CONSENSYS

Blockchain / Crosschain / DeFi Bridge Design

Peter Robinson, October 13, 2021

Acknowledgements

Thank you to everyone who reviewed my slides:

- Domink Schmid
- Marc Mecuri
- Weijia Zhang
- Raghavendra Ramesh
- Ricardo Silva
- Faina Shalts
- Kevin Weaver

https://pixabay.com/photos/gothic-old-roman-bridge-stone-5027110/

Agenda

- What are blockchain / crosschain / DeFi bridges?
- Topology & Terminology
- Crosschain Protocol Stack
- Design Choices: Crosschain Messaging Layer
- Design Choices: Crosschain Function Call Layer
- Design Choices: Crosschain Application Layer
- Security Considerations
- Scalability
- Bridge Fees
- Standards
- Philosophical Questions

CONSENSYS

What are Blockchain / Crosschain / DeFi Bridges?

Bridge?

Bridges allow you to:

- Swap assets on one blockchain for assets on another blockchain.
  Transfer assets from one blockchain to another blockchain.
  - Assets are typically ERC 20 fungible tokens, ERC 721 non-fungible tokens.
- Communicate arbitrary data and messages across blockchains.
- Have functions in contracts on one blockchain call functions in contracts on another blockchain.

A ***Bridge*** is also known as a:

- Blockchain Bridge
- Crosschain Bridge
- DeFi Bridge
- Token Bridge

Why?

Why would you want to move value from one blockchain to another?

- Transaction fees.
- Block confirmation times.
- Functionality.
- Liquidity pools.
- Capital utility / efficiency.

Why?

Why would you want to move value from one blockchain to another?

- Transaction fees:
  - Different blockchains have different fees. These are typically driven by the number of people competing to use a blockchain. Note that lower transaction fees might come with more centralization and lower security.
- Block confirmation times:
  - Typically, you should wait between six and twelve block confirmations* on Ethereum MainNet.This translates to between 1 ½ and 3 minutes. Other blockchains may offer faster block times and faster or instant finality.
    Note 1: that faster confirmation times may come with more centralization and lower security.
    Note 2: the Ethereum Merge will change the block confirmation times.
- Functionality:
  - A contract you wish to use may be hosted on a different blockchain to the one you have your value on.

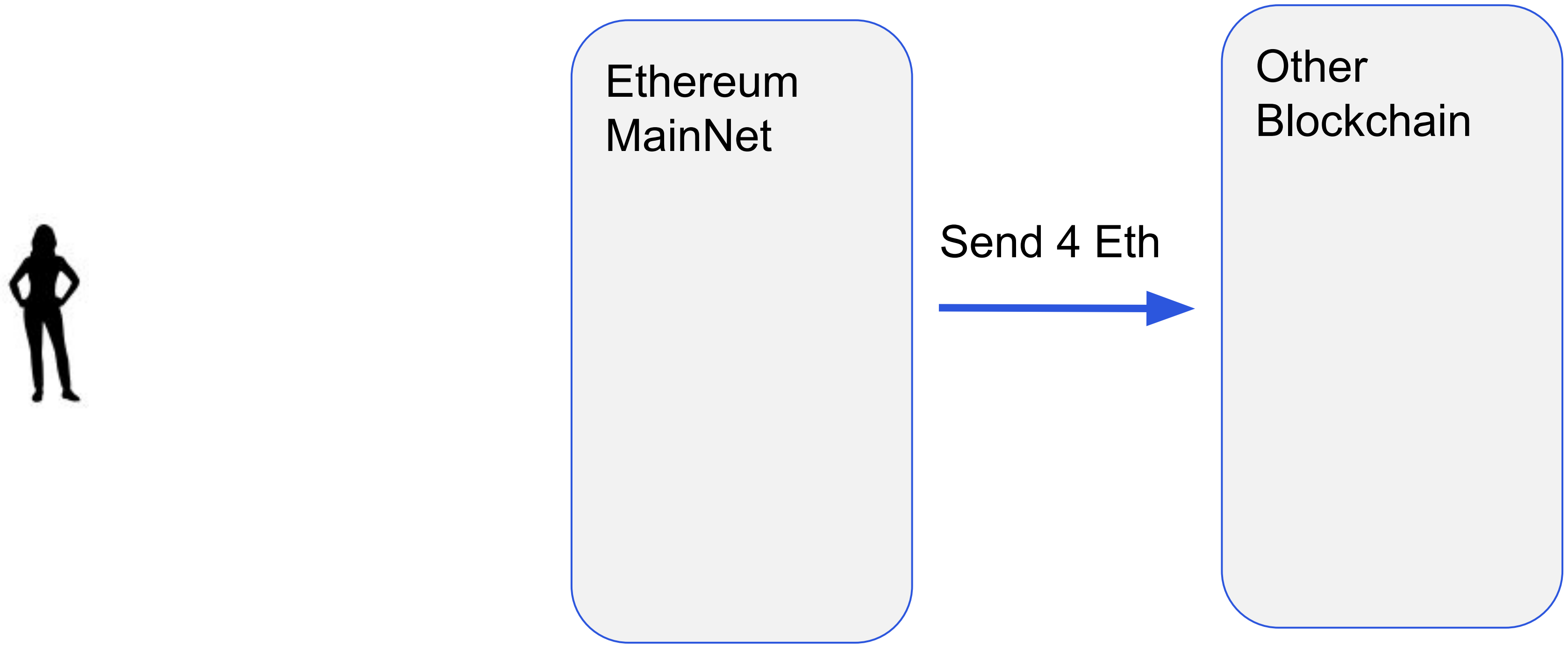
* See https://doi.org/10.1017/S0269888920000296 or open access version: https://arxiv.org/abs/1906.04421

Why?

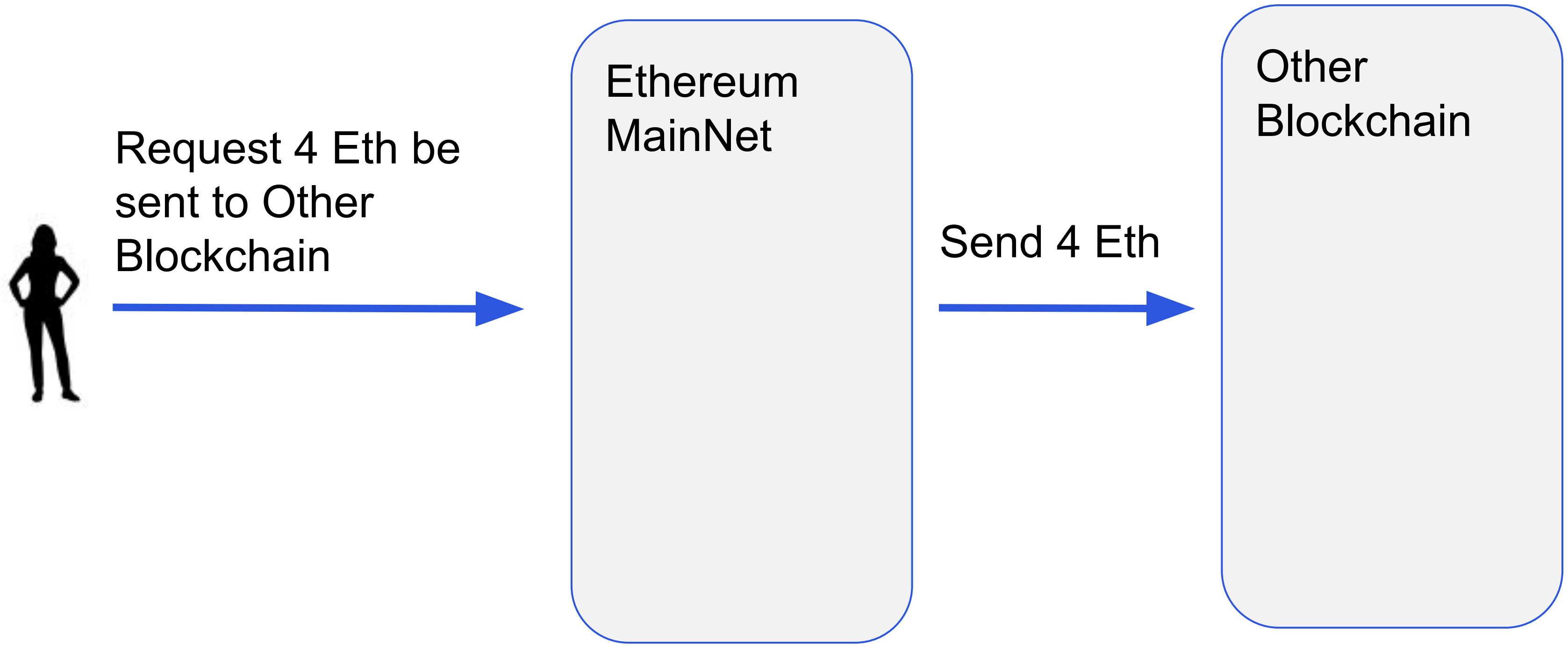
Why would you want to move value from one blockchain to another?

- Liquidity pools:
  - Moving your value to a blockchain with a larger liquidity pool may make it easier to trade.
- Capital utility / efficiency
  - Able to use all assets from all blockchains in the one transaction.

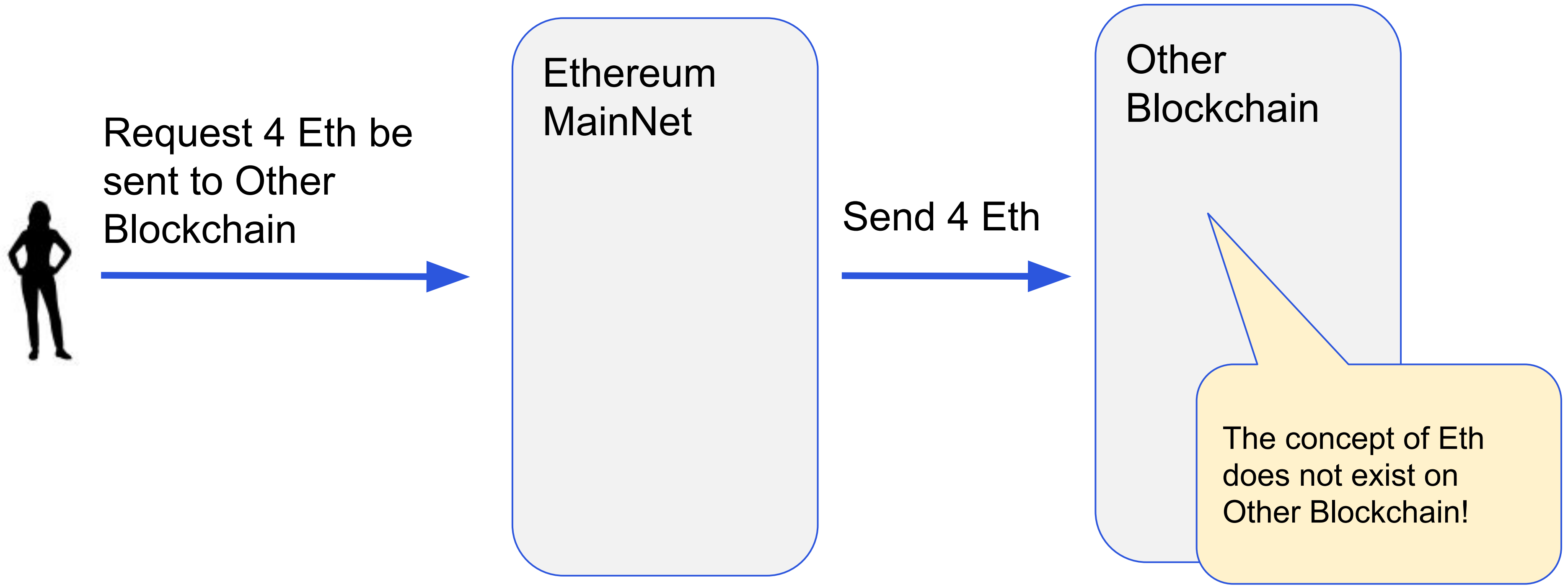
Value Transfer



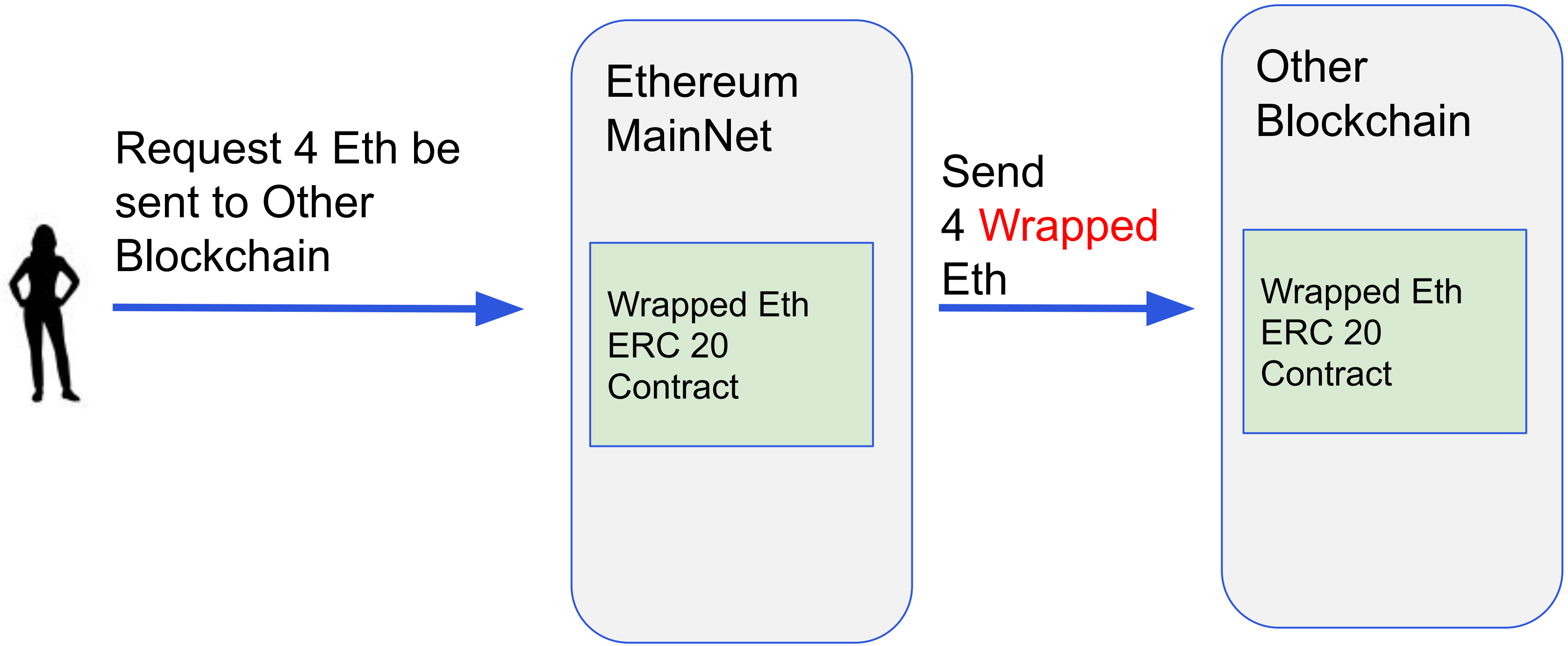
Value Transfer



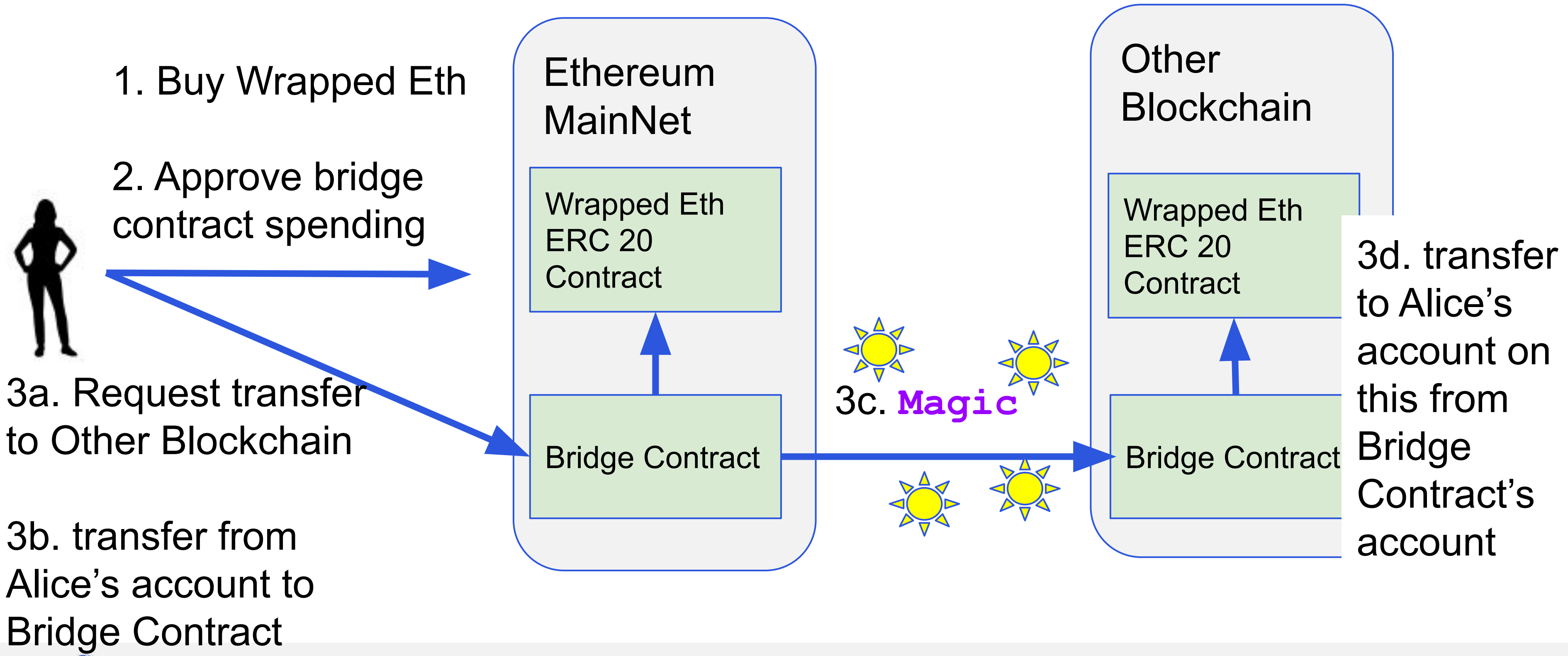
Value Transfer


Request 4 Eth be sent to Other Blockchain
Ethereum MainNet
Send 4 Eth
Other Blockchain
The concept of Eth does not exist on Other Blockchain!

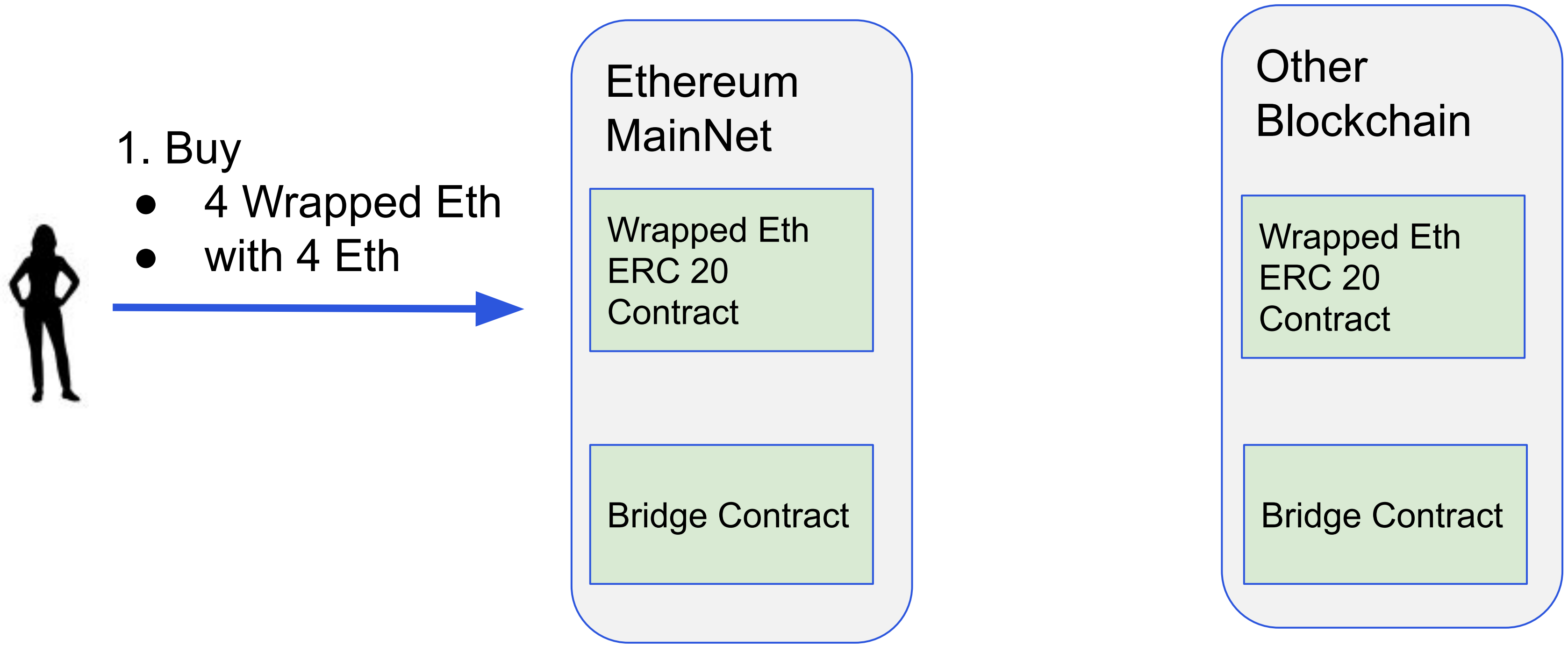
Value Transfer


Request 4 Eth be sent to Other Blockchain
Ethereum MainNet
Wrapped Eth ERC 20 Contract
Send 4 Wrapped Eth
Other Blockchain
Wrapped Eth ERC 20 Contract

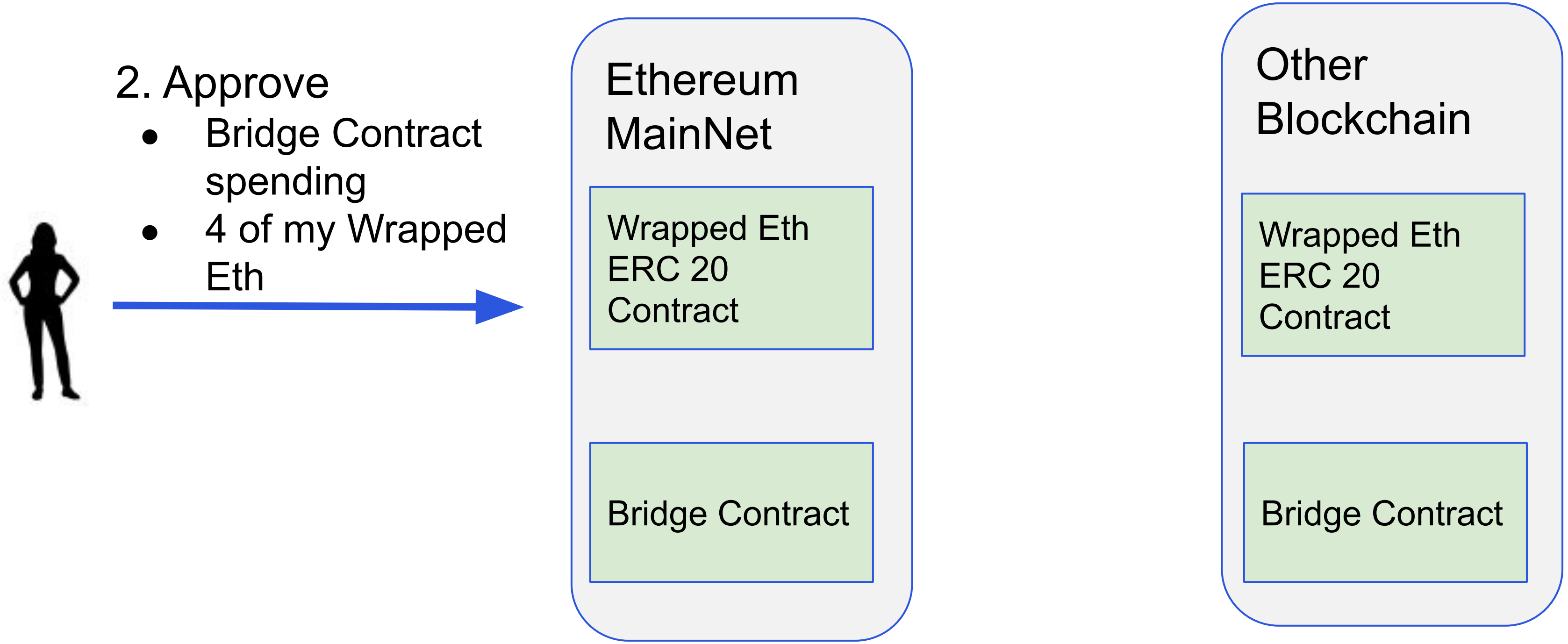
Value Transfer


1. Buy Wrapped Eth
2. Approve bridge contract spending
3a. Request transfer to Other Blockchain
3b. transfer from Alice's account to Bridge Contract
Ethereum MainNet
Wrapped Eth ERC 20 Contract
Bridge Contract
3c. Magic
Other Blockchain
Wrapped Eth ERC 20 Contract
Bridge Contract
3d. transfer to Alice's account on this from Bridge Contract's account

