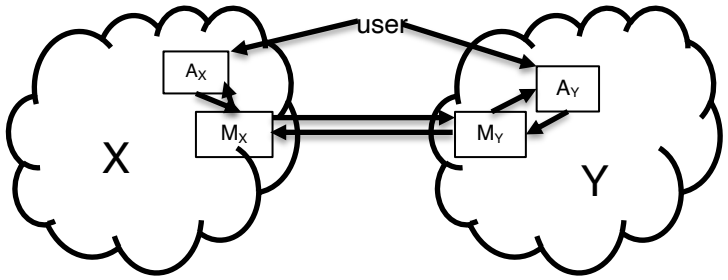
Ideal communication flow: (X → Y)

- Alice sends tx to A_X (e.g., deposit k coins)



Cross-Chain Messaging System

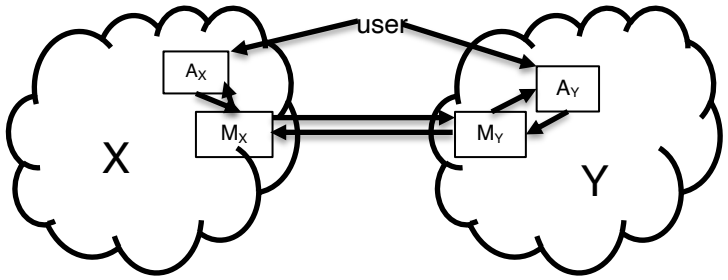
Ideal communication flow: (X → Y)



- Alice sends tx to A_X (e.g., deposit k coins)
- A_X puts msg in M_X's outbound msg queue:
  < recipient = A_Y, data = "Alice deposited k coins" >

Cross-Chain Messaging System

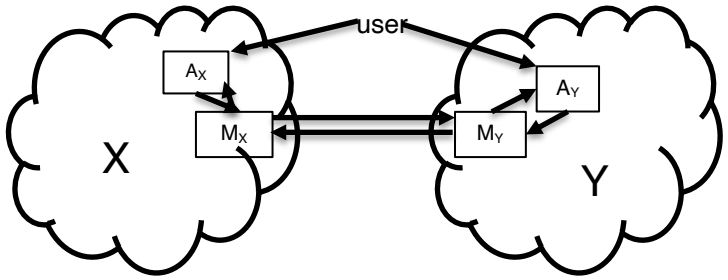
Ideal communication flow: (X → Y)



- Alice sends tx to A_X (e.g., deposit k coins)
- A_X puts msg in M_X's outbound msg queue:
  < recipient = A_Y, data = "Alice deposited k coins" >
- [abstraction] X signs msg, M_X sends signed msg to M_Y

Cross-Chain Messaging System

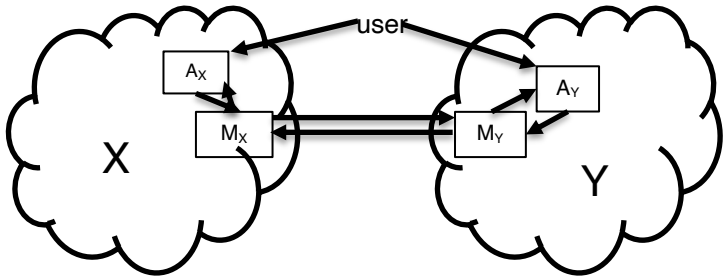
Ideal communication flow: (X → Y)



- Alice sends tx to A_X (e.g., deposit k coins)
- A_X puts msg in M_X's outbound msg queue:
  < recipient = A_Y, data = "Alice deposited k coins" >
- [abstraction] X signs msg, M_X sends signed msg to M_Y
- when M_Y receives msg < A_Y, m>, verifies signature (by X) ➔ if valid, sends payload m to recipient contract A_Y
  - e.g., A_Y mints k wrapped coins and sends to Alice's account on Y

Cross-Chain Messaging System

Ideal communication flow: (X → Y)

