Interoperability between blockchain

How to bridge chains

Slides from Dan Boneh

Many L1 blockchains

Bitcoin: Bitcoin scripting language (with Taproot)

Ethereum: EVM. Currently: high Tx fees (better with Rollups)

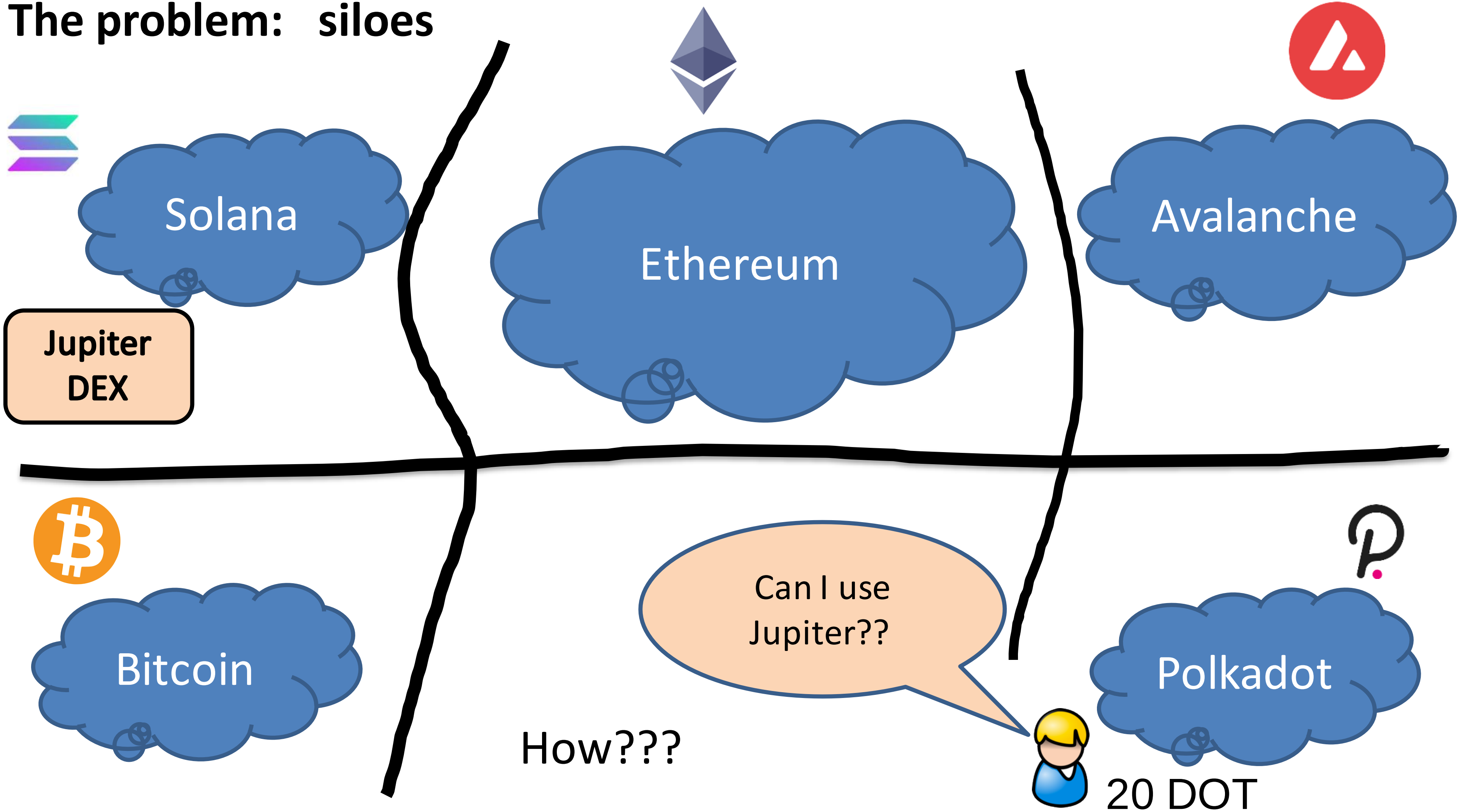
EVM compatible blockchains: **Avalanche, BSC**, …

- Higher Tx rate ⟹ lower Tx fees
- EVM compatibility ⟹ easy project migration and user support

Other fast non-EVM blockchains: **Solana, Cosmos**, …

- Higher Tx rate ⟹ lower Tx fees

The problem: siloes



Solana

Jupiter DEX

Ethereum

Avalanche

Bitcoin

Can I use Jupiter??

How???

Polkadot

20 DOT

Interoperability

Interoperability:

- User owns funds or assets (NFTs) on one blockchain system
  Goal: enable user to move assets to another chain

Composability:

- Enable a DAPP on one chain to call a DAPP on another

Not a problem if the entire world used Ethereum

- In reality: many blockchain systems that need to interoperate
- The solution: **bridges**

A first example: BTC in Ethereum

How to move BTC to Ethereum ?? Goal: enable BTC in DeFi.

$\Longrightarrow$ need new ERC20 on Ethereum pegged to BTC

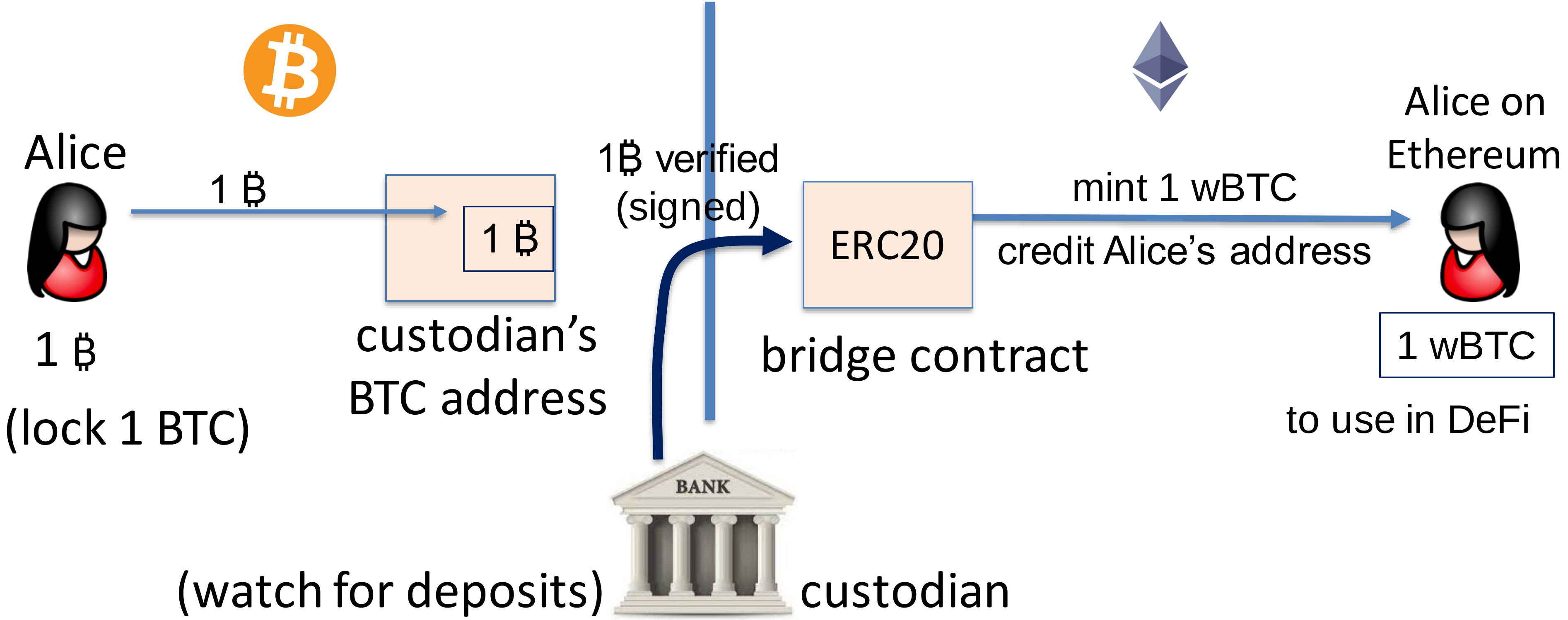
(e.g., use it for providing liquidity in DeFi projects)

The solution: **wrapped coins**

- Asset X on one chain appear as wrapped-X on another chain
- For BTC: several solutions (e.g., wBTC, tBTC, …)