Value Transfer

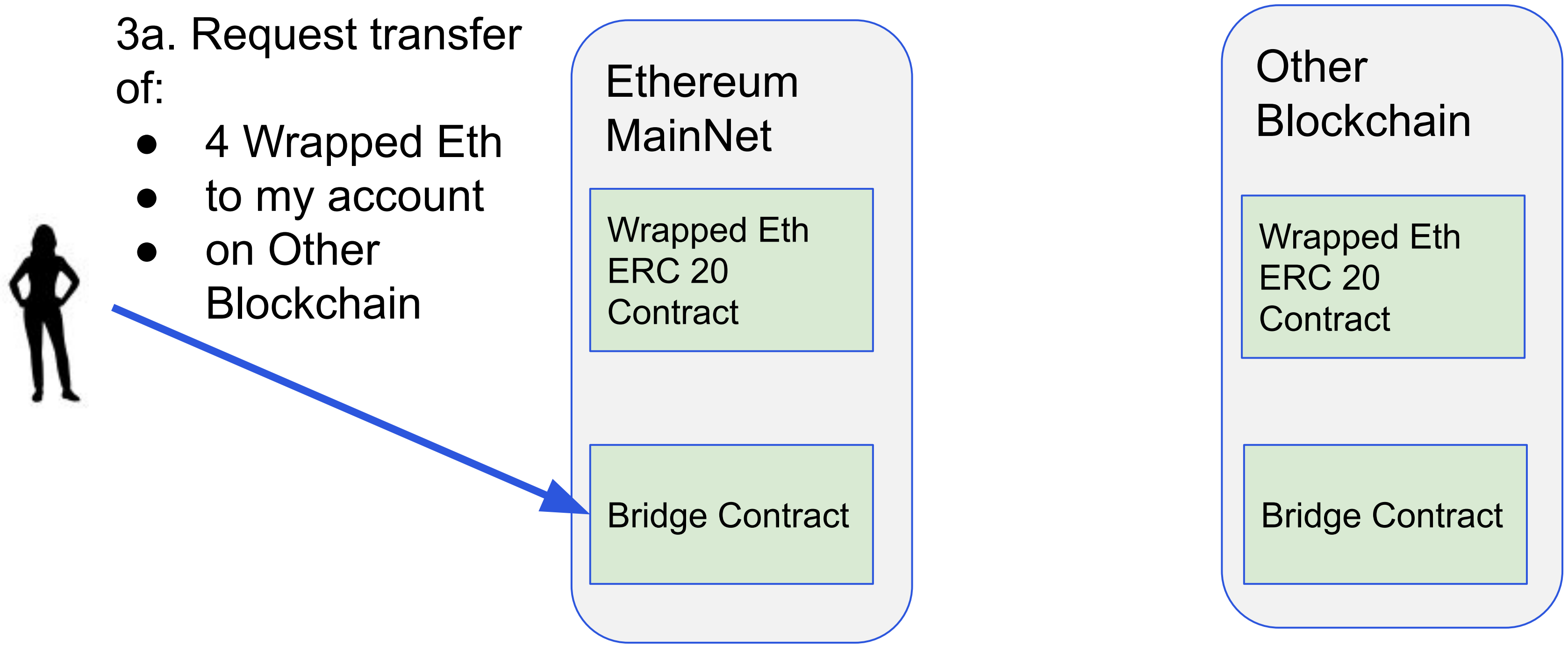


Value Transfer



Value Transfer

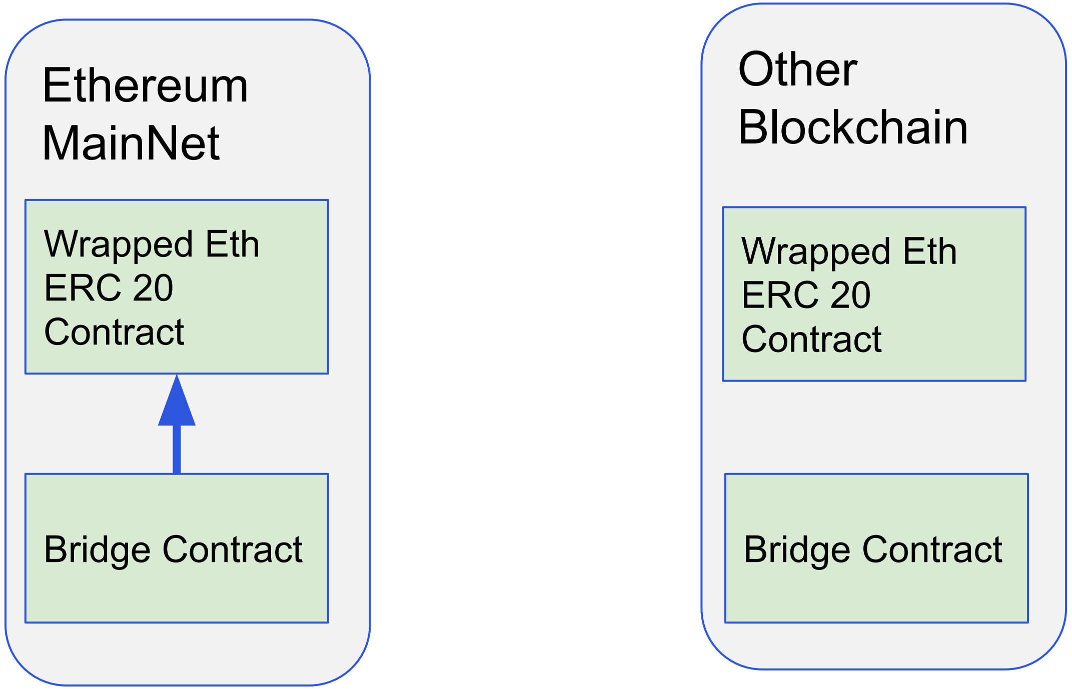
3a. Request transfer of:
- 4 Wrapped Eth
- to my account
- on Other Blockchain



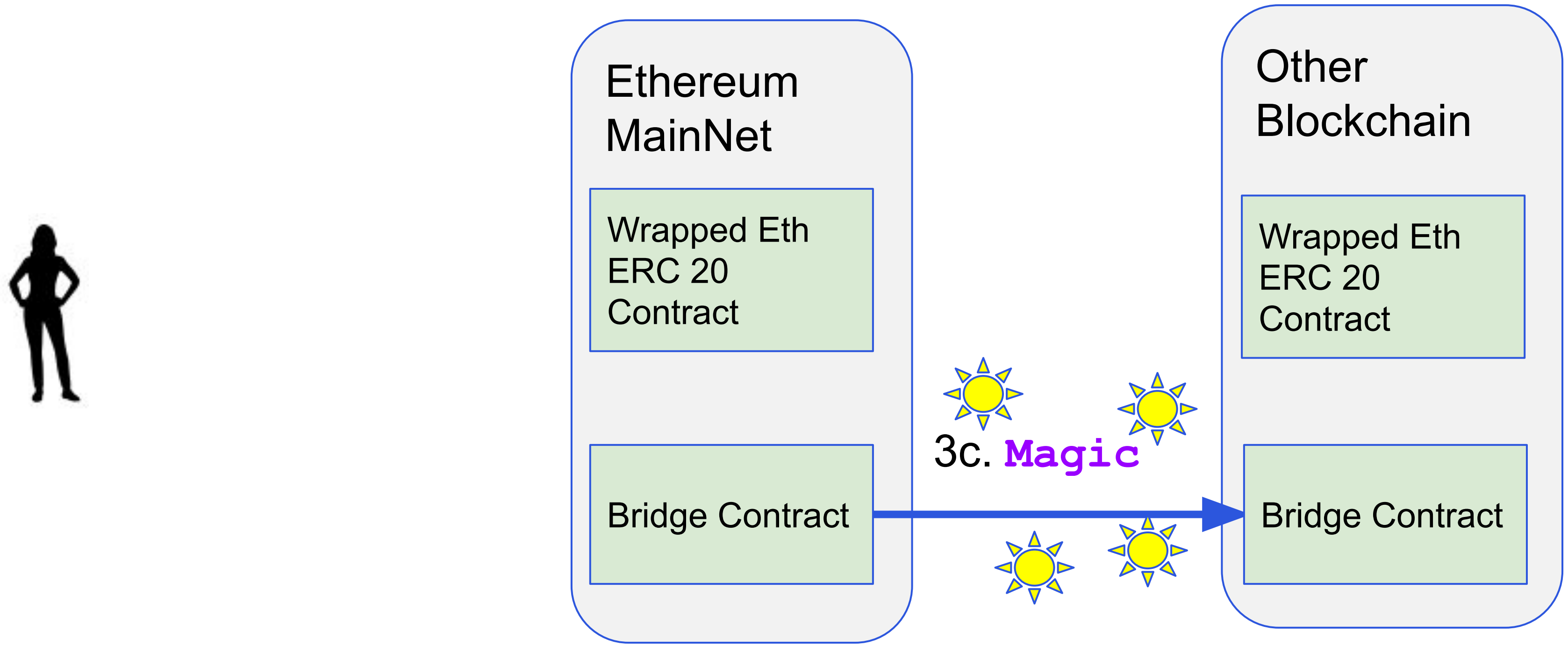
Value Transfer

3b. transfer from:

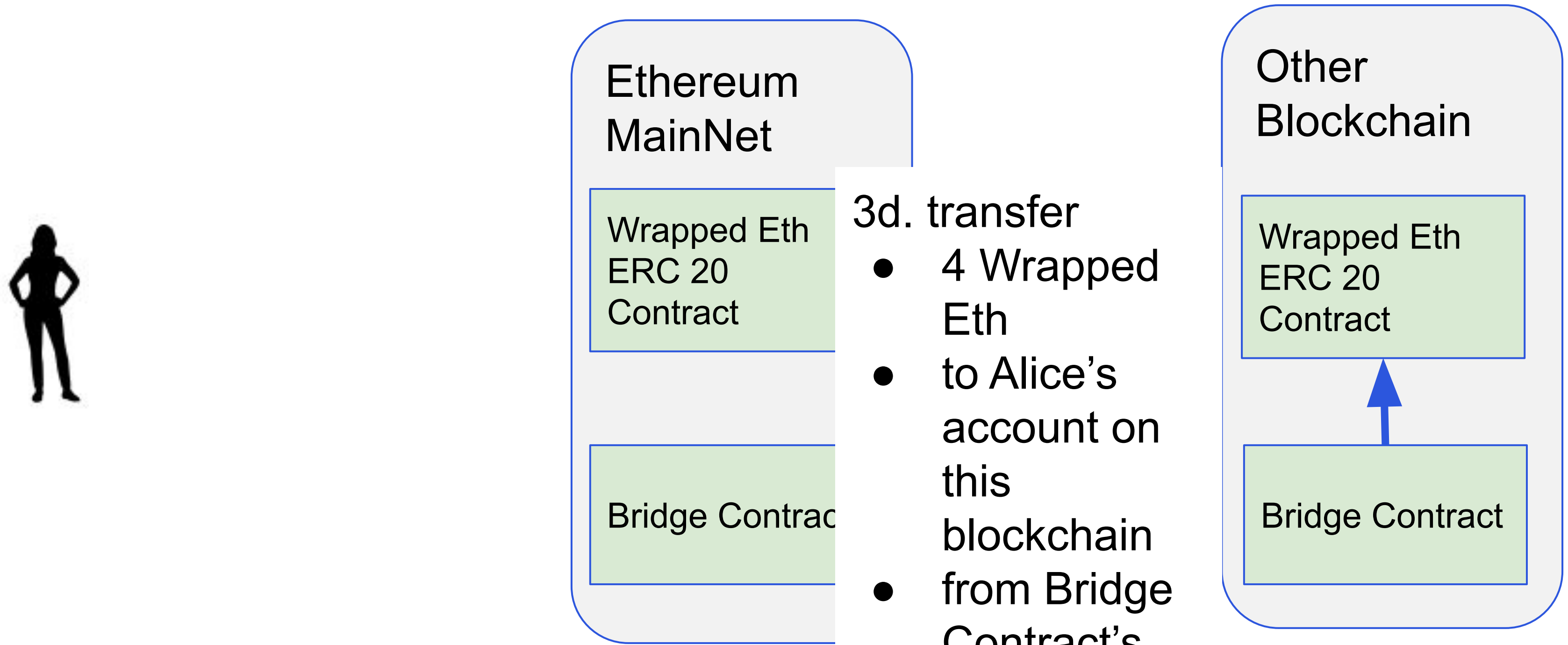
- Alice’s account
- to Bridge Contract account
- 4 Wrapped Eth



Value Transfer

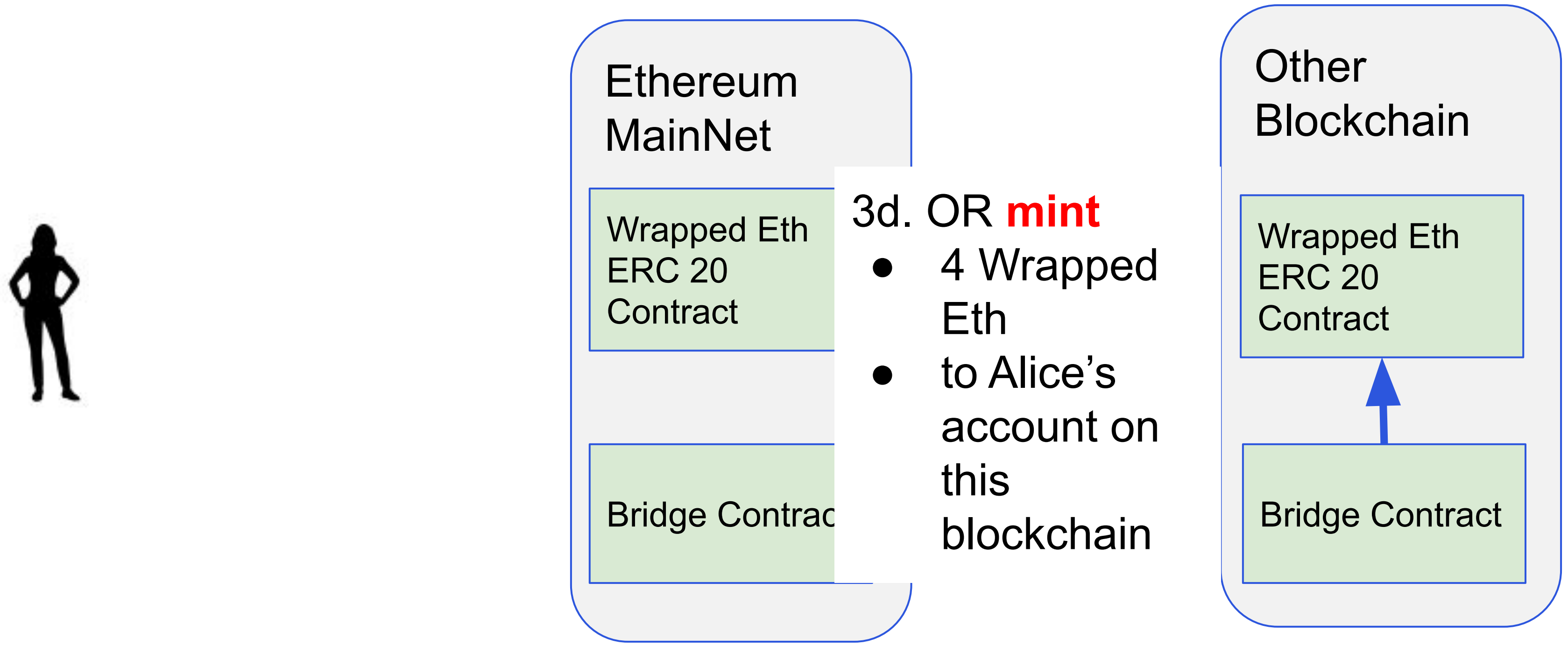


Value Transfer



3d. transfer
- 4 Wrapped Eth
- to Alice's account on this blockchain
- from Bridge Contract's account

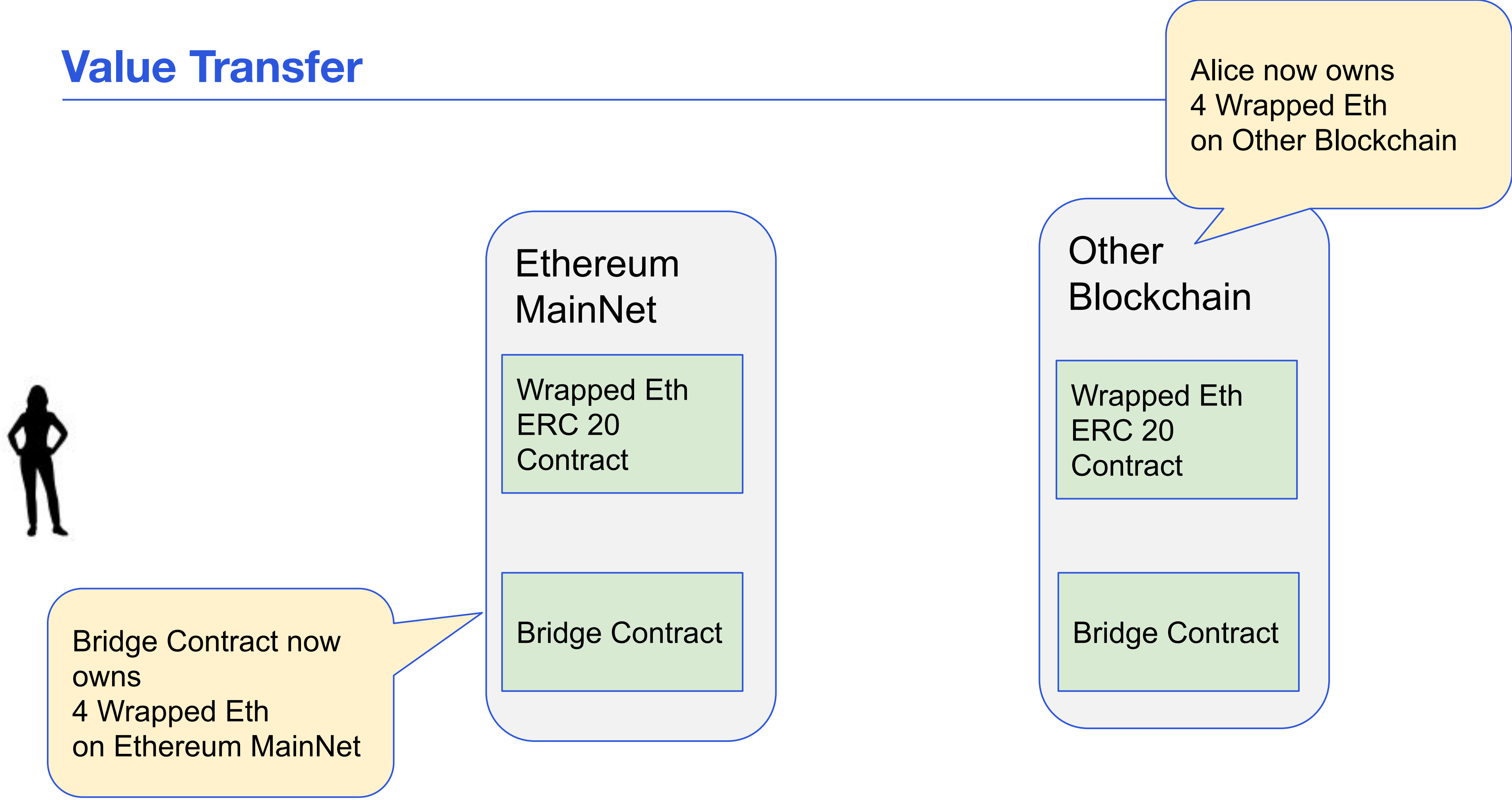
Value Transfer



3d. OR **mint**

- 4 Wrapped Eth
- to Alice’s account on this blockchain

Value Transfer



Crosschain Use-Cases

Lots of other crosschain use-cases:

- Supply chain financing.
- Supply chain provenance / selective transparency.
- Cross-border supply chain.
- Inter-CBDC payments.

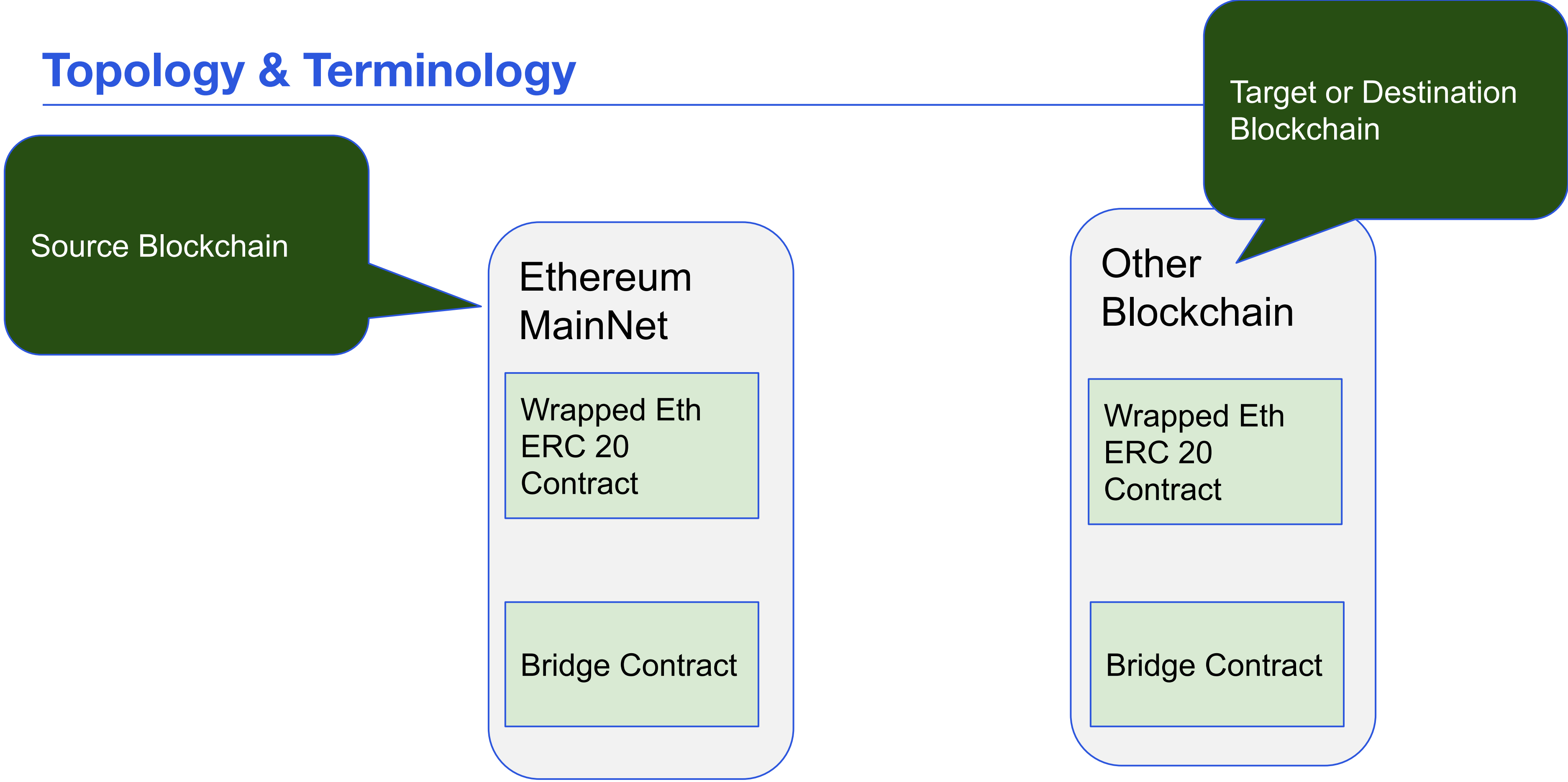
EEA: Crosschain Interoperability Working Group

Crosschain Interoperability
Use Case

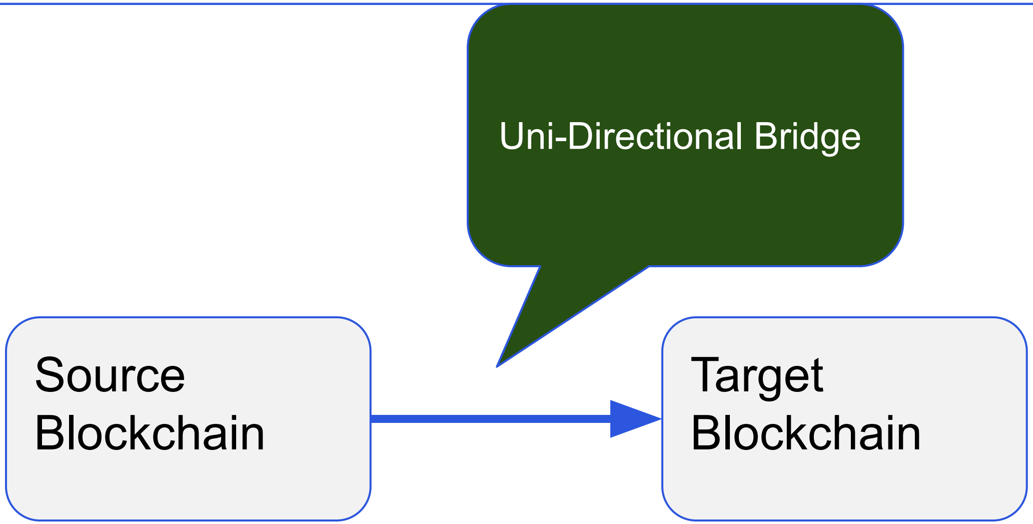
CONSENSYS

Topology & Terminology

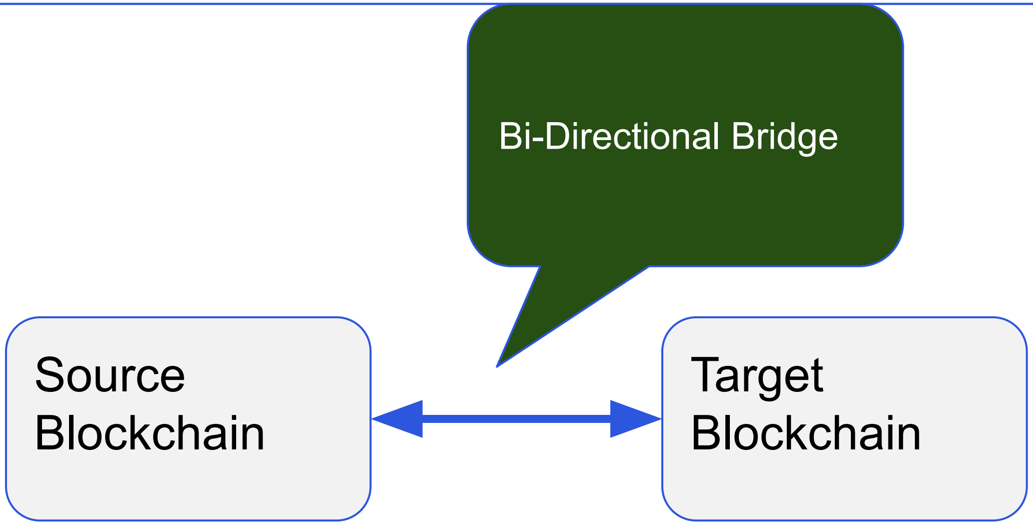
Topology & Terminology


Source Blockchain
Target or Destination Blockchain
Ethereum MainNet
Wrapped Eth ERC 20 Contract
Bridge Contract
Other Blockchain
Wrapped Eth ERC 20 Contract
Bridge Contract

Topology & Terminology

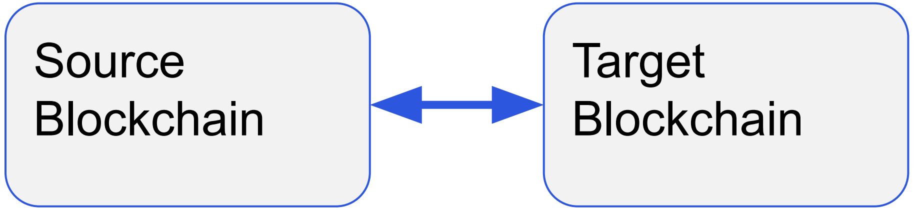


Topology & Terminology

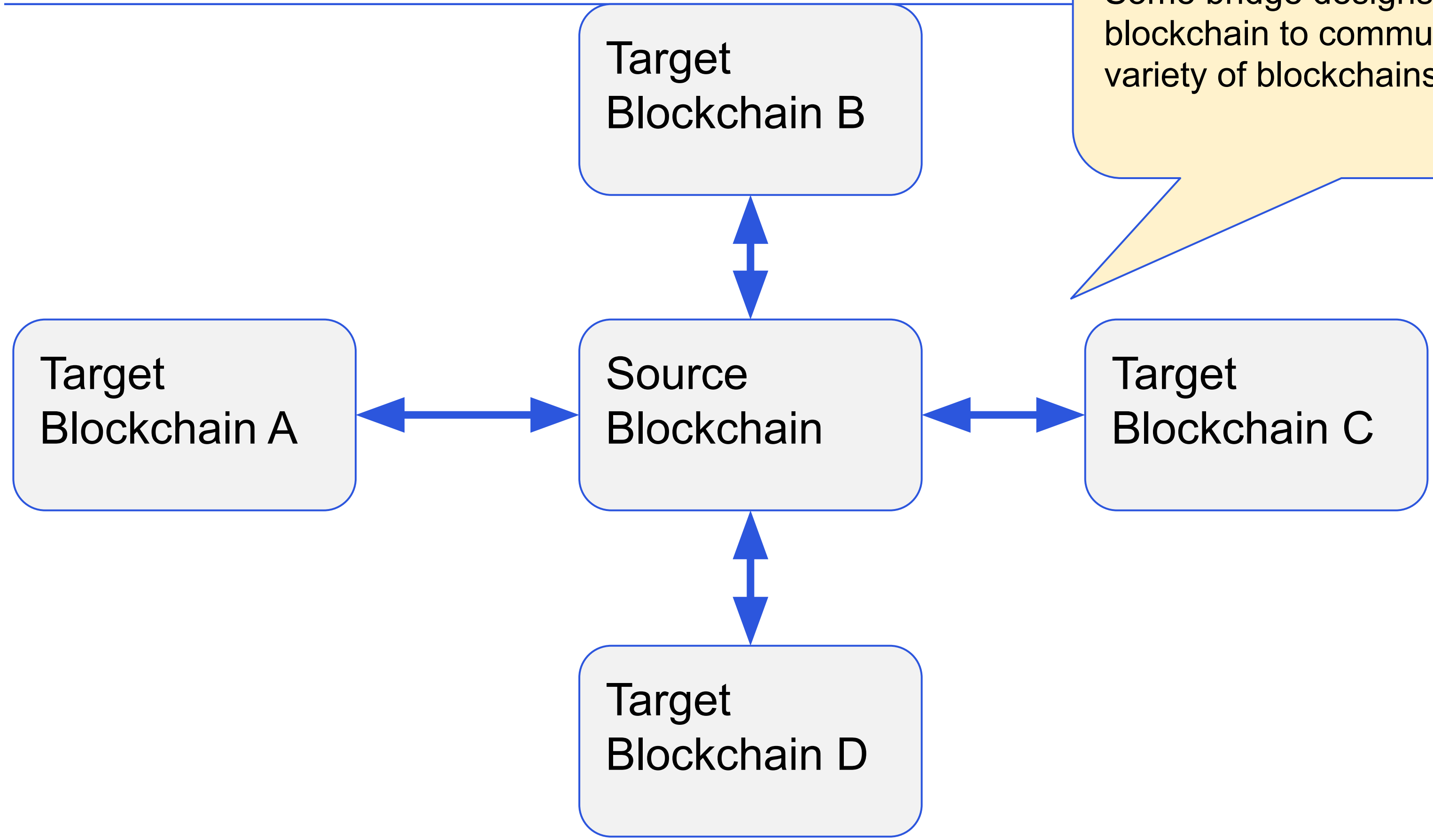


Topology & Terminology

Some bridge designs only work between two specific blockchains.