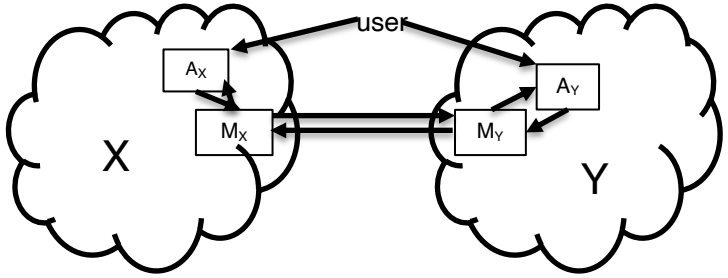


- Alice sends tx to A_X (e.g., deposit k coins)
- A_X puts msg in M_X's outbound msg queue:
  < recipient = A_Y, data = "Alice deposited k coins" >
- [abstraction] X signs msg, M_X sends signed msg to M_Y
- when M_Y receives msg < A_Y, m>, verifies signature (by X) ➔ if valid, sends payload m to recipient contract A_Y
  - e.g., A_Y mints k wrapped coins and sends to Alice's account on Y
  - in effect, A_Y trusting X's signature that Alice's tx finalized on X

Messaging System ➔ Bridge

Note: can build a bridge (to transfer assets) from such a messaging system.

- lock/unlock via A_X, mint/burn via A_Y

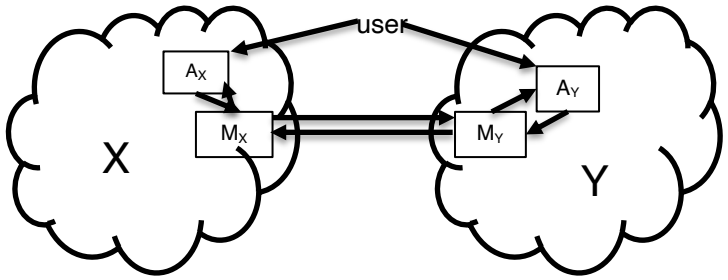


Assumptions:

Messaging System ➔ Bridge

Note: can build a bridge (to transfer assets) from such a messaging system.

- lock/unlock via A_X, mint/burn via A_Y



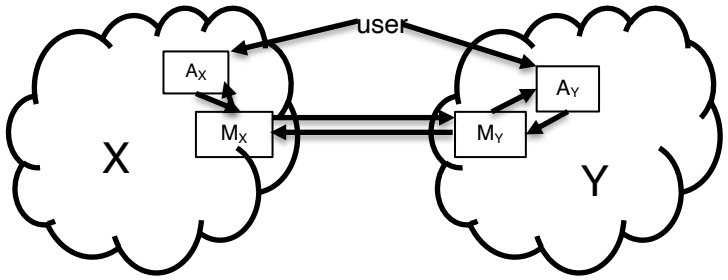
Assumptions:

- no smart contract bugs
  - e.g., A_X doesn't send fabricated deposits to M_X

Messaging System ➔ Bridge

Note: can build a bridge (to transfer assets) from such a messaging system.

- lock/unlock via A_X, mint/burn via A_Y



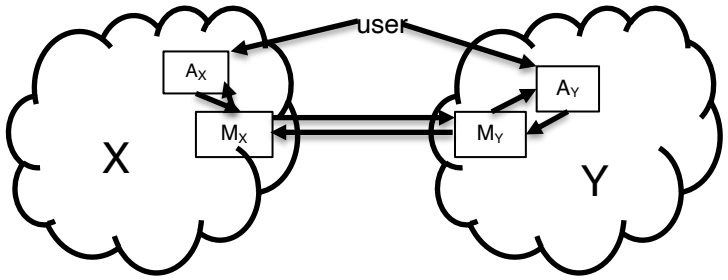
Assumptions:

- no smart contract bugs
  - e.g., A_X doesn't send fabricated deposits to M_X
- X, Y both consistent and live

Messaging System ➔ Bridge

Note: can build a bridge (to transfer assets) from such a messaging system.