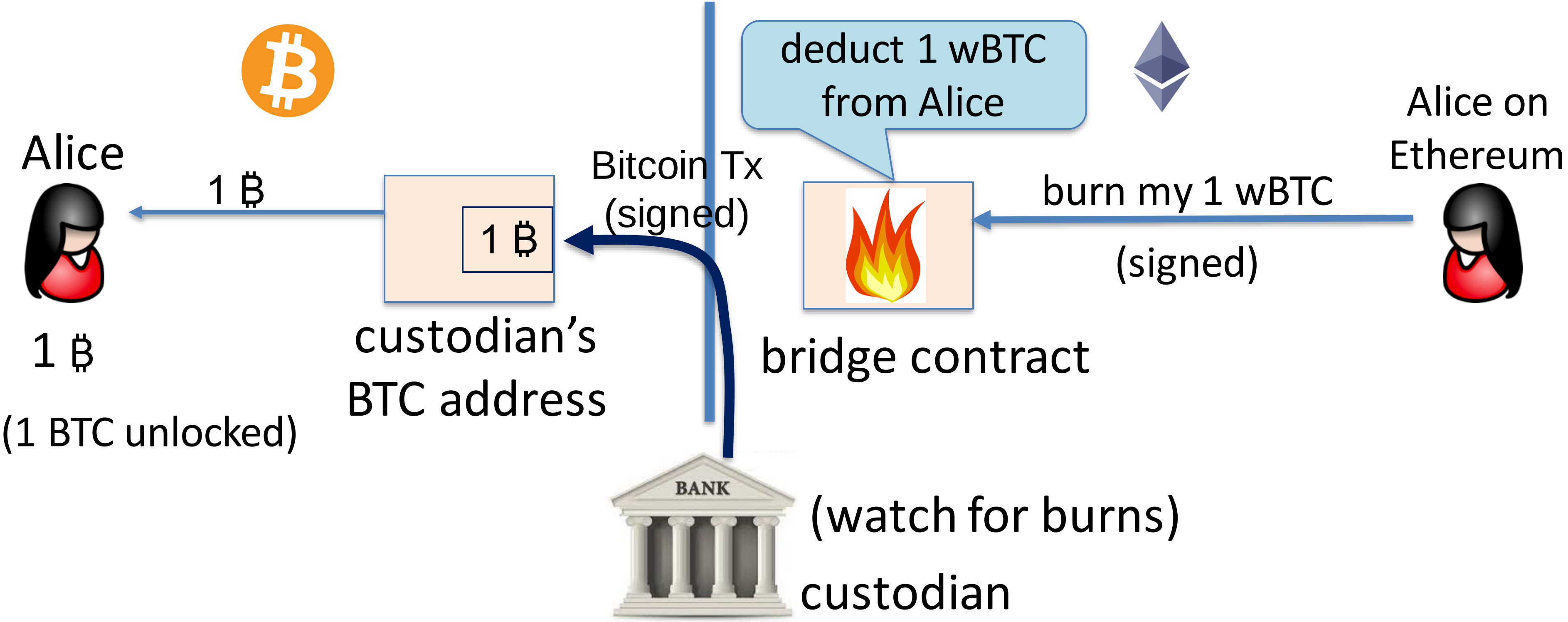
wBTC and tBTC: a lock-and-mint bridge

Let’s start with wBTC: **moving 1 BTC to Ethereum**



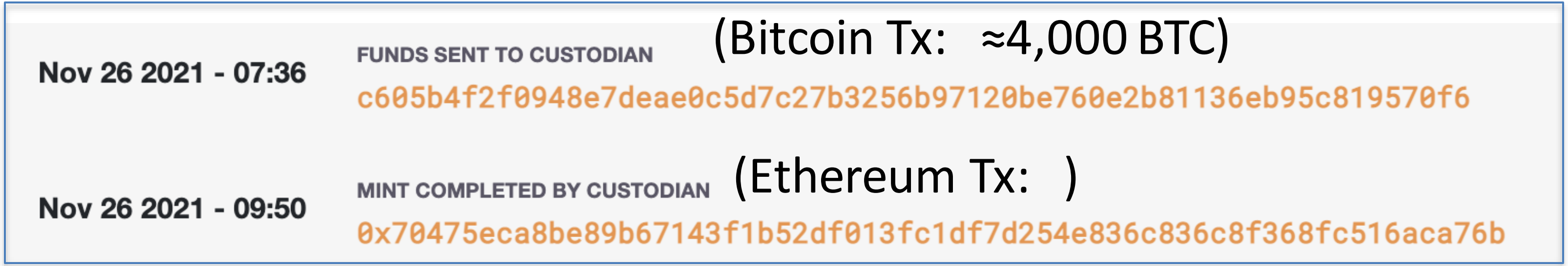
Alice wants her 1 BTC back

Moving 1 wBTC back to the Bitcoin network:



wBTC

Example BTC → Ethereum:



Nov 26 2021 - 07:36

FUNDS SENT TO CUSTODIAN (Bitcoin Tx: ≈4,000 BTC)

c605b4f2f0948e7deae0c5d7c27b3256b97120be760e2b81136eb95c819570f6

Nov 26 2021 - 09:50

MINT COMPLETED BY CUSTODIAN (Ethereum Tx:)

0x70475eca8be89b67143f1b52df013fc1df7d254e836c836c8f368fc516aca76b

Why two hours? … make sure no Bitcoin re-org

The problem: trusted custodian

Can we do better?

tBTC: no single point of trust

Alice requests to mint tBTC:

random three registered custodians are selected and
they generate P2PKH Bitcoin address for Alice

signing key is 3-out-of-3 secret shared among three
(all three must cooperate to sign a Tx)

Alice sends BTC to P2PKH address, and received tBTC.

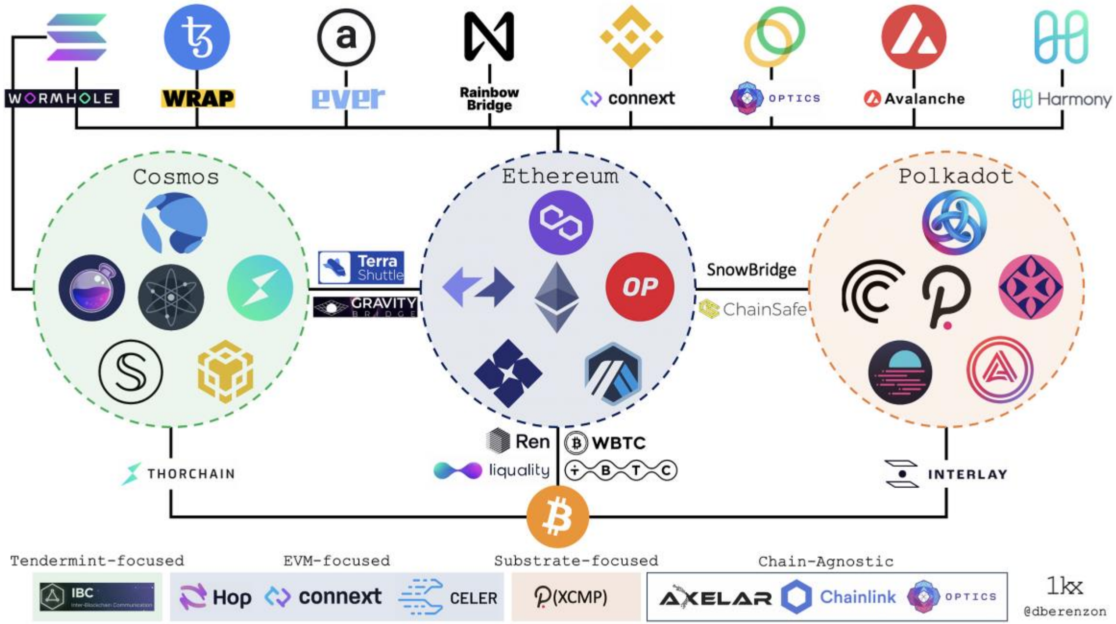
Custodians must lock 1.5x ETH stake for the BTC they manage

- If locked BTC is lost, Alice can claim staked ETH on Ethereum.

Bridging smart chains (with Dapp support)

A very active area:

- Many super interesting ideas
- Figure already outdated

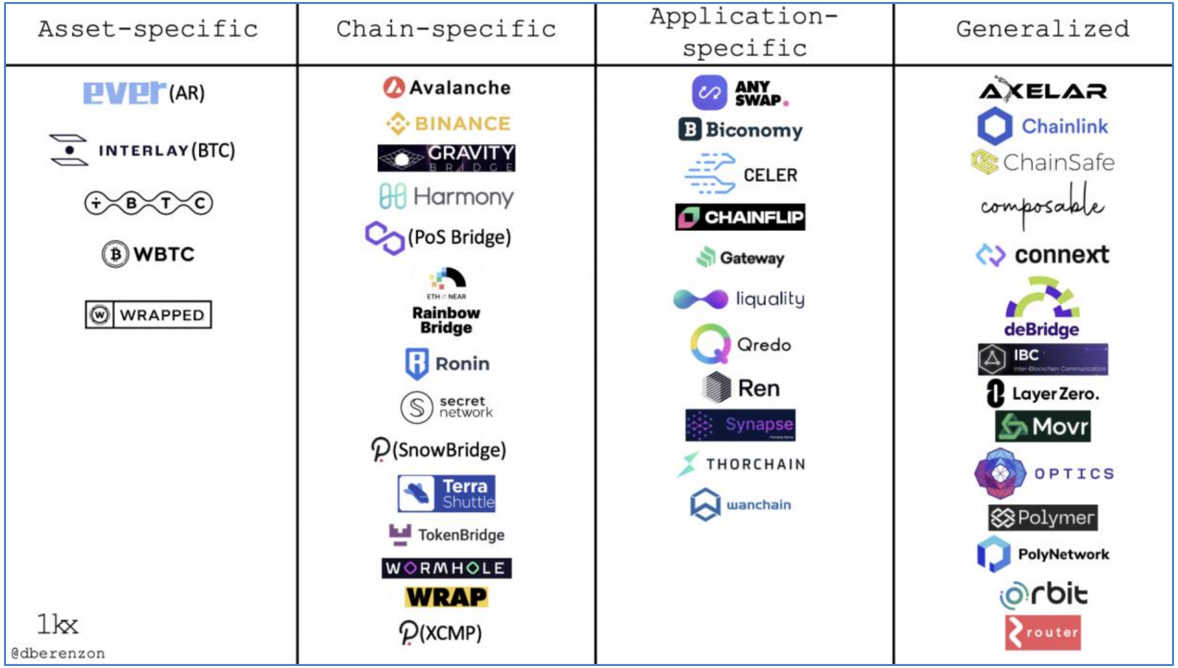


https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

Bridging smart chains (with Dapp support)

