Feasibility Report

Polkadot vs Cosmos for implementing "Aragon Chain"

Created by ChainSafe Systems

Authors: *Stephanie Zhang, Austin Abell, Elizabeth Binks and Amer Ameen*

Editors: *Colin Schwarz, Amer Ameen*

Contents

Overview

This report explores the implementation of Aragon Chain with EVM compatibility and feature parity on either Polkadot or Cosmos. An additional requirement is that applications, including Aragon applications & contracts developed in Solidity, must be supported on Aragon Chain.

Based on our research, these requirements should not be an issue if Aragon Chain is implemented using Ethermint on Cosmos or an EVM-based parachain on the Polkadot network. We expect that existing tooling and software (e.g. Metamask, block explorers, etc.) will be compatible with either network with minimal development effort. This is based on the expectation that Ethermint and any potential EVM-based parachain will implement RPC calls that are equivalent to those in Ethereum. That being said, in order to present a comprehensive assessment, we have also outlined options other than an EVM-based chain on Polkadot. In this case, for existing tooling and software to be compatible, there would need to be a more significant development effort to ensure compatibility with Aragon Chain.

Polkadot

Polkadot is a network of blockchains (called parachains) that are interoperable via a relay chain and share security with the entire network.

Security

Security on Polkadot is shared across the network. This means that, on Polkadot, Aragon would not have to ensure that there is sufficient security (i.e. validators) on its parachain. That being said, there is an added cost to operating a parachain, since you have to lease a parachain slot. The cost of this has not been determined, since these parachain slots will be auctioned off. On Polkadot, validators take turns validating the state transitions of each parachain. The relay chain coordinates the security of the network by randomly assigning validators to validate a particular parachain for a particular block. The state of the relay chain is dependent on the state of each of the parachains and consensus is reached using the GRANDPA consensus algorithm. This means the state of a parachain cannot be reversed without the state of the relay chain (i.e. the state of every parachain) being reversed.

Message Passing

Shared security implies that each parachain is as secure as the entire Polkadot network. Thus, a parachain can trust any message that it receives from any other parachain. In addition to coordinating validators to secure the network, the relay chain also passes messages from parachain to parachain using a scheme called Interchain Message Passing (ICMP). Interchain messaging in Polkadot works in the following way: each block in a parachain will have a field that says, “relay chain, please relay this message to chain ABC”. Specific transactions will be able to trigger the inclusion of these messages in a block. Note that these messages can be any arbitrary data, neither the sending chain nor the relay chain necessarily needs to understand them. Once a block is produced that contains these “please relay”

message fields, the validators of the relay chain will facilitate their delivery. Since the network of relay chain validators overlaps with the sending parachain's network as well as the receiving parachain's network, the relay chain will consistently be able to deliver these messages. Once received, the arbitrary data sent will be decoded, and if valid, will cause some action to occur on the receiving parachain.

Smart Contracts and EVM Compatibility

Although the relay chain will not have smart contract capabilities, other parachains will. A parachain's runtime (i.e. state transition function) can be written in any language that compiles to WebAssembly (Wasm). The runtime for each parachain is unrestricted; the only requirement being that the interface for a parachain is implemented (execute_block, validate_transaction, etc.). To create a smart contract parachain, the runtime would need to support smart contracts. One way to create a custom runtime is by using runtime modules which are packages that provide specific functionality. To make a parachain that supports smart contracts, we could import a module from the runtime library that provides that functionality. At this point, there is a parachain for running Wasm contracts (edgeware), but there isn't a parachain for running EVM compatible smart contracts. We expect there to be an EVM compatible parachain and an EVM runtime module in the future, but the timeframe is uncertain.

Potential Aragon Chain Implementations on Polkadot

The four options described below port Aragon functionality to Polkadot. The first two are Aragon-specific parachains, whereas the third option is to use an existing smart contract parachain and the fourth is to deploy Aragon Chain as a fully custom parachain.

In order to maintain compatibility with existing tooling and software (wallets and other UIs) the RPC calls on the parachain must match the Ethereum RPC calls. For options one and two, this is possible, since the RPC calls for an Aragon-specific parachain could be customized to Aragon's needs. However, for option three, the RPC calls would not be customizable, so there is no guarantee that existing tooling and software could be used. Polkadot specific tooling exists, but it is not as mature as the tooling available for Ethereum. That being said, we think it is likely that an EVM compatible parachain would attempt to be compatible with Ethereum tooling and software by developing RPC calls to match Ethereum.