- lock/unlock via A_X, mint/burn via A_Y



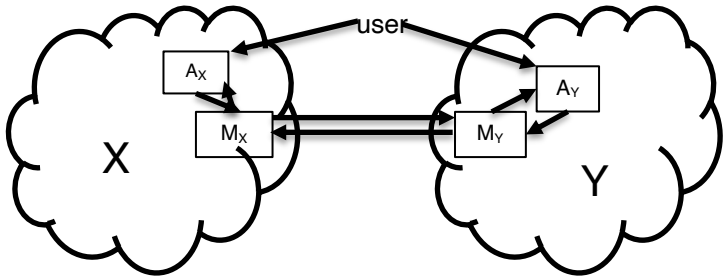
Assumptions:

- no smart contract bugs
  - e.g., A_X doesn't send fabricated deposits to M_X
- X, Y both consistent and live
- X, Y both sign + send messages whenever asked to (liveness)

Messaging System ➔ Bridge

Note: can build a bridge (to transfer assets) from such a messaging system.

- lock/unlock via A_X, mint/burn via A_Y



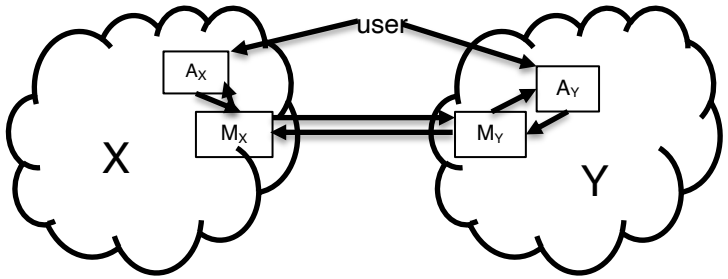
Assumptions:

- no smart contract bugs
  - e.g., A_X doesn't send fabricated deposits to M_X
- X, Y both consistent and live
- X, Y both sign + send messages whenever asked to (liveness)
- X, Y don't sign + send messages they're not asked to (safety)

Signing and Sending Messages

Recall abstraction: (X → Y)

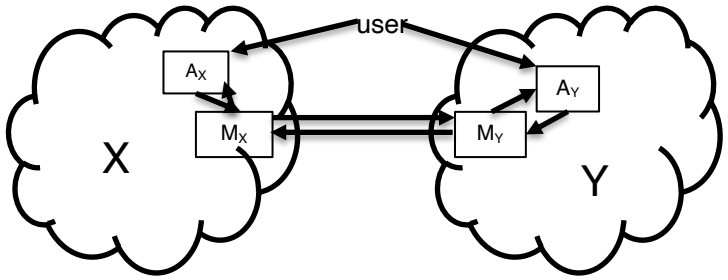
- X signs msg, M_X sends signed msg to M_Y



Signing and Sending Messages

Recall abstraction: (X → Y)

- X signs msg, M_X sends signed msg to M_Y



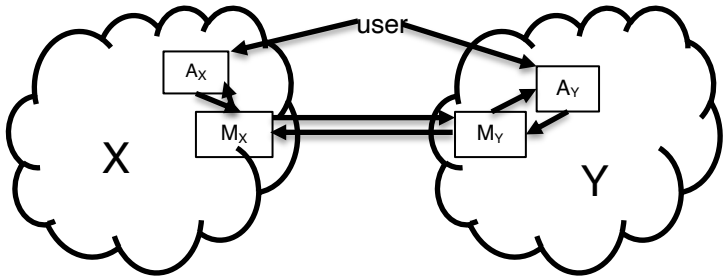
Questions:

1. what does it mean for a blockchain protocol to “sign a msg”?
2. who relays signed messages between X and Y?

Signing and Sending Messages

Recall abstraction: (X → Y)

- X signs msg, M_X sends signed msg to M_Y



Questions:

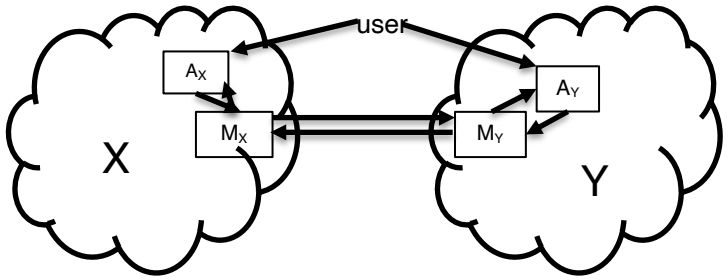
1. what does it mean for a blockchain protocol to “sign a msg”?
2. who relays signed messages between X and Y?

