Use Case : Ethereum USD Stable Coin on Linea

Mohamed Alaoui

Overview

A leading builder company seeks to expand the reach of its USD-collateralized stablecoin. Currently issued on Ethereum, the company aim to make it available on Linea as well with the following requirements :

- Leverage robust and secure technologies and protocols
- Enable native issuance of USDB on Linea in a subsequent stage

Hypothesis and naming convention :

The builder company issue 1:1 USD-collateralized coin named USDB on ethereum and and aim to make it available in Linea as USDB.e before natively issue the USDB token on Linea

Agenda

I. *Phasing strategy and rationale behind it*

I. *Bridge Token from Ethereum (L1) to Linea (L2)*

I. *Native issuance strategy*

I. *Liquidity management*

Agenda Checkpoint I

Phasing Strategy

Phasing strategy rationale

Bridged Vs Native Stable coin

Bridged

Pros

- Solving the “cold start” problem for generating activity on new layers
- Helps developers bootstrap liquidity and allow users to experiment new use cases earlier
- Bridged form can be quickly put in place other chain and bridges
- Less issuer cost and effort to be present in multiple layer due to the setup and deployment

Cons

- Bridging risk and related custody risk
- Liquidity fragmentation risk - friction due to non fungibility between different bridged token
- Expensive redemption as it needs potentiel multiple cross chain movement

Native

Pros

- Backed by USD and always redeemable 1:1 by the official issuer (fully reserved)
- Official form of the stable coin on a given layer, regulated and more secure
- Cheaper or free fee for redemption and lower possible cost for bridging
- User attraction by tapping maximum chain to build ecosystem around the native token
- Larger ticket size for transfer, capital efficiency and deep liquidity

Cons

- Raising of new layers make it difficult to issue natively as it need time and strategy to onramp
- Complexe regulation and auditing setup for multi-chain issuance and unified off-chain reserve management

This Pros & Cons map make it legit to propose 2 phases :

1. Use bridged token at the phase one to stimulate the ecosystem
2. Use native token in a subsequent phase taking into account a potential smooth migration strategy

Phasing strategy rationale

How can we ensure the smoothest possible transition?

The idea is to propose a path to solve the cold start problem, and avoid the pitfalls of liquidity fragmentation and heavy migrations from the bridged token to the new native token?

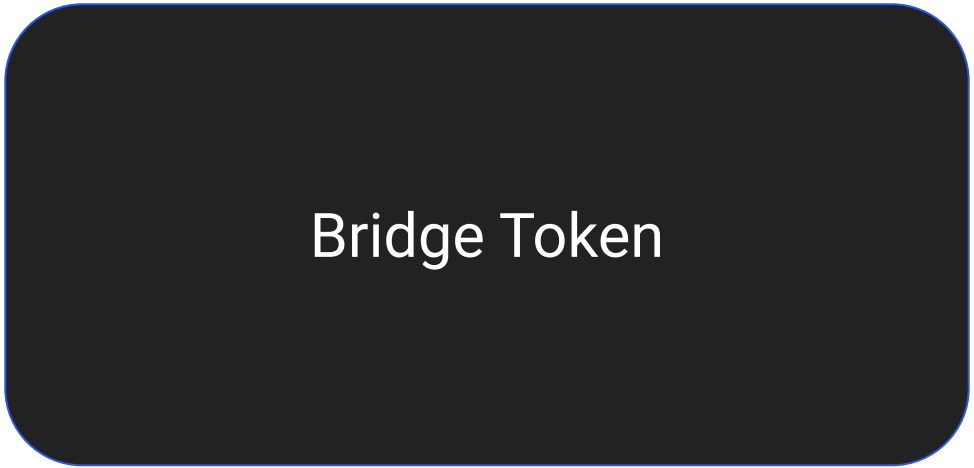
Phase I : Bridged stable Coin

- Propose a Bridged token standard with the optionality to upgrade seamlessly to native token
- The standard guarantee a secured way to transfer ownership of a bridged token contract to the issuer (builder) to facilitate an upgrade to native token
- So the key words are : **Secured, Audited, Standardized and extendible**

Phase II : Native stable Coin

- Initial state : Third party team deployed the standardized Bridged token to bootstrap initial liquidity
- The issuer decide to issue natively the stable coin as the adoption of the standard bridged token increased (or even the non standard proliferated) with significant supply, amount of holders, and number of app integrations.
- Builder and third-party jointly decide to securely transfer the ownership of the bridged token contract to the issuer (builder)
- After the transfer of ownership, the issuer upgrades bridged token to native one and seamlessly retains existing supply, holders, and app integrations.