Option 1: Deploy Aragon Chain as a parachain that supports Wasm (or Ewasm) smart contracts.

1. Develop a runtime that supports Wasm and deploy it as a parachain (Aragon Chain).
    - Wasm or Ewasm could be used, but it is more likely that a reliable Solidity to Ewasm compiler will be available before a Solidity to Wasm compiler.
    - Ewasm (Ethereum flavored WebAssembly) is a restricted subset of Wasm that is intended to be used to develop smart contracts in Ethereum. Details of the specific differences between Wasm and Ewasm can be found here.
2. Convert existing Aragon Solidity smart contracts to Ewasm.

- This would be done using a compiler such as https://github.com/hyperledger-labs/solang
    - This compiler is still a proof-of-concept, so it is not in a state to be used to generate Wasm for any production software. There is no estimated timeframe for a production release.
    - See their TODO list here for an idea of what is still not possible https://github.com/hyperledger-labs/solang/blob/master/TODO.md.
    - Ethereum is moving to Ewasm for Ethereum 2.0 so it is presumed that compiling Solidity to Ewasm should be a standard practice in the future.
- An alternative would be to re-write the smart contracts in a language that compiles to Wasm (such as Rust, Go, C, C++, Java, Python). Although this would require significantly more effort, it would remove the dependency on a Solidity to Wasm compiler.

3. Deploy Aragon Ewasm smart contracts to Aragon Chain.
4. Modify existing wallets and software to support the new infrastructure.
    - Assuming Aragon Chain RPC calls have been developed to match Ethereum, this should be relatively trivial.

Option 2: Deploy Aragon Chain as a parachain that can execute EVM code.

1. Develop a runtime that can execute EVM code and deploy it as a parachain (Aragon Chain).
    - In this custom runtime, optimizations could be made to the EVM for Aragon's functionality.
2. Deploy existing Aragon smart contracts on Aragon Chain.
    - It would be as simple as deploying EVM bytecode to Aragon Chain.
3. Modify existing wallets and software to support the new infrastructure.
    - Assuming Aragon Chain RPC calls have been developed to match Ethereum, this should be relatively trivial.

Option 3: Deploy Aragon smart contracts on a smart contract parachain that can execute Wasm or EVM code.

1. Select an existing smart contract parachain.
    - Edgeware (Wasm smart contracts) is currently the only smart contract platform. We expect there to be an EVM compatible smart contract parachain in the future, however the timeframe is uncertain.
2. Deploy existing Aragon smart contracts on Aragon Chain.
    - As described above, the process is different for a Wasm parachain than for an EVM parachain.
        - Wasm parachain: Aragon smart contracts would either have to be compiled to Wasm using a Solidity to Wasm compiler or rewritten in a language that compiles to Wasm before deployment.

        - EVM parachain: existing Aragon smart contracts could be deployed to the EVM parachain as EVM bytecode.
    - Since this is not an Aragon Chain, optimizations would be limited to the smart contract layer (like on the Ethereum mainnet).
3. Modify existing wallets and software to support the new infrastructure.
    - This could require significant effort if the RPC calls are significantly different on the parachain.

Option 4: Deploy Aragon Chain as a fully custom parachain.

1. Based on the functionality of the existing Aragon smart contracts, design and develop a custom Aragon runtime for Aragon Chain.
    - This is a significant undertaking as the existing work would essentially be a prototype for this new custom blockchain. The logic of the smart contracts would be represented in the runtime itself.
2. Deploy Aragon Chain with the custom Aragon runtime.
    - This should be trivial.
3. Modify existing wallets and software to support the new infrastructure.
    - The RPC calls for this Aragon Chain would be developed to match Ethereum's RPC calls, although more research would need to be done to understand if we can match Ethereum's RPC calls for all cases.

Cosmos

Cosmos is a network of independent blockchains, called zones, that interact with each other through connector blockchains, called Hubs. Hubs are root custodians. They facilitate message passing and token transfers between zones. Hubs can be connected to other hubs, which would have their own set of zones, all of which would be able to interoperate. The first Hub, the Cosmos Hub has launched and is now operational.

Security

Blockchains on Cosmos are secured by Tendermint BFT Consensus. Tendermint is notable for its simplicity, performance, and fork-accountability (see Appendix for more info). Each Zone is responsible for organizing a set of validators. This set of validators is known and each active validator is expected to participate in securing the zone. Hubs are responsible for guaranteeing the security of the messages that are passed through them. As mentioned in the whitepaper, the Cosmos Hub does not verify or execute transactions committed on other zones, so it is the responsibility of users to send tokens to zones that they trust.