Answer to question #2: user themselves, or a third party.

- important, but not the hard part
- next: menu of answers to question #1

Externally Validated Bridges

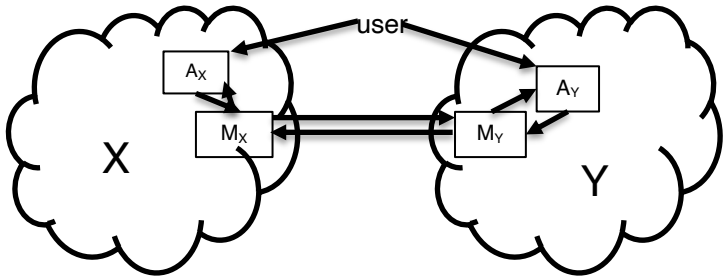
Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



- e.g., that Alice really did lock k coins in A_X
- proxy for "<A_Y,m> signed by X"
- ideally, app-specific (each pair (A_X, A_Y) specifies its own rules)

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.

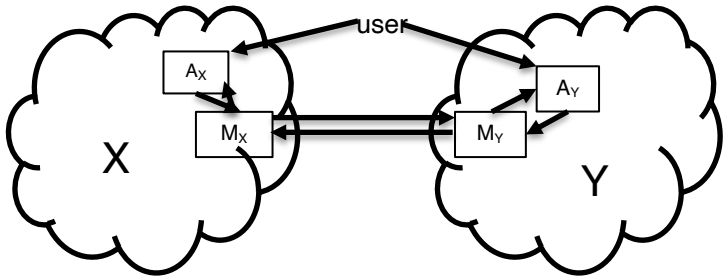


- e.g., that Alice really did lock k coins in A_X
- proxy for "<A_Y,m> signed by X"
- ideally, app-specific (each pair (A_X, A_Y) specifies its own rules)

Solution #1: trusted third party (TTP).

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



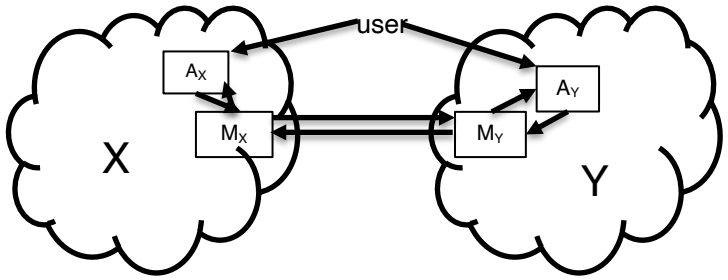
- e.g., that Alice really did lock k coins in A_X
- proxy for "<A_Y,m> signed by X"
- ideally, app-specific (each pair (A_X, A_Y) specifies its own rules)

Solution #1: trusted third party (TTP).

- TTP's pk hard-wired into A_Y
  - TTP responsible for monitoring all confirmed txs (on X) involving A_X

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



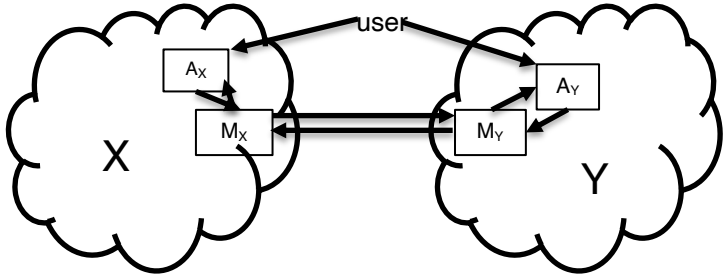
- e.g., that Alice really did lock k coins in A_X
- proxy for "<A_Y,m> signed by X"
- ideally, app-specific (each pair (A_X, A_Y) specifies its own rules)

Solution #1: trusted third party (TTP).