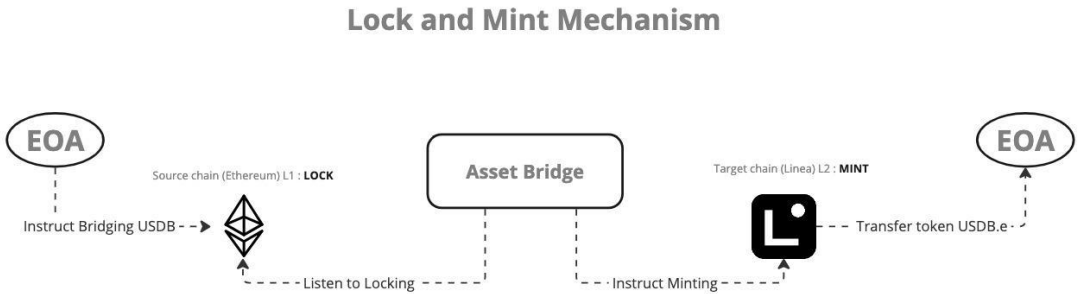
Agenda Checkpoint II



Phase I : Bridging

High Level Process

- Lock & Mint Strategy
- Lock on the source chain (Ethereum) and Mint on the target chain Linea
- It requires secure and reliable cross-chain communication to ensure a level of atomicity for the lock and mint process : **Asset Bridge**
- This bridging strategy could be used by any USDB token holder in Ethereum



Phase I : Bridging

Proposed implementation

Proposition

- Consensys Linea team operate the official **Canonical Token Bridge** optimized for builders and technical partners
- The **Canonical Token Bridge** components allow partners to have full control over the bridging if needed and give them the opportunity to implement complex cross-chain logic with some responsibility (building, auditing, and operating their own token bridge)
- The Canonical Token Bridge relies on **Messaging Services** for cross-chain interactions (Cross-chain messaging between Ethereum and Linea)

Components

- **Canonical Token Bridge** main component : **TokenBridge** smart contracts deployed on Ethereum and Linea (bunch of smart contracts)
- **MessageService** smart contracts deployed on Ethereum and Linea (bunch of smart contracts)
- Off-chain **Postman** (PostBots) to operate between both layer via listening and posting strategy

Resources

- Bridge UI :https://bridge.linea.build/
- Contracts : Contracts
- Addresses :
  - TokenBridge
  - MessageService
- Offchain components : Linea SDK
- Audits :
  - Consensys Diligence
  - Openzeppelin

Phase I : Bridging

TokenBridge Implementation focus

Key Features

- Instruct bridging :
  - Without Permit: User allows the bridge to transfer tokens on their behalf by calling allowance() on the TokenBridge. User calls the bridgeToken() method.
  - With Permit: User calls bridgeTokenWithPermit() to pass permit data in a single transaction.

- Triggering the Delivery:
  - MessagingService implementation focus : completeBridging is wrapped in a claiming message

```
// Single entry point to bridge tokens
function bridgeToken(
    address _token,
    uint256 _amount,
    address _recipient
  ) public payable nonZeroAddress(_token) nonZeroAddress(_recipient)
nonZeroAmount(_amount) whenNotPaused nonReentrant {
// Implementation
}

// Finalize the bridging
function completeBridging(
    address _nativeToken,
    uint256 _amount,
    address _recipient,
    uint256 _chainId,
    bytes calldata _tokenMetadata
  ) external nonReentrant onlyMessagingService onlyAuthorizedRemoteSender whenNotPaused
{
// Implementation
}
```

Phase I : Bridging

MessagingService Implementation focus

Key Features

- Allows a contract on the source chain to interact with a contract on the target chain : L1 TokenBridge triggering mint on the L2 TokenBridge.