Message Passing

Inter-Zone communication would take place through Cosmos's Inter-Blockchain Communication protocol and would allow for seamless transit of assets and messaging between distinct zones.

Below is a breakdown of the current specification of the Inter-Blockchain Communication Protocol:

1. A transaction or message is packaged into a packet (format) and is sent to the IBC module on Blockchain A.
2. The handler in the IBC module on Blockchain A then performs the following steps in order to send the packet to Blockchain B (Source):
    a. Handler checks that the channel & connection are open to send packets.
    b. Handler checks that the calling module owns the sending port.
    c. Handler checks that the packet metadata matches the channel & connection information.
    d. Handler checks that the timeout height specified has not already passed on the destination chain.
    e. Handler increments the send sequence counter associated with the channel.
    f. Handler stores a constant-size commitment to the packet data.
3. The packet data is then sent through an off-chain relayer to the corresponding channel in Blockchain B's IBC module (see: Source).
4. The handler in the IBC module on Blockchain B then performs the following steps in order to receive and process the relayed packet from Blockchain A (Source):
    a. Handler checks that the channel & connection are open to receive packets.
    b. Handler checks that the calling module owns the receiving port.
    c. Handler checks that the packet metadata matches the channel & connection information.
    d. Handler checks that the packet sequence is the next sequence the channel end expects to receive (for ordered channels).
    e. Handler checks that the timeout height has not yet passed.
    f. Handler checks the inclusion proof of packet data commitment in the outgoing chain's state.
    g. Handler sets the opaque acknowledgement value at a store path unique to the packet (if the acknowledgement is non-empty or the channel is unordered).
    h. Handler increments the packet receive sequence associated with the channel end (ordered channels only).
5. Blockchain B decodes then acts on the data contained within the packet.
6. After Blockchain B has processed and acknowledged the packet, Blockchain A processes the acknowledgement of the package it previously sent and cleans its packet commitment.

For more information on how the Inter-Blockchain Communication Protocol will work please see the specification here.

Smart Contracts and EVM Compatibility

Ethermint is an implementation of the Ethereum Virtual Machine on top of Tendermint consensus. Once completed, Ethermint will provide the same functionality as the Ethereum mainnet.

Potential Aragon Chain Implementation

We would set up Aragon Chain to run as an Ethermint zone to take advantage of the built in functionality identical to that of a standalone EVM based chain. Ethermint is being built to be completely interoperable with the Ethereum mainnet through Cosmos' Inter-Blockchain Communication Protocol (in combination with the Ethereum Peg Zone). There is currently no known timeline for completion of the IBC and a bidirectional Peg Zone, but Cosmos developers are already deploying applications on Ethermint. This means that existing smart contracts, Web3 utilities, and tooling will be compatible with Aragon Chain with minimal development effort (the RPC calls related to Proof of Work will change due to Proof of Stake which may require some updates to existing software). Furthermore, Cosmos is currently working towards implementing an API layer that will be compatible with Web3. All existing Aragon Core smart contracts could be seamlessly deployed as-is to the new Aragon Chain.

Chain-Level Optimization for Aragon Smart Contracts

For either approach, in order to optimize, we would have to take Aragon's core contracts and reimplement functionality at a lower level where advantageous. This could mean potentially turning contracts (or portions of them) into opcodes.

Polkadot Chain-Level Optimizations

If we use an EVM-based parachain (Option 2) we could add custom instructions for Aragon. In this case, we would need to modify the runtime and add more opcodes. If we choose to compile Solidity to Wasm and execute it on a parachain (Option 1), we could optimize this, but it would need to be on the compiler level, perhaps through manual optimizations as there is no direct equivalent of "new opcodes". As a result, we would have to write special instructions to be executed in a specific way. This is a more complex optimization.

For Option 4, creating a custom parachain with the Aragon smart contract functionality built into the runtime of the parachain, we would be able to perform the most significant optimizations. This is essentially the value-proposition of a parachain, since you are not bound by the limitations of a virtual machine that is intended to be a much more general platform. In this case, we would define custom extrinsics on the DAO chain: e.g. vote, submit proposal, stake etc. The runtime would deal with these different extrinsics and update the state accordingly. We could also define custom block format and add extra fields if we'd like. For example currentProposals, currentActiveDAOs, staking info etc. We could get different users to vote to update the DAO chain's runtime, possibly adding new extrinsics or other governance mechanisms as the chain progresses. If desired, the DAO chain could be governed by an

actual DAO. This would be the cheapest (in terms of fees) and most optimized option but would also be the most significant undertaking of all the options.