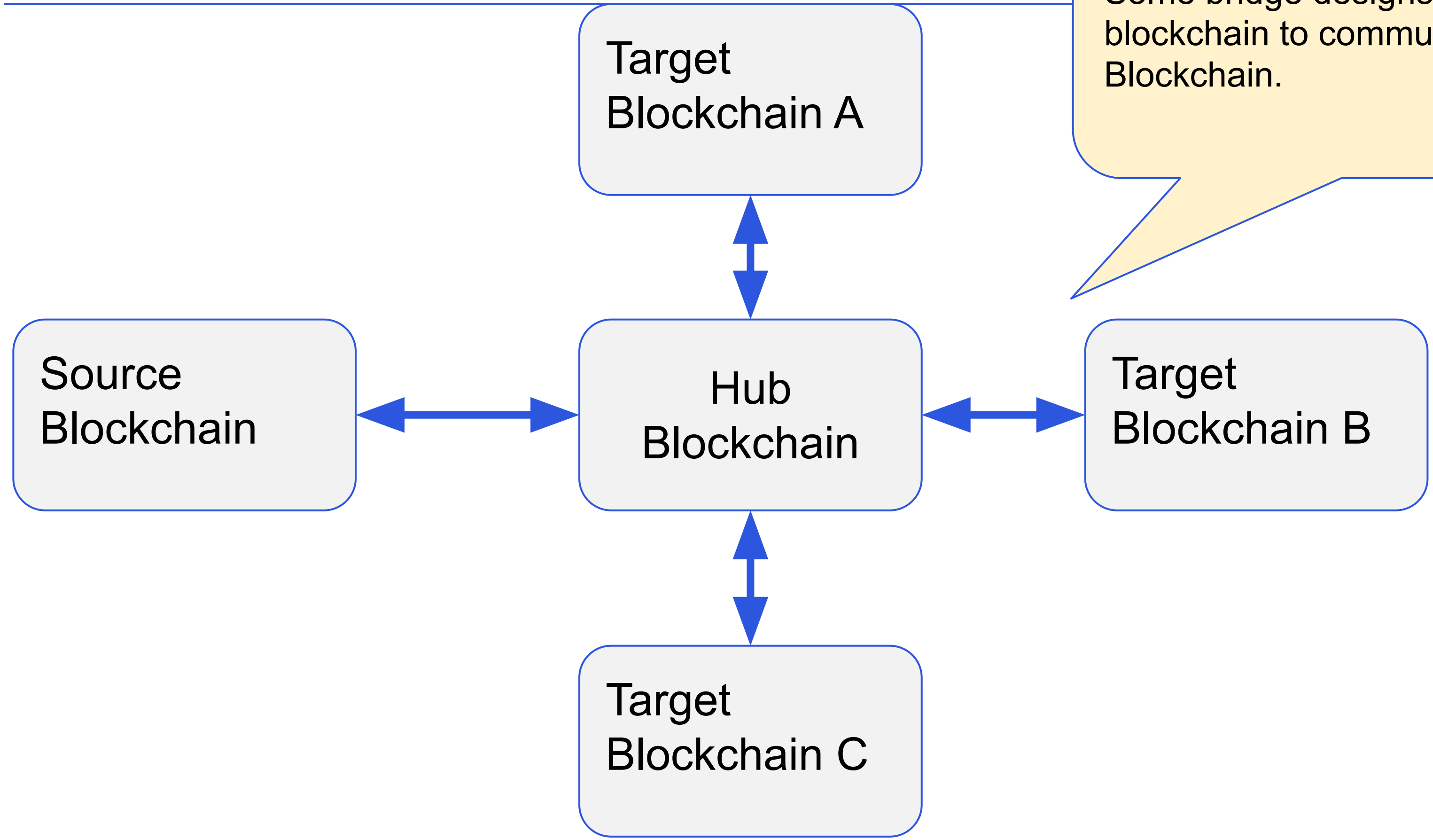


Topology & Terminology



Some bridge designs allow a blockchain to communicate with a variety of blockchains.

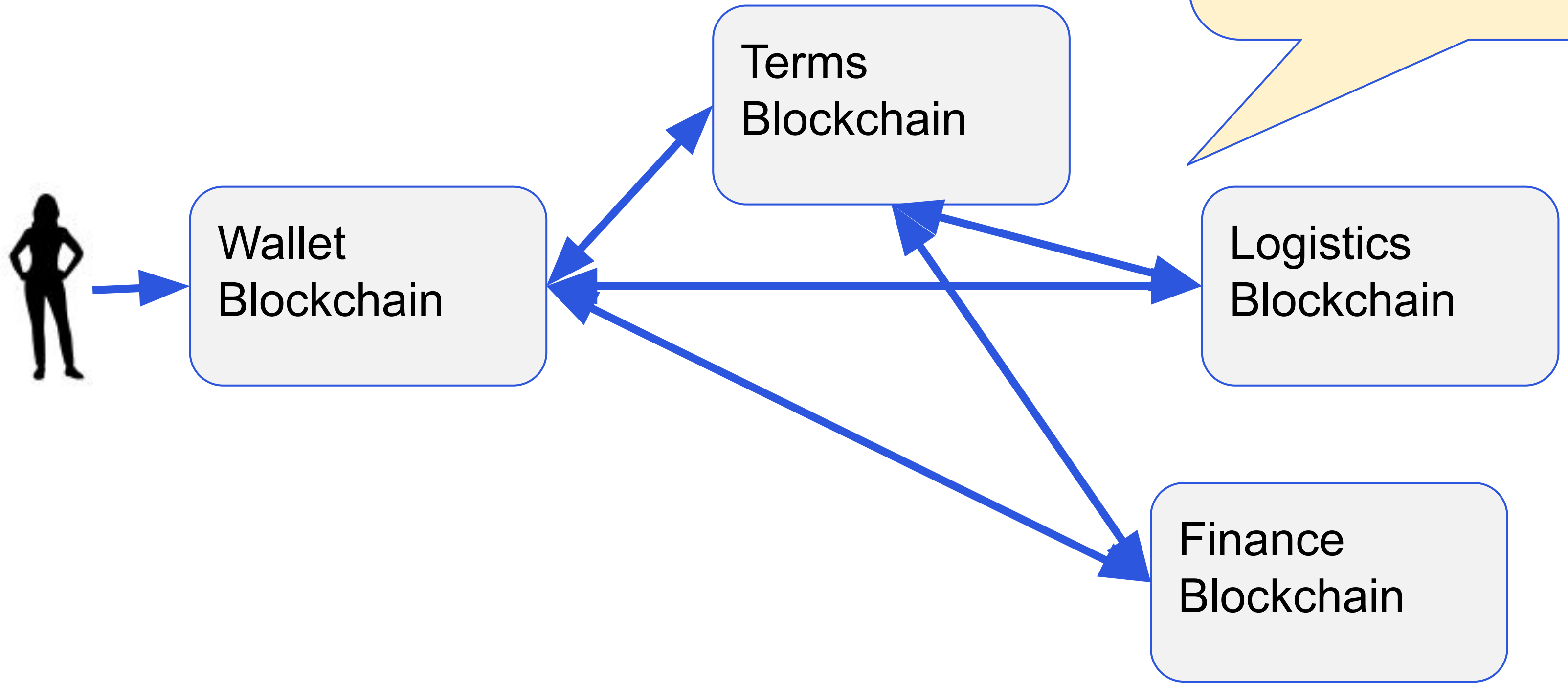
Topology & Terminology



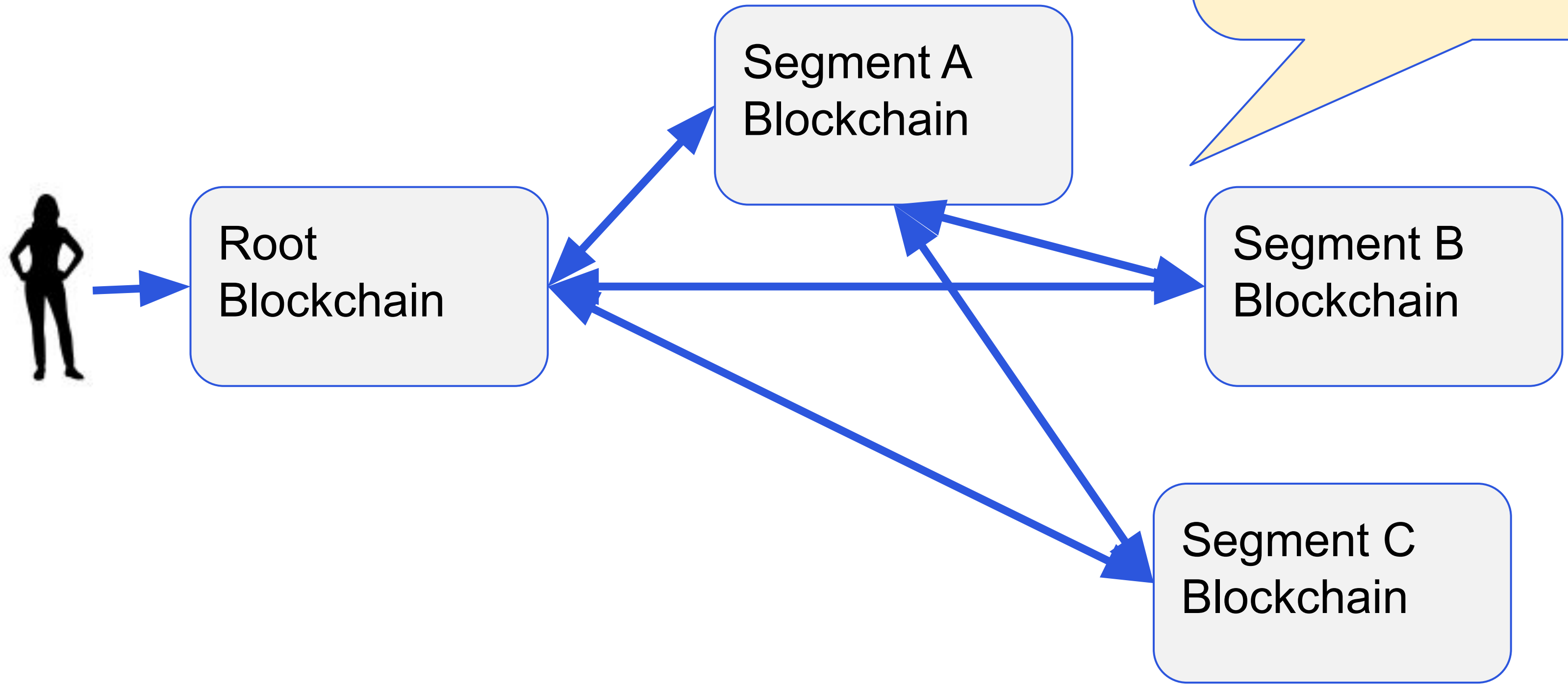
Some bridge designs allow a blockchain to communicate via a Hub Blockchain.

Topology & Terminology

Some crosschain use-cases involve multiple blockchains.



Topology & Terminology

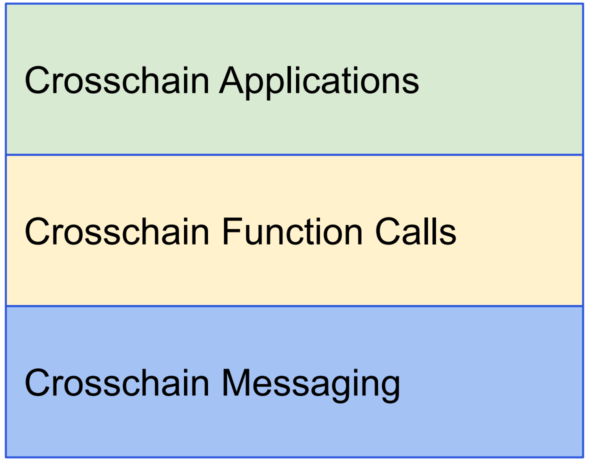


In these cases Root and Segment blockchain might be used rather than Source and Target or Destination

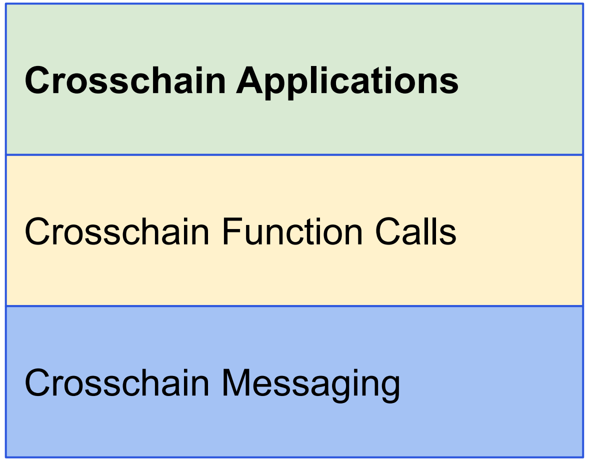
CONSENSYS

Crosschain Protocol Stack

Crosschain Protocol Stack



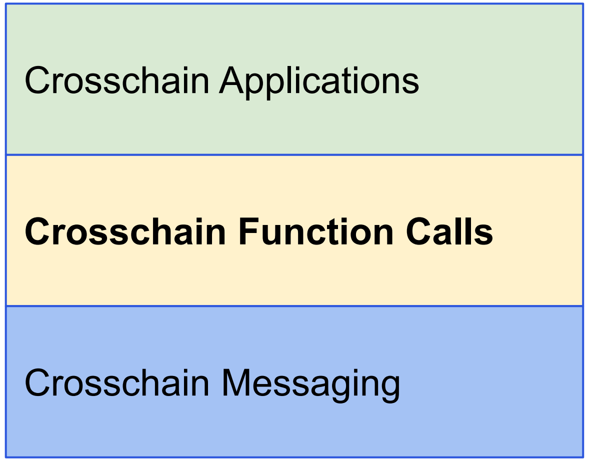
Crosschain Protocol Stack



The **Crosschain Applications** layer consists of applications that operate across blockchains.

There may be software components created in this layer that allow complex applications to be created more easily.

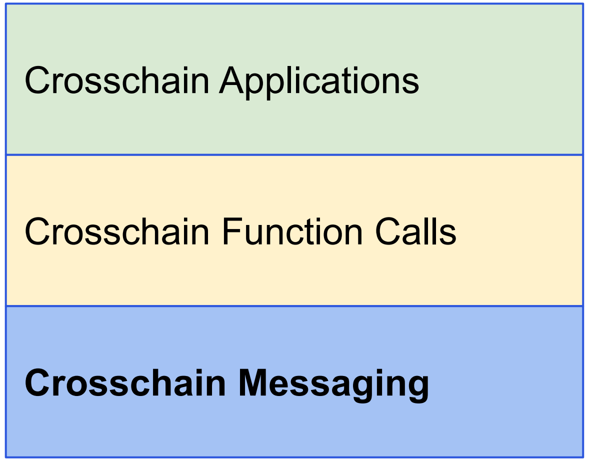
Crosschain Protocol Stack



The **Crosschain Function Calls** layer executes function calls across blockchains.

The updates due to the function call protocol can be atomic or not-atomic.

Crosschain Protocol Stack



The **Crosschain Messaging** layer ensures that information can be verified as having come from a certain blockchain.

Crosschain Protocol Stack

Having a layered architecture with **clearly defined interfaces between layers** has the following advantages:

- **Interoperability**: Applications will work with a variety of Function Call implementations that will work with a variety of Messaging implementations.
- **Flexibility**: different Crosschain Message Verification technique could be used for each blockchain / roll-up* used in an overall crosschain function call.
- **Infrastructure Reuse**: Different Crosschain Function Call techniques can share the same deployed Crosschain Message Verification infrastructure.
- **Focus on what you are good at**: Organisations can focus on creating solutions for a single part of the protocol stack stack. That is, rather than having to create the entire stack, a company might choose to focus on an application, a better Function Call approach, or some Messaging infrastructure.
- **Experimentation**: Not having to build the entire protocol stack should make experimentation much easier.

* Rollups: https://www.youtube.com/watch?v=Ohi6R7m5BWk&t=4s

Design Choices: Questions to think through

Questions to think through:

- Who is being trusted?
- How many transactions are needed to facilitate a transfer?
- Who is submitting what transactions to which blockchain?
- Can a user who has no value on the destination blockchain use this?
- How are infrastructure components (relayers, attestors etc) compensated?
- Do infrastructure components need to handle requests from users?
- How many components need to be compromised / bribed for the system to fail?
- What attacks are possible?
- Would you use this technique to transfer a billion $?????

CONSENSYS

Design Choices: Crosschain Messaging Layer

Crosschain Messaging: Ethereum Events

Ethereum transactions can emit *events*:

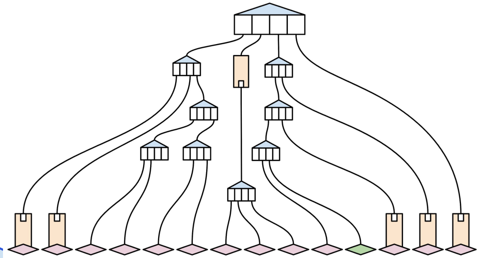
```
function transfer(address _recipient, uint256 _amount) external {
  ....

  emit Deposit(msg.sender, _recipient, _amount);
}

event Deposit(address _from, address _to, uint256 _amount);
```

Crosschain Messaging: Ethereum Events

- Events are stored in transaction receipts.
- The root of a Merkle Patricia Trie of transaction receipts is included in the Ethereum Block Header.

Block 100

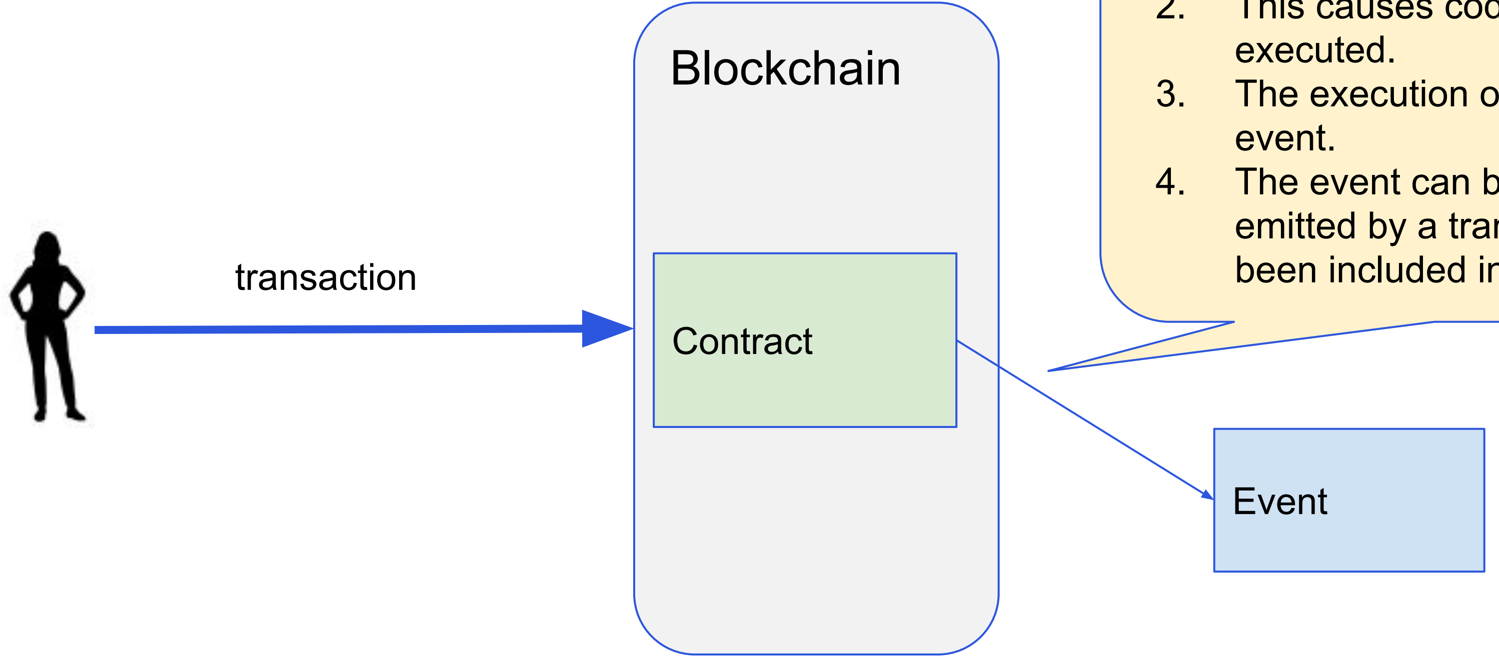
Block 101:
… , transaction receipt root, ...



Transaction Receipt:
- State root or status,
- Cumulative gas used,
- Bloom Filter,
- Event logs,
- Revert Reason

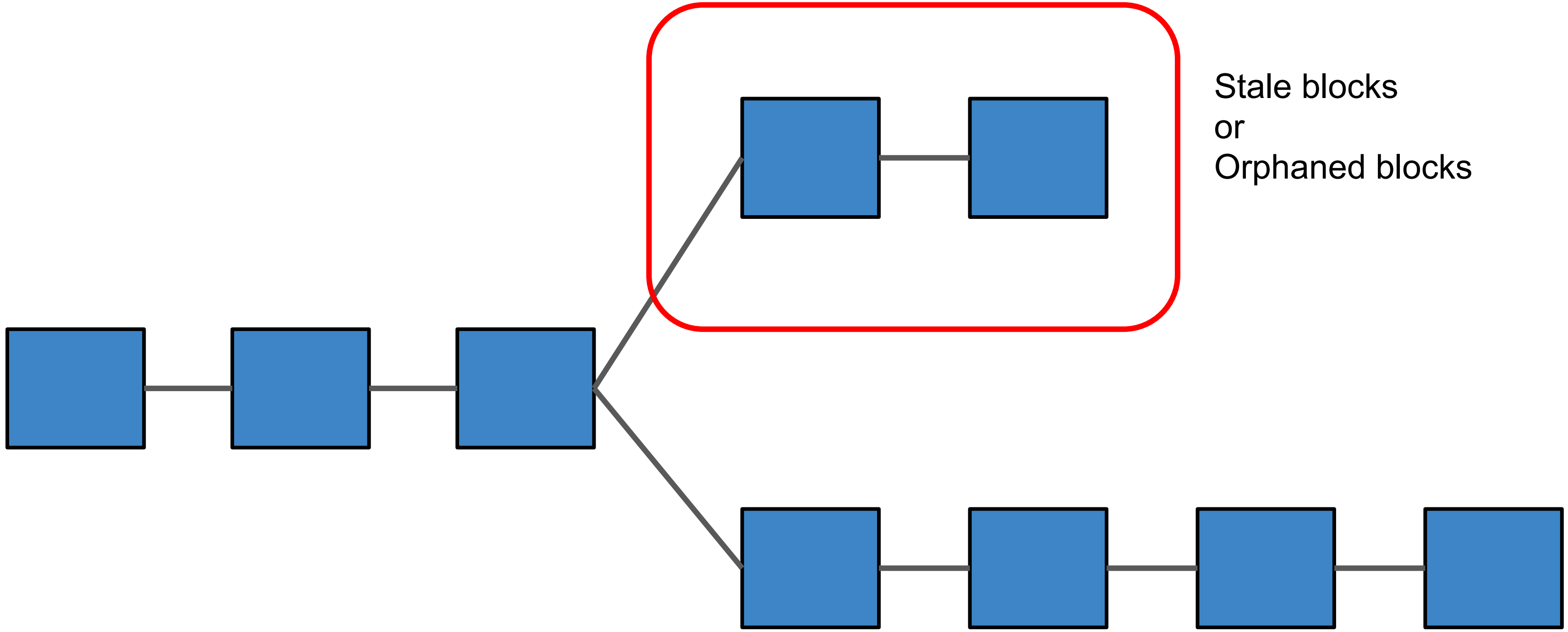
Event log:
- Address of contract that emitted the event,
- Topics. These are the event function signature and indexed parameters.
- Data. The encoded parameters.

Crosschain Messaging: Ethereum Events

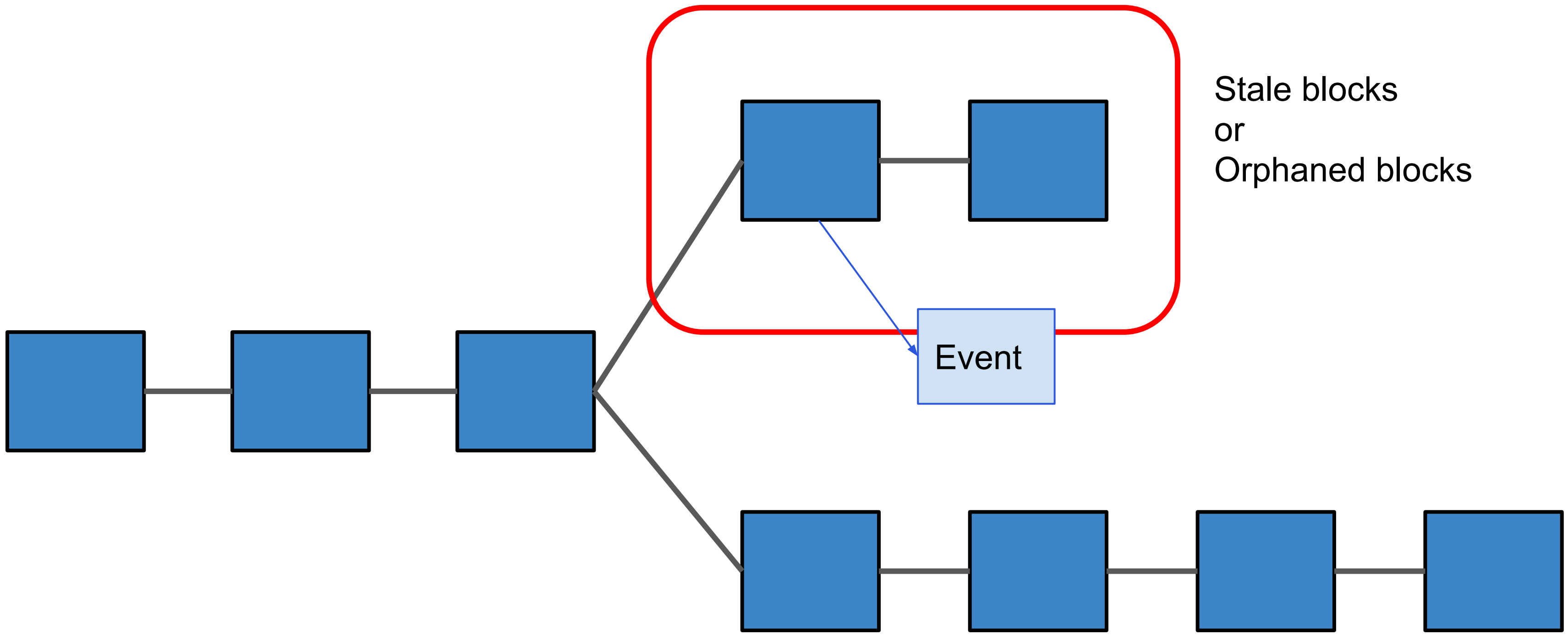


1. The user submits a transaction.
2. This causes code in the contract to be executed.
3. The execution of the code emits an event.
4. The event can be proven to be emitted by a transaction that has been included in a block.

Crosschain Messaging: Finality



Crosschain Messaging: Finality



Crosschain Messaging: Finality

- **A Crosschain Messaging system should only use events that belong to transactions that have been included in blocks that are (probably) final.**

- Block confirmation times:
  - Typically, you should wait between six and twelve block confirmations* on Ethereum MainNet.This translates to between 1 ½ and 3 minutes.
  - Other blockchains may offer faster block times and faster or instant finality.
  - Note that faster confirmation times may come with more centralization and lower security.
  - Note: the Ethereum Merge will change the block confirmation times.

