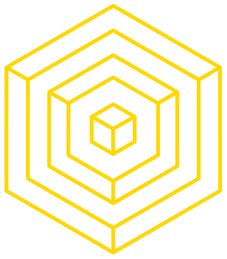
Lecture 11:
Interoperability (Cosmos)

Sankalp Aggarwal



LECTURE OUTLINE

1. WHY INTEROPERABILITY
2. PEACE RELAYS
3. LIGHT CLIENTS PROBLEM
4. INTER-BLOCKCHAIN COMMUNICATION (IBC)

1 WHY INTEROPERABILITY

INTUITION BEHIND INTEROPERABILITY

WHY DOES COSMOS EXIST

- The internet was created to connect independent and isolated networks using a common standard to communicate between clients (IP)
- State of the blockchain ecosystem - very different stacks and **unable to connect to each other**
- **Isolated innovations** - i.e. I want the privacy of ZCash but the turing completeness of EVM
- **No way to send data or assets between blockchains!**
- **Not atomic swaps**
    - Atomic swaps are just as simple as one person having coin A and another having B and they mutually exchange their coins for one another’s

INTEROPERABILITY POSSIBILITY

INCREASE SCALING AND ABILITY

- Imagine we have a lottery
- I'd move my bitcoin to this lottery chain
- I make a bunch of money and I want to move it to some chain running some ZCash like protocol
- I wanna then move my cleanly made bitcoin to a DEX chain and trade it for some DAI (stablecoin)
- And when I wanna make a bet, I'll move it to the Ethereum chain, where it's in a contract
    - The bet might be I breed an orange CryptoKitty within 6 months
    - Once the bet is over I can send the kitty from the CryptoKitty chain to the Ethereum chain in order to win the bet