- TTP's pk hard-wired into A_Y
  - TTP responsible for monitoring all confirmed txs (on X) involving A_X
- message <A_Y,m> regarded as valid by M_Y/A_Y ⇔ signed by TTP

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



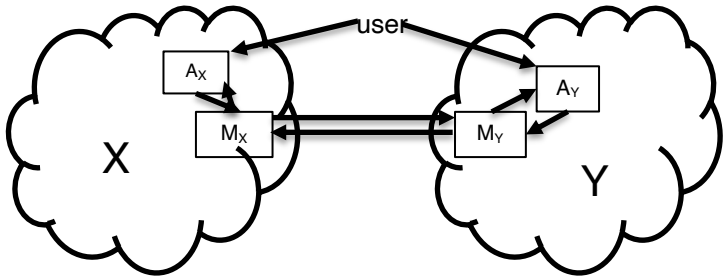
- e.g., that Alice really did lock k coins in A_X
- proxy for "<A_Y,m> signed by X"
- ideally, app-specific (each pair (A_X, A_Y) specifies its own rules)

Solution #1: trusted third party (TTP).

- TTP's pk hard-wired into A_Y
  - TTP responsible for monitoring all confirmed txs (on X) involving A_X
- message <A_Y,m> regarded as valid by M_Y/A_Y ⇔ signed by TTP
  - note: safety and liveness both depend entirely on TTP

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.

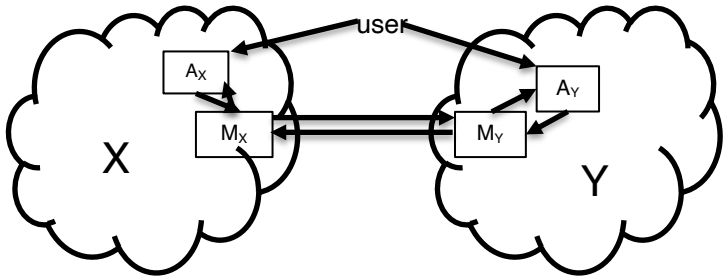


Solution #1: trusted third party (TTP).

Solution #2: k-of-n multisig. (k,n = app-specific parameters)

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



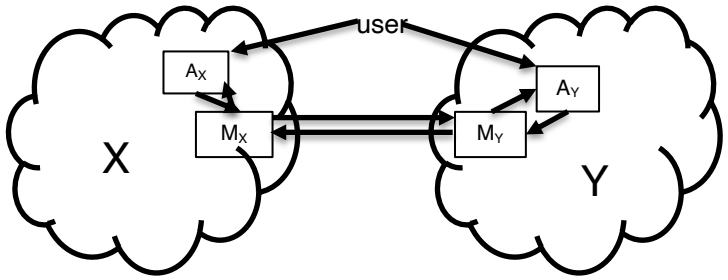
Solution #1: trusted third party (TTP).

Solution #2: k-of-n multisig. (k,n = app-specific parameters)

- n pks hard-wired into A_Y
- message <A_Y,m> regarded as valid by M_Y/A_Y ⇔ signed by at least k of the n corresponding private keys
  - k closer to n ➔ favors safety over liveness (k closer to 1 ➔ the reverse)

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



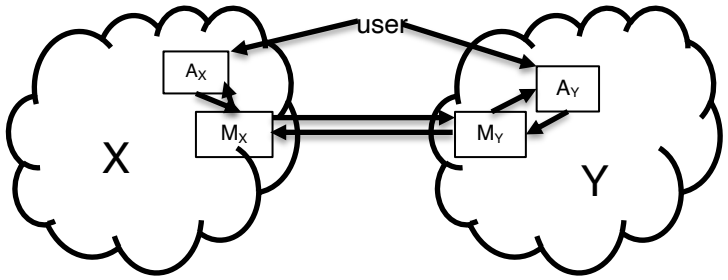
Solution #1: trusted third party (TTP).