* See https://doi.org/10.1017/S0269888920000296 or open access version: https://arxiv.org/abs/1906.04421

Non-EVM Blockchains

There are a lot of blockchains that use the Ethereum Virtual Machine (EVM).

- Ethereum MainNet, Binance, Polygon, SKALE, CheapEth, and many, many more.

Optimistic Rollups are EVM compatible too:

- Arbitrum, Optimism, and in the future many more.

zkRollups:

- All are moving towards some degree of EVM compatibility.

However, there are other non-EVM compatible blockchains and blockchain-like platforms out there!

- Bitcoin, Polkadot, Fuel, Hyperledger Fabric, Corda, and many, many more.

Most blockchains have some sort of event or message that can be emitted based on programmable logic.

Design Choices: Trust

You are always trusting someone.

For example, with Ethereum MainNet, you trust that at least 50% of the PoW miners are honest.

- **Trustless**: Trust no one.
- **Semi Trusted / Almost trustless**: trust that at least one party is honest.
- **Semi Trusted**: trust that at least a threshold number (for example ⅔ majority) are honest.
- **Trusted**: trust that a single party is honest.

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Mechanism:
  - Use hashes and preimages to **swap value.**
  - H = Message Digest (R)

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

HTLCs aren’t exactly a Crosschain Messaging protocol, but it is where they most closely fit.

Hash Timelock Contracts (HTLC):

- Mechanism:
    - Use hashes and preimages to **swap value.**
    - H = Message Digest (R)

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Mechanism:
    - Use hashes and preimages to swap value.
    - H = Message Digest (R)

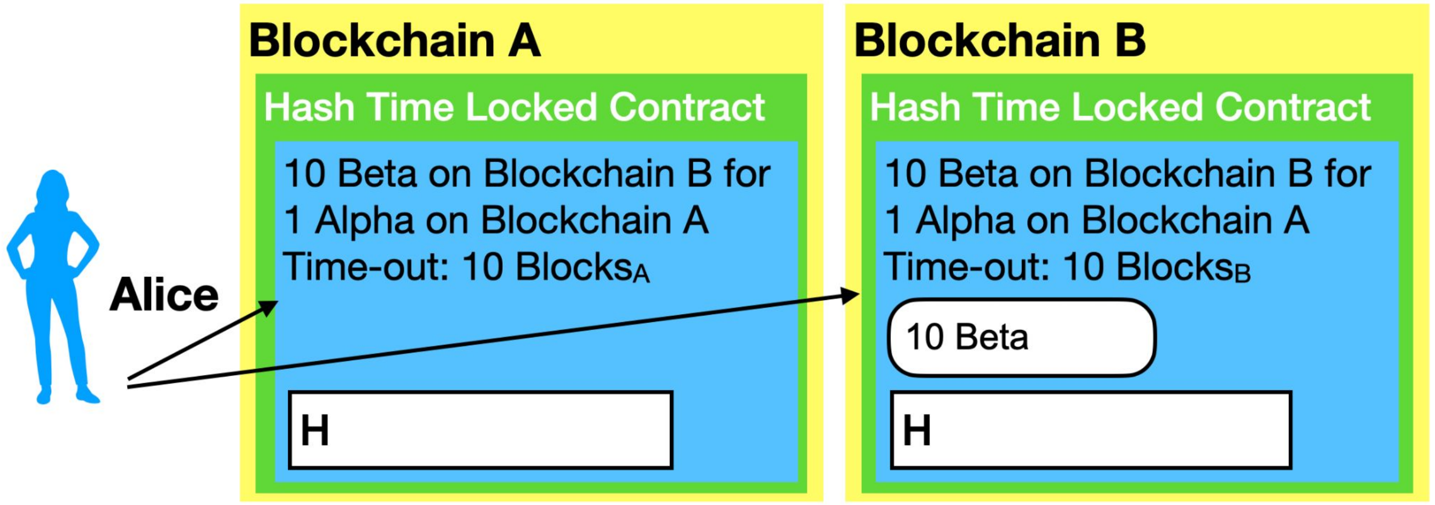
- Step 1: Set-up the deal. This is probably done off chain.

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Step 2: Alice sets up contracts and deposits coins on target blockchain.
  - Only Bob can withdraw on Blockchain B prior to the time-out. Only Alice can withdraw on Blockchain A prior to timeout.

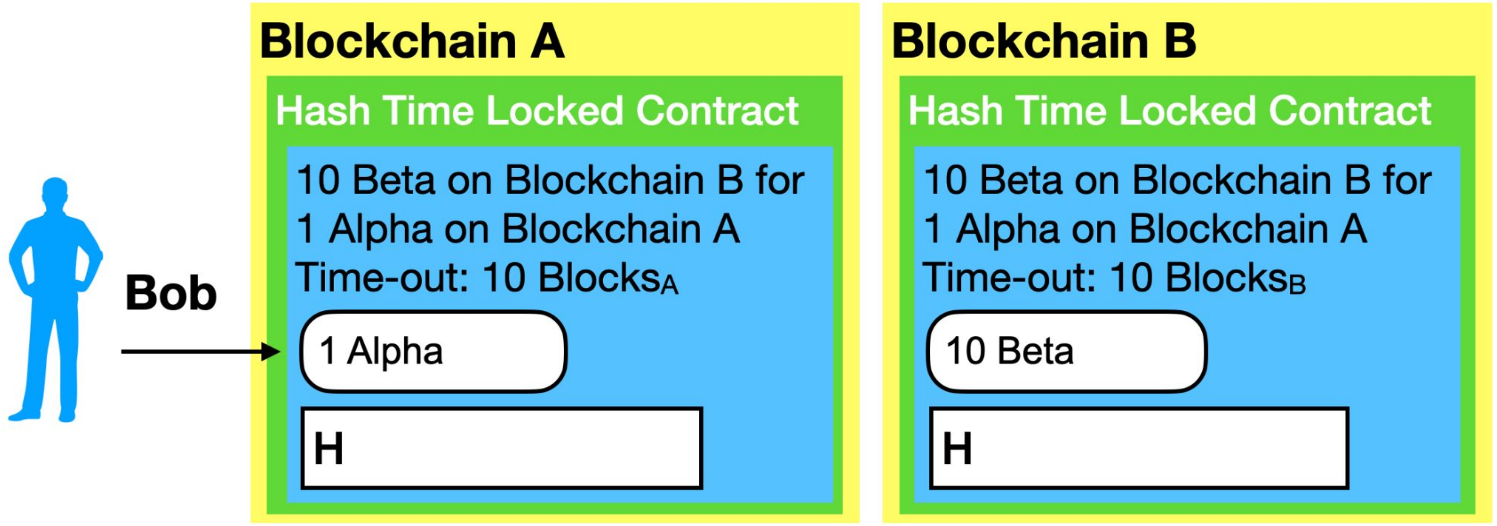


* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Step 3: Bob deposits coins on Blockchain A.

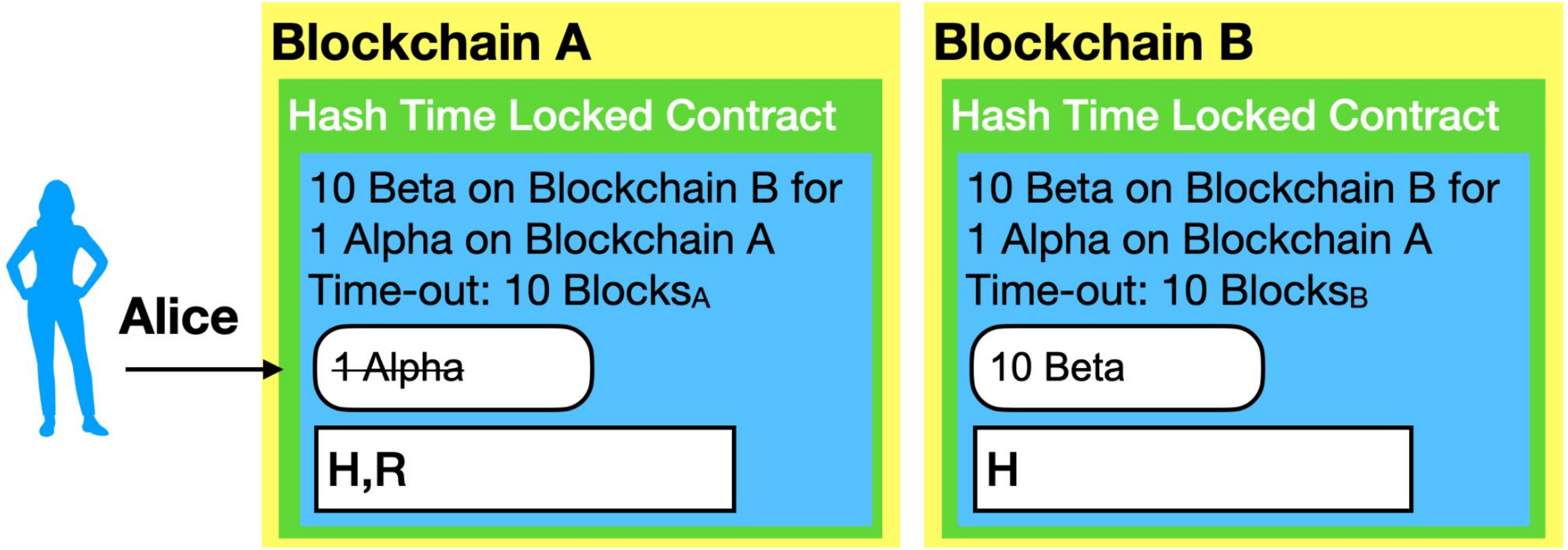


* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Step 4: Alice submits R such that H = Message Digest (R), and withdraws tokens on Blockchain A.



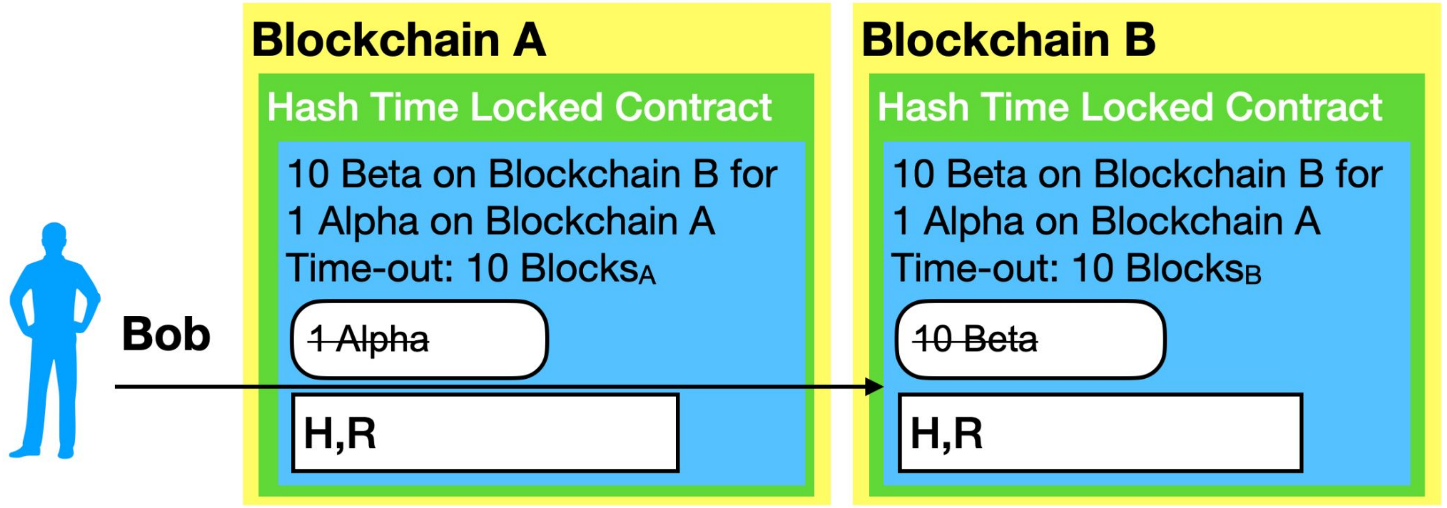
Alice
Blockchain A
Hash Time Locked Contract
10 Beta on Blockchain B for
1 Alpha on Blockchain A
Time-out: 10 Blocks_A
~~1 Alpha~~
H,R
Blockchain B
Hash Time Locked Contract
10 Beta on Blockchain B for
1 Alpha on Blockchain A
Time-out: 10 Blocks_B
10 Beta
H

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Step 5: Bob submits R such that H = Message Digest (R) and withdraws tokens on Blockchain B.



* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC):

- Advantages:
  - Trustless mechanism.
  - Atomic (usually)
- Disadvantages:
  - Limited to value transfer.
  - Subject to griefing attacks:
    - Bob could not do step 3. Alice has to wait until the timeout.
    - Alice could not submit R in step 4. Bob's coins are locked up until the time-out.
  - Requires parties to observe the blockchain:
    - There are multiple sub-steps where each party is observing the blockchain, waiting for parts of the transfer to occur.
  - Attacks:
    - The timeout should be greater for Blockchain A than Blockchain B. Otherwise, Alice waits until just before the timeout to submit R and withdraw.
    - If Bob is offline / the blockchain is busy and Bob can't execute step 5 he will lose his coins on both blockchains.

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Hash Timelock Contracts (HTLC): As a bridge.

- Two transactions on source blockchain.
- Two transactions on destination blockchain.
- Alice needs to be able to submit transactions on the source blockchain and Bob needs to be able to submit transactions on the destination blockchain.
  - How do they pay for gas before they have any value on the blockchain?
- For consortium blockchains using a IBFT consensus, the blockchain validators could be bribed to refuse to submit transactions, thus having the transaction timeout.
  - If there is just one honest validator this won't be a problem.
  - If the validators can be bribed then the blockchain is in big trouble.

Note: Deposits have been mooted as a way of stopping griefing attacks. They require more transactions, which increases the latency and cost of each transfer.

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Trustless

Modified HTLC as a bridge.

- **Connext** have modified the HTLC protocol such that Alice is a component called a Router.
- The Router does step 5 for Bob.
- Advantages:
  - Bob doesn't need to be able to pay for gas on the destination chain.
- Disadvantages:
  - Bob has to trust the Router.
  - Bob has to quickly do another transaction if the Router cheats, setting up a separate transfer so they will have some value on the destination blockchain so they can complete the first transfer.

https://medium.com/connext/nxtp-a-simpler-xchain-protocol-88760697ea04
https://docs.connext.network/QuickStart/transfer
https://github.com/connext/nxtp

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Mechanism:
  - Use the PoW hashing power of the source blockchain to be sure that information can be trusted.
  - Operate a light client of a PoW blockchain on another blockchain.

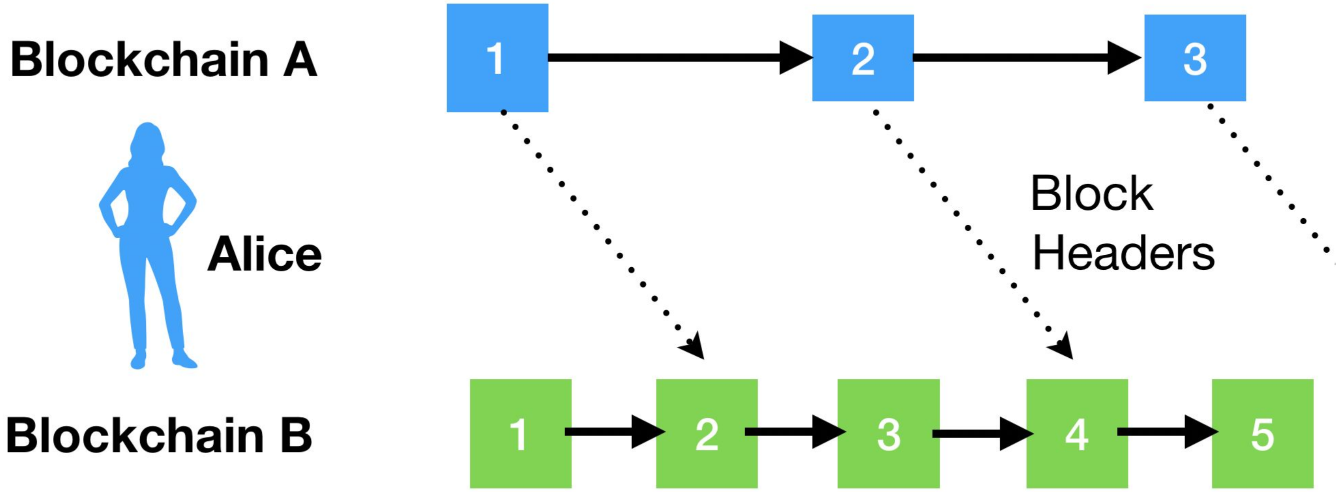
- Step 1: Deploy a contract to the destination blockchain. Specify the genesis block or another starting block of the source blockchain and number of block confirmations to use.

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Step 2: One or more (hopefully many) relayers compete to transfer block headers to the destination blockchain. Block headers are accepted if they match the PoW algorithm for the source blockchain.

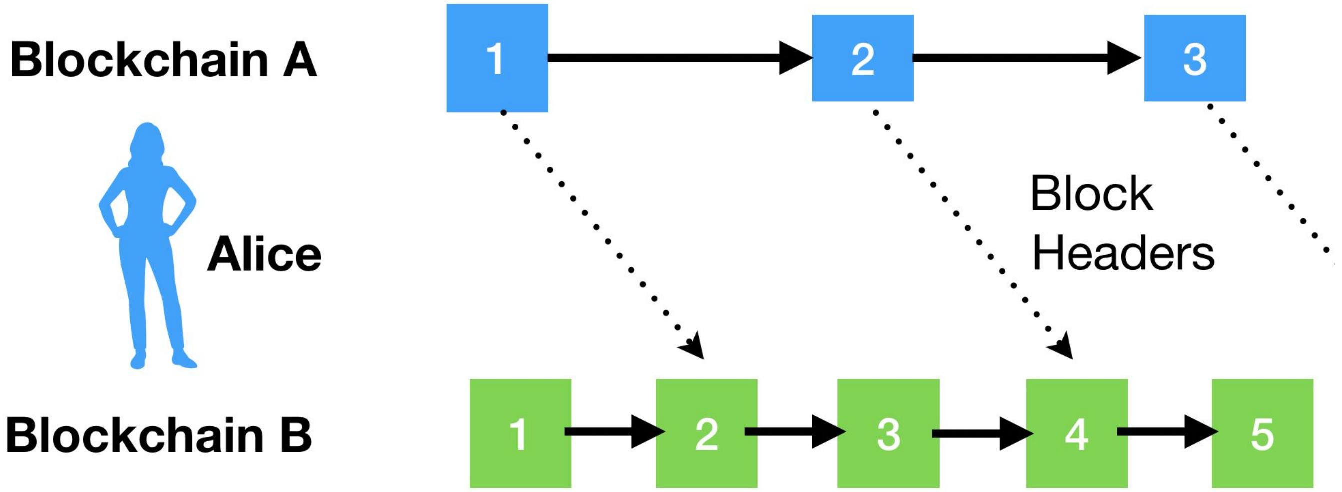


CONSENSYS * For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Step 3: Block headers are usable on the destination blockchain after enough block headers have been added to the end of the chain. This indicates the number of block confirmations.

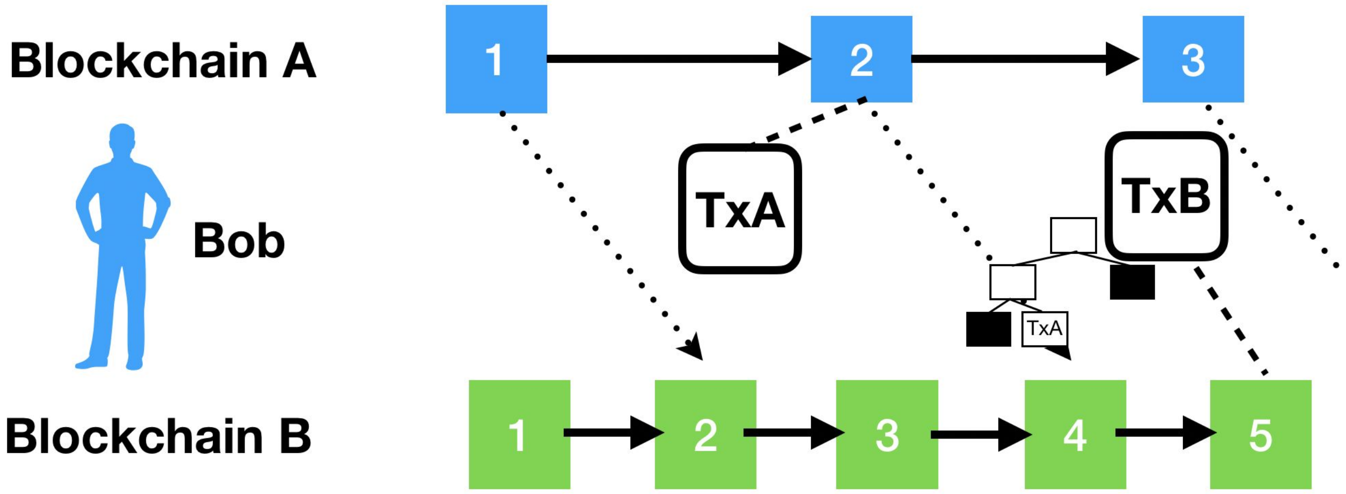


* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Step 4: Bob executes a transaction on the source blockchain (Bitcoin).

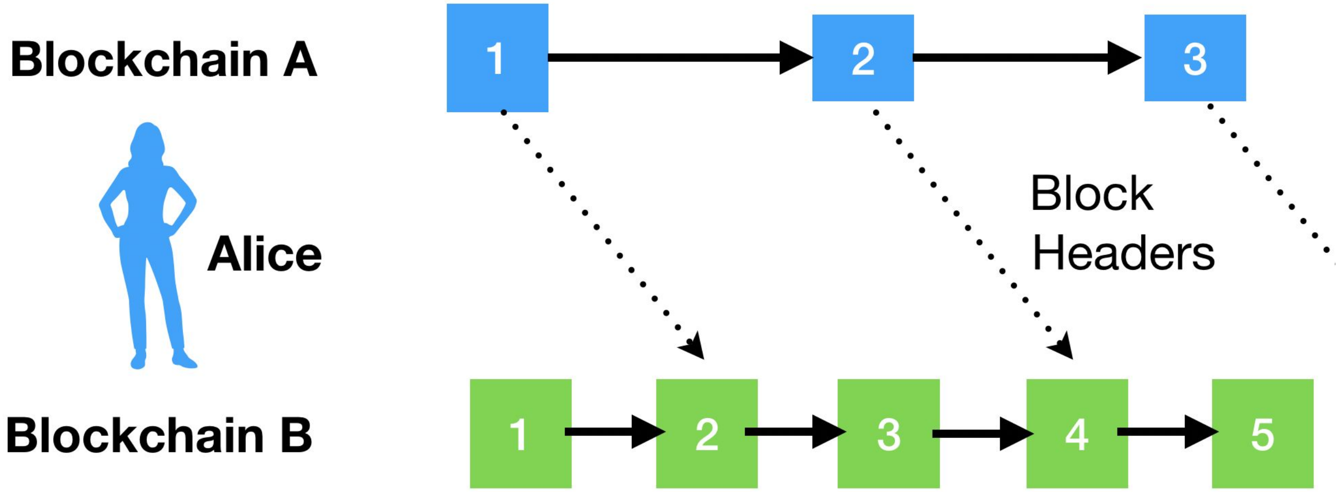


* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Step 5: The block header for the transaction containing Bob's transaction is relayed to the destination blockchain. Enough block headers are transferred that build on the block header to meet the block confirmation rules.

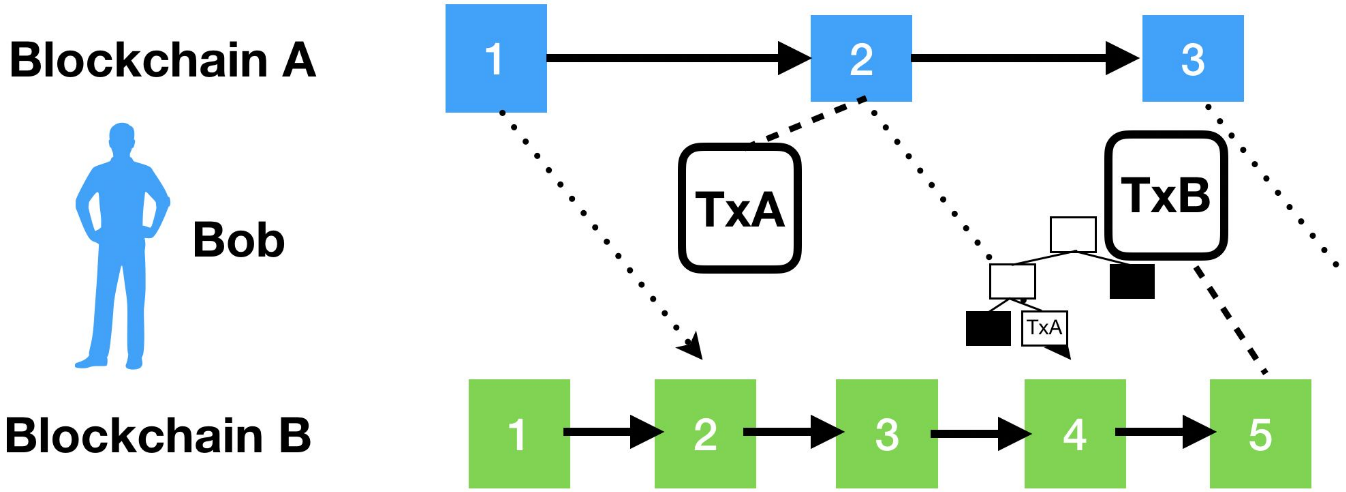


* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Step 6: Bob submits a transaction on the destination blockchain including the transaction from the source chain and a Merkle Proof showing the transaction was included in a block on the source blockchain (Bitcoin).



* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

Compensation model:

- Users pay to confirm a transaction using the contract.
- The Relayer that transferred the block header gets compensated.

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Advantages:
  - Only need one honest relayer.
    - Malicious relayers can submit block headers from forks of the canonical chain.
    - The malicious relayers can not create a longer change that the canonical chain.
    - Any honest relayer can submit block headers from the canonical chain. The forks will then not be useable.
- Disadvantages:
  - Liveness issues: all block headers must be transferred; even if there are no transactions.
  - Becomes uneconomical if not enough transactions. Last transaction in 2018.
    - https://etherscan.io/address/0x41f274c0023f83391de4e0733c609df5a124c3d4
  - PoW source blockchain's hashing power must be large enough that 51% attacks are infeasible: hence Ethereum MainNet and Bitcoin.
  - PoW algorithm must be able to be implemented on target blockchain.

* For details of methodology, see: https://doi.org/10.1016/j.comnet.2021.108488

Design Choices: Semi-Trusted / Almost Trustless

Simple Payment Verification / Light Client / BTC Relay:

- Attack*: Lys et al identified a safety issue:
  - Eve has $1000 of Eth on Ethereum MainNet.
  - Eve sets up the Eth in a contract with the transfer to whoever submits a transaction demonstrating they have transferred BTC to her account on Bitcoin.
  - Eve convinces Alice and Bob to transfer $1000 of BTC to her account on Bitcoin, based on the Eth in the contract on Ethereum MainNet.
  - Only Alice or Bob will be able to get the Eth based on confirming their transaction.
  - Eve gets $2000 of BTC, Alice gets $1000 of Eth, and Bob has $0.
- Attack*: Lys et al identified a liveness issue:
  - Eve convinces Alice to have $1000 of Eth on Ethereum MainNet locked in a contract, waiting for a Bitcoin transaction confirmation.
  - Eve then walks away and never transfers the BTC.
  - Alice has her $1000 locked in the contract forever.
- Attack: Martin Swende identified a software defect:
  - BTC transactions could be confirmed without paying for block headers by using an exposed internal API that didn't do the appropriate checks.

https://hal.archives-ouvertes.fr/hal-03212152
https://swende.se/blog/BTCRelay-Auditing.html

Design Choices: Semi-Trusted / Almost Trustless

Rainbow Bridge by Near Protocol:

- For Ethereum MainNet to Near, uses Ethereum MainNet's PoW to secure the bridge.
- Massive feat. Ethereum MainNet's PoW algorithm is time-memory hard, not just time hard like Bitcoin's.