2 BASIC SIDECHAINS

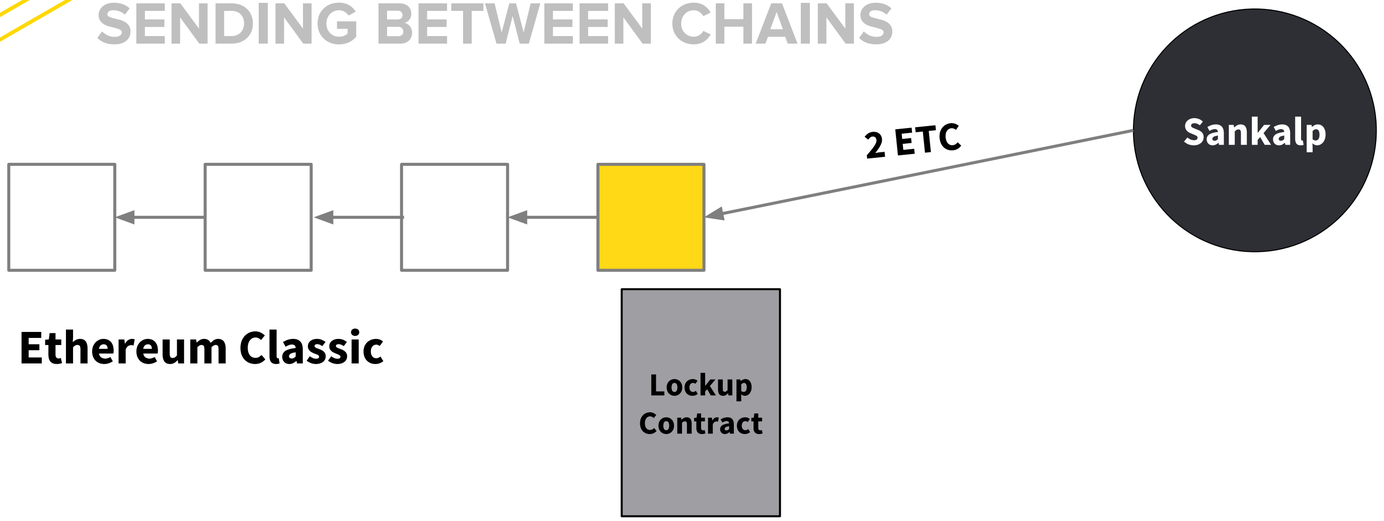
WHAT ARE PEACE RELAYS

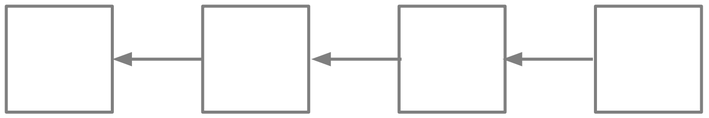
SENDING BETWEEN CHAINS

- This concept of moving tokens over chains like this is call Sidechains.
- Coined by Blockstream
- **Peace relay** was a way to send tokens between Ethereum and Ethereum Classic
  - Simple way of doing multi-chain transfers
- Peace Relay is an example of a side chain protocol
  - Proof of Freeze and Thaw (not Proof of Burn)