A very active area:

- Many super interesting ideas
- Figure already outdated



Asset-specific	Chain-specific	Application-specific	Generalized
ever (AR)	Avalanche	ANY SWAP.	AXELAR
INTERLAY (BTC)	BINANCE	Biconomy	Chainlink
tBTC	GRAVITY BRIDGE	CELER	ChainSafe
WBTC	Harmony	CHAINFLIP	composable
WRAPPED	(PoS Bridge)	Gateway	connext
	ETH NEAR Rainbow Bridge	liquality	deBridge
	Ronin	Qredo	IBC Inter-Blockchain Communication
	secret network	Ren	LayerZero.
	(SnowBridge)	Synapse	Movr
	Terra Shuttle	THORCHAIN	OPTICS
	TokenBridge	wanchain	Polymer
	WORMHOLE		PolyNetwork
	WRAP		orbit
	(XCMP)		router

1kx
@dberenzon

https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

Two types of bridges

Type 1: a lock-and-mint bridge

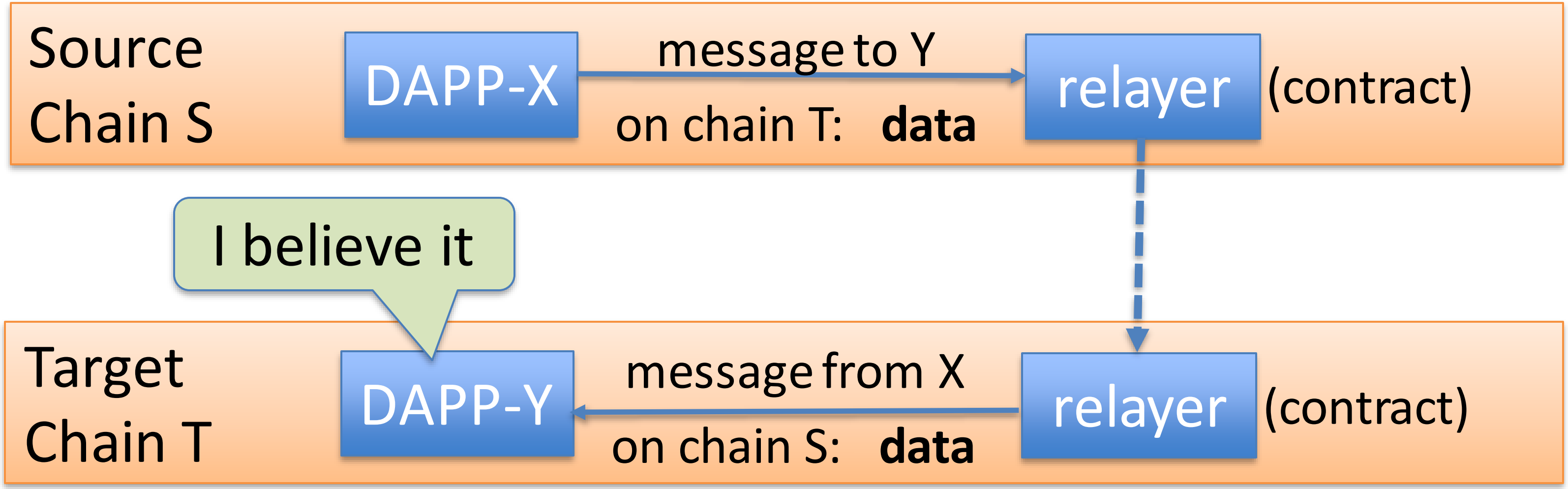
- SRC → DEST: user locks funds on SRC side,
  wrapped tokens are minted on the DEST side
- DEST → SRC: funds are burned on the DEST side,
  and released from lock on the SRC Side

Type 2: a liquidity pool bridge

- Liquidity providers provide liquidity on both sides
- SRC → DEST: user sends funds on SRC side,
  equivalent amount released from pool on DEST side

Bridging smart chains (with Dapp support)

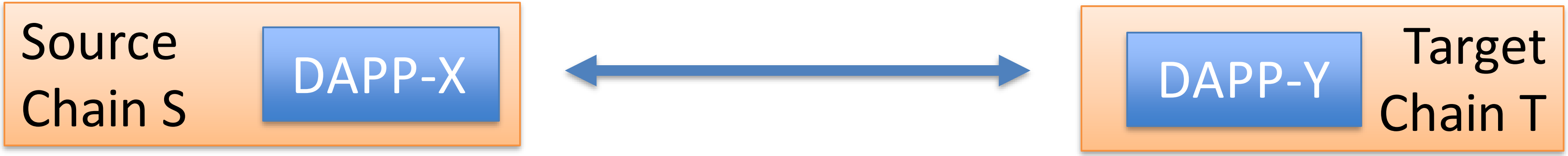
Step 1 (hard): a secure cross-chain messaging system



Step 2 (easier): build a bridge using messaging system

Bridging smart chains (with Dapp support)

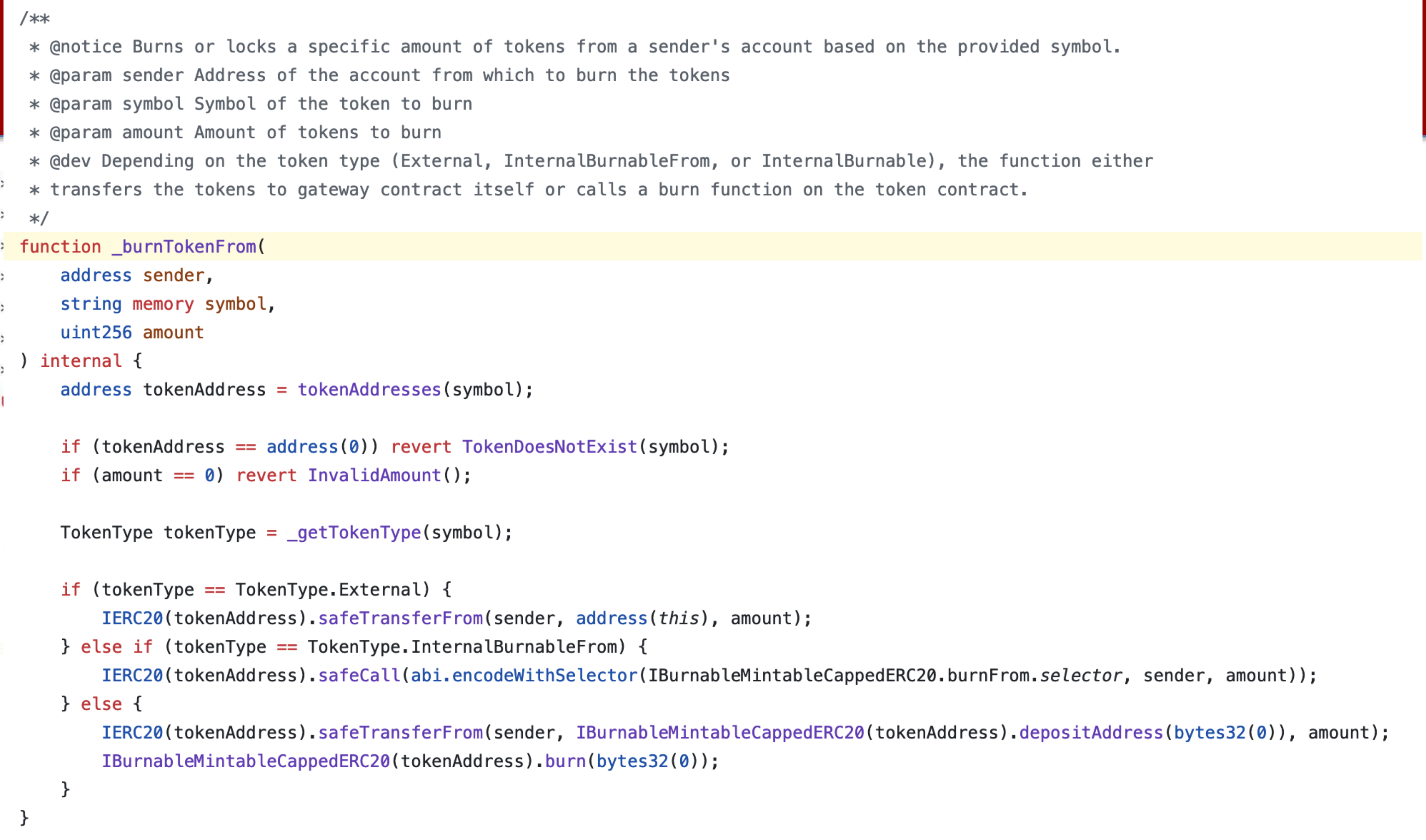
Step 1 (hard): a secure cross-chain messaging system