Issues:

- Ethereum MainNet merge is due to happen in Q1, 2022. At this point Ethereum MainNet will use PoS.

Design Choices: Semi Trusted and Trusted

- All(?) approaches come down to a threshold number of relayers, validators, or attestors signing or doing multi-party computation, on some information from the source blockchain.
- For a trusted system, there is only 1 relayer / validator / attestor, and the threshold is 1.
- Public keys of signers are stored on target blockchains.
    - For Ethereum, the address is stored as a proxy for the public key.

Semi Trusted and Trusted Schemes

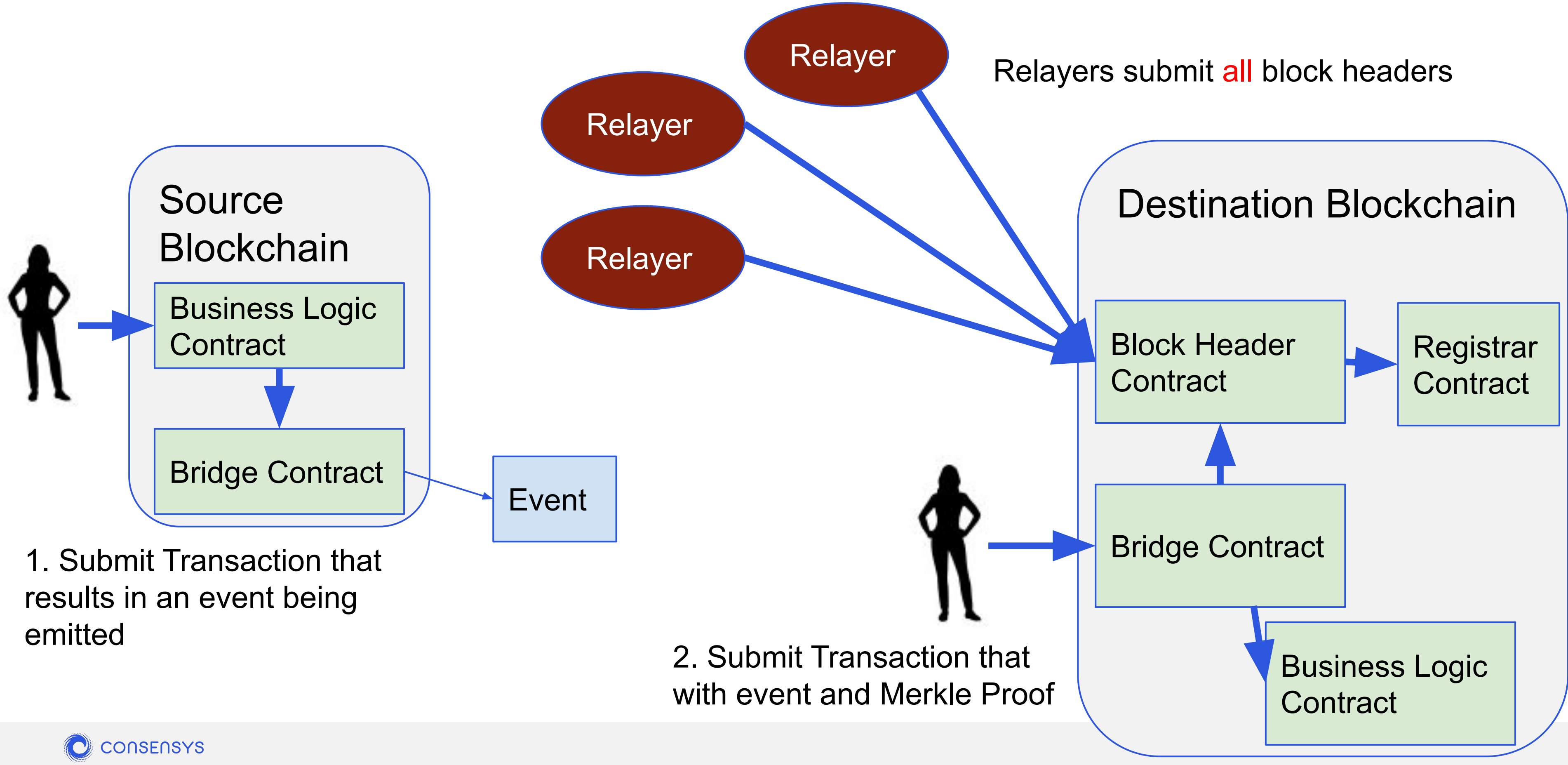
- The next slides walk through the myriad of similar but different techniques.
- They explore the trade-offs being made.

Choices:

- Sign individual messages, or sign a Merkle Root?
- Create a Merkle Tree of messages, or use block headers and transactions within blocks?
- Have relayers / attestors cooperatively sign, or have them separately submit information to blockchains?
- Do relayers / attestors have to respond to requests from users?
- Who submits the transaction on the destination blockchain?

Block header transfer, separately submit header

This is approximately how Clearmatics' Ion Framework works



Block header transfer, separately submit header

This is approximately how Clearmatics' Ion Framework works

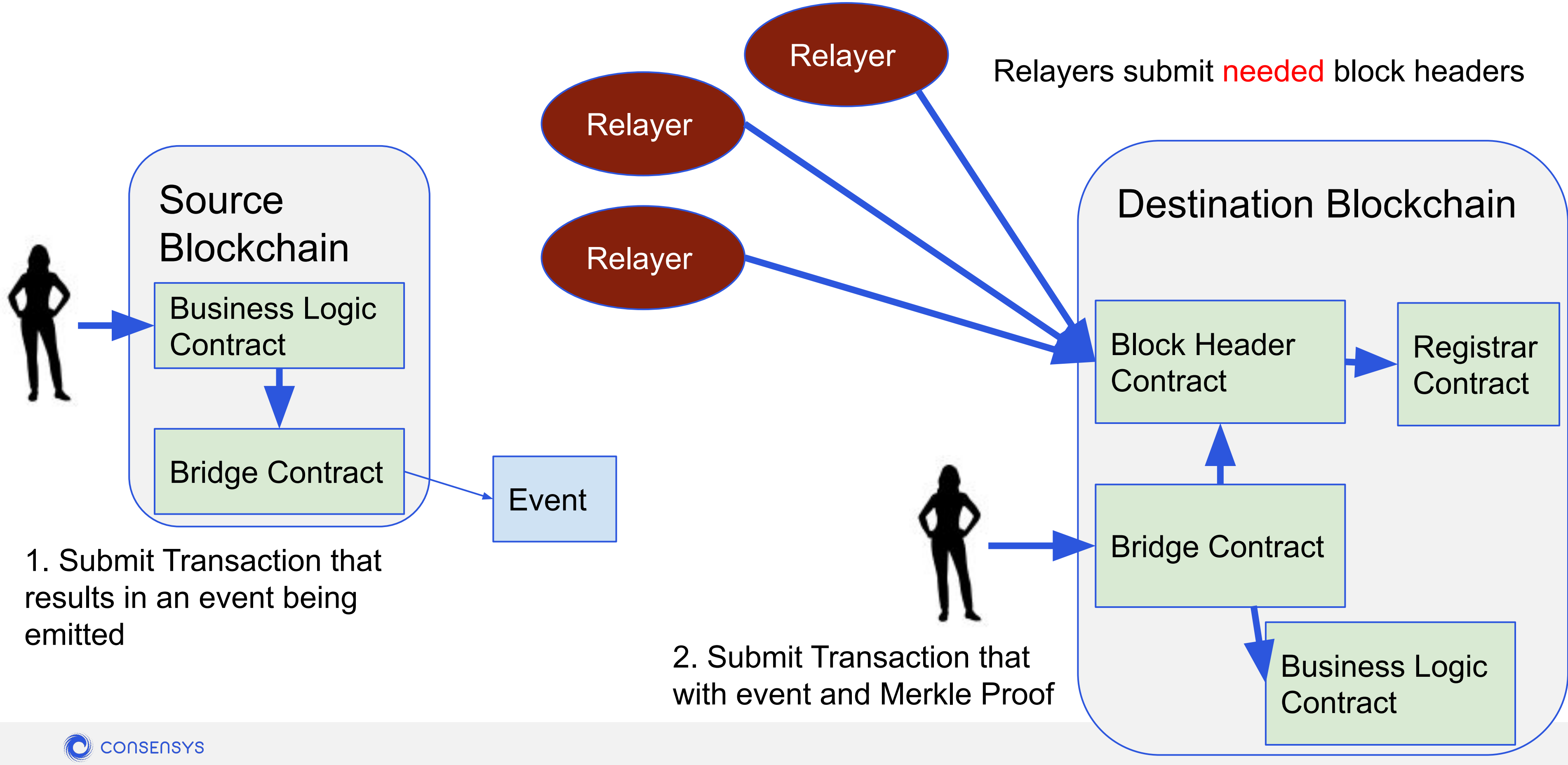
Advantage:

- Simple.
- Relayers do not need to cooperate.
- Users do not interact with Relayers.

Disadvantage:

- One transaction for each relayer, for each block to submit signed block headers!
- If there are multiple target blockchains, this process has to be repeated for all target blockchains.
- Relayers are paying to submit transactions on the destination blockchain.
- User has to be able to submit a transaction on the destination blockchain.

Block header transfer, each separately submit block header



Block header transfer, each separately submit block header

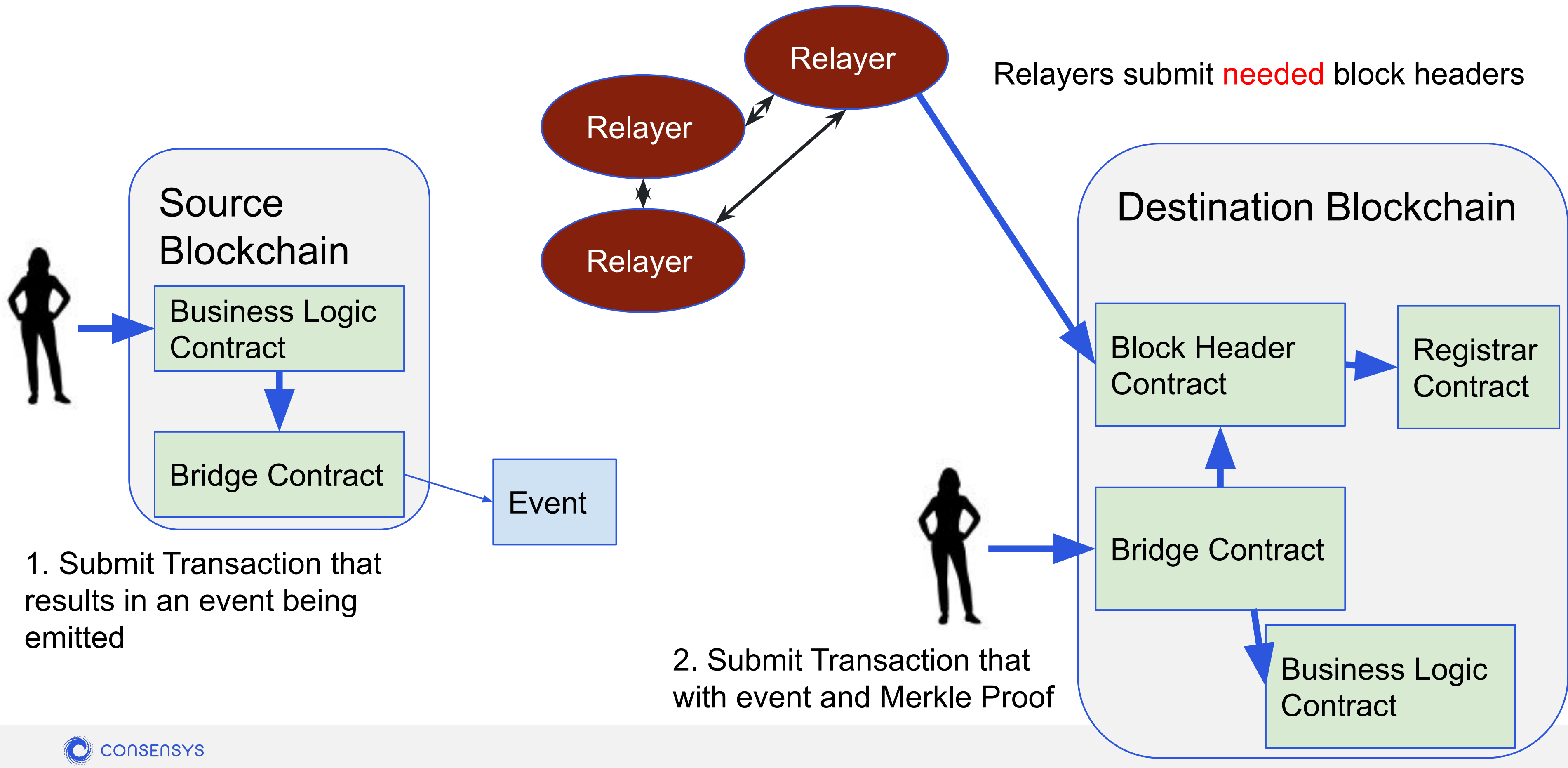
Advantage:

- One transaction per Relayer per block header that is needed.
- Relayers do not need to cooperate.
- Users do not interact with Relayers.

Disadvantage:

- Relayers need to analyse transactions in blocks to determine which block headers need to be transferred.
- If there are multiple target blockchains, relayers need to understand which blockchain an event will be used on, OR block headers that might be needed need to be communicated with all blockchains.
- Relayers are paying to submit transactions on the destination blockchain.
- User has to be able to submit a transaction on the destination blockchain.

Block header transfer, Relayers cooperate to multiply sign



Block header transfer, Relayers cooperate to multiply sign

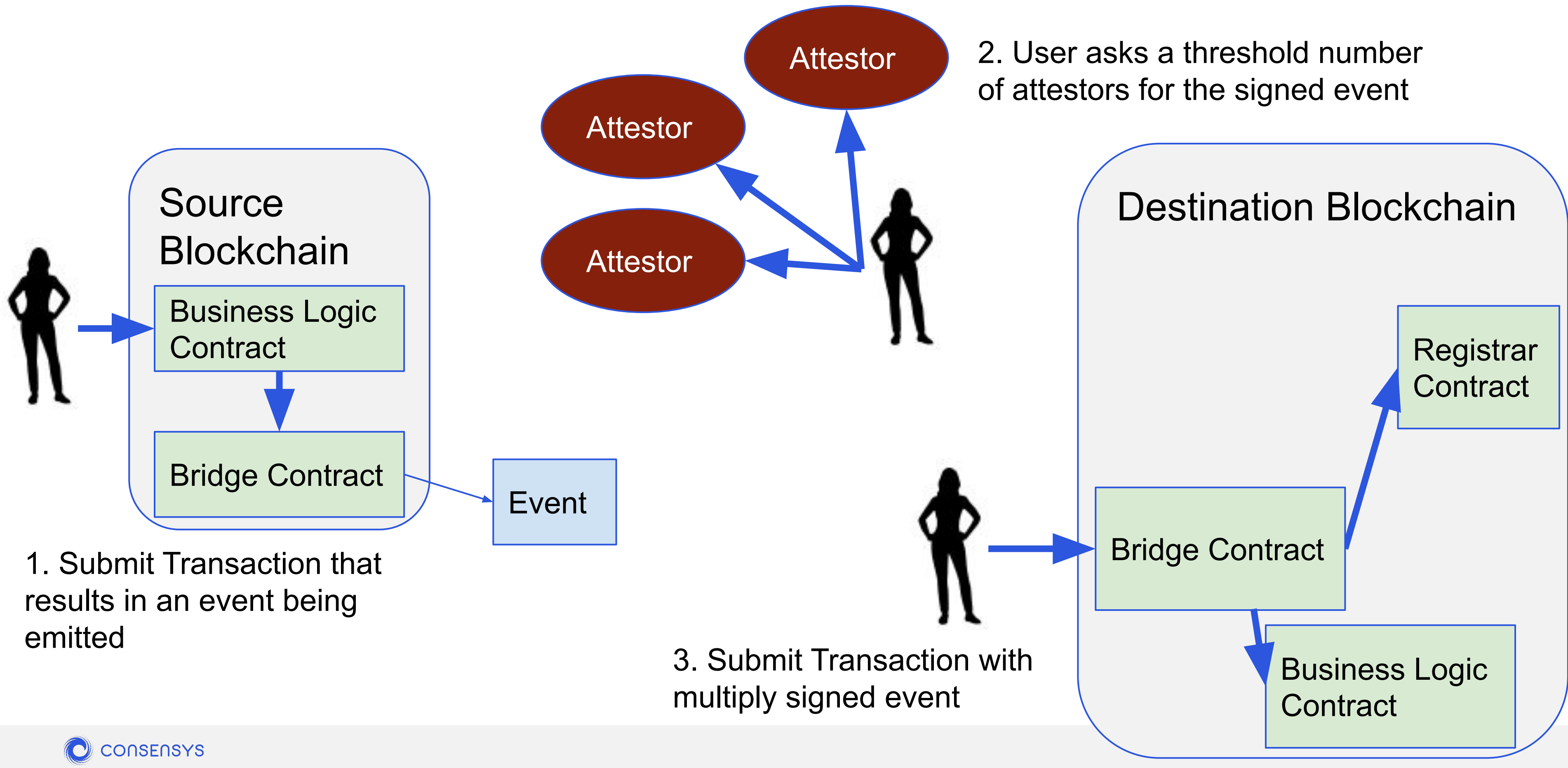
Advantage:

- One transaction per block header that is needed.
- Users do not interact with Relayers.

Disadvantage:

- Relayers need to cooperate.
- Relayers need to analyse transactions in blocks to determine which block headers need to be transferred.
- If there are multiple target blockchains, relayers need to understand which blockchain an event will be used on, OR block headers that might be needed need to be communicated with all blockchains.
- Relayers are paying to submit transactions on the destination blockchain.
- User has to be able to submit a transaction on the destination blockchain.

Event sign, separate Attestor sign



1. Submit Transaction that results in an event being emitted

2. User asks a threshold number of attestors for the signed event

3. Submit Transaction with multiply signed event

Event sign, separate Attestor sign

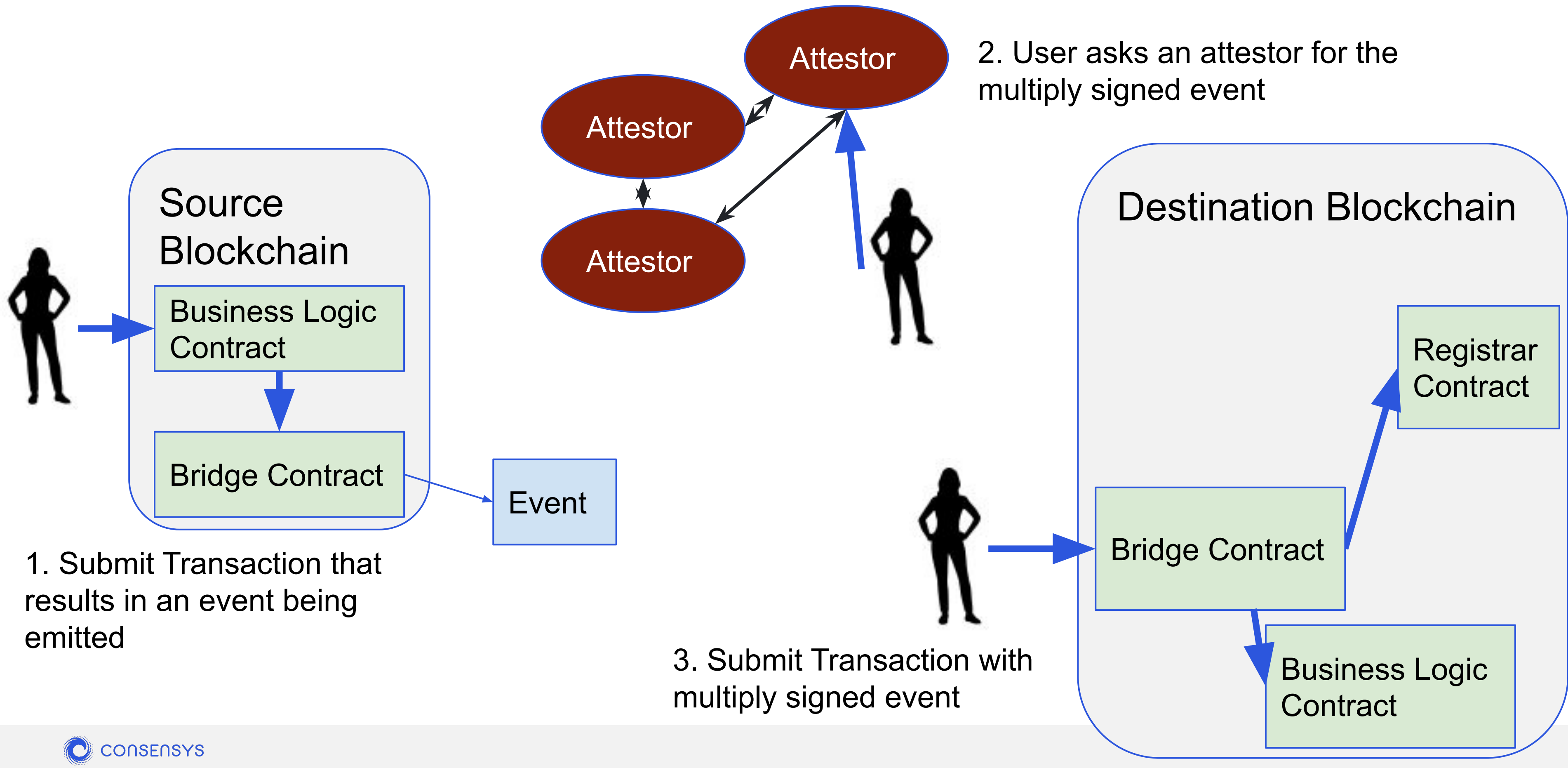
Advantage:

- Attestors do not submit transactions on the destination blockchain.
- Attestors sign all transactions from a certain contract. They don't need to know understand the target blockchain.
- Attestors do not need to cooperate.

Disadvantage:

- Users interact with Attestors.
- User has to be able to submit a transaction on the destination blockchain.

Event sign, Attestors cooperate to multiply sign


Attestor
Attestor
Attestor
2. User asks an attestor for the multiply signed event
Source Blockchain
Business Logic Contract
Bridge Contract
Event
Destination Blockchain
Registrar Contract
Bridge Contract
Business Logic Contract
1. Submit Transaction that results in an event being emitted
3. Submit Transaction with multiply signed event

Event sign, Attestors cooperate to multiply sign

Advantage:

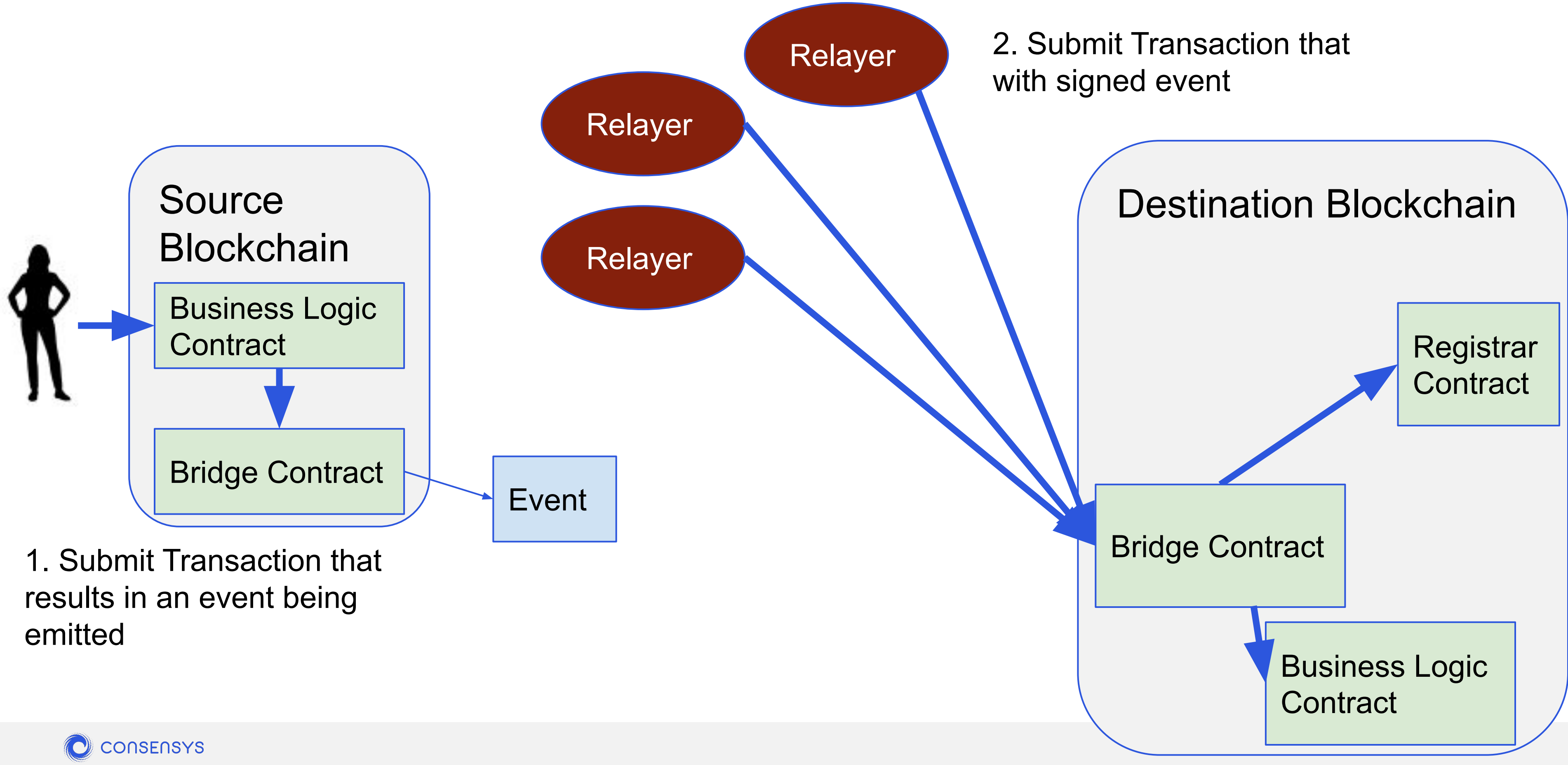
- Attestors do not submit transactions on the destination blockchain.
- Attestors sign all transactions from a certain contract. They don’t need to know understand the target blockchain.

Disadvantage:

- Attestors need to cooperate.
- Users interact with Attestors.
- User has to be able to submit a transaction on the destination blockchain.