PEACE RELAY

SENDING BETWEEN CHAINS



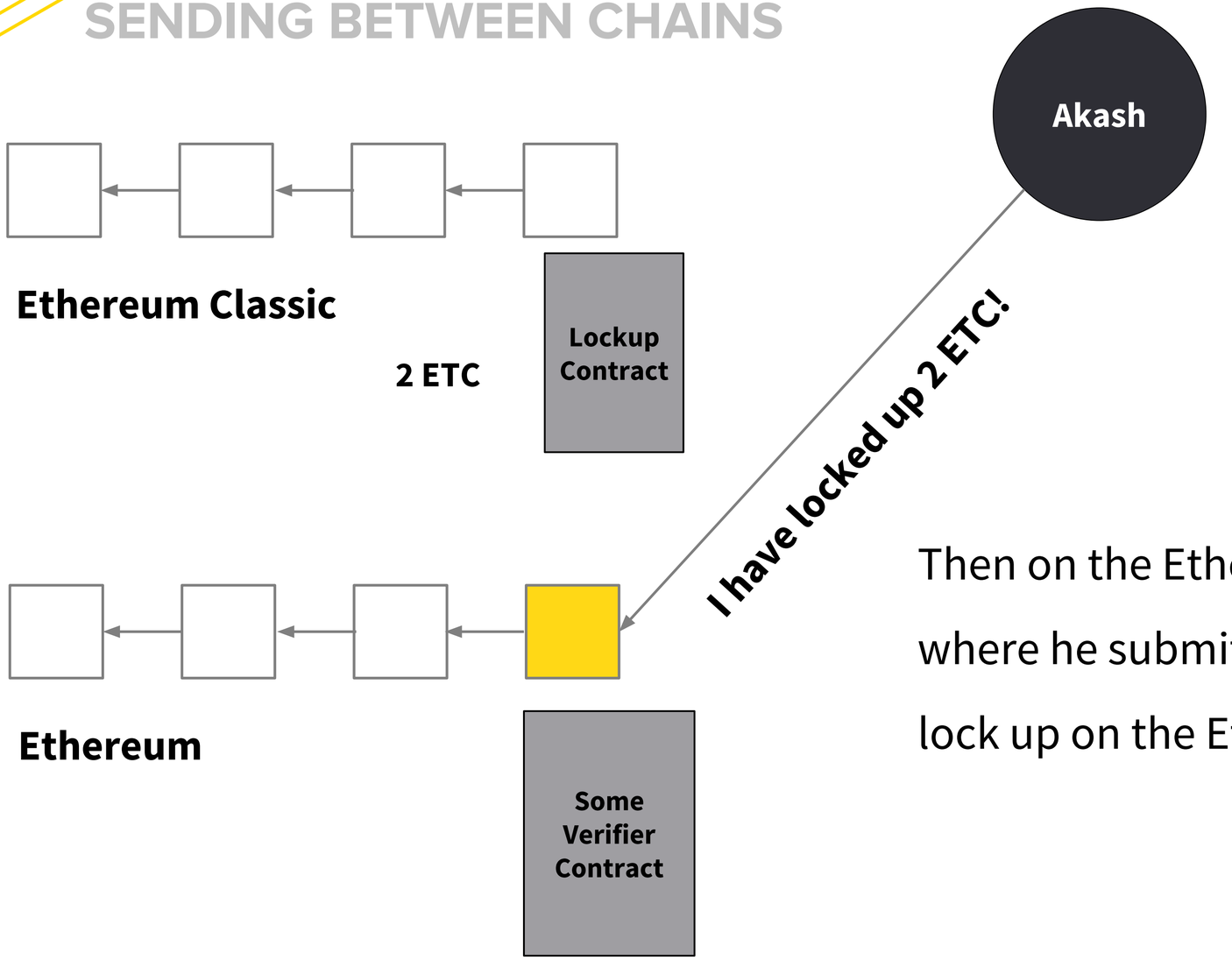


Ethereum

Sankalp takes ETC and locks it up in a contract on the Ethereum Classic chain

PEACE RELAY

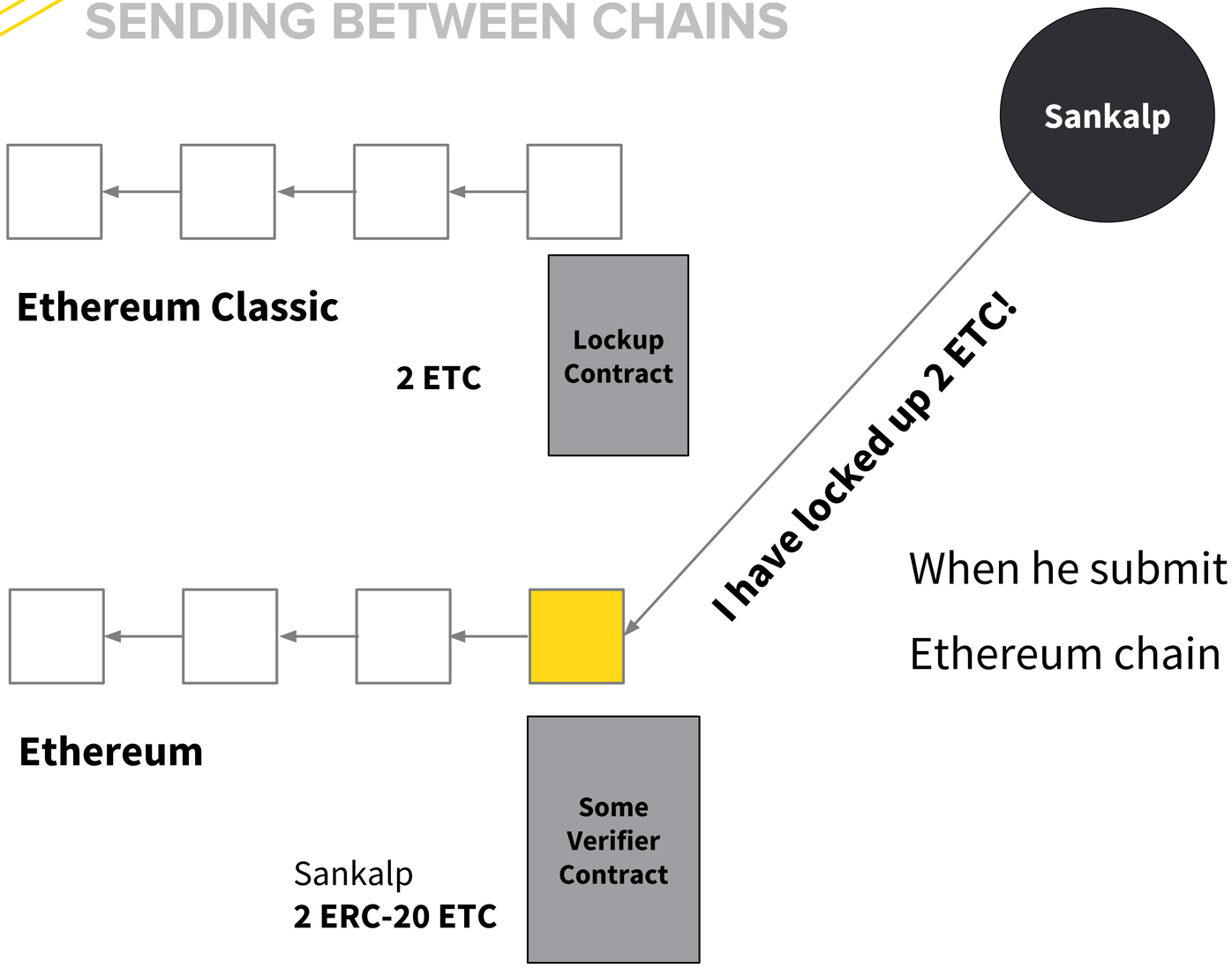
SENDING BETWEEN CHAINS



Then on the Ethereum chain there's a contract where he submits proof **(proof of freeze)** of the lock up on the Ethereum Classic chain