Step 2 (easier): build a bridge using messaging system

- DAPP-X → DAPP-Y: "I received 3 CELO, ok to mint 3 wCELO"
- DAPP-Y → DAPP-X: "I burned 3 wCELO, ok to release 3 CELO"

If messaging system is secure, no one can steal locked funds at S

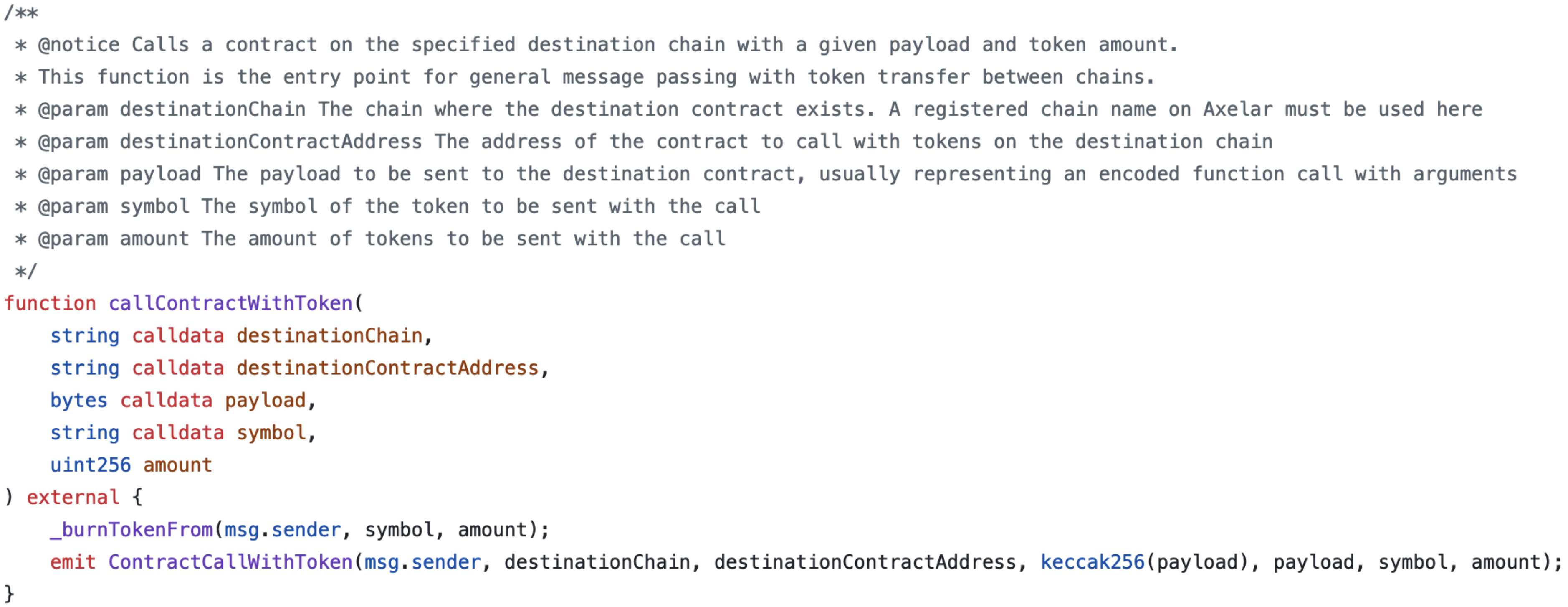


```
/**
 * @notice Burns or locks a specific amount of tokens from a sender's account based on the provided symbol.
 * @param sender Address of the account from which to burn the tokens
 * @param symbol Symbol of the token to burn
 * @param amount Amount of tokens to burn
 * @dev Depending on the token type (External, InternalBurnableFrom, or InternalBurnable), the function either
 * transfers the tokens to gateway contract itself or calls a burn function on the token contract.
 */
function _burnTokenFrom(
    address sender,
    string memory symbol,
    uint256 amount
) internal {
    address tokenAddress = tokenAddresses(symbol);

    if (tokenAddress == address(0)) revert TokenDoesNotExist(symbol);
    if (amount == 0) revert InvalidAmount();

    TokenType tokenType = _getTokenType(symbol);

    if (tokenType == TokenType.External) {
        IERC20(tokenAddress).safeTransferFrom(sender, address(this), amount);
    } else if (tokenType == TokenType.InternalBurnableFrom) {
        IERC20(tokenAddress).safeCall(abi.encodeWithSelector(IBurnableMintableCappedERC20.burnFrom.selector, sender, amount));
    } else {
        IERC20(tokenAddress).safeTransferFrom(sender, IBurnableMintableCappedERC20(tokenAddress).depositAddress(bytes32(0)), amount);
        IBurnableMintableCappedERC20(tokenAddress).burn(bytes32(0));
    }
}
```

Axelar Gateway (sending side)

- To send token:
  1. Approve the gateway contract as the spender of your ERC20 contract (e.g., call `USDC.approve(gatewayAddr, amount)`)
     - E.g., on:
       Ethereum: 0x4F4495243837681061C4743b74B3eEdf548D56A5
       Avalanche: 0x5029C0EFf6C34351a0CEc334542cDb22c7928f78
  2. Call the gateway `sendToken()`

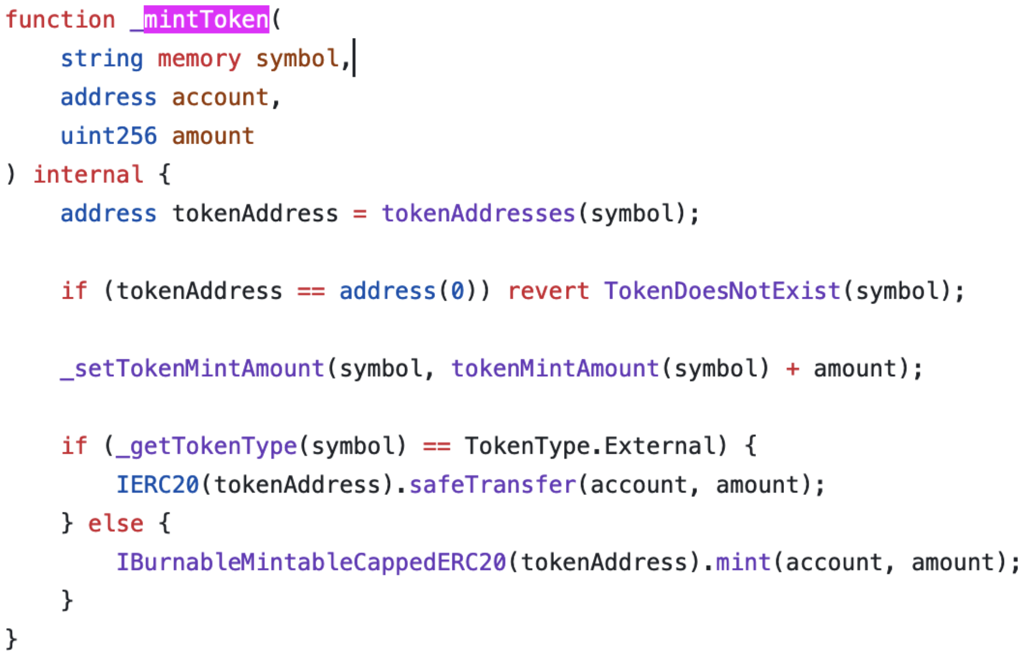
Axelar Gateway (sending side)



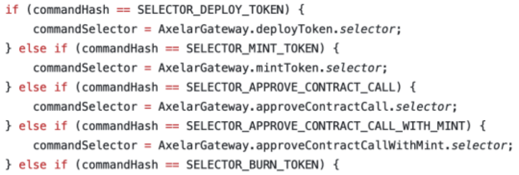
```
/**
 * @notice Calls a contract on the specified destination chain with a given payload and token amount.
 * This function is the entry point for general message passing with token transfer between chains.
 * @param destinationChain The chain where the destination contract exists. A registered chain name on Axelar must be used here
 * @param destinationContractAddress The address of the contract to call with tokens on the destination chain
 * @param payload The payload to be sent to the destination contract, usually representing an encoded function call with arguments
 * @param symbol The symbol of the token to be sent with the call
 * @param amount The amount of tokens to be sent with the call
 */
function callContractWithToken(
    string calldata destinationChain,
    string calldata destinationContractAddress,
    bytes calldata payload,
    string calldata symbol,
    uint256 amount
) external {
    _burnTokenFrom(msg.sender, symbol, amount);
    emit ContractCallWithToken(msg.sender, destinationChain, destinationContractAddress, keccak256(payload), payload, symbol, amount);
}
```

generic cross-chain call

Axelar Gateway (receiving side)



```
function _mintToken(
    string memory symbol,
    address account,
    uint256 amount
) internal {
    address tokenAddress = tokenAddresses(symbol);

    if (tokenAddress == address(0)) revert TokenDoesNotExist(symbol);

    _setTokenMintAmount(symbol, tokenMintAmount(symbol) + amount);

    if (_getTokenType(symbol) == TokenType.External) {
        IERC20(tokenAddress).safeTransfer(account, amount);
    } else {
        IBurnableMintableCappedERC20(tokenAddress).mint(account, amount);
    }
}
```