Relayer separately submit signed event

This is approximately how ChainSafe's ChainBridge works



Relayer separately submit signed event

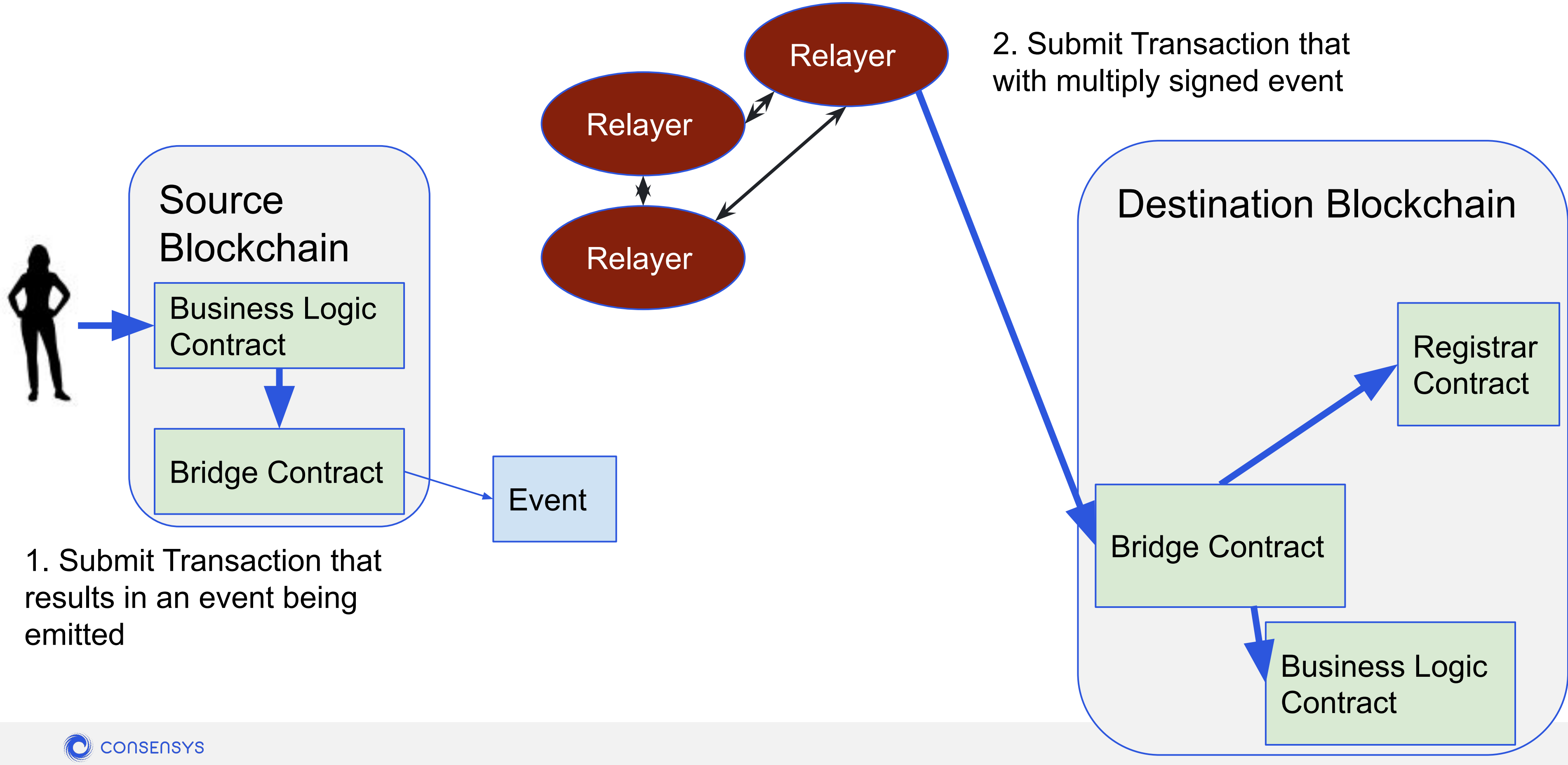
Advantage:

- Relayers do not need to cooperate.
- Users do not interact with Relayers.
- Users do not need to be able to submit transactions on the destination blockchain.

Disadvantage:

- One transaction per Relayer per event.
- Relayers submit transactions on the destination blockchain.
- Relayers need to know understand the target blockchain for an event.

Relayer each separately submit signed event



Relayer each separately submit signed event

Advantage:

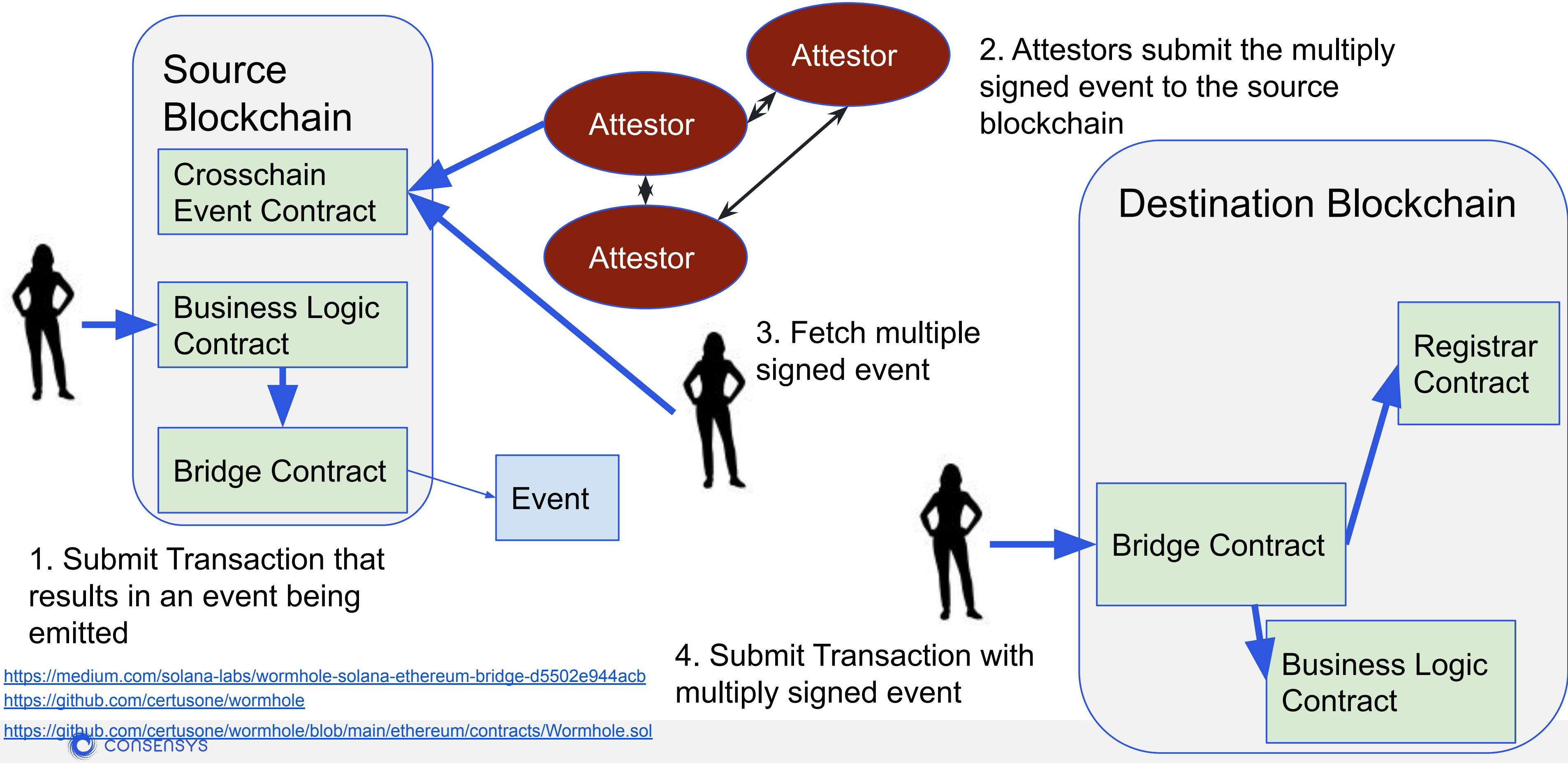
- Relayers submit one transaction per event.
- Users do not interact with Relayers.
- Users do not need to be able to submit transactions on the destination blockchain.

Disadvantage:

- Relayers need to cooperate.
- Relayers submit transactions on the destination blockchain.
- Relayers need to know understand the target blockchain for an event.

Attestors submit multiply signed event to source

This is approximately how the Wormhole Solana - Ethereum MainNet bridge works


Source Blockchain
Crosschain Event Contract
Business Logic Contract
Bridge Contract
Event
Attestor
Attestor
Attestor
2. Attestors submit the multiply signed event to the source blockchain
3. Fetch multiple signed event
Destination Blockchain
Registrar Contract
Bridge Contract
Business Logic Contract
1. Submit Transaction that results in an event being emitted
4. Submit Transaction with multiply signed event

https://medium.com/solana-labs/wormhole-solana-ethereum-bridge-d5502e944acb
https://github.com/certusone/wormhole
https://github.com/certusone/wormhole/blob/main/ethereum/contracts/Wormhole.sol

Attestors submit multiply signed event to source

This is approximately how the Wormhole Solana - Ethereum MainNet bridge works

Advantage:

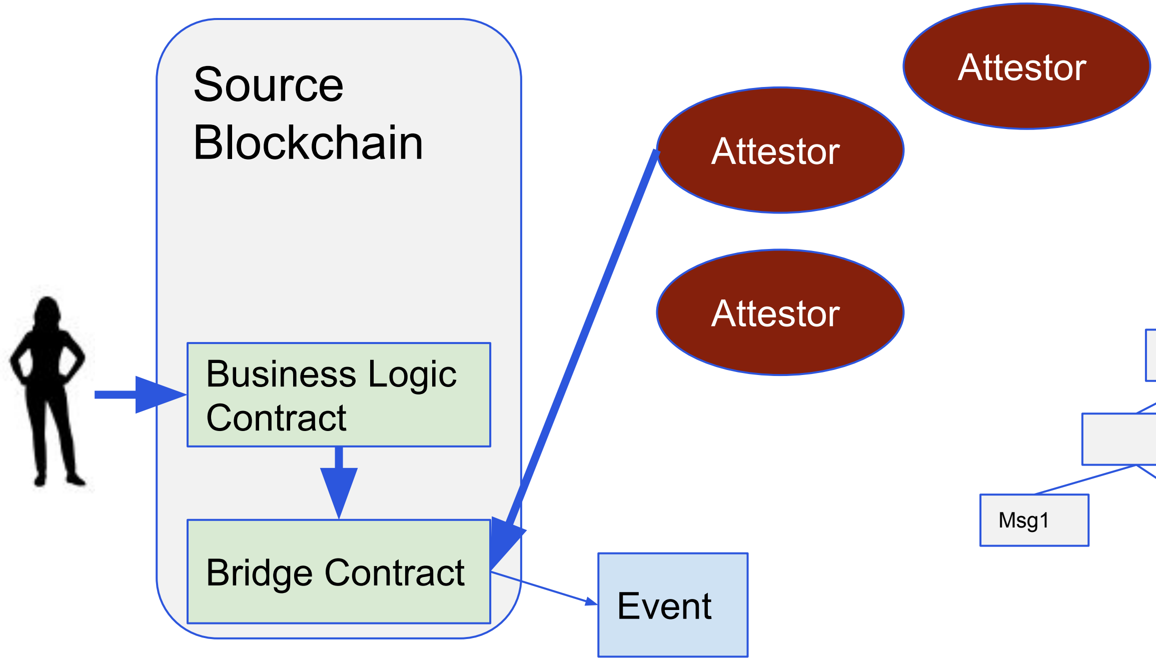
- Relayers do not submit any transactions to the destination blockchain.
- Relayers need to need to understand the target blockchain for an event.
- Users do not interact with Relayers.

Disadvantage:

- Relayers need to cooperate.
- User has to be able to submit a transaction on the destination blockchain.

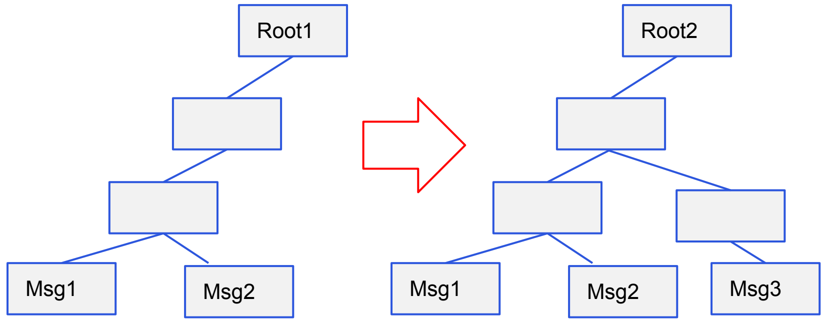
Incremental Merkle Tree

This is approximately how the Celo Optics bridge works



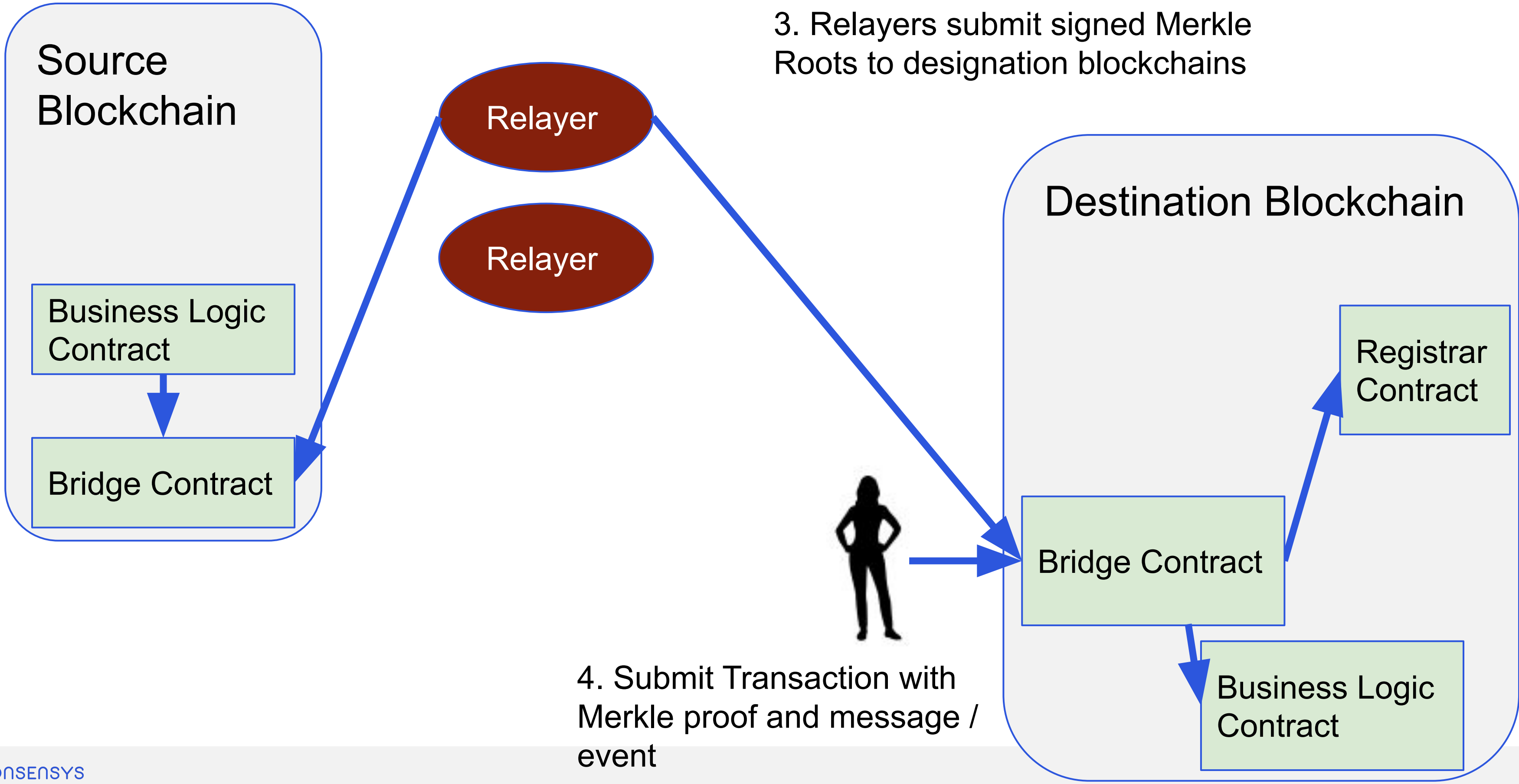
1. Submit Transaction that results in a message / event being emitted

2. Attestors (Updaters) order messages, and submit signed Merkle Roots to source blockchain



Incremental Merkle Tree

This is approximately how the Celo Optics bridge works



3. Relayers submit signed Merkle Roots to designation blockchains

4. Submit Transaction with Merkle proof and message / event

Celo Optics

- Celo Optics relies on only one Relayer transferring a Merkle Root.
    - This is to save gas.
- The Merkle Root can not be used until a “Fraud Window” has expired.
- Observers are expected to submit malicious Merkle Roots back to the source blockchain so that malicious Attestors (Updaters) can be slashed.
- Applications need to check each message and reject / not act on fraudulent messages.
    - This seems like a big imposition on applications.

Attestors submit multiply signed event to source

This is approximately how the Celo Optics bridge works

Advantage:

- Relayers do not submit any transactions to the destination blockchain.
- Relayers need to need to understand the target blockchain for an event.
- Users do not interact with Relayers.

Disadvantage:

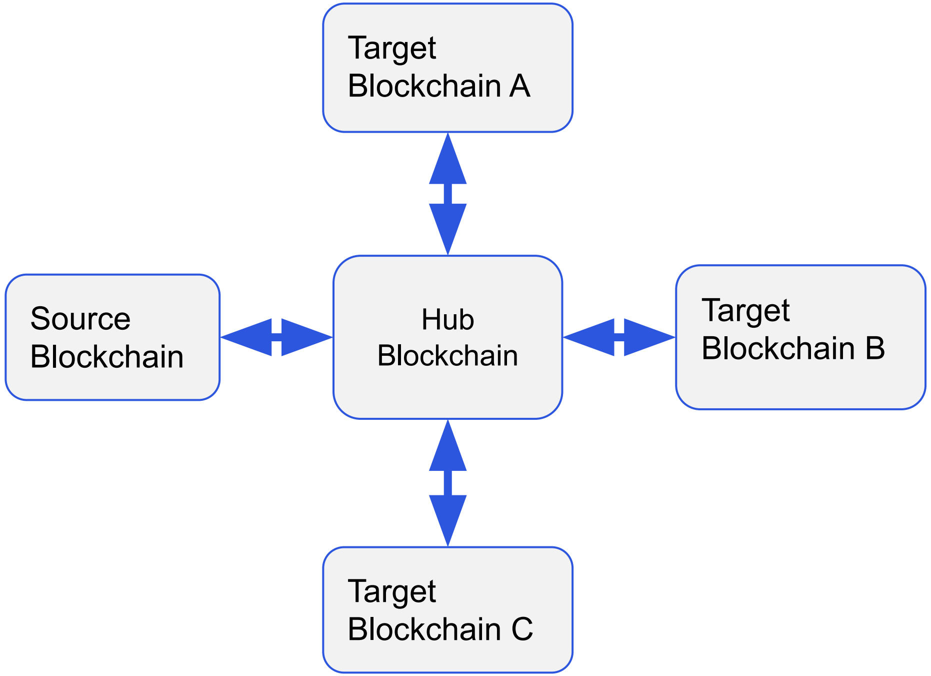
- Relayers need to cooperate.
- User has to be able to submit a transaction on the destination blockchain.

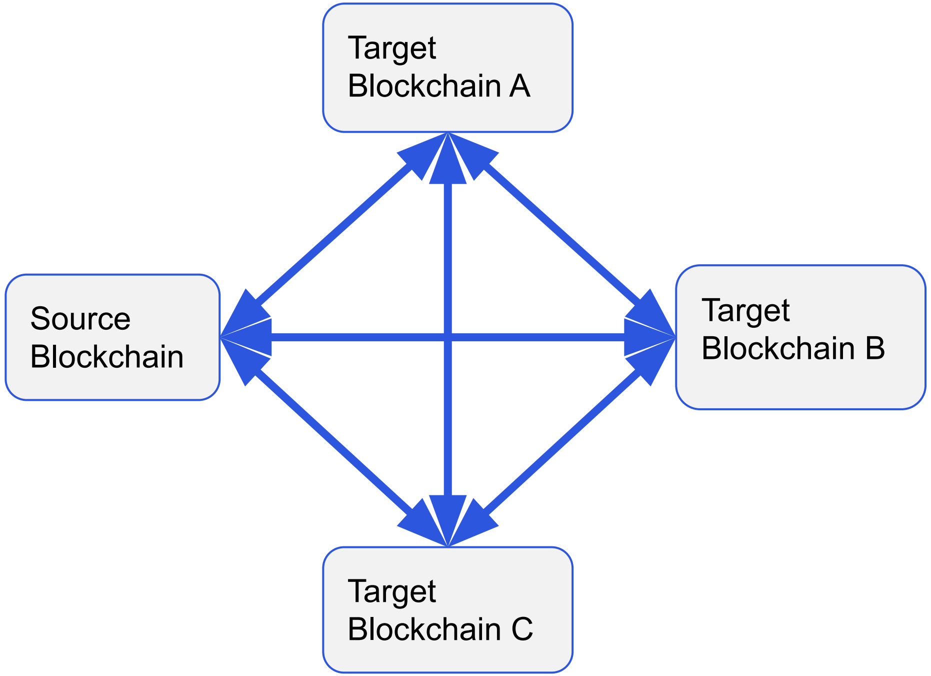
Signature Schemes

- ECDSA: Used for transaction signing in Ethereum. Fast.
- BLS:
  - Signature verification is orders of magnitude slower that ECDSA.
  - Mathematical properties allow signatures and public keys to be added.
- BLS Aggregated Signatures / Threshold Signatures:
  - Attestors independently generate their private keys and public keys.
  - Add multiple signatures, indicate who has signed.
  - Ethereum 2 is using this technique.
- BLS Threshold Signatures:
  - Attestors independently generate private key shares and cooperate to generate a combined public key.
  - Cooperate to create combined signatures.
  - Hides which attestors signed.
  - Complicated set-up.
- Schnorr Aggregated Signatures / Threshold Signatures:
  - Relayers independently generate their private keys and public keys.
  - Combine multiple signatures, indicate who has signed.
  - Wanchain is using this technique.

Hub Blockchains

WanChain, PolyNetwork, and Cosmos use hub blockchain.

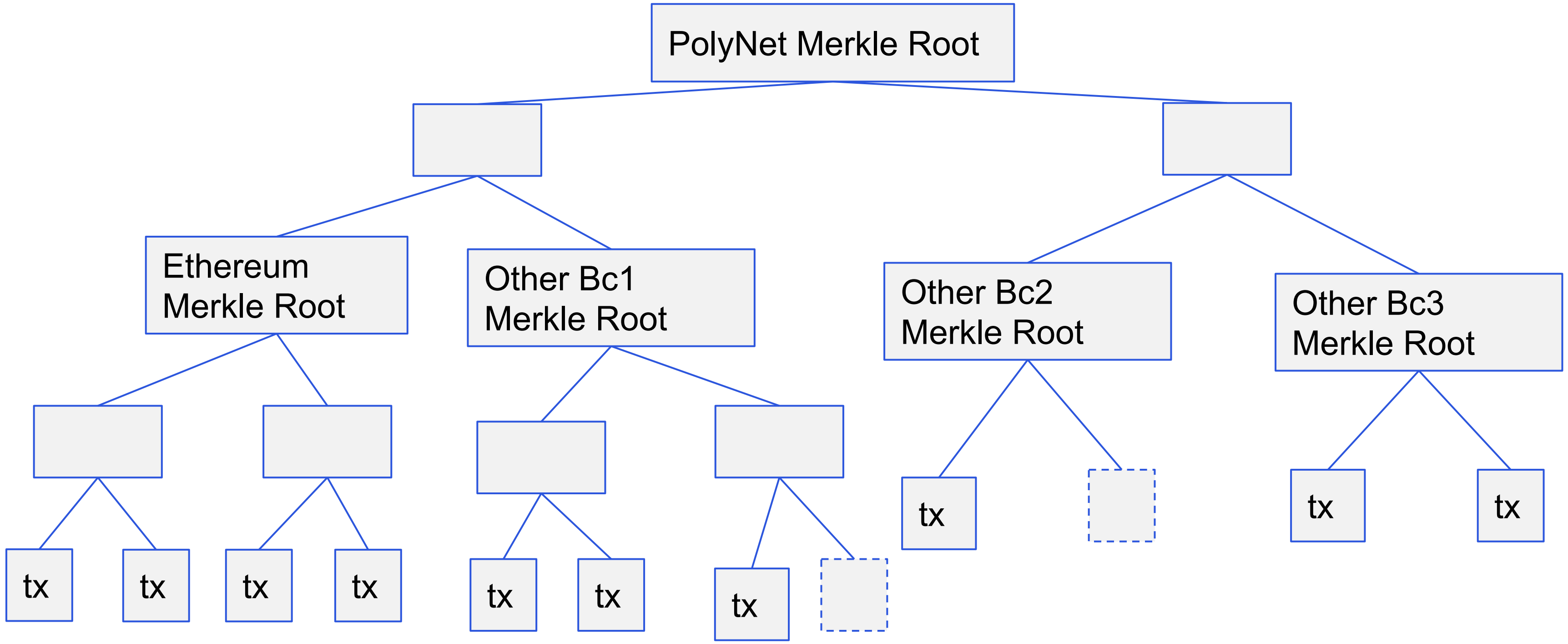




Hub Blockchains

- Reduce the number of bridges required if all blockchains need to be interconnected.
- Do all blockchains need to be connected?
- Hub blockchains will charge $ to use their services.
  - Most bridges, whether via a hub or not are likely to charge $ for their service.
- Hubs increase the latency of crosschain transactions.

Hub Blockchains with Aggregated Merkle Tree



Staking and Slashing

Attestors / Relayers can be required to stake some $.

Questions to consider:

- How much is being staked?
- How does the amount being staked relate to the amount that could be stolen?
- In what situations can a relayer be slashed?
- How is the misbehaviour proven? That is, what cryptographic enforcement is there?

Transaction Issues

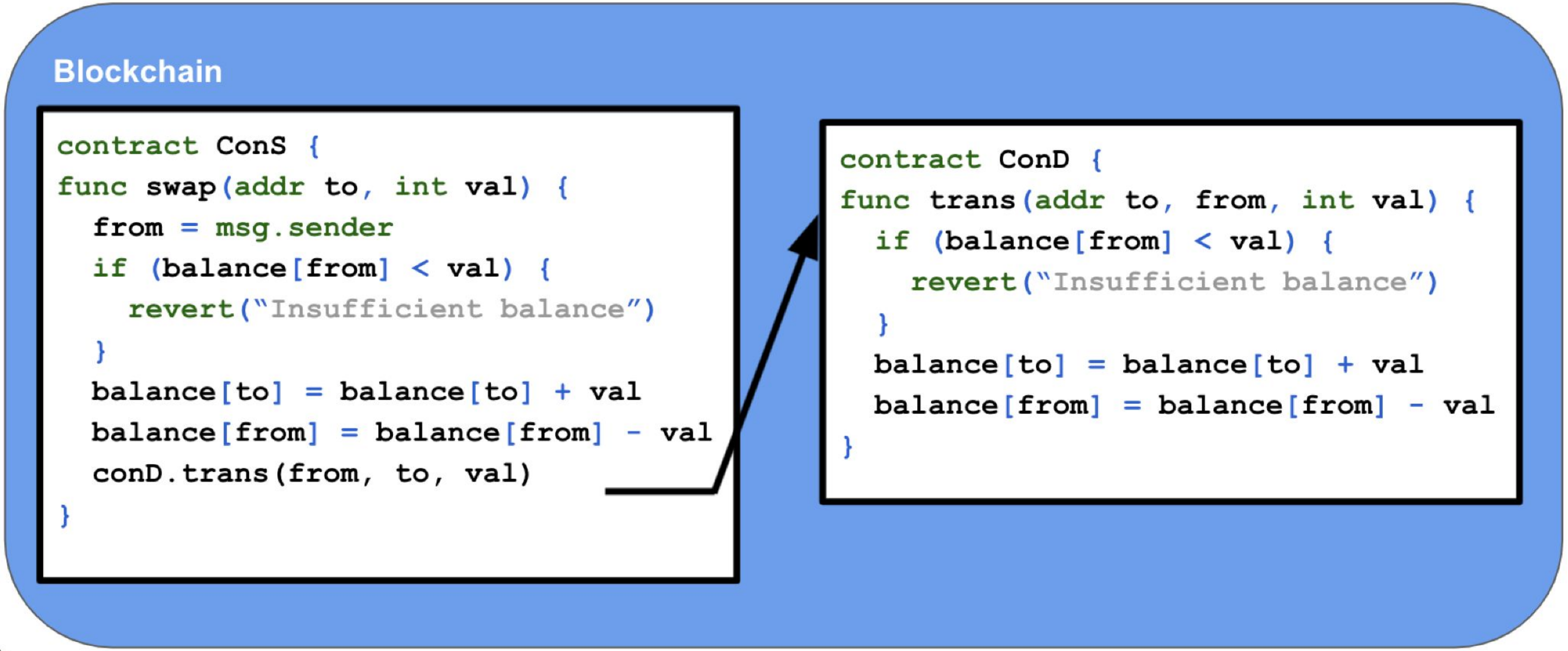
What happens when a transaction fails?

- Do relayers retry the transaction?
- Should a repriced transaction be submitted (that is, same nonce, higher tip / gas price)?