Cosmos Chain-Level Optimizations

Assuming Aragon Chain would be built using Ethermint, we would be able to modify the underlying EVM module to implement custom opcodes for Aragon's desired functionality. These opcodes would make the operations in existing common interactions in Aragon's core smart contracts more efficient.

It would also be possible to create an "Aragon Hub", creating an additional lower cost zone (also running on Ethermint) where DAO operations would take place and relevant smart contracts could be deployed as precompiles. Users on the main Aragon Hub could directly interact with this zone and perform complex contract calls through Cosmos's Inter-Blockchain Communication protocol (more details on this protocol below).

Additionally, Aragon could look into utilizing Cosmos's native governance module (and/or building on top of it) on Aragon Chain in the place of the current voting smart contract. See here for more on Cosmos governance.

The State of Polkadot Cross-Chain Communication

Polkadot plans to build a bridge that would connect an Ethereum parachain (to be launched at genesis) to the Ethereum mainnet (see prototype here) and that would have cross-parachain communication. The bridge is still in the research stage and there is no expected completion date at the time of writing. However, this is the main piece of software that needs to exist for an Aragon parachain to interact with the Ethereum mainnet. This system for cross-chain communication would consist of two primary components: the bridge software, that monitors the Polkadot parachain and the associated Ethereum contract(s), and the parachain that is being bridged. This would be either an Ethereum parachain which mirrors the Aragon mainnet or an Aragon Chain itself. Alternatively, a bridge could be built between the Ethereum mainnet and Aragon Chain directly. However, this may require more time since the current bridging spec for parachains is still in the research phase. It is important to note that Polkadot uses sr25519 for signing and has support for ed25519 curve keys. Ethereum uses the secp256k1 curve and uses ECDSA as its signing algorithm. Keys on one chain will not necessarily correspond to keys on the other and will thus need separate wallets. However, a parachain can be configured to use whatever signing it wants so we could implement secp256k1 and ECDSA into the Aragon parachain if we choose this approach.

Below we've outlined a process for cross-chain communication, assuming that Aragon is built on an EVM-based parachain. There steps are based off the current Parity Bridge linked above. However, the bridge is subject to change so the steps may change accordingly.

Ethereum Parachain to Ethereum Mainnet

1. Sender on the Ethereum parachain (Polkadot) calls the relay message function (containing transaction data and intended recipient) which emits a RelayMessage event.
2. Bridge authorities witness RelayMessage event.
3. Bridge authorities create and sign a message of the transaction referenced in the RelayMessage event and submit it.
4. Bridge authorities watch until some threshold amount of authorities have signed and submitted the message associated with the transaction.
5. Bridge authorities then relay the message to the smart contract on the Ethereum mainnet.
6. The smart contract on Ethereum mainnet receives the message, verifies that a sufficient amount of recognized authorities have signed the message, emits an AcceptMessage event and then executes the transaction in the message.

Ethereum Mainnet to Ethereum Parachain

1. Sender on the Ethereum mainnet calls the relay message function (containing transaction data and intended recipient) on the smart contract that the bridge is monitoring.
2. The relay message function emits a RelayMessage event.
3. Bridge authorities witness the RelayMessage event and create and sign an AcceptMessage transaction.
4. Bridge authorities watch until some threshold amount of distinct authorities has signed and submitted an AcceptMessage transaction, then execute the transaction on the Aragon parachain (Polkadot).

For more information on the above, this article explains how an Ethereum parachain will be launched, hopefully for genesis, while this article describes bridging for PoW chains.

The State of Cosmos Cross-Chain Communication

Cosmos is currently developing an Inter-Blockchain Communication protocol to facilitate cross-chain communication. Cosmos is also developing a chain that hosts an Ethereum specific bridge that supports messaging and token transfer bi-directionally from any blockchain built on the Cosmos SDK to Ethereum (see https://github.com/swishlabsco/peggy and note that it is currently in Alpha and unidirectional: Ethereum to Cosmos). Since Aragon's chain would require compatibility with existing Ethereum ecosystem tooling and utilities, it would presumably be running as an "Ethermint Zone". This assumes Aragon would be using the Cosmos Hub's bridge and not launching their own bridge chain. This could be accomplished by setting Aragon's own threshold of stake required to validate actions but would cause a loss in the added security provided by having a larger pool of validators. Alternatively, a custom bridge could be used. Attaching the Peggy module to a sort of "Aragon Hub" would also be an option,

although perhaps less viable as Aragon would need sufficient trusted active validators to secure the bridge zone.