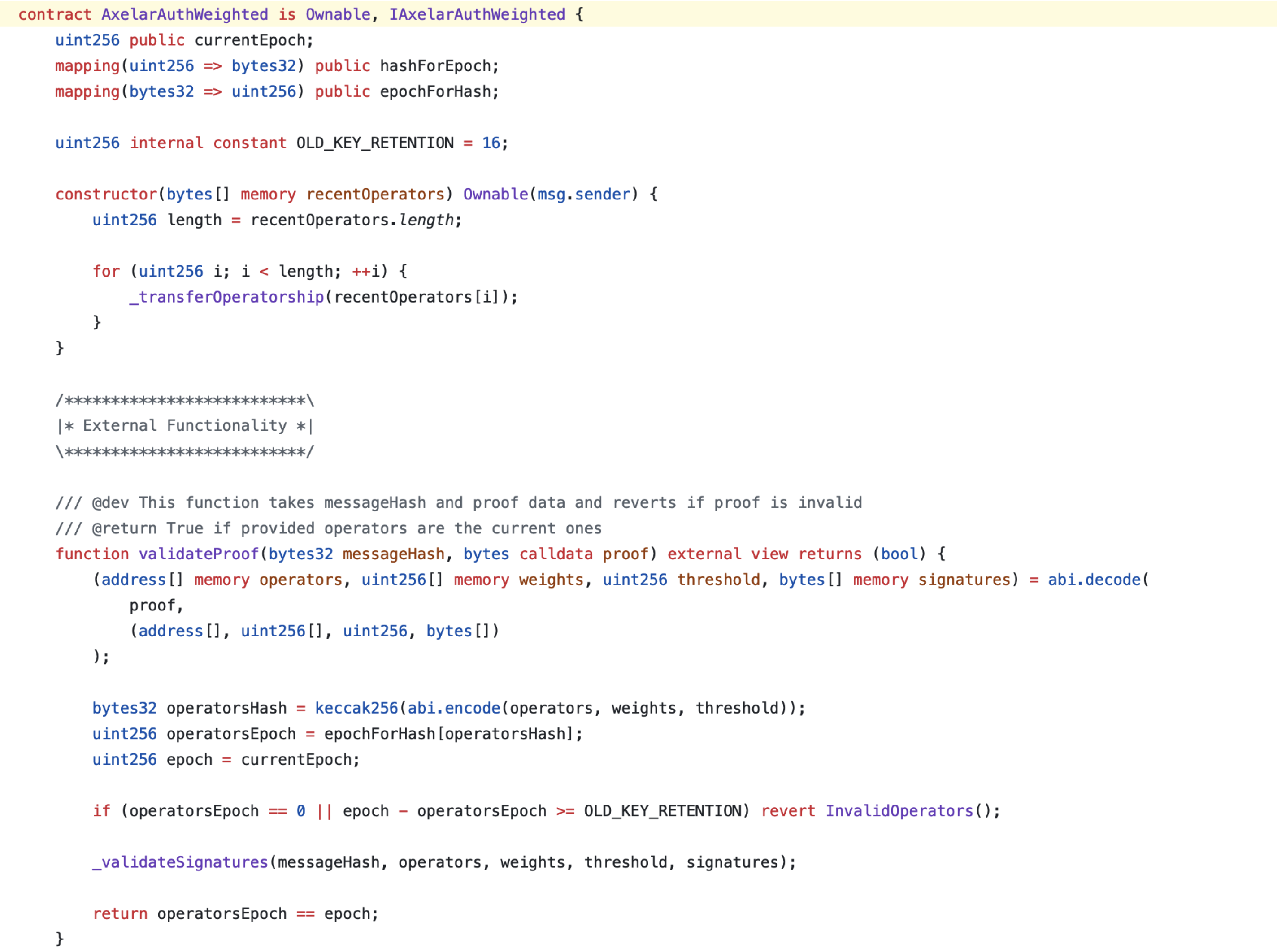


```
if (commandHash == SELECTOR_DEPLOY_TOKEN) {
    commandSelector = AxelarGateway.deployToken.selector;
} else if (commandHash == SELECTOR_MINT_TOKEN) {
    commandSelector = AxelarGateway.mintToken.selector;
} else if (commandHash == SELECTOR_APPROVE_CONTRACT_CALL) {
    commandSelector = AxelarGateway.approveContractCall.selector;
} else if (commandHash == SELECTOR_APPROVE_CONTRACT_CALL_WITH_MINT) {
    commandSelector = AxelarGateway.approveContractCallWithMint.selector;
} else if (commandHash == SELECTOR_BURN_TOKEN) {
```

full

Axelar Gateway (receiving side)

- Who can call execute()?
  - Anyone: this is a public function!
- What prevents Mallory from just sending herself tokens?
  - On-chain, verify proof that message on the other chain is real (e.g., that token was actually burned)

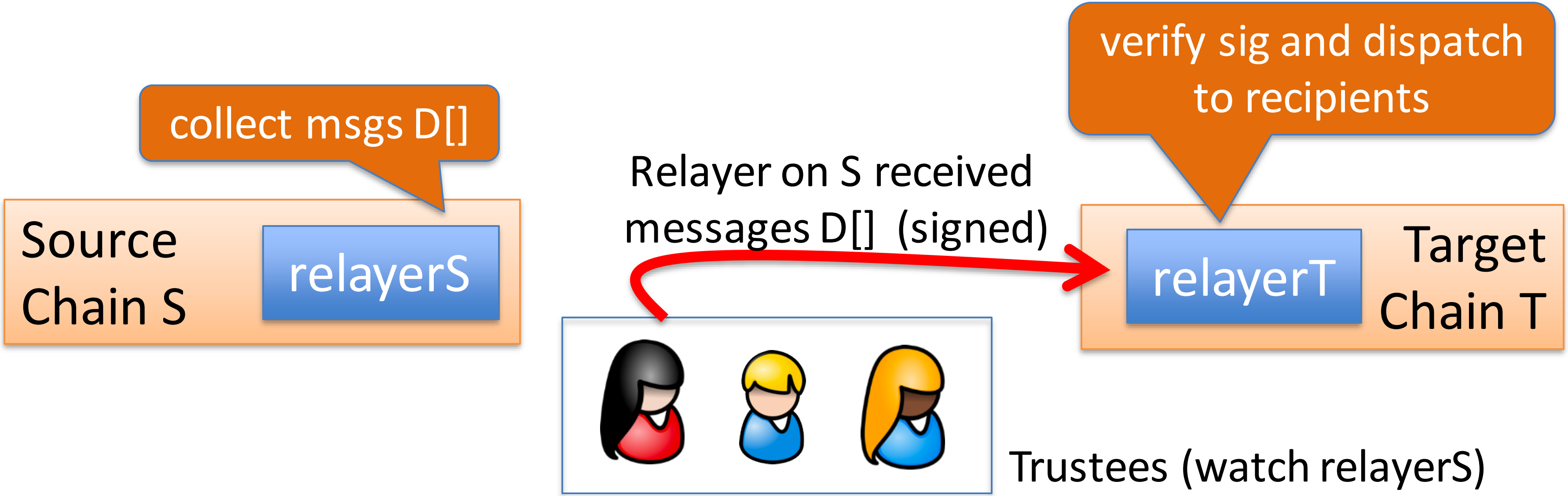


```
contract AxelarAuthWeighted is Ownable, IAxelarAuthWeighted {
    uint256 public currentEpoch;
    mapping(uint256 => bytes32) public hashForEpoch;
    mapping(bytes32 => uint256) public epochForHash;

    uint256 internal constant OLD_KEY_RETENTION = 16;

    constructor(bytes[] memory recentOperators) Ownable(msg.sender) {
        uint256 length = recentOperators.length;

        for (uint256 i; i < length; ++i) {
            _transferOperatorship(recentOperators[i]);
        }
    }

    /**************************\
    |* External Functionality *|
    \**************************/

    /// @dev This function takes messageHash and proof data and reverts if proof is invalid
    /// @return True if provided operators are the current ones
    function validateProof(bytes32 messageHash, bytes calldata proof) external view returns (bool) {
        (address[] memory operators, uint256[] memory weights, uint256 threshold, bytes[] memory signatures) = abi.decode(
            proof,
            (address[], uint256[], uint256, bytes[])
        );

        bytes32 operatorsHash = keccak256(abi.encode(operators, weights, threshold));
        uint256 operatorsEpoch = epochForHash[operatorsHash];
        uint256 epoch = currentEpoch;

        if (operatorsEpoch == 0 || epoch - operatorsEpoch >= OLD_KEY_RETENTION) revert InvalidOperators();

        _validateSignatures(messageHash, operators, weights, threshold, signatures);

        return operatorsEpoch == epoch;
    }
```

full

Primarily two types of messaging systems

(1) **Externally verified**: external parties verify message on chain S



collect msgs D[]