CONSENSYS

Design Choices: Crosschain Function Call Layer

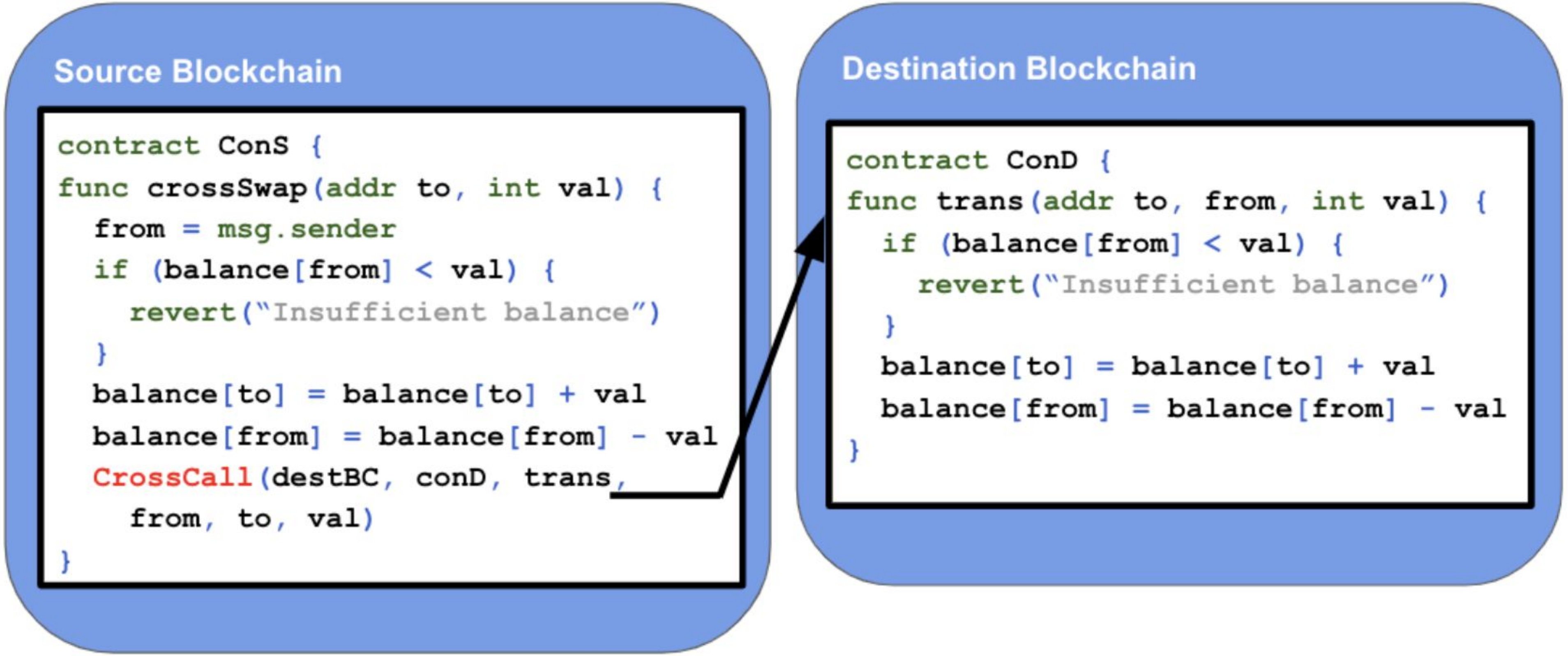
Function Calls: Single Blockchain



```
contract ConS {
func swap(addr to, int val) {
  from = msg.sender
  if (balance[from] < val) {
    revert("Insufficient balance")
  }
  balance[to] = balance[to] + val
  balance[from] = balance[from] - val
  conD.trans(from, to, val)
}
```

```
contract ConD {
func trans(addr to, from, int val) {
  if (balance[from] < val) {
    revert("Insufficient balance")
  }
  balance[to] = balance[to] + val
  balance[from] = balance[from] - val
}
```

Function Calls: Crosschain



```
contract ConS {
func crossSwap(addr to, int val) {
  from = msg.sender
  if (balance[from] < val) {
    revert("Insufficient balance")
  }
  balance[to] = balance[to] + val
  balance[from] = balance[from] - val
  CrossCall(destBC, conD, trans,
    from, to, val)
}
```

```
contract ConD {
func trans(addr to, from, int val) {
  if (balance[from] < val) {
    revert("Insufficient balance")
  }
  balance[to] = balance[to] + val
  balance[from] = balance[from] - val
}
```

Function Call Layer Design: Simple Function Calls (not atomic)

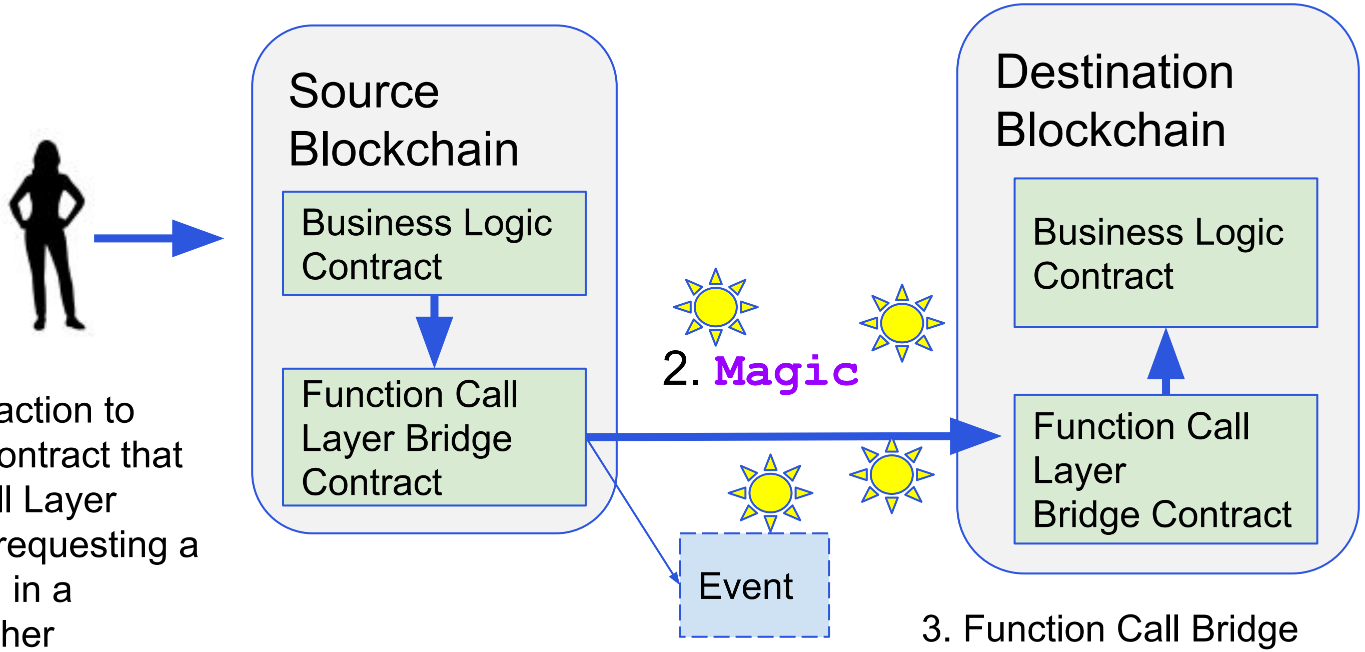
Simple Function Calls:

- Call a function on a contract on a blockchain indicated in an event.

Example implementation:

- https://github.com/ConsenSys/gpact
- https://github.com/ConsenSys/gpact/tree/main/functioncall/sfc

Function Call Layer Design: Simple Function Calls (not atomic)



1. Submit Transaction to business logic contract that the Function Call Layer Bridge contract requesting a call to a function in a contract on another blockchain

3. Function Call Bridge contract calls a function in a contract

Function Call Layer Design: Simple Function Calls (not atomic)

- Updates across blockchains are NOT atomic.
- That is: the transaction on the destination blockchain might fail.

Reasons why a transaction might fail:

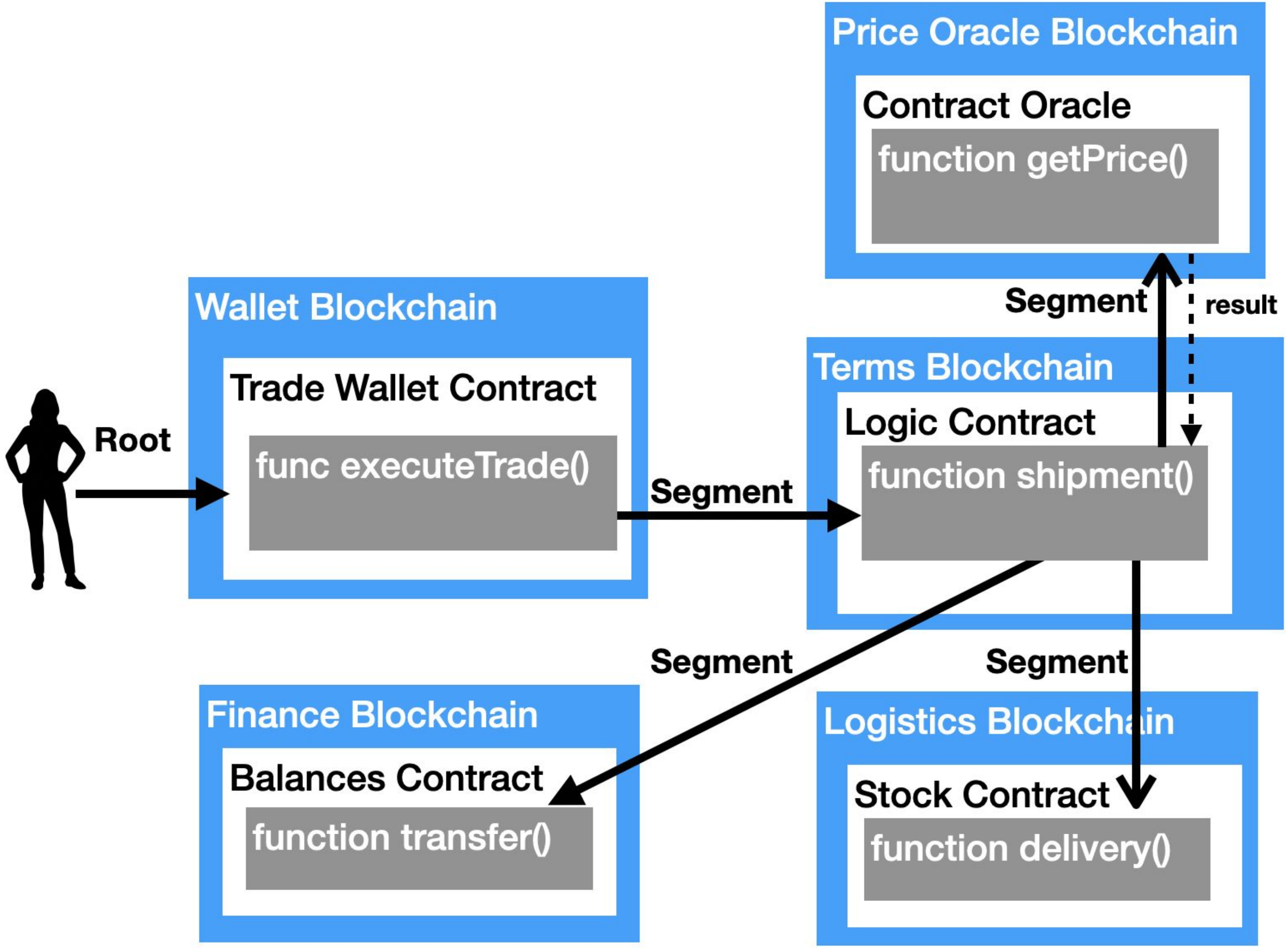
- The **revert** in the trans function could be triggered.
- The account that submitted the transaction on the destination account might not have enough Ether to pay for the gas to execute the transaction.
- The account might not have permission to execute the transaction. This could be at the Solidity contract level, where permissioning may have been configured to limit which accounts can call a function. In a permissioned blockchain network, only certain accounts might be permissioned to submit transactions.
- The transaction may have been submitted with a gas price that is too low given the current gas price required to have a transaction included in a block.
- The transaction may be incorrectly configured, for instance with an incorrect nonce value.

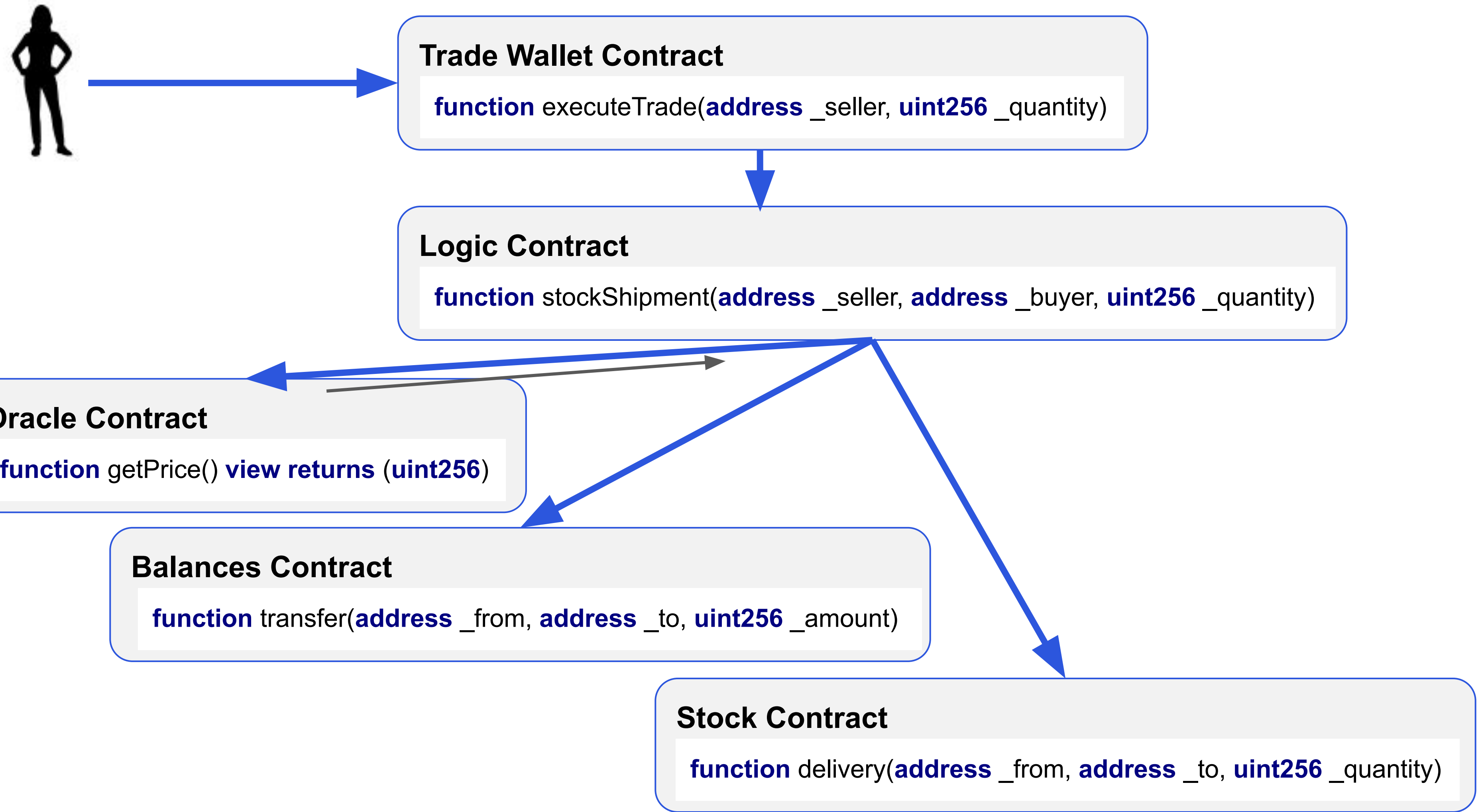
General Purpose Atomic Crosschain Transactions (GPACT)

- Atomic crosschain function call protocol:
    - Updates across blockchains are either all committed or are all discarded.
- Two phase commit scheme for call execution trees.
- Provides composable, synchronous execution model that applications developers are accustomed to.

Open source repo:

- https://github.com/ConsenSys/gpact
- https://github.com/ConsenSys/gpact/tree/main/functioncall/gpact


Price Oracle Blockchain
Contract Oracle
function getPrice()
Segment
result
Wallet Blockchain
Trade Wallet Contract
func executeTrade()
Root
Segment
Terms Blockchain
Logic Contract
function shipment()
Segment
Segment
Finance Blockchain
Balances Contract
function transfer()
Logistics Blockchain
Stock Contract
function delivery()



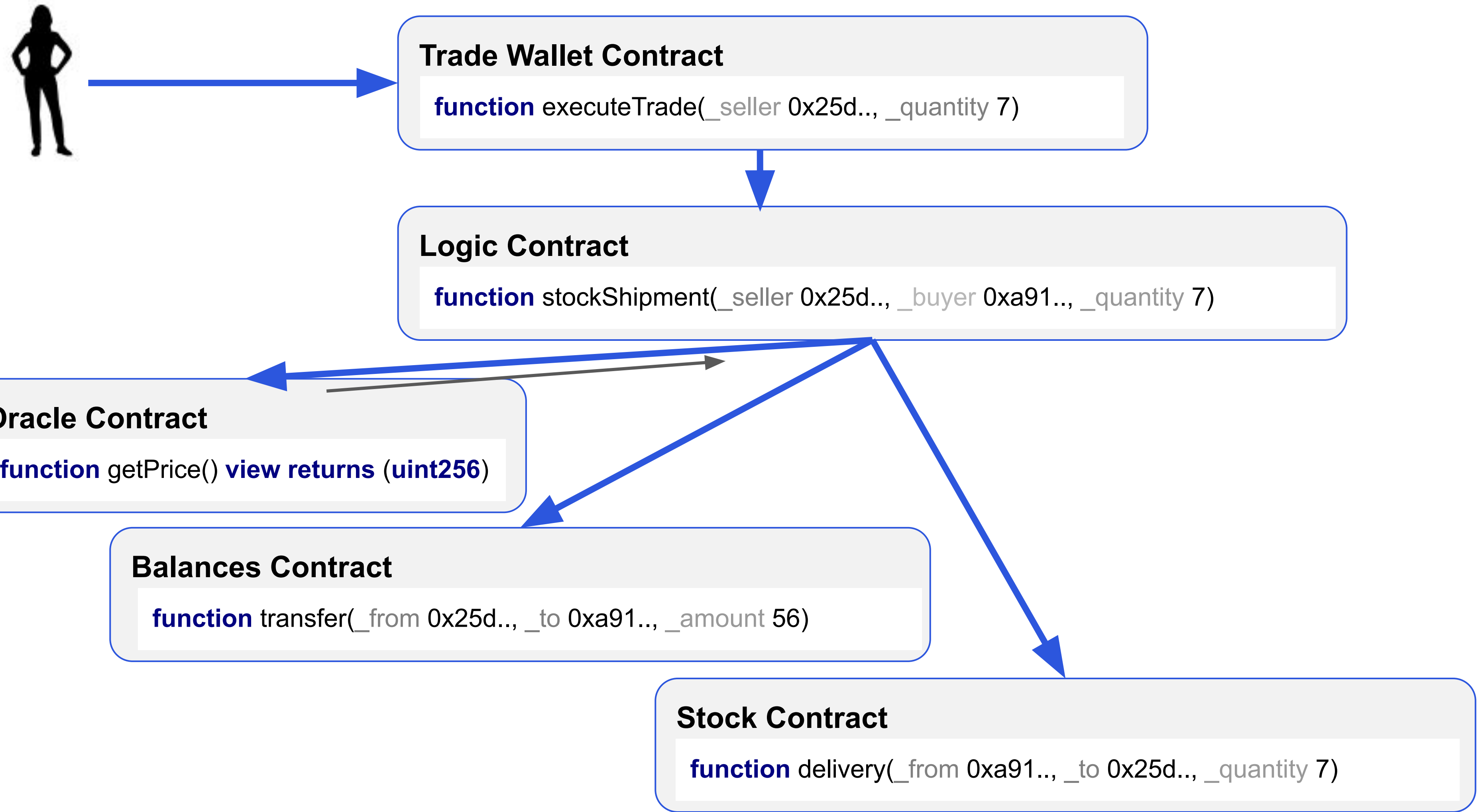
```
function executeTrade(address _seller, uint256 _quantity)
```

```
function stockShipment(address _seller, address _buyer, uint256 _quantity)
```

```
function getPrice() view returns (uint256)
```

```
function transfer(address _from, address _to, uint256 _amount)
```

```
function delivery(address _from, address _to, uint256 _quantity)
```



```
function executeTrade(_seller 0x25d.., _quantity 7)
```

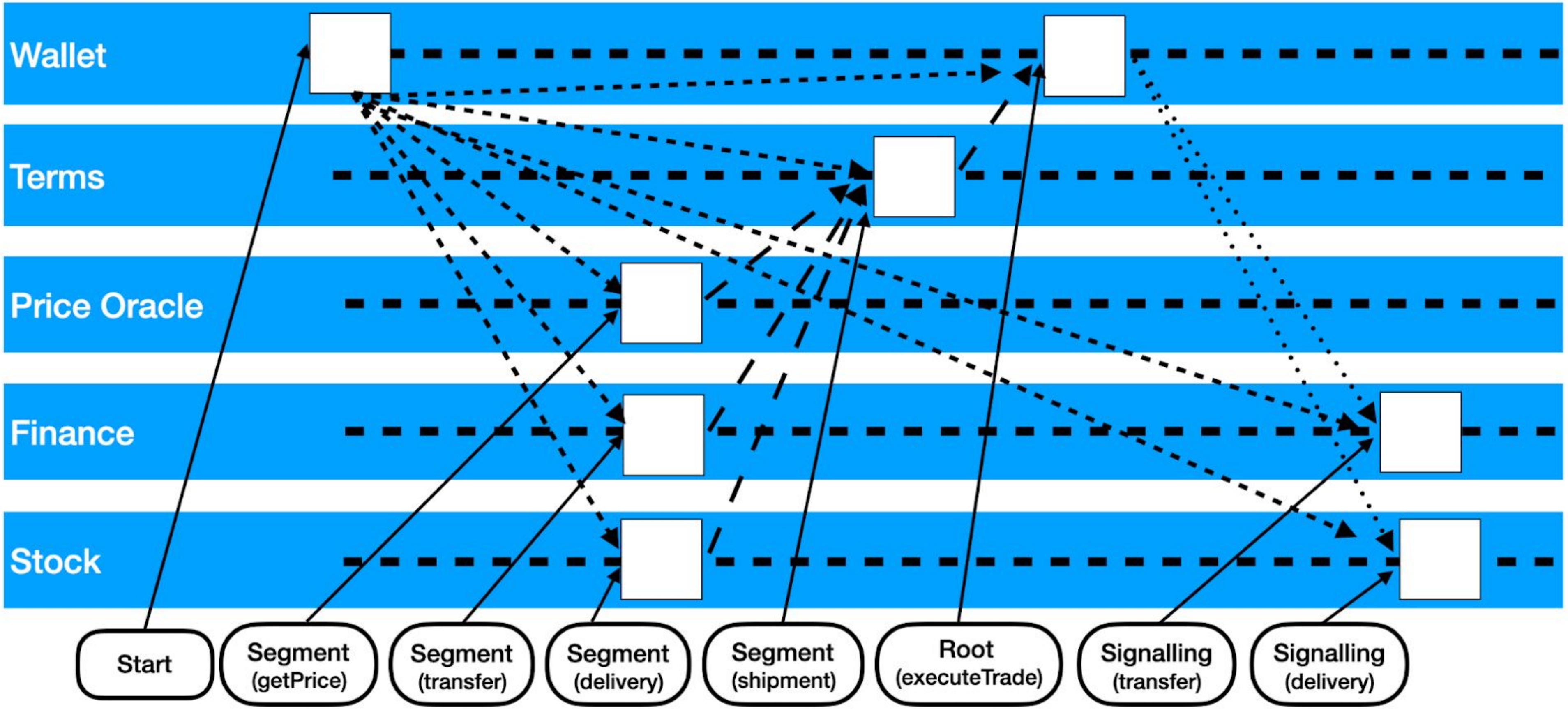
```
function stockShipment(_seller 0x25d.., _buyer 0xa91.., _quantity 7)
```

```
function getPrice() view returns (uint256)
```

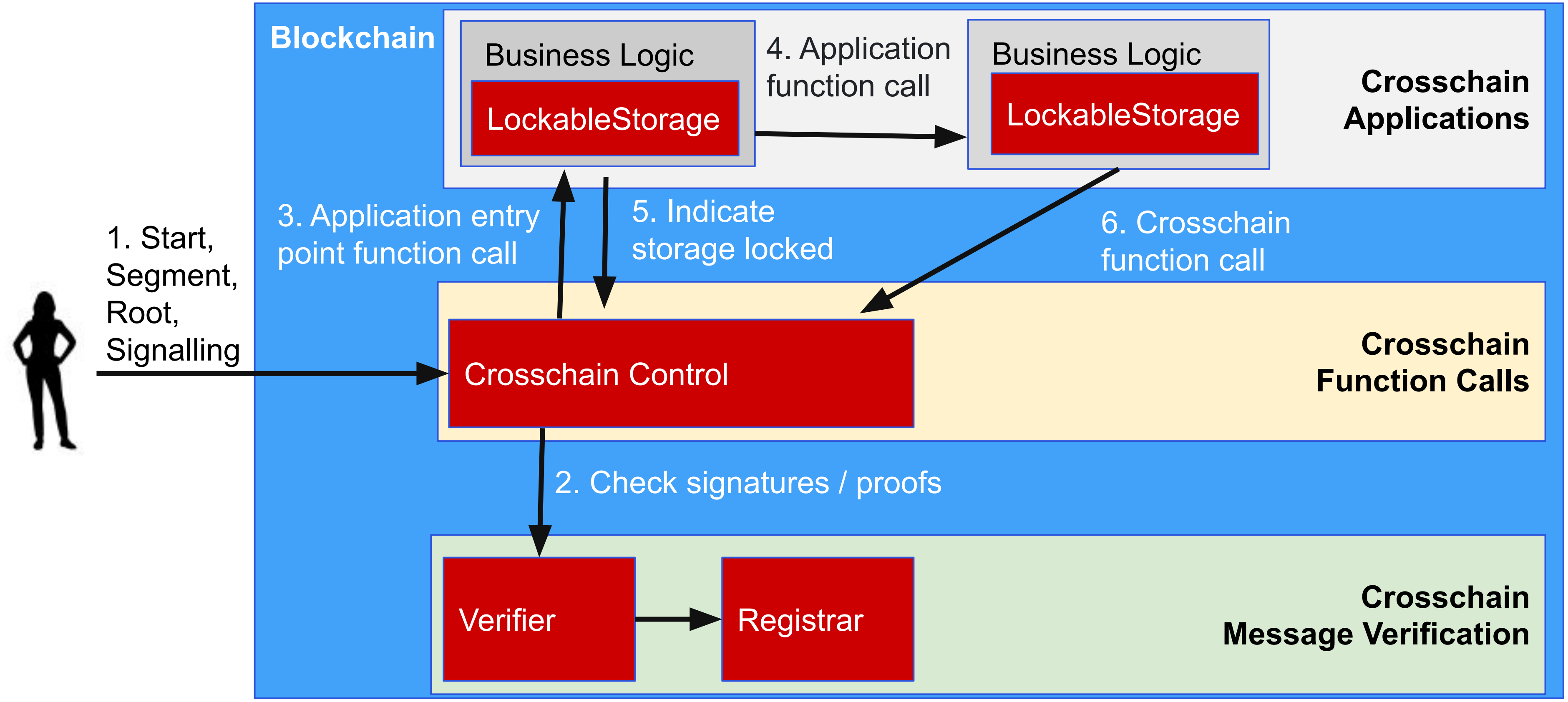
```
function transfer(_from 0x25d.., _to 0xa91.., _amount 56)
```

```
function delivery(_from 0xa91.., _to 0x25d.., _quantity 7)
```

Example Sequence



GPACT


Blockchain
Business Logic
LockableStorage
4. Application function call
Business Logic
LockableStorage
Crosschain Applications
3. Application entry point function call
5. Indicate storage locked
6. Crosschain function call
1. Start, Segment, Root, Signalling
Crosschain Control
Crosschain Function Calls
2. Check signatures / proofs
Verifier
Registrar
Crosschain Message Verification

Function Call Layer Design: GPACT

Application design considerations with GPACT:

- Unpredictable values can not affect parameter values that are committed to.
- Storage locations that affect parameter values must be accessible.
- Applications need to be designed with locking in mind.
    - Note that locking that allows for parallel execution is straightforward.
- The root blockchain must be accessible by owners of applications.
    - They may have to submit Signalling transactions.
- Gas cost around three times as much as non-atomic approaches.
- Front running (affects all blockchain and all crosschain protocols).