Solution #2: k-of-n multisig. (k,n = app-specific parameters)

Solution #3: consensus protocol w/permissionless validator set.

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



Solution #1: trusted third party (TTP).

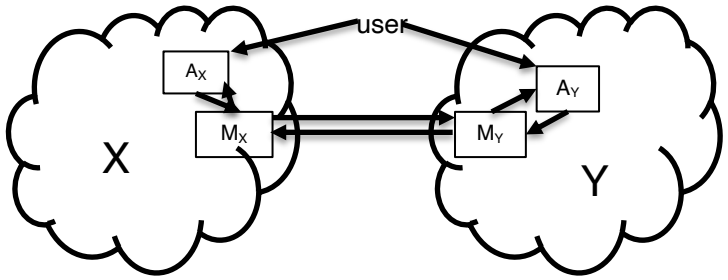
Solution #2: k-of-n multisig. (k,n = app-specific parameters)

Solution #3: consensus protocol w/permissionless validator set.

- specific to bridge, distinct from validators for X and Y
- e.g., using proof-of-stake for sybil-resistance/voting weights (see Pt III)
- like a multisig but allow free entry/exit to set of signing parties

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.

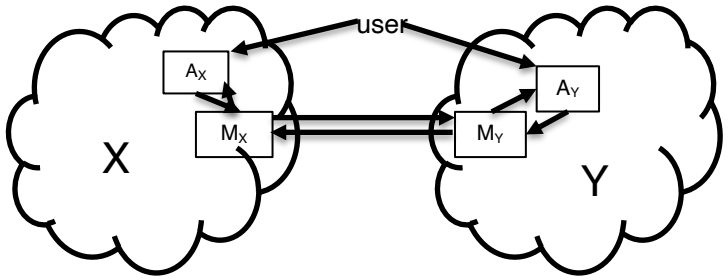


Solutions: (in increasing order of sophistication)
trusted third party, k-of-n multisig, permissionless consensus.

To discourage safety violations: (i.e., signing fake messages)

Externally Validated Bridges

Wanted: convincing proof (to M_Y or A_Y) that A_X really did want to send a msg m to A_Y.



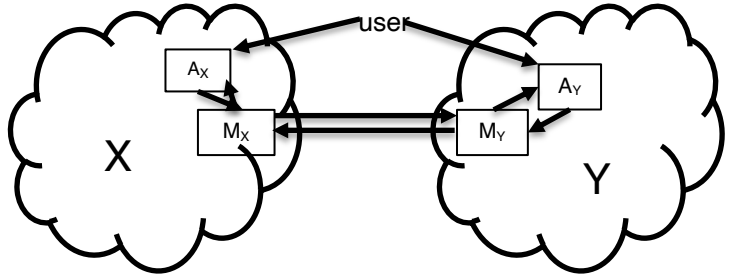
Solutions: (in increasing order of sophistication)
trusted third party, k-of-n multisig, permissionless consensus.

To discourage safety violations: (i.e., signing fake messages)

- require all parties to lock up collateral in M_X/A_X (for X → Y direction)
- anyone can post a signed fake message to M_X/A_X (via a tx on X), triggers the confiscation of collateral of all signing parties
  - could prove msg is fake using e.g. Merkle proof of non-membership

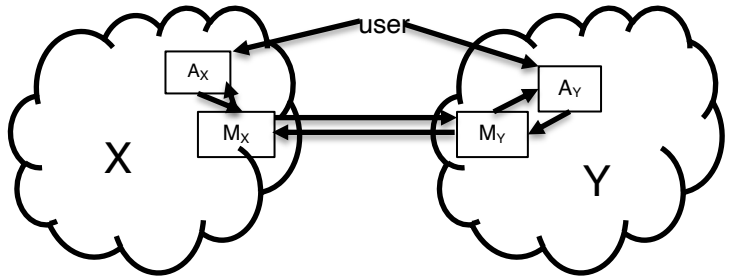
“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:



“Trustless” Bridges

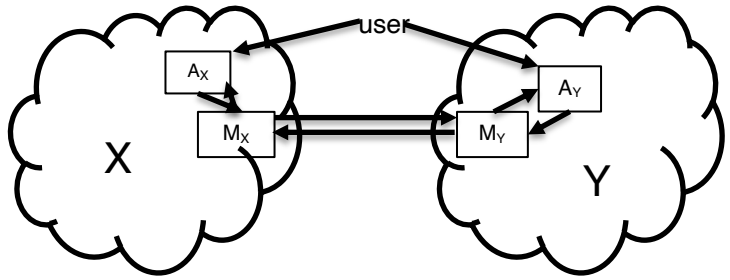
Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:



- header of appropriate block B of X
  - assume block header includes root of Merkle tree with leaves = txs

“Trustless” Bridges

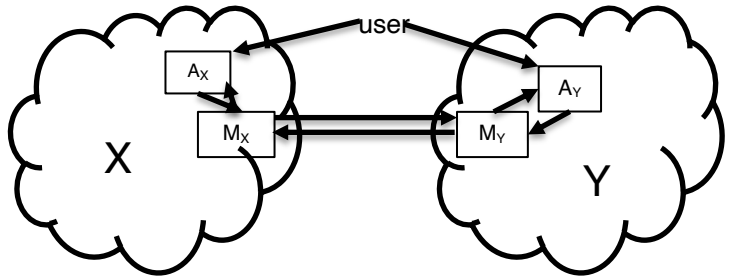
Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:



- header of appropriate block B of X
  - assume block header includes root of Merkle tree with leaves = txs
- Merkle proof showing that t was included in B

“Trustless” Bridges

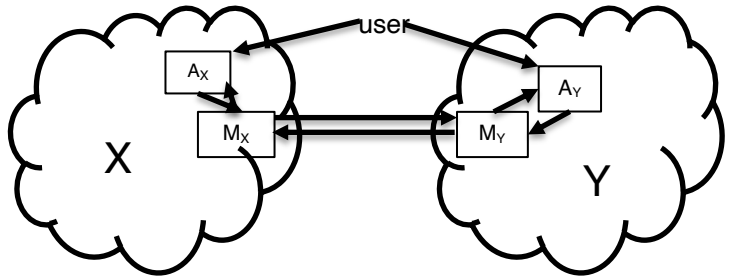
Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:



- header of appropriate block B of X
  - assume block header includes root of Merkle tree with leaves = txs
- Merkle proof showing that t was included in B
- evidence that B was indeed finalized by validators of X
  - e.g., for Tendermint, signatures from > 2n/3 of X’s validators
  - piggyback on existing trust assumption on validators of X

“Trustless” Bridges

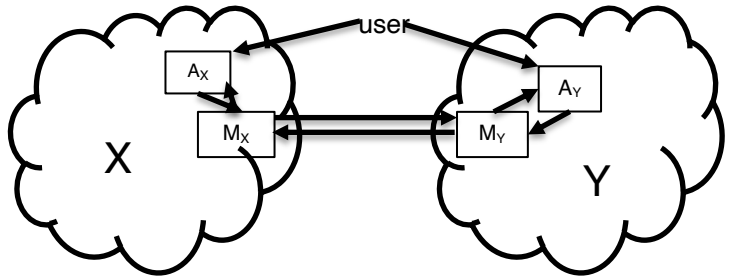
Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:



- header of appropriate block B of X
  - assume block header includes root of Merkle tree with leaves = txs
- Merkle proof showing that t was included in B
- evidence that B was indeed finalized by validators of X
  - e.g., for Tendermint, signatures from > 2n/3 of X’s validators
  - piggyback on existing trust assumption on validators of X
- ➔ A_Y accepts msg from M_Y ⇔ accompanied by such evidence

“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y:

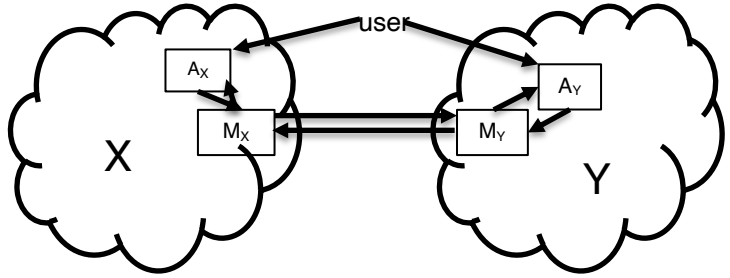


- header of appropriate block B of X
- Merkle proof showing that t was included in B
- evidence that B was indeed finalized by validators of X
  - e.g., for Tendermint, signatures from > 2n/3 of X’s validators
  - piggyback on existing trust assumption on validators of X
- ➔ A_Y accepts msg from M_Y ⇔ accompanied by such evidence

Note: A_Y effectively acting as a light client for X.

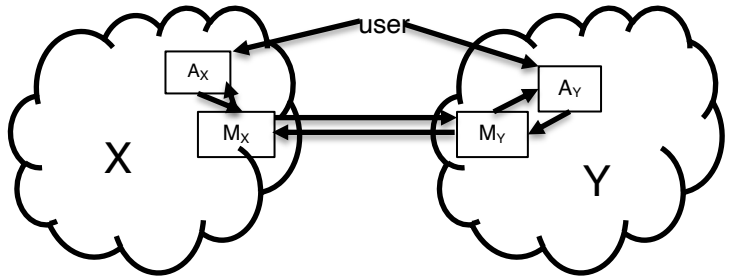
“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y a block header, Merkle pf of tx inclusion, evidence of finalization.



“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y a block header, Merkle pf of tx inclusion, evidence of finalization.

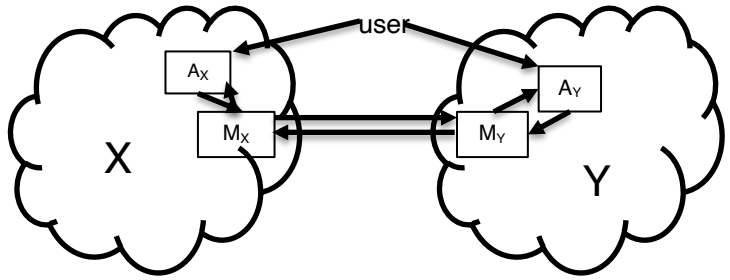


Challenge #1: Assumes public keys of X’s validators hard-wired into A_Y.

- much harder if X’s validator set changing over time
  - in many cases, still possible in principle (with additional evidence)

“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y a block header, Merkle pf of tx inclusion, evidence of finalization.

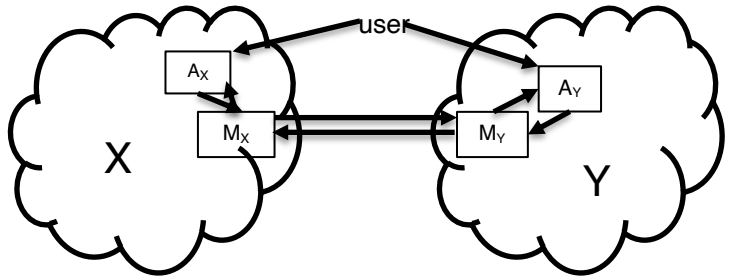


Challenge #1: Assumes pks of X’s validators hard-wired into A_Y.

Challenge #2: Verification of evidence (i.e., light client logic) too much work/too expensive for an L1 to carry out.

“Trustless” Bridges

Idea: to prove to A_Y that tx t really was finalized on X, post to A_Y a block header, Merkle pf of tx inclusion, evidence of finalization.



Challenge #1: Assumes pks of X’s validators hard-wired into A_Y.

Challenge #2: Verification of evidence (i.e., light client logic) too much work/too expensive for an L1 to carry out.

- possible solution: provide SNARK proof of existence of such evidence that a light client would accept as valid
  - only post proof of knowledge, not evidence itself; L1 only verifies correctness of SNARK proof, does not carry out light client logic