Source Chain S

relayerS

Relayer on S received messages D[] (signed)

verify sig and dispatch to recipients

relayerT

Target Chain T

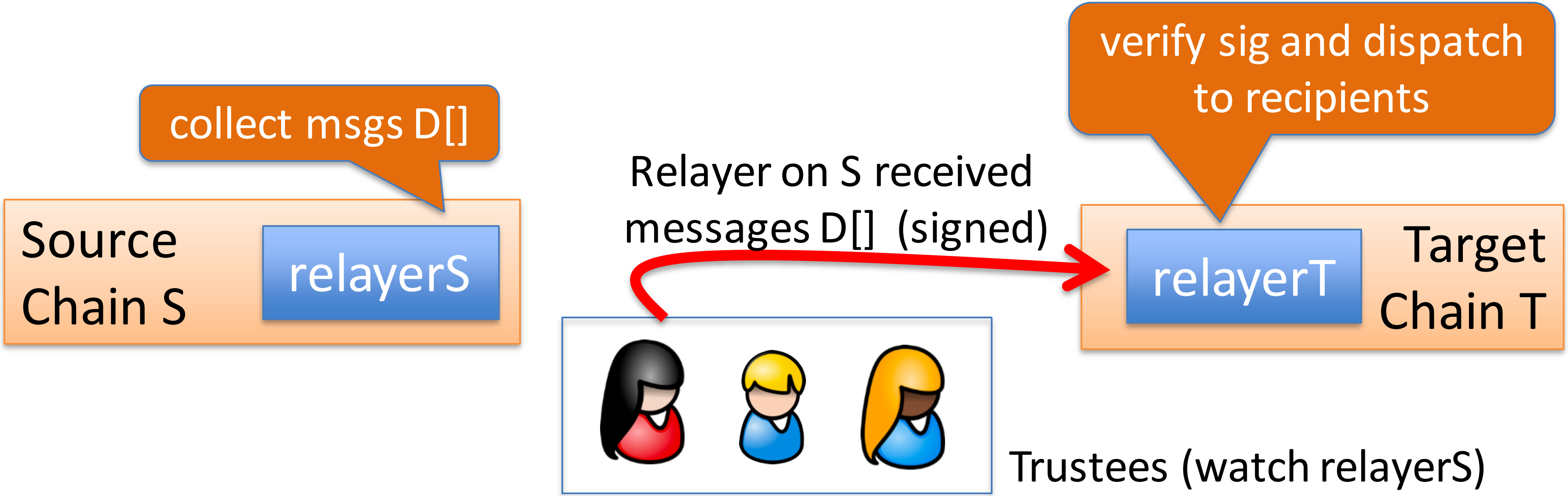
Trustees (watch relayerS)

RelayerT dispatches only if all trustees signed

⟹ **if** DAPP-Y trusts trustees, it knows DAPP-X sent message

Primarily two types of messaging systems

(1) **Externally verified**: external parties verify message on chain S



collect msgs D[]

Source Chain S

relayerS

Relayer on S received messages D[] (signed)

verify sig and dispatch to recipients

relayerT

Target Chain T

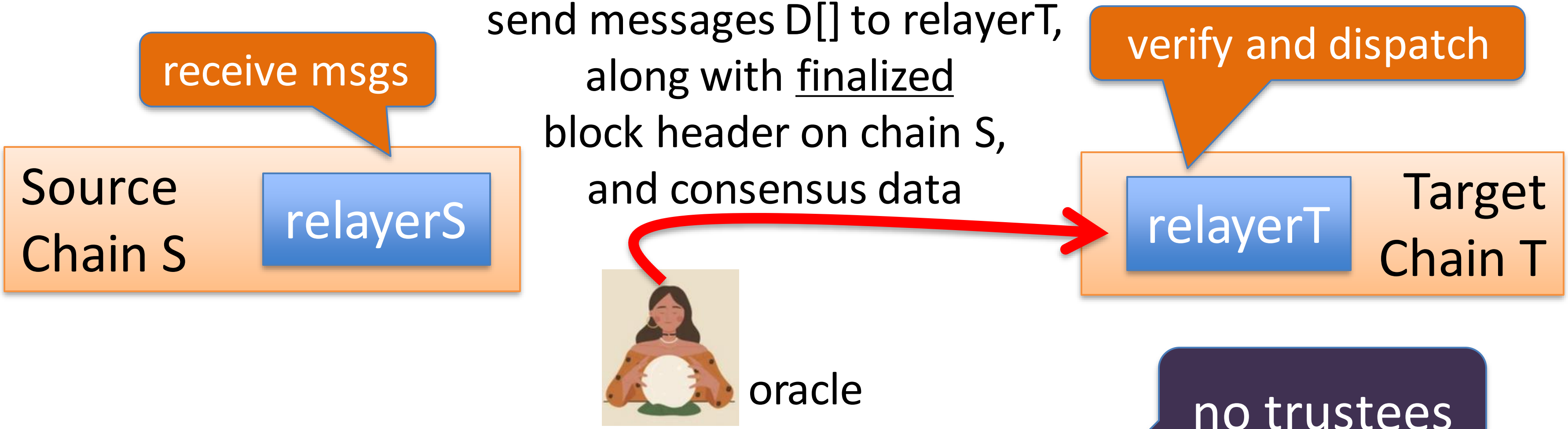
Trustees (watch relayerS)

What if trustees sign and post a fake message to relayerT?

- anyone can send trustee's signature to relayerS $\Longrightarrow$ trustee slashed on S

Primarily two types of messaging systems

(2) **On-chain verified**: chain T verifies block header of chain S

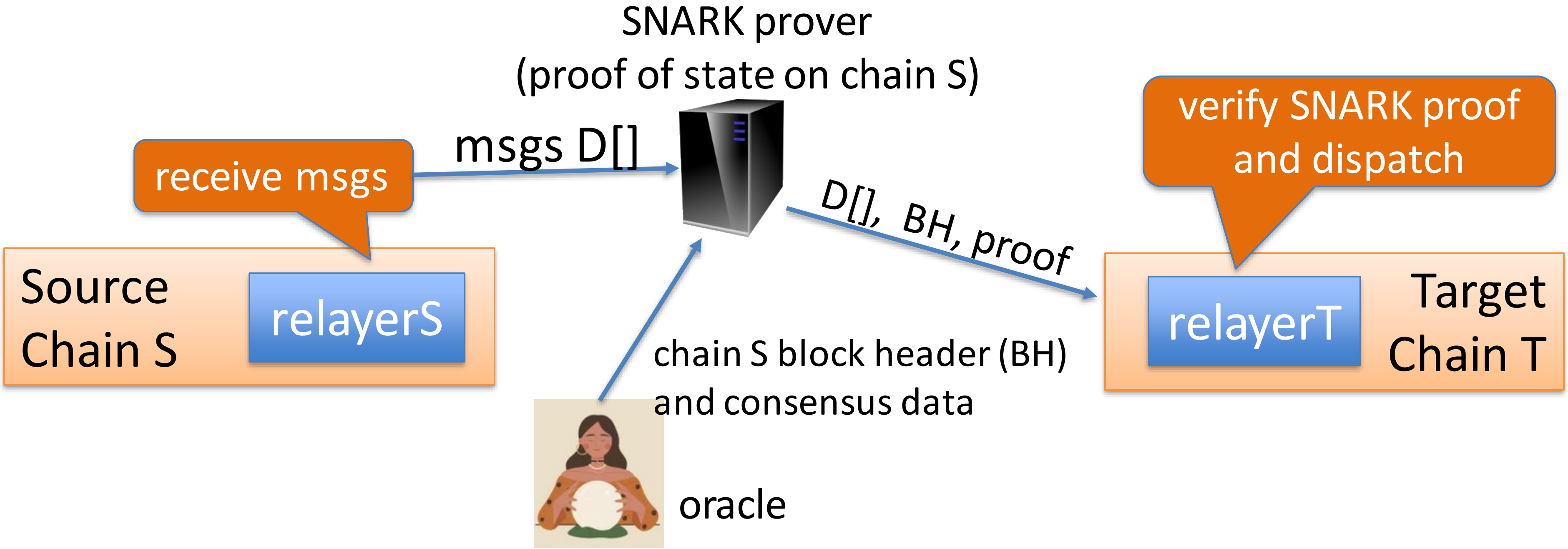

receive msgs
Source Chain S
relayerS
send messages D[] to relayerT, along with finalized block header on chain S, and consensus data
verify and dispatch
relayerT
Target Chain T
oracle