Function Call Layer Design: GPACT vs Non-Atomic

GPACT	**Non-Atomic**
Atomic updates across blockchains.	----
Crosschain transaction fails handled by protocol.	Crosschain transaction failures need to be handled within the application.
Higher gas costs (10x single blockchain)	Lower gas costs (3x single blockchain)
Contracts designed with locking in mind	Any contract
Composable programming model that application developers are accustomed to. From the perspective of the application developer, ● All transactions occur simultaneously. ● Functions return values immediately.	Non-standard programming model: ● Each segment of the overall crosschain transaction is separate. ● Functions can not return values.

Function Call Layer Design: GPACT vs Non-Atomic

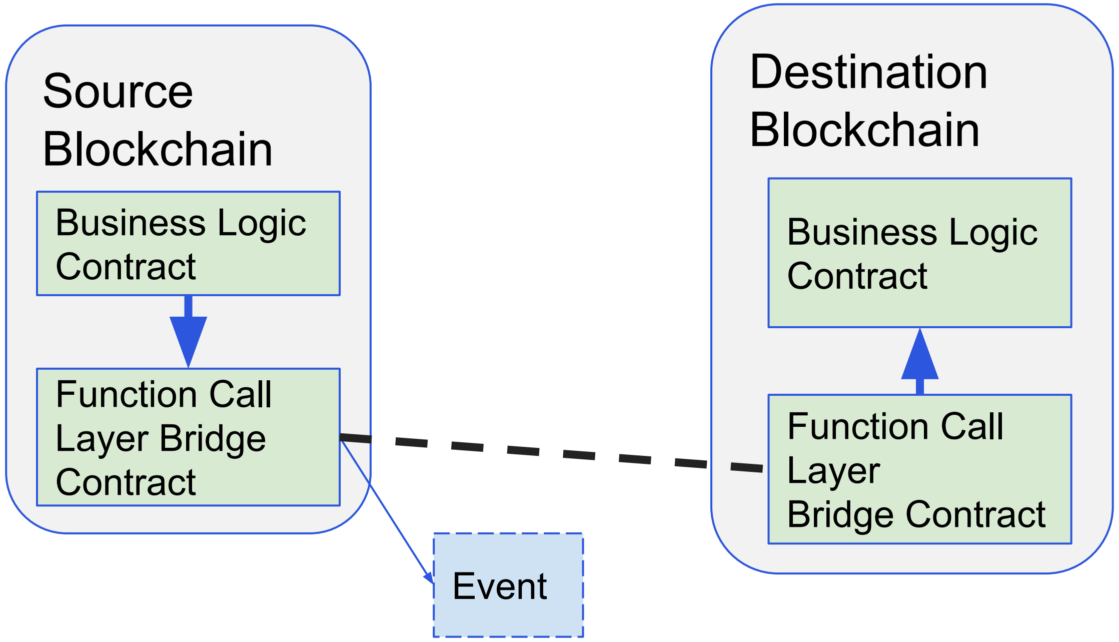
These numbers are implementation specific. Fewer checks / less functionality means closer to single blockchain performance.

GPACT	Non-Atomi
Atomic updates across blockchains.	----
Crosschain transaction fails handled by protocol.	Cross ain t har ed within the application.
Higher gas costs (10x single blockchain)	Lower gas costs (3x single blockchain)
Contracts designed with locking in mind	Any contract
Composable programming model that application developers are accustomed to. From the perspective of the application developer, ● All transactions occur simultaneously. ● Functions return values immediately.	Non-standard programming model: ● Each segment of the overall crosschain transaction is separate. ● Functions can not return values.

Function Call Layer Design: Trusted Event Emitter

Trusted Event Emitter:

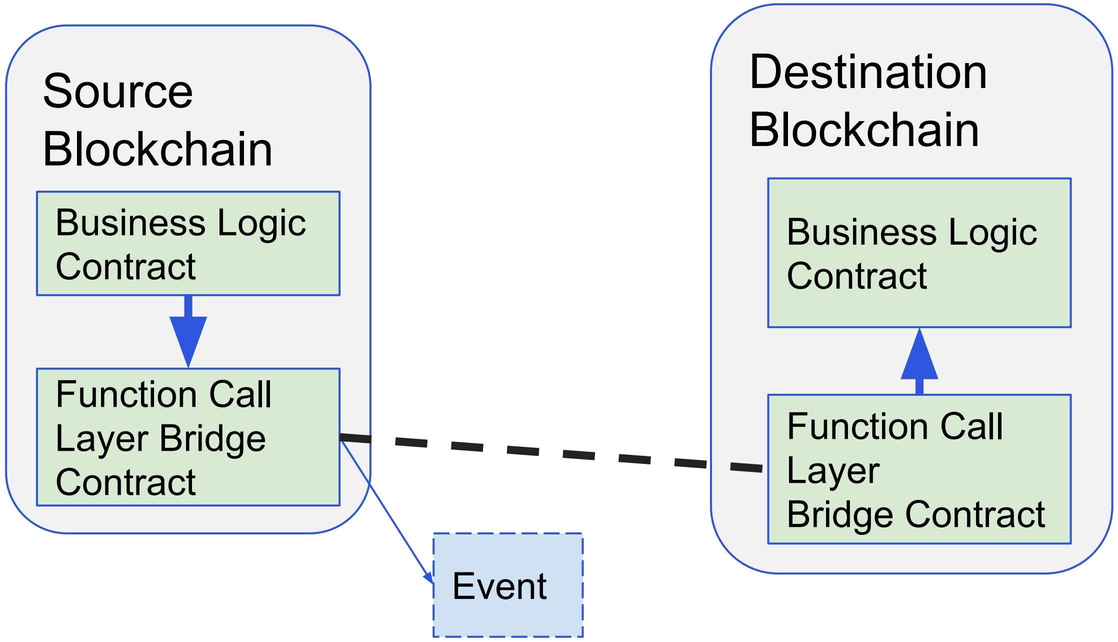
- Function Call Layer Bridge Contracts must know the addresses of other Function Call Layer Bridge Contracts on other blockchains.
- They need to check that events have only been emitted by trusted Function Call Layer Bridge Contracts.



Function Call Layer Design: Relay Protection

Replay Protection:

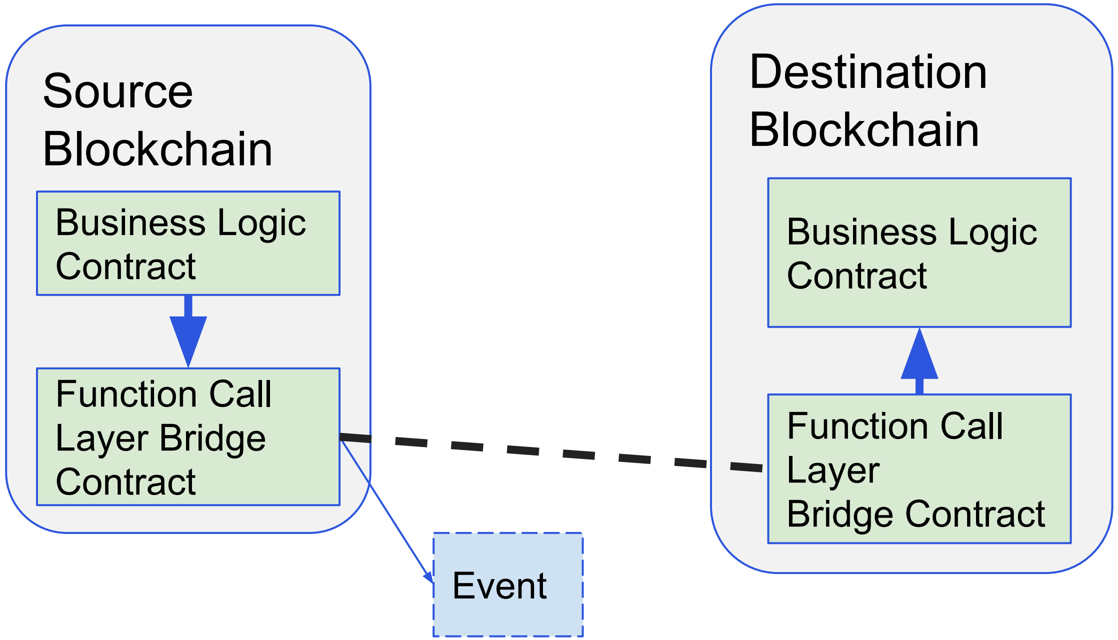
- The Function Call Layer Bridge Contract needs to create the event information needs such that it can not be successfully presented on the destination more than once.
- Unique identifiers are a common way to prevent replay attacks.
- However, storing all part unique identifiers will waste storage space on the blockchain.



Function Call Layer Design: Age Events

Timeout / Age Events:

- Include a timestamp in event information.
- Ignore “old” events. “old” could be deemed five or ten minutes, or maybe less.
- Timeouts could be used to reduce the cost of storing the unique identifiers by removing old unique identifiers.



Function Call Layer Design: Application Access Control

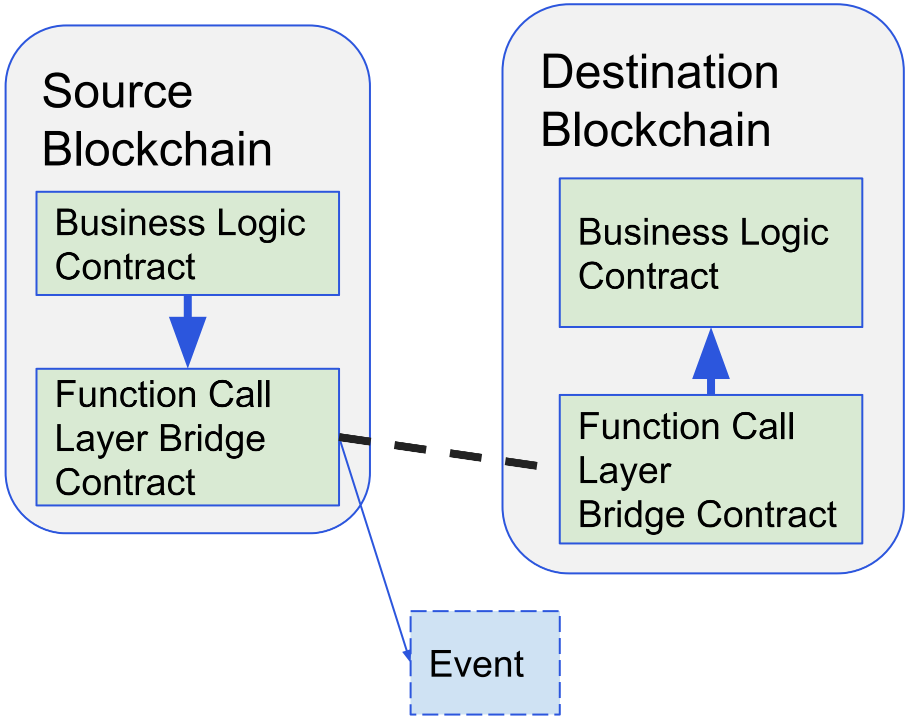
Application Access Control for single blockchain:

```
function someFunc(uint256 _param) external {
    require(msg.sender == authorised, “Not authorised!”);
    ...
}
```

Function Call Layer Design: Application Access Control

Application Access Control for crosschain:

```
function someFunc(
  uint256 _sourceBcId,
  address _sourceAddress,
  uint256 _param) external {

  require(msg.sender == fcBridge”);
  require(_sourceBcId == authBcId);
  require(_sourceAddress == authA);
  ...
}
```



Function Call Layer Design: Application Access Control

Function Call Layer Design must provide applications with:

- Source blockchain identifier.
- Source contract address.
- For multi-blockchain function calls, the root blockchain identifier.

Function Call Layer Design: Arbitrary Execution

Function Call Layer Bridges can be designed to:

- Allow any function in any contract to be called.
  - Arbitrary execution.

OR

- Restrict which contracts and functions can be called.

Arbitrary execution: Any function on any contract:

- Simpler and cheaper: less configuration and fewer checks.
- More flexible.

Restricted execution: Restrict which contracts and functions can be called:

- Fewer security challenges.

Function Call Layer Design: Pausable

- All functions on the bridge should be able to be stopped.
- The functions to pause and unpause the bridge need to be access control protected.

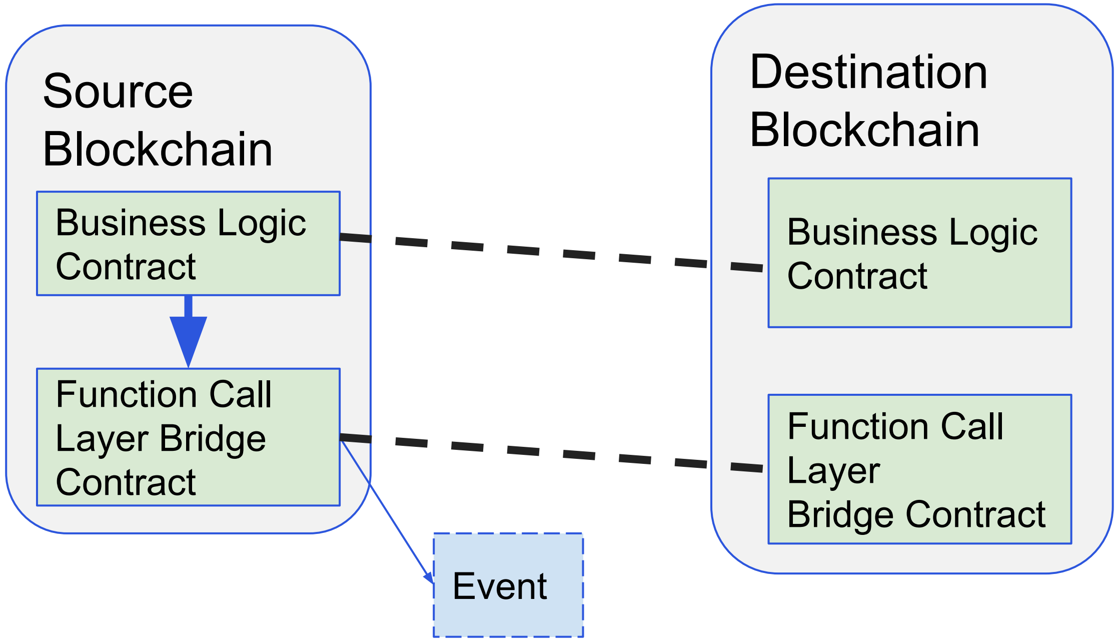
CONSENSYS

Design Choices: Crosschain Application Layer

Crosschain Application Design: Access Control

Access control:

- Use access control provided by function call layer to implement an **Allow List** of application contracts on other blockchains that are authorised to call the function.



Crosschain Application Layer Design: Pausable

- All functions on the application bridge should be able to be stopped.
- The functions to pause and unpause the bridge need to be access control protected.

- If the Function Call Bridge is used by multiple applications, you can not rely on the Function Call Layer “Pausable” feature to pause the application bridge.

For Applications using non-Atomic Function Calls

Admin control to allow failed transactions to be rolled back:

```
function adminTransfer(
   address _erc20Contract,
   address _recipient,
   uint256 _amount) onlyReallyTrustedAdmin external {
   ...
}
```

1. Who is this trusted admin?
2. What if the $ under control is huge?
3. Use a multi-sig wallet!

For GPACT: Locking Design

If a storage location in an application contract needs to be locked...

Lock whole contract	**Lock just the storage location**
Less gas	More gas
Less complex	More complex (though template contracts handle the complexity)
One crosschain transaction at a time	Many crosschain transactions at a time

ERC 20 Mass Conservation vs Minting and Burning

- ERC 20 Minting Burning:
  - The Function Call Bridge contract is likely to have the Minter role.
  - If there is a crosschain bug, an attacker could mint coins.
- ERC 20 Mass Conservation:
  - The Function Call Bridge contract could be given control over some number of tokens (“active tokens”).
  - Move ownership of tokens from the bridge to another account when is holds too many tokens.
  - Give tokens to the bridge when it is running out of token.
  - If there is a crosschain bug, the worst an attacker could steal is the “active tokens”.

Limiting value on a bridge

- How much should validators stake? US$50K? How many validators?
    - Stake x signing threshold = amount that can be slashed.
- The amount of value on the bridge should be less than the amount that is ~~staked~~ that can be slashed.

Questions:

- Which bridge / which layer?
    - Messaging layer?
    - Function call layer?
    - Application layer?
- How could a Messaging layer or Function Call layer implementation know about the value on the bridge?
- How does this work with ERC 721 NFT tokens?
- What if the bridge is a Trade - Finance application?
- How are transactions on the bridge temporarily rejected, held, or paused?

Batching / Aggregating Crosschain Transactions

Crosschain transactions cost money. Consider batching up transfers:

- Hold all transfers to a certain blockchain for all users for a period of time, and then the transfers as a single transaction.
- Allow multiple tokens to be transferred in the one crosschain transfer.

Trade-off is that error handling will be more complex.

CONSENSYS

Security Considerations

Separation of Data Channel and Control Channel

- Have separate addresses for different roles.
- Use role based access control contracts from Open Zeppelin:
    - https://docs.openzeppelin.com/contracts/4.x/access-control

```
function processData(
  uint256 _param) external {
}

function addNewBridgeContract(
  uint256 _sourceBcId,
  address _sourceContract) external {
}
```

Source and Destination Blockchain Security

Source & destination blockchain security:

- What type of security do the validators of the source / destination blockchain have?
- What would it take to compromise a validator of the source / destination blockchain?
- What happens if all trust in a source blockchain / destination blockchain is lost?

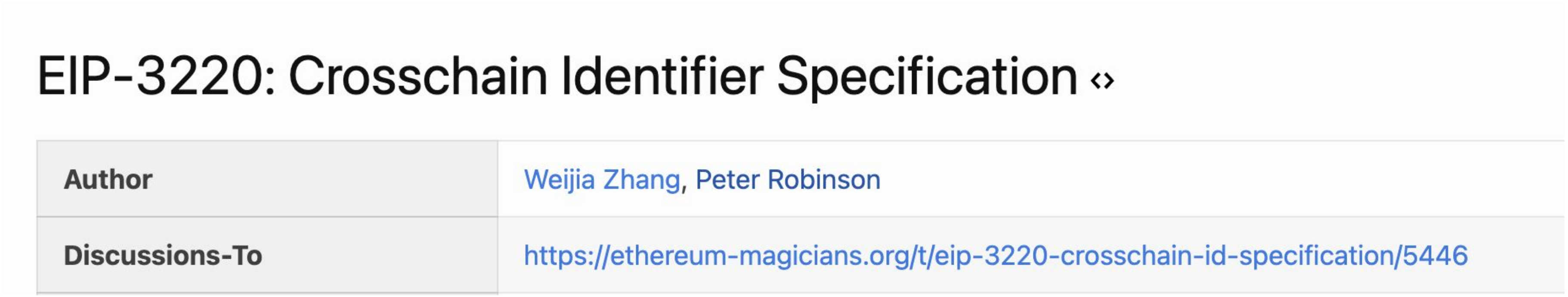
Attack Surface Expansion

- For single blockchain applications, the attack surface is the blockchain.
- For crosschain applications, the attack surface expands to the connected blockchains.
- The control an application has may be less when a hub blockchain is used.

Blockchain Boot Nodes and Blockchain Identifiers

- 256 bit blockchain identifier represent each blockchain.
- Need to be very careful to ensure the blockchain identifier matches the actual blockchain.
- When setting up a bridge, how do you know if you are interacting with the real blockchain?
- Where do you obtain blockchain boot node information from?

https://eips.ethereum.org/EIPS/eip-3220



EIP-3220: Crosschain Identifier Specification ‹›

Author	Weijia Zhang, Peter Robinson
Discussions-To	https://ethereum-magicians.org/t/eip-3220-crosschain-id-specification/5446

Upgrade

- You may want to upgrade your contact to fix defects.
- Upgradeable contract patterns:
  - Business logic contract with separate data holder contract.
    https://www.youtube.com/watch?v=VhzafmGGmzo&t=10s
  - Open Zeppelin upgradeable pattern:
    https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

- However, wIll there be a trust issue with an upgradeable contract?
  - Users may be concerned that the logic of the contract can be changed at any point.

Attestors and Relayers

- Who are the Attestors / Relayers?
- Why should you trust them?
- Do they use staking and slashing?
- Is staking and slashing enough given how much you plan to transfer?
- How are the Attestors / Relayers allocated to your bridge?
- For PoS / PoA sidechains, do the Attestors / Relayers match the network validators?

The Problem with Watchers, Fishermen, & Fisherwomen

- Some protocols have “Watchers” or “Fishermen” and “Fisherwomen”.
- These nodes watch for bad behaviour and are paid some slashing reward or other reward if they report bad behaviour.
- However, if 99.99% of the time there is no bad behaviour, then the Watchers are not being paid.
- External incentivization is required.

But…

- If you pay Watchers to watch the network, and they never report anything, how do you know they are doing their job?

EEA Crosschain Security Guidelines

ENTERPRISE
ETHEREUM
ALLIANCE

EEA CIW - Crosschain Security Guidelines Version 1.0

Crosschain Interoperability WG - Working Draft

24 September 2021

How to securely operate a crosschain bridge.

https://entethalliance.github.io/crosschain-interoperability/crosschainsecurityguidelines.html

CONSENSYS

Scalability

Scalability

How many Transactions Per Second (TPS) ?

Scalability

Answer: it depends

Scalability

Limits caused by the bridge:

- Relayers / Attestors signing every message (rather than Merkle Roots or block headers).
- Relayers / Attestors coordinating to sign (rather than signing individually).
- Number of transactions that need to be submitted on the source and destination blockchain.
- Amount of gas used by bridge.
- Latency of value moving across the bridge and limits on the value on the bridge at any one time.
- Use of a hub blockchain (what is its maximum tps?)

Limits caused by the source and destination blockchain:

- Performance of source and destination blockchain.
- Congestion.

Limits caused by application:

- What code is being executed?
- Lockable contract design.

CONSENSYS

Bridge Fees

Bridge Fees

Bridge fees could be charged at the:

- Crosschain Application Layer.
- Crosschain Function Call Layer.
- For use of a hub blockchain.

Fees could be charged on the Source Blockchain to the account identified by msg.sender:

- Charged some tokens for each transfer.
- Require Eth to be sent as part of call.
- Pay a subscription for account to be added to an Allow List.
- Attestors on sign if an off-chain payment is made.

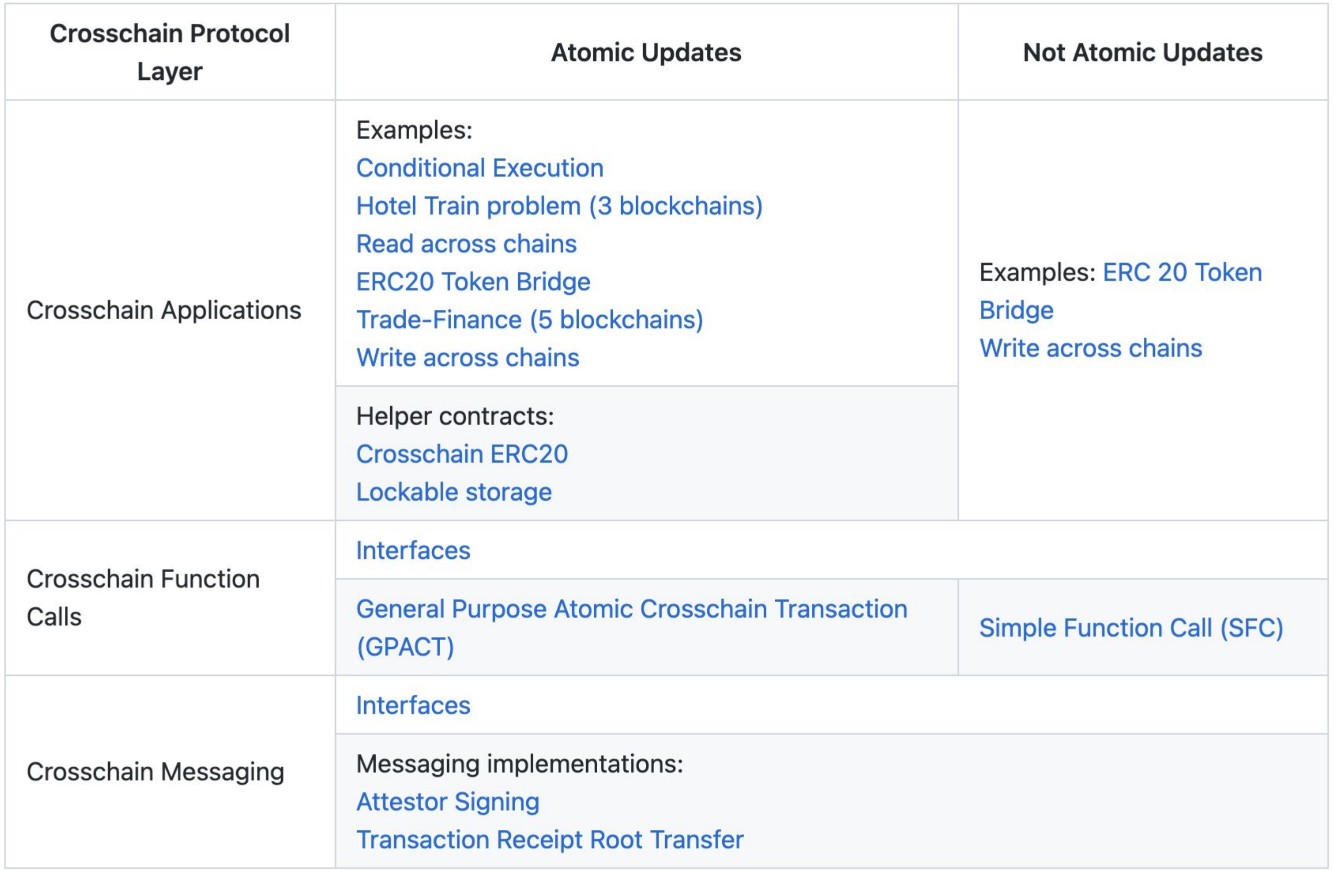
CONSENSYS

Standards

Standards

- Enterprise Ethereum Alliance Crosschain Interoperability Working Group is actively standardising the Crosschain Protocol Stack.
- The initial definition of the interfaces between layers, along with implementations is available here: https://github.com/ConsenSys/gpact

Standards



Crosschain Protocol Layer	Atomic Updates	Not Atomic Updates
Crosschain Applications	Examples: Conditional Execution Hotel Train problem (3 blockchains) Read across chains ERC20 Token Bridge Trade-Finance (5 blockchains) Write across chains	Examples: ERC 20 Token Bridge Write across chains
	Helper contracts: Crosschain ERC20 Lockable storage	
Crosschain Function Calls	Interfaces	
	General Purpose Atomic Crosschain Transaction (GPACT)	Simple Function Call (SFC)
Crosschain Messaging	Interfaces	
	Messaging implementations: Attestor Signing Transaction Receipt Root Transfer	

CONSENSYS

Philosophical Questions

Philosophical Questions

What is the bridge?

Function call layer, the messaging layer, application layer?

Is 1 wrapped XYZ coin on Ethereum MainNet worth 1 wrapped XYZ coin on a sidechain?

Is bridge contract upgradability good or bad?

When is money on a bridge?

If you were transferring a billion dollars, which technology would you use?

CONSENSYS

Revisiting the Design Questions

Design Choices: Questions to think through

Questions to think through:

- Who is being trusted?
- How many transactions are needed to facilitate a transfer?
- Who is submitting what transactions to which blockchain?
- Can a user who has no value on the destination blockchain use this?
- How are infrastructure components (relayers, attestors etc) compensated?
- Do infrastructure components need to handle requests from users?
- How many components need to be compromised / bribed for the system to fail?
- What attacks are possible?
- Would you use this technique to transfer a billion $?????

Future Talks

October 27: Ethereum State Expiry

- Raghavendra Ramesh, 12:30 pm Brisbane

November 3: Exploring Ethereum's Data Stores: A Cost and Performance Comparison

- Periklis Kostamis, 5pm Brisbane (8am Athens)

November 10: MEV Attacks and how you could mitigate them with Hyperledger Besu

- Antony Denyer, 5pm Brisbane (7am London)

November 27: Formal Verification of the QBFT Consensus Protocol

- Roberto Saltini, 12:30 pm Brisbane

December 1: Peter’s Solidity Recruitment Test

- Peter Robinson, 12:30 pm Brisbane

December 8: Networking & Recruitment, 12:30 pm Brisbane

January 12, 2022: More Advanced Solidity

- Peter Robinson, 12:30 pm Brisbane