PEACE RELAY

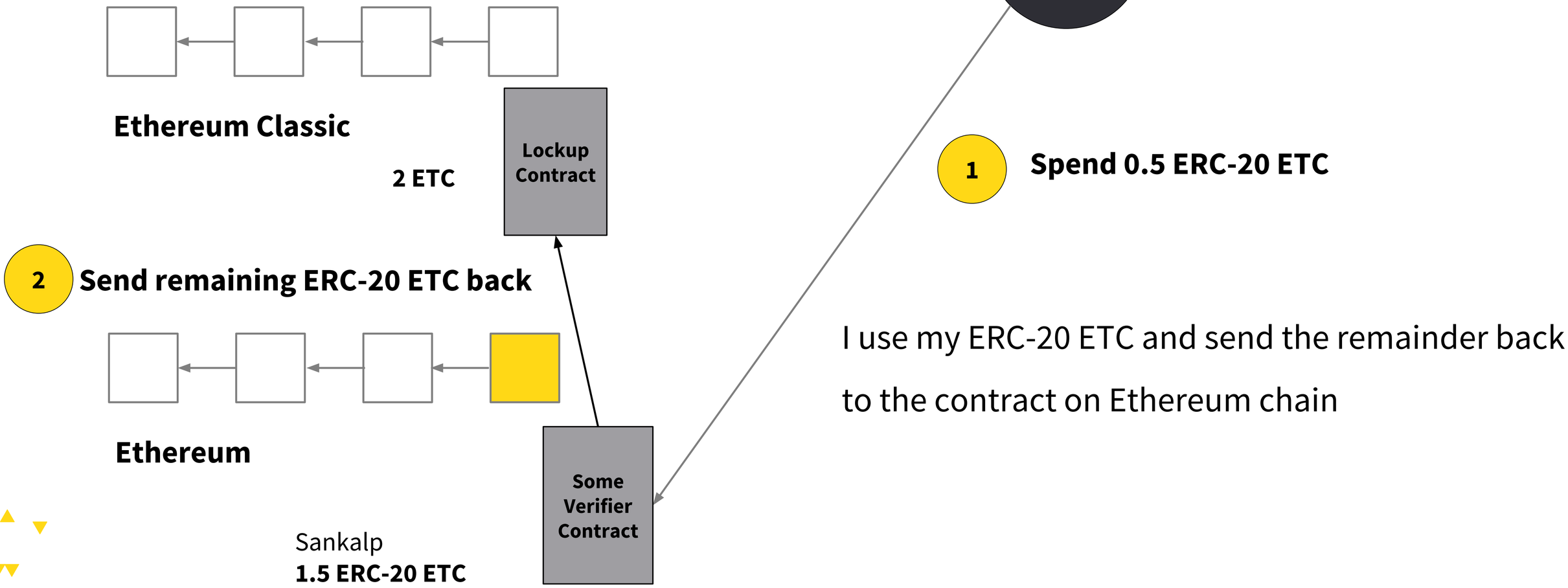
SENDING BETWEEN CHAINS



When he submit proof, I get an ERC-20 on the Ethereum chain that is representative of ETC

PEACE RELAY

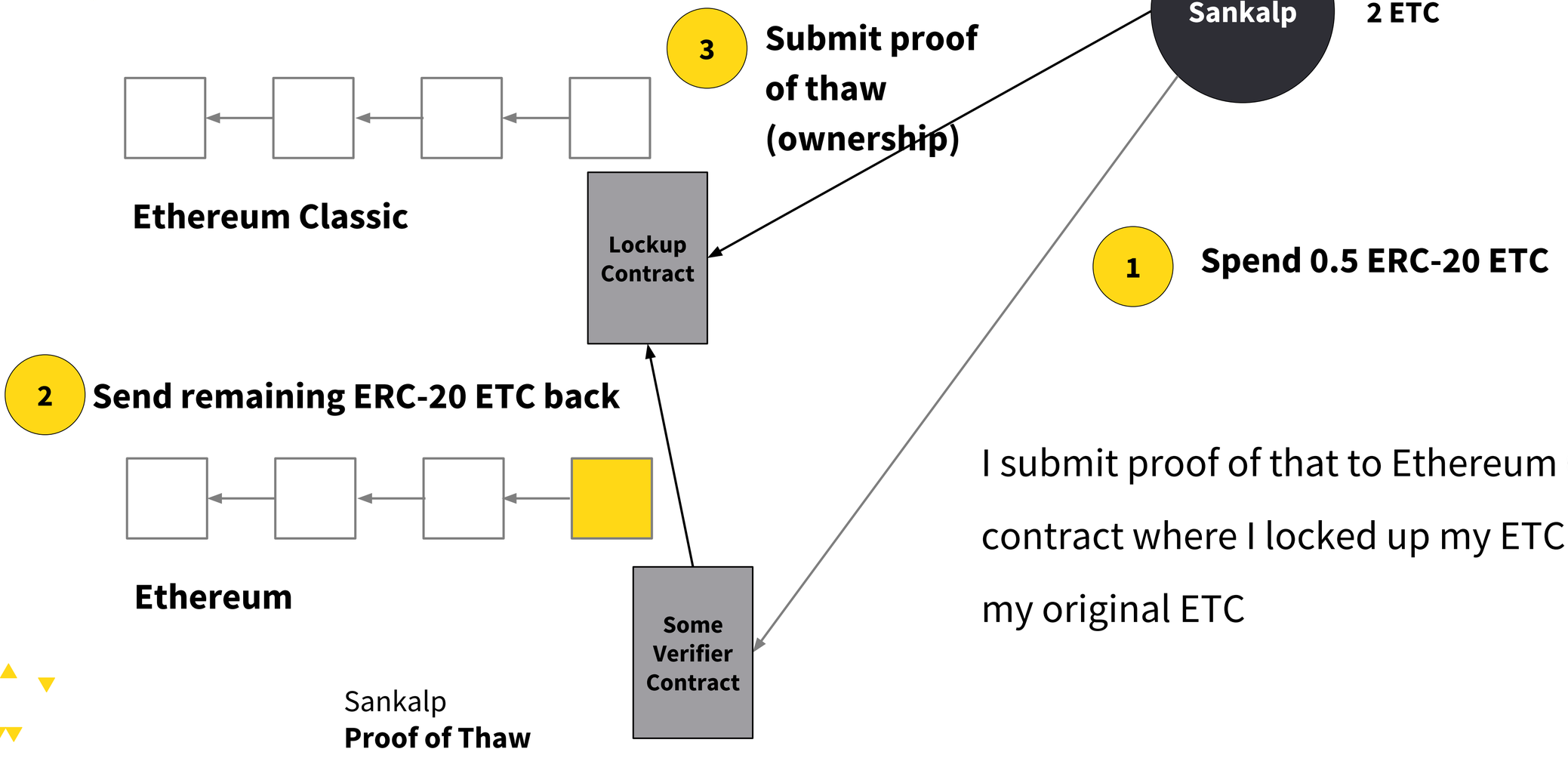
SENDING BETWEEN CHAINS



I use my ERC-20 ETC and send the remainder back to the contract on Ethereum chain

PEACE RELAY

SENDING BETWEEN CHAINS



I submit proof of that to Ethereum Classic contract where I locked up my ETC and I get back my original ETC

PEACE RELAY

ISSUES, WHY WE NEED IBC

- Very EVM specific, and pretty complex, so it doesn't work for general purpose across many chains
- Does not handle edge cases
    - What happens if your tokens don't make it to the other chain, as in your locked up your funds in the contract which requires interaction from the other chain to unlock?
- Doesn't define common standards for how to parse packets.
- **Light clients for proof of work are computationally difficult to