relayerT runs a light-client for chain S to verify that relayerS received messages D[]

no trustees

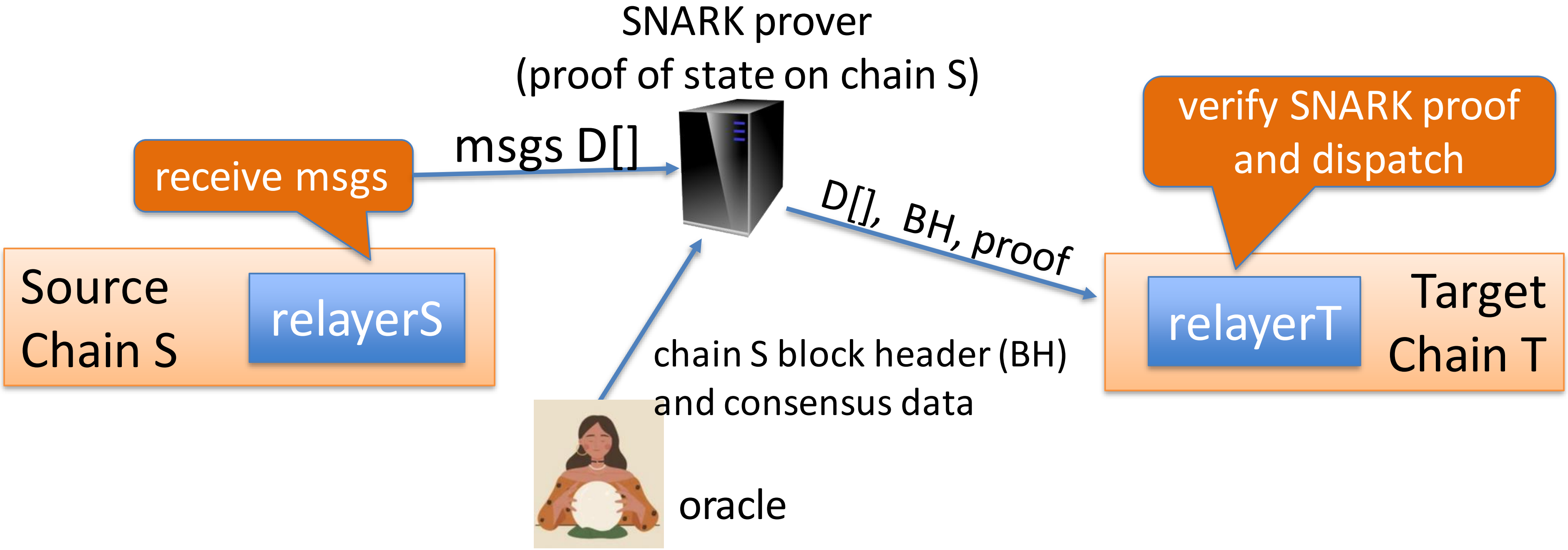
assumes security of light client

Primarily two types of messaging systems


SNARK prover
(proof of state on chain S)
verify SNARK proof and dispatch
msgs D[]
receive msgs
D[], BH, proof
Source Chain S
relayerS
relayerT
Target Chain T
chain S block header (BH) and consensus data
oracle

Problem: high gas costs on chain T to verify state of source chain S.
Solution: zkBridge: use SNARK to reduce work for relayerT

Primarily two types of messaging systems



SNARK prover

(proof of state on chain S)

receive msgs

msgs D[]