- Push and Pull Mechanisms:
  - Push: Auto-execution on the target layer if a fee is paid
  - Pull: Users or Protocols are responsible for triggering the transaction from dApp or code (Postman)

```
// Sending a message
function sendMessage(address _to, uint256 _fee, bytes calldata _calldata) external {
    // Implementation
}

// Claiming a message
function claimMessage(bytes32 messageHash) external {
    // Implementation
}
```

- With Canonical Message Service, data and assets can be permissionless and reliably transferred from one blockchain to another.
- One of the most important things that the Message Service transfers is information about the current state of the Ethereum network, from Ethereum to Linea, and in return, an updated Merkle tree and a zk-proof from Linea to Ethereum, every time Linea reports back about activity on the network. In other words, the canonical message service transmits the rollup data.
- Generic : It is general-purpose, public infrastructure which can be used by developers, integrated into dapps, and triggered by end users.

=> Perfect candidate to manage the stable coin bridging

Phase I : Bridging

Postman (PostBots) Implementation focus

Key Features

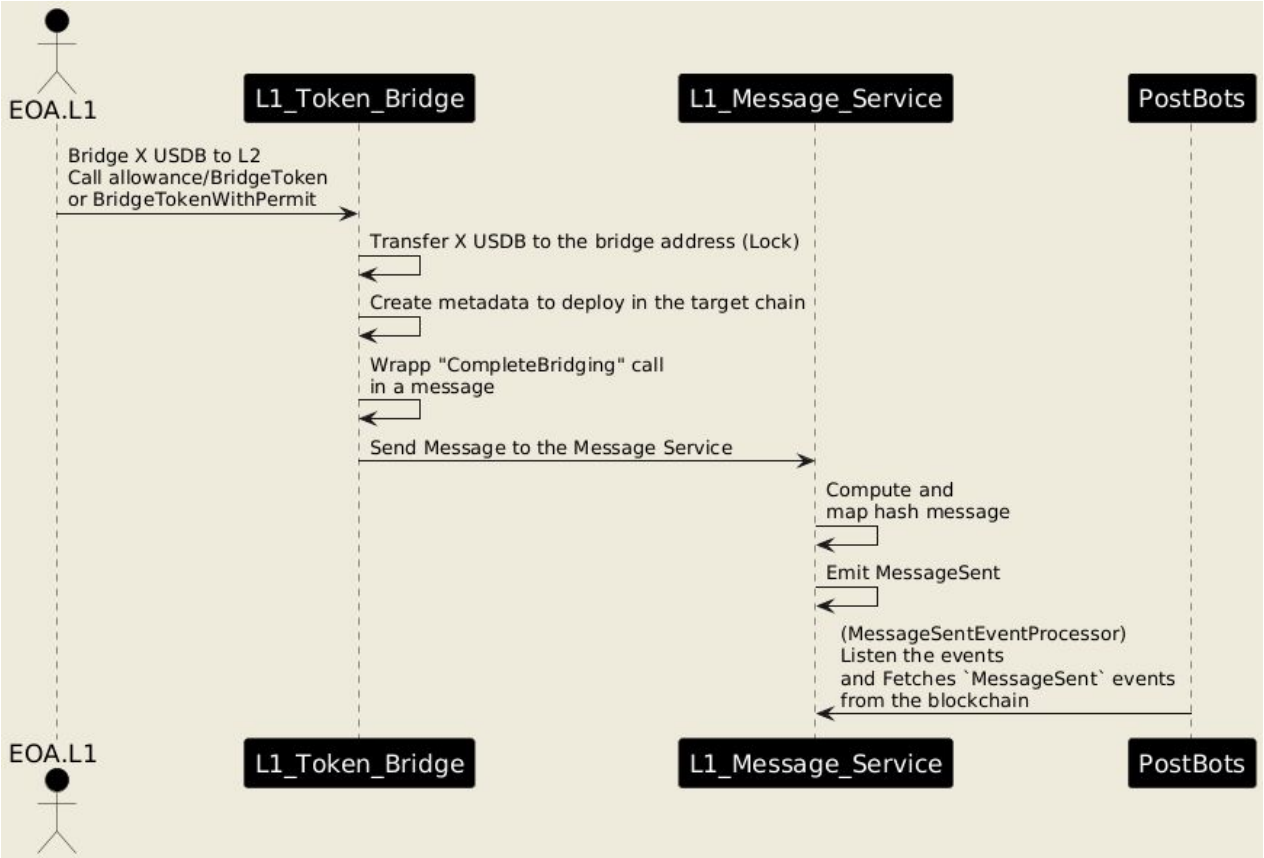
- The Postman SDK listen to the sendMessage event
- The Postman SDK can execution the message on the destination layer if some conditions are respected (Gas estimation, profit margin, rate limits).
- Triggering the Delivery:
  - If the message is not delivered by the postman, it can be manually claimed by calling claimMessage.

Main low level off-chain components involved :

- MessageSentEventProcessor::process
  - Listen to the event emitted by the smart contract and store in a DB
- MessageClaimingProcessor::process
  - Poll the DB, consider various factors and decide to claim or not

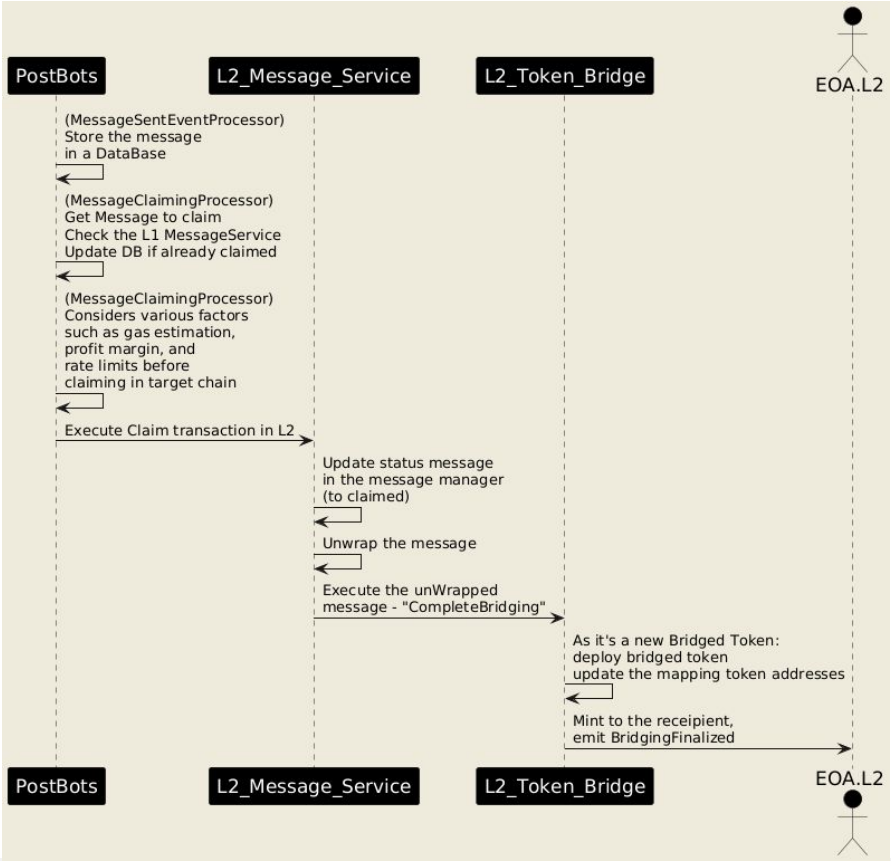
Lock & Mint orchestration

Sequence diagram - L1 part



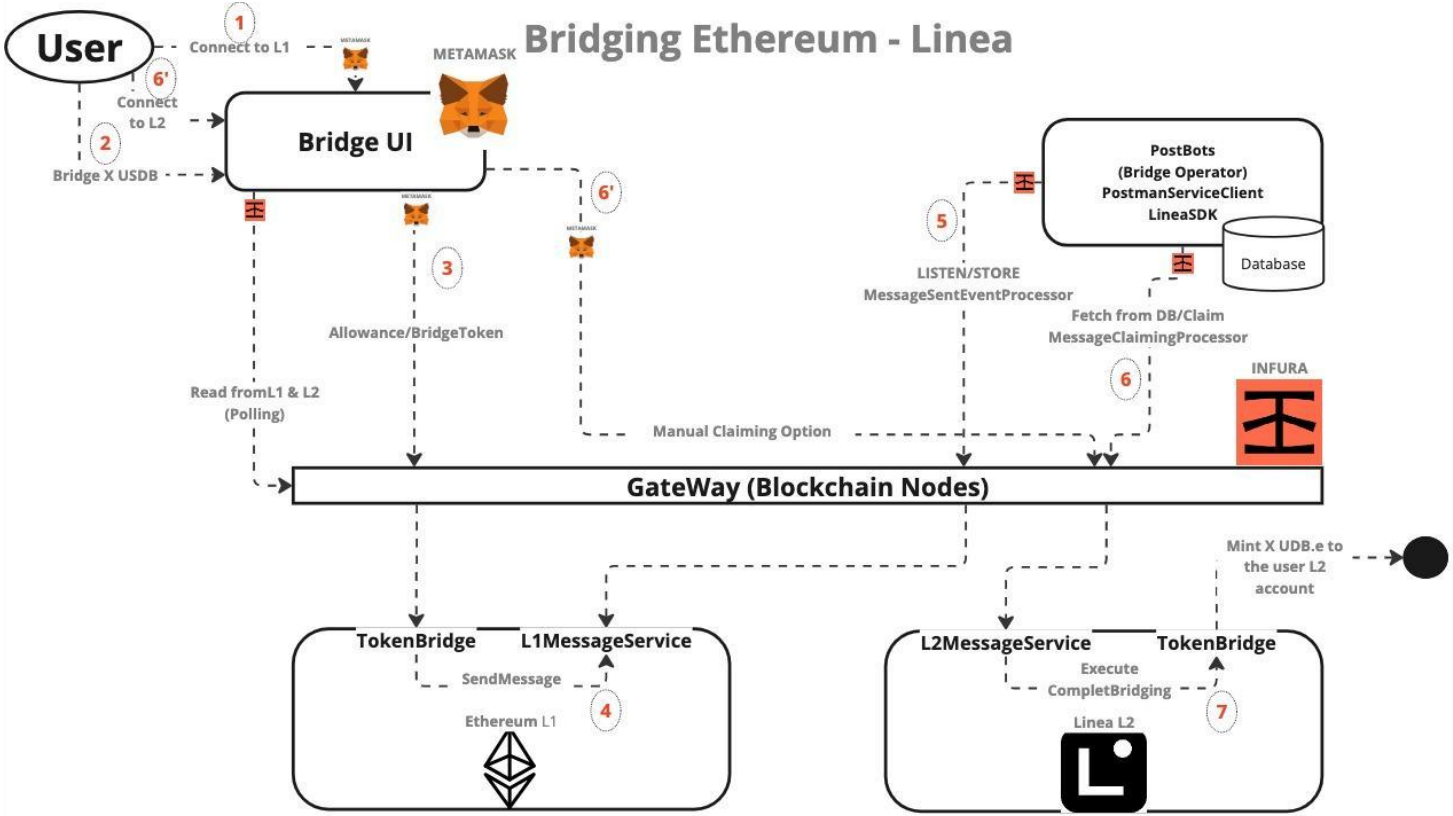
Lock & Mint orchestration

Sequence diagram - L2 part



Technical architecture to operate the Bridging

Architecture Diagram



Redeem Bridged Token

Bridging back : Burn & Transfer

Key Points

- In case of Bridged Token redemption :
  - Bridged Token is mapped with the original native one : USDB – USDB.e
  - The TokenBridge smart contract is able to detect the mapping and trigger and decide to burn the Bridged Token before sending a message to Message Service to complete the bridging
  - The claim process form the PostBot or the UI should take care to claim with the finalization proof of L2 transactions related to the Burning
  - After the L2 hard finalization detected in L1 we can proceed and transfer the native token in L1 to the user

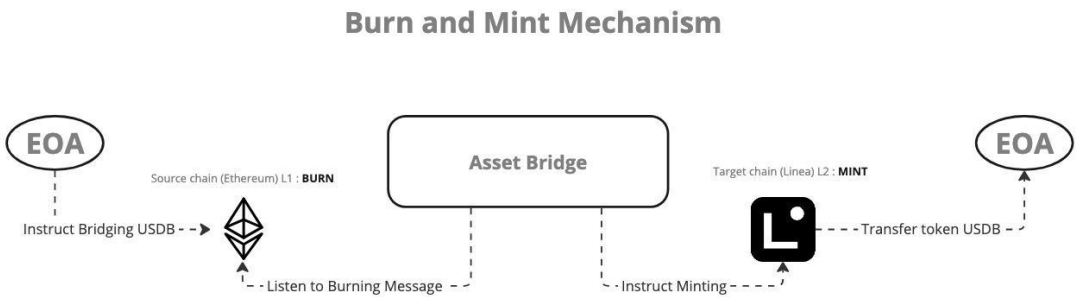
Agenda Checkpoint III

Native Issuance

Phase II : Native Issuance on L2 with Burn & Mint

High Level Process

- The builder deployed a native USDB stable coin on L2
- Bridging L1 USDB to L2 USDB follow a Burn & Mint mechanism to keep the supply reserved
- Burn on the source chain (Ethereum) and Mint on the target chain Linea
- This burn and mint could be used by any USDB token holder in Ethereum
- We are not talking about L2 Builder user minting account (L2 minting against fiat)



Phase II : Native Issuance on L2 with Burn & Mint

How to achieve the Burn & Mint ?

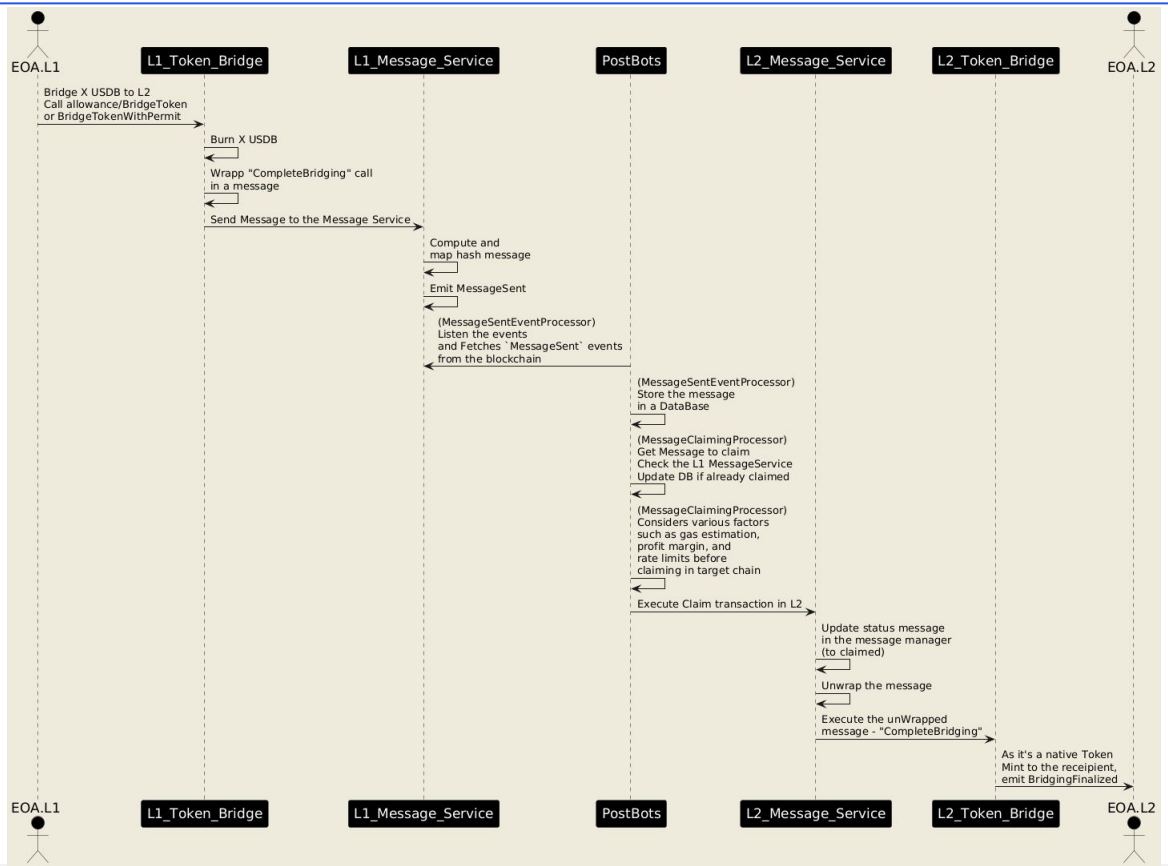
Key Points

- The Builder deploy a native stablecoin contract on L2
- Linea Canonical Token Bridge can set a custom mapping on L2 between the new native deployed contract on L2 and the native contract on L1
- This mapping allow the mint process for the new native contract on L2

```
//Linea can set a custom ERC20 contract for specific ERC20.
//For security purpose, Linea can only call this function if the token has
//not been bridged yet.
 function setCustomContract(
    address _nativeToken,
    address _targetContract
  ) external nonZeroAddress(_nativeToken) nonZeroAddress(_targetContract)
onlyOwner isNewToken(_nativeToken) {
// Implementation
}
```

Phase II : Native Issuance on L2 with Burn & Mint

Sequence Diagram



Agenda Checkpoint IV

Liquidity Management

Phase II : How to avoid Liquidity Fragmentation ?

What to do with the existing bridged token ?

Key Points

1. The builder pushed a standardized Bridged Token specification and encourage the adoption
2. Third-party team follows the standard to deploy their bridge contracts, or retains the ability to upgrade their bridge contracts in the future to incorporate the required functionality.
3. If and when a joint decision is made by the third-party team and the builder to securely transfer ownership of the bridged token contract to the Builder and perform an upgrade to native token, the following will take place:
    a. Third-party team **will pause** bridging activity and reconcile in-flight bridging activity to harmonize the total supply of native locked on the origin chain with the total supply of bridged token on the destination chain. (Bridge contract should be pausable)
    b. Third-party team will securely re-assign the contract roles of the bridged token contract to the builder.
4. The builder and the third-party team will jointly coordinate to burn the supply of native locked in the bridge contract on the origin chain and upgrade the bridged token contract on the destination chain to native token.
5. The native token contract seamlessly retains the existing supply, holders, and app integrations of the original bridged token contract.

Phase II : How to avoid Liquidity Fragmentation ?

Bridge contracts constraint

Key Points

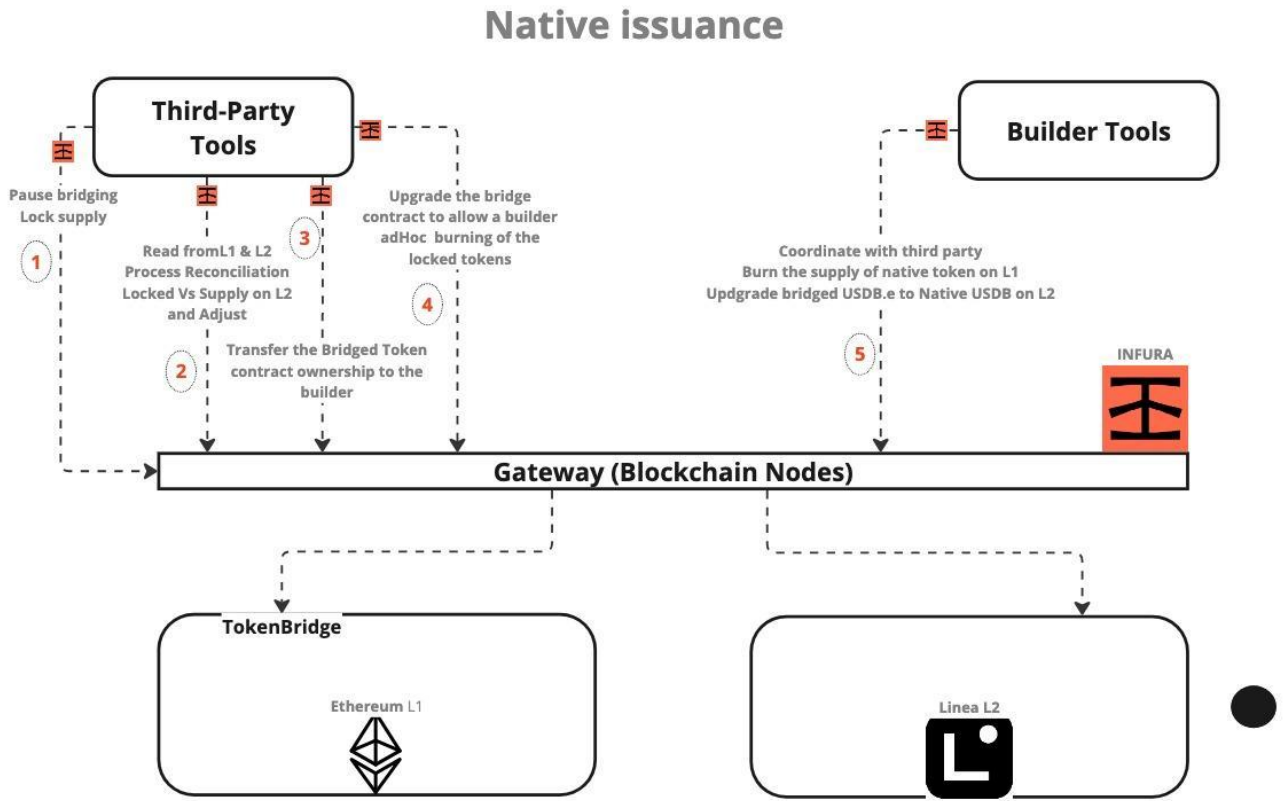
- Bridge contracts play an integral role in the process, and must be upgradable in order to add the following functionality, which is required to support the upgrade process :
    - Ability to pause USDB bridging to create a lock on the supply (source and destination layer)
    - Ability to burn native locked token (source layer)

- Ability to pause USDB bridging :
    - The bridges must be able to support a USDB supply lock, whereby the USDB locked on the source blockchain will (at some point soon after) precisely match the circulating bridged USDB supply on the destination blockchain. (Must be present before an upgrade can take place)

- Ability to burn locked USDB :
    - Burn locked USDB in the source blockchain bridge contract. To support this, the builder will temporarily assign the bridge contract holding the USDB balance the role of a zero-allowance USDB minter. This means that the bridge may burn its own held balance but not mint new supply.
    - Be only callable by an address that the Builder specifies closer to the time of the upgrade
    - Burn the amount of USDB held by the bridge that corresponds precisely to the circulating total supply of bridged USDB established by the supply lock.

Phase II : How to avoid Liquidity Fragmentation ?

Architecture

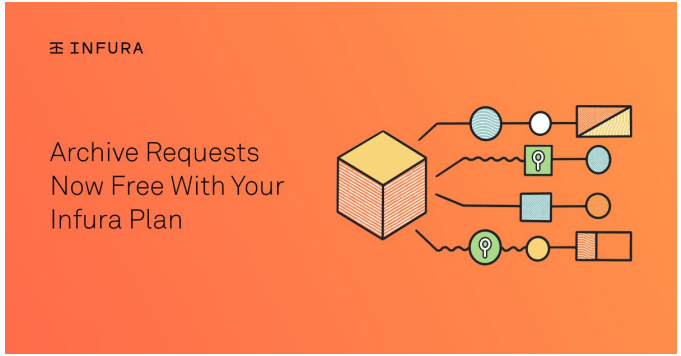


Network Data Focus

Key Points

- Operate a bridge and manage a multi-chain Stable Coin issuance need data accessibility and optimized gateway. Builder or third parties can deploy nodes and manage his own gateway but this work had been already done in the ecosystem in salable, secured and reliable way.

- Consensys propose infura to tackle this crucial point :
  - Client facing RPC API nodes for :
    - Receiving layer state to serve if needed
    - Handle incoming transaction
  - Archive nodes :
    - Data request service to handle intensive sollicitation
  - Https/Websocket protocol



Quick Recap on main points

Native issuance present some challenges which need to be tackled to smoothly migrate the ecosystem to use the new native token :

- Bridged Token Standard proposed by the builder and implemented by the third party bridges/rollup is one of the pillar of this smooth migration
- Bridge contracts capabilities as pausability and upgradability are crucial to achieve smooth migration.

Benefits for the ecosystem and the value chain :

- **Layers** : Get bridged token into the hands of developers and users early with the potential for a seamless upgrade to native token in the future.
- **Developers**: Build on bridged token with a contract address that will persist after an upgrade to native token. No code change needed to support a new asset.
- **Users**: Store, pay, trade, borrow, and lend with bridged token that automatically becomes native token upon an upgrade. No need to swap to a new asset.
- **Builders**: Avoid the time-consuming liquidity migration process of educating and incentivizing users to move from bridged token to native token.

Agenda Checkpoint V

Thank You !
Let's open the floor for discussion