The process for cross-chain Cosmos to Ethereum communication would be as follows. As a disclaimer, this is theoretical, since none of it has been implemented yet.

1. Message is relayed from Aragon Chain to the Cosmos Hub via the Inter-Blockchain Communication Protocol (IBC).
2. Message is relayed to the Ethereum Peg Zone from Cosmos Hub via the IBC.
3. Within the Ethereum Peg Zone, the message is decoded into an Ethereum readable format (note that the standard format of a message packet has not yet been documented) and signed by at least ⅔ total signers (Signers are also Cosmos Validators who secure the Cosmos Hub, it would require that at least ⅔ of active validators securing the hub itself are malicious for the transaction to be modified in any way).
4. Relayers watch the message (now translated into an equivalent transaction) until it is signed by at least ⅔ of total signers, then post batches of these transactions to the appropriate Ethereum bridge smart contract.
5. Action takes place on the Ethereum mainnet.

The process from cross-chain communication from Ethereum to Cosmos is as follows. This is based on current Peggy documentation and implementation, so the names are changed according to Peggy naming conventions.

1. Transaction is sent to the bridge smart contract on the Ethereum mainnet which triggers an event.
2. Event is witnessed by witnesses who then wait 100 blocks to ensure finality before parsing the information associated with that transaction and building an unsigned Cosmos transaction.
3. Unsigned Cosmos transactions are then signed and submitted to the Ethereum Peg Zone Bridge.
4. The Ethereum Peg Zone Bridge takes the signed transactions and converts them into generic claims. It then submits them to the Oracle module.
5. Once at least ⅔ active witnesses have submitted the same claim, or a threshold of stake that has been set, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status (or Failure status if some threshold of stake of active witnesses do not agree with the claim).
6. If the claim is returned with a Successful status the Ethereum Peg Zone Bridge performs the action indicated, relaying the transaction to the Cosmos Hub, possibly via the IBC, depending on how it ends up being built out.
7. The Cosmos Hub would then relay the transaction via the IBC to Aragon Chain.

Below is a breakdown of the current specification of the Inter-Blockchain Communication Protocol as first shown starting on page 8. Please note that incentives and fee structures for the following have not been specified to our existing knowledge, though we expect that relayers will be awarded a portion of the transaction fees used to send the IBC packet to the module, and that the fees will be payable with multiple tokens:

1. An action or message is packaged into a packet (format defined by: Source) and is sent to the IBC module on Blockchain A.
2. The handler in the IBC module on Blockchain A then performs the following steps in order to send the packet to Blockchain B (Source). This would presumably be performed at the individual node level:
    a. Handler checks that the channel & connection are open to send packets.
    b. Handler checks that the calling module owns the sending port.
    c. Handler checks that the packet metadata matches the channel & connection information.
    d. Handler checks that the timeout height specified has not already passed on the destination chain.
    e. Handler increments the send sequence counter associated with the channel.
    f. Handler stores a constant-size commitment to the packet data.
3. The packet data is then sent through an off-chain relayer to the corresponding channel in Blockchain B's IBC module (see: Source).
4. The handler in the IBC module on Blockchain B then performs the following steps in order to receive and process the relayed packet from Blockchain A (Source). This would also presumably be performed by an individual node:
    a. Handler checks that the channel & connection are open to receive packets.
    b. Handler checks that the calling module owns the receiving port.
    c. Handler checks that the packet metadata matches the channel & connection information.
    d. Handler checks that the packet sequence is the next sequence the channel end expects to receive (for ordered channels).
    e. Handler checks that the timeout height has not yet passed.
    f. Handler checks the inclusion proof of packet data commitment in the outgoing chain's state.
    g. Handler sets the opaque acknowledgement value at a store path unique to the packet (if the acknowledgement is non-empty or the channel is unordered).
    h. Handler increments the packet receive sequence associated with the channel end (ordered channels only).
5. Blockchain B decodes then acts on the data contained within the packet.
6. After Blockchain B has processed and acknowledged the packet, Blockchain A then processes the acknowledgement of the package it previously sent and cleans its packet commitment.

For more information on how the Inter-Blockchain Communication Protocol will work please see the full specification here.

Interchain Communication Scenarios: Polkadot and Cosmos Flow Breakdowns

Note that each of the following flows rely on the software described in the previous section being complete and functional as currently documented.

Scenario One: An organization deployed on Aragon Chain controlling an "agent" on Ethereum mainnet. (See this article on Aragon agents)

For Cosmos

(Assuming that Cosmos' Inter-Blockchain Communication Protocol and Ethereum Peg Zone are completed and function as currently documented):

If we assume Aragon Chain would be running as an "Ethermint Zone" to take advantage of the built-in functionality identical to a standalone EVM based chain, the following flow would be appropriate:

1. Organization sends a packet containing agent direction transaction relayed from Aragon Chain to the Cosmos Hub via the IBC.
2. Packet containing agent direction transaction is relayed to the Ethereum Peg Zone from the Cosmos Hub via the IBC.
3. Within the Ethereum Peg Zone, the message is decoded into an Ethereum readable format (note that the standard format of a message packet has not yet been documented) and signed by at least ⅔ total signers.
4. Relayers watch the message (now translated into an equivalent transaction) until it is signed by at least ⅔ of total signers, then post the transaction to the appropriate bridge smart contract on the Ethereum mainnet.
5. The bridge smart contract sends a transaction giving the directions to the agent.

For Polkadot:

1. An organization deployed on a parachain controls an Ethereum agent by sending a parachain message through the relay chain to an Ethereum-bridged parachain.
2. On the Aragon parachain, a transaction is sent that results in an interchain message being broadcast.
3. This message is received by the relay chain which routes it to the Ethereum bridge parachain.
4. This causes some action to occur on the Ethereum parachain.
5. Since the Ethereum parachain is bridged to Ethereum, a corresponding action will happen on Ethereum.

In this scenario, we need to trust that the Ethereum to Polkadot bridge is functional and not faulty. As well, we must assume that the message passing through the relay chain has not been corrupted (ie. Polkadot isn't being attacked) and that the message is actually delivered to the Ethereum bridge parachain (no significant network partitions or delays).

Scenario Two: An organization deployed on Ethereum mainnet controlling an "agent" on Aragon Chain.

For Cosmos

1. The organization sends a transaction containing an agent direction to the bridge smart contract present on the Ethereum mainnet.
2. The event is witnessed by witnesses who then parse the information associated with that transaction and build an equivalent unsigned Cosmos transaction.
3. The unsigned Cosmos transaction is signed and submitted to the Ethereum Peg Zone Bridge.
4. The Ethereum Peg Zone Bridge takes the signed transaction and converts it to a generic claim. It then submits it to the Oracle module.
5. Once at least ⅔ active witnesses have submitted the same claim, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status.
6. If the claim is returned with a Successful status, the Ethereum Peg Zone Bridge then relays the transaction (converted to a packet) to the Cosmos Hub via the IBC.
7. The Cosmos Hub would then relay the transaction packet via the IBC to Aragon Chain.
8. The transaction packet is decoded and sent to the corresponding agent on Aragon Chain.

For Polkadot

Assuming there is a functional Ethereum to EVM parachain bridge:

1. The organization sends a transaction to the parachain bridge contract on Ethereum.
2. This transaction is picked up by the bridge and the corresponding action occurs on the Ethereum parachain.
3. The Ethereum parachain sends a message to Aragon Chain.
4. This message is relayed by the relay chain.

In this scenario, we again need to trust that the Ethereum to Polkadot bridge is functional and not faulty. Again, we assume that the message passing through the relay chain has not been corrupted (i.e. Polkadot isn't being attacked) and that the message is actually delivered from the Ethereum bridge parachain to Aragon Chain (no significant network partitions or delays).

Scenario Three: An organization wants to hold DAI (native to the Ethereum mainnet) in a vault on Aragon Chain.

For Cosmos

1. A transaction is sent to a bridge smart contract present on the Ethereum mainnet. This transaction is of the value transfer of DAI that the organization wants to hold in a vault on Aragon Chain.
2. This DAI is locked in the smart contract, triggering a lockup event.

3. The lockup event is witnessed by witnesses who then parse the information associated with that transaction and build an equivalent unsigned Cosmos transaction.
4. Unsigned Cosmos transactions are then signed and submitted to the Ethereum Peg Zone Bridge.
5. The Ethereum Peg Zone Bridge takes the signed transaction and converts it to a generic claim. It then submits it to the Oracle module.
6. Once at least ⅔ active witnesses have submitted the same claim, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status.
7. If the claim is returned with a Successful status, the Peg Zone mints a synthetic version of DAI (still an ERC-20).
8. The bank module sends this DAI to the Vault on Aragon Chain's address via the IBC through the Cosmos Hub to the IBC Aragon Chain.

For Polkadot

The vault on Aragon Chain would own a synthetic version of DAI that is pegged to DAI on Ethereum. To release DAI from the vault back into the Ethereum mainnet, the vault would:

1. Burn the DAI.
2. Send a message to the Ethereum bridge parachain to release the DAI on Ethereum.
3. This would cause a transaction to occur on the Ethereum parachain which will get picked up by the bridge.
4. The bridge would send a transaction that will allow for the release of mainnet DAI.

Alternatively, the organization could change the permissions on its vaulted DAI, allowing an Aragon Chain user/app to spend it. In this case the withdrawer of the DAI would either keep the synthetic DAI on Aragon Chain, or send it through the bridge to the Ethereum mainnet.

Scenario Four: The native staking token for Aragon Chain would be linked to a token native to the Ethereum mainnet (ANT or a derivative token).

For Cosmos

Using ANT on Aragon Chain:

ANT held on the Ethereum mainnet to recipient on Aragon Chain.

1. A transaction is sent to a bridge smart contract present on the Ethereum mainnet. This transaction contains the value of ANT the user wants to transfer and the address of the recipient on Aragon Chain.
2. The ANT is locked in the smart contract, emitting a lockup event.

3. The lockup event is witnessed by witnesses who then parse the information associated with that transaction and build an equivalent unsigned Cosmos transaction.
4. The unsigned Cosmos transaction is then signed and submitted to the Ethereum Peg Zone Bridge.
5. The Ethereum Peg Zone Bridge takes the signed transaction and converts it to a generic claim. It then submits it to the Oracle module.
6. Once at least ⅔ active witnesses have submitted the same claim, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status.
7. If the claim is returned with a Successful status, the Peg Zone mints a synthetic version of ANT (Would still be an ERC-20 MiniMe token).
8. The bank module sends this ANT to the recipient address.

ANT Held on Aragon Chain to recipient on the Ethereum mainnet.

1. User sends token transfer via the IBC to the Cosmos Hub with directions to relay to the Ethereum Peg Zone, and the address of the intended recipient.
2. Token transfer is relayed to the Ethereum Peg Zone from the Cosmos Hub via the IBC.
3. Within the Ethereum Peg Zone, the ANT is locked. A lockup message is emitted containing details of the recipient and the amount of ANT locked.
4. Relayers watch the message (now translated into an equivalent transaction) until it is signed by at least ⅔ of total signers, then post the transaction to the appropriate bridge smart contract on the Ethereum mainnet.
5. The bridge smart contract unlocks an equivalent amount of ANT and sends a transaction to the intended recipient.

Derivative token on Aragon Chain that is linked to ANT via Bonding Curve:

To mint the initial balances for the native staking token on the Aragon Chain, Aragon could replicate the state of accounts currently holding ANT on their chain as a sort of spoon with the new derivative token (this is similar to what Cosmos was planning to do with OmiseGo as seen in this article.

If one wishes to link any further minting of ANT, the following could be done:

1. Set up witnesses to observe the ANT contract and await events such as minting.
2. A minting event is witnessed by witnesses who then parse the information associated with that transaction and build an equivalent unsigned Cosmos transaction.
3. An unsigned Cosmos transaction is then signed and submitted to the Ethereum Peg Zone Bridge.
4. The Ethereum Peg Zone Bridge takes the signed transaction and converts it to a generic claim.
5. The Ethereum Peg Zone Bridge then submits the generic claim to the Oracle module.
6. Once at least ⅔ active witnesses have submitted the same claim, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status.
7. If the claim is returned with a Successful status, the Peg Zone mints a synthetic version of DAI (still an ERC-20).

8. The bank module sends this DAI to the vault on the Aragon Chain's address via the IBC through the Cosmos Hub to the IBC Aragon Chain.

For Polkadot

The below scenarios assume that Aragon Chain is a POA EVM-based parachain.

ANT held on Ethereum mainnet to a recipient on Aragon Chain.

1. Sender on the Ethereum mainnet calls the relay message function with a value transfer equivalent to what they wish to send. The transaction must contain transaction data and the intended recipient and is sent to a smart contract that the bridge is monitoring.
2. The value transfer is locked in the smart contract
3. The relay message function emits a RelayMessage event.
4. Bridge authorities witness the RelayMessage event and create and sign an AcceptMessage transaction.
5. Bridge authorities watch until some threshold amount of distinct authorities have signed and submitted an AcceptMessage transaction, then execute the transaction on the Aragon parachain (Polkadot).

ANT Held on Aragon Chain to recipient on the Ethereum mainnet.

1. Sender on Aragon Chain (Polkadot) calls the relay message function (containing transaction data and intended recipient) with a value transfer equivalent to what they wish to send the recipient on the Ethereum mainnet. The Ethereum mainnet then emits an RelayMessage event.
2. Value transfer is Locked.
3. Bridge authorities witness the RelayMessage event.
4. Bridge authorities create and sign a message of the transaction referenced in the RelayMessage event and submit it.
5. Bridge authorities watch until some threshold amount of authorities have signed and submitted the message associated with the transaction.
6. Bridge authorities then relay the message to the smart contract on the Ethereum mainnet. The smart contract on the Ethereum mainnet receives the message, verifies that a sufficient amount of recognized authorities have signed the message, emits an AcceptMessage event and then executes the transaction in the message.

To mint the initial balances for the native staking token on Aragon Chain within the Polkadot context, we would create a parachain:

1. The distribution of this parachain's native token would be connected through a bridge to Ethereum.

2. In the runtime of the parachain there would have to be specific conditions met to mint the native token. For example, a message being received from the Ethereum bridge parachain that says “if such and such has happened with ANT on the mainnnet, perform the corresponding action here.”
3. On the Ethereum side, the bridge will monitor the ANT contract.
4. If it is required to mirror token transfers, we would have the bridge relay messages from Ethereum mainnet to Aragon Chain for each token transfer.

Scenario Five: The Aragon Court deployed to Ethereum mainnet wants to allow subscriptions and disputes to be initialized by users on the Aragon Chain.

For Cosmos

A user seeking to initialize a subscription or dispute:

1. Sends a packet containing their instruction relayed from Aragon Chain to the Cosmos Hub via the IBC.
2. A packet containing user instructions is relayed to the Ethereum Peg Zone from the Cosmos Hub via the IBC.
3. Within the Ethereum Peg Zone, the message is decoded into an Ethereum readable format (note that the standard format of a message packet has not yet been documented) and signed by at least ⅔ total signers.
4. Relayers watch the message (now translated into an equivalent transaction) until it is signed by at least ⅔ of total signers, then post the transaction to the appropriate bridge smart contract on the Ethereum mainnet.
5. The bridge smart contract sends a transaction initializing the subscription or dispute on the Ethereum mainnet deployed Aragon Court.

For Polkadot

On Aragon Chain, we would allow for transactions that cause an initialization of subscriptions/disputes.

1. When one of these transactions is sent, it would cause a message to be relayed to the Ethereum parachain.
2. This message would be picked up by the bridge.
3. The bridge would then send a transaction to the Aragon Court.
4. This will allow for a subscription/dispute initialization.

Scenario Six: The Aragon Court deployed to Aragon Chain wants to allow subscriptions and disputes to be initialized by users on Ethereum mainnet (requiring a value transfer to pay for fees).

For Cosmos

1. A transaction with value transfer and instructions to initialize a subscription or dispute is sent to a bridge smart contract present on the Ethereum mainnet.
2. Value is locked in the smart contract, triggering a lockup event.
3. The lockup event is witnessed by witnesses who then parse the information associated with that transaction and build an equivalent unsigned Cosmos transaction.
4. The unsigned Cosmos transaction is then signed and submitted to the Ethereum Peg Zone Bridge.
5. The Ethereum Peg Zone Bridge takes the signed transaction and converts it to a generic claim. It then submits it to the Oracle module.
6. Once at least ⅔ active witnesses have submitted the same claim, the Oracle returns the claim to the Ethereum Peg Zone Bridge with a Successful status.
7. If the claim is returned with a Successful status, the Peg Zone Bridge relays the transaction (converted to a packet) to the Cosmos Hub via the IBC.
8. The Cosmos Hub would then relay the transaction packet to Aragon Chain, also via the IBC.
9. The transaction packet is decoded and the subscription or dispute is initialized on the deployed Aragon Court.

For Polkadot

1. A transaction is sent on the Ethereum mainnet to a contract monitored by the bridge.
2. The contract locks up some value and states “I want to initialize x”.
3. The bridge picks up this transaction and sends a transaction to the Ethereum parachain.
4. This parachain then sends a message to Aragon Chain that initializes a subscription/dispute in the deployed Aragon Court.