verify SNARK proof and dispatch

D[], BH, proof

Source Chain S

relayerS

relayerT

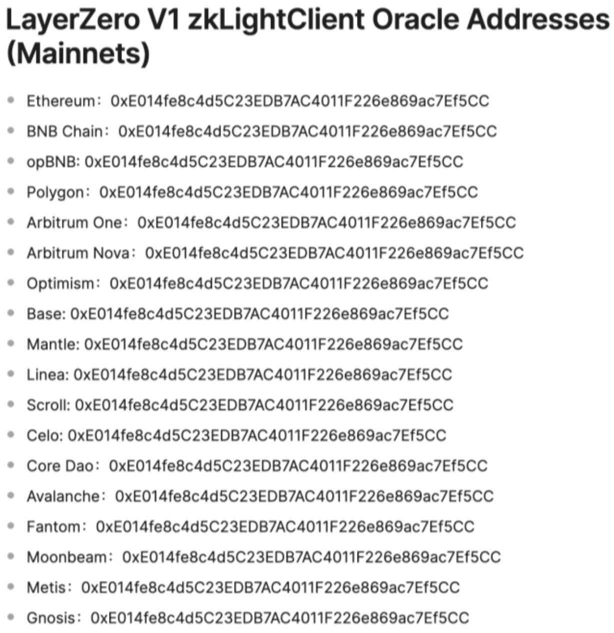
Target Chain T

chain S block header (BH) and consensus data

oracle

zkBridge

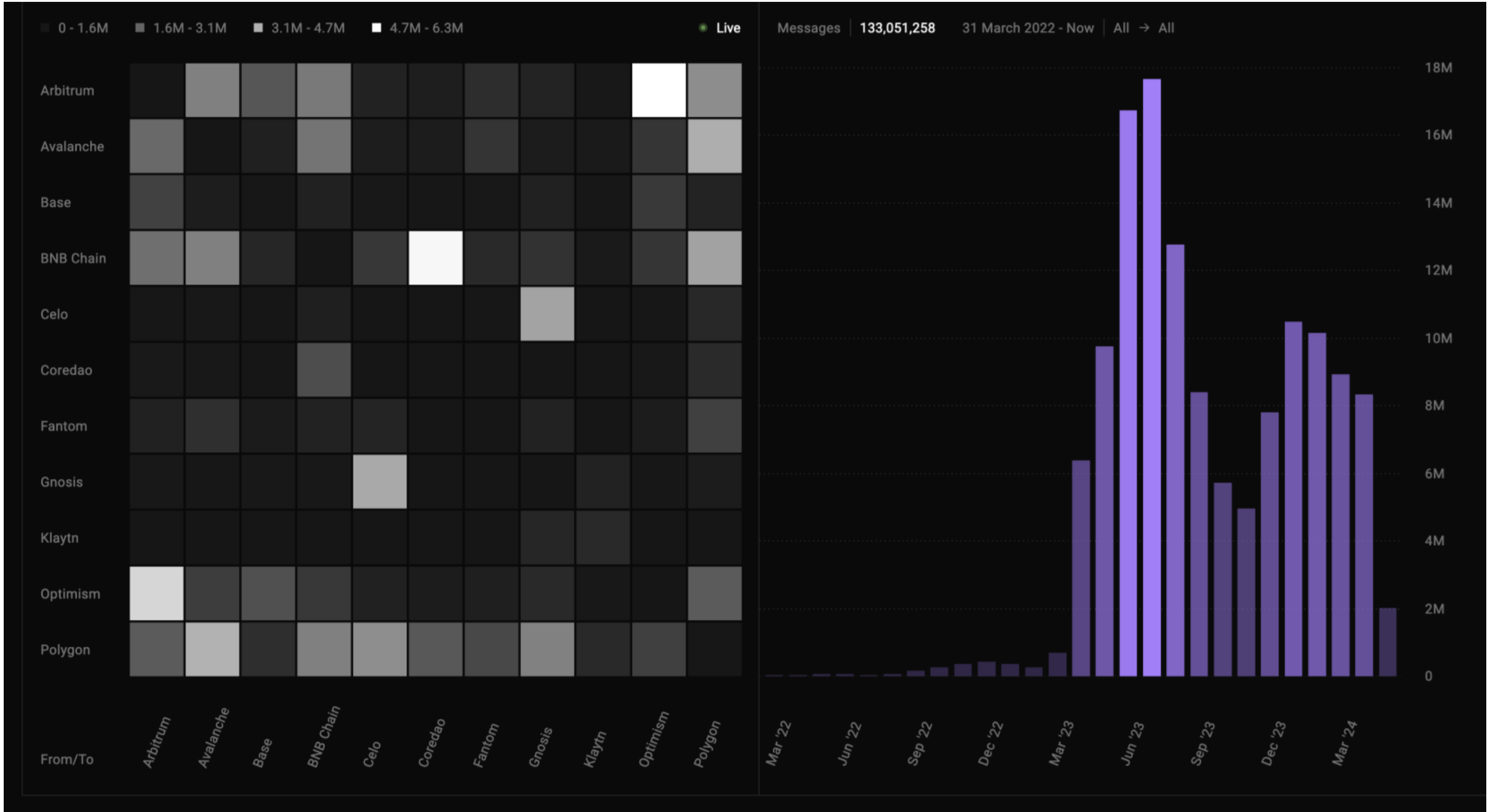
- Used with the LayerZero bridge



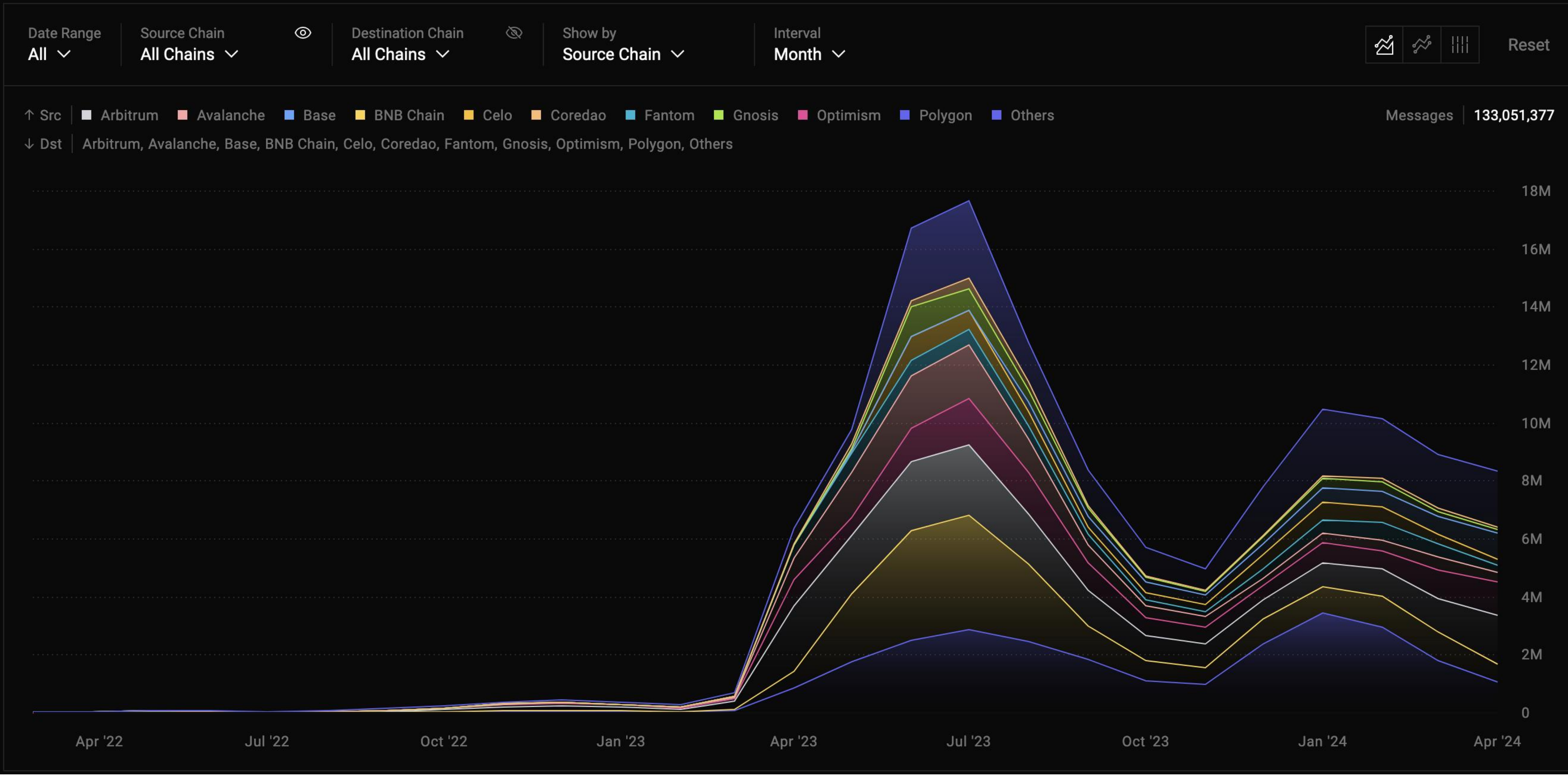
LayerZero V1 zkLightClient Oracle Addresses (Mainnets)

- Ethereum：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- BNB Chain：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- opBNB: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Polygon：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Arbitrum One：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Arbitrum Nova：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Optimism：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Base: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Mantle: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Linea: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Scroll: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Celo: 0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Core Dao：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Avalanche：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Fantom：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Moonbeam：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Metis：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC
- Gnosis：0xE014fe8c4d5C23EDB7AC4011F226e869ac7Ef5CC

Activity (Layer Zero)



Messages 133,051,258 31 March 2022 - Now All → All



Date Range All

Source Chain All Chains

Destination Chain All Chains

Show by Source Chain

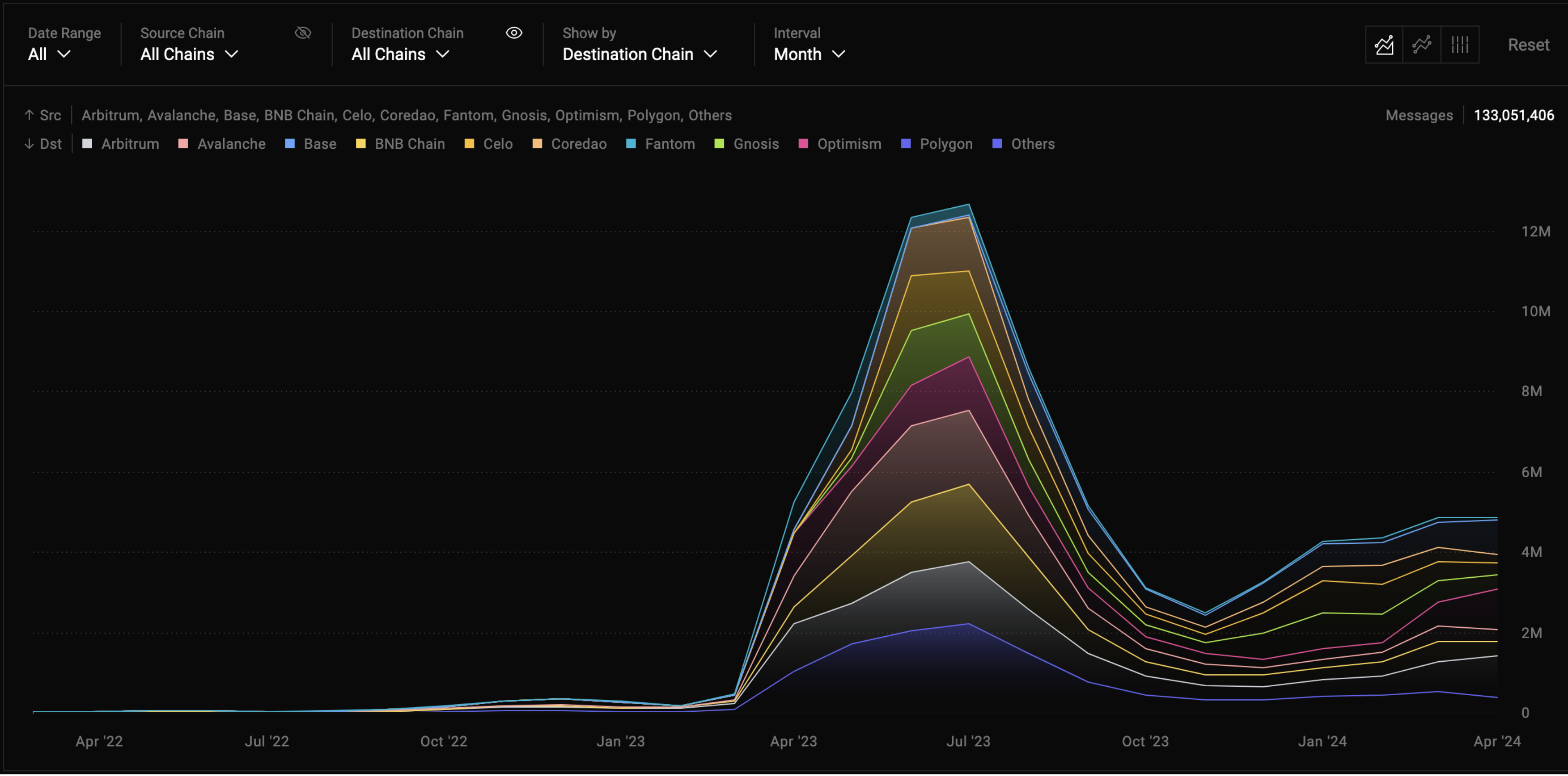
Interval Month

Reset

↑ Src | ■ Arbitrum ■ Avalanche ■ Base ■ BNB Chain ■ Celo ■ Coredao ■ Fantom ■ Gnosis ■ Optimism ■ Polygon ■ Others

↓ Dst | Arbitrum, Avalanche, Base, BNB Chain, Celo, Coredao, Fantom, Gnosis, Optimism, Polygon, Others

Messages | 133,051,377



Date Range All

Source Chain All Chains

Destination Chain All Chains

Show by Destination Chain

Interval Month

Reset

↑ Src Arbitrum, Avalanche, Base, BNB Chain, Celo, Coredao, Fantom, Gnosis, Optimism, Polygon, Others

Messages 133,051,406

Bridging: the future vision

User can hold assets on any chain

- Assets move cheaply and quickly from chain to chain
- A project's liquidity is available on all chains
- Users and projects choose the chain that is best suited for their application and asset type

We are not there yet … except on Avalanche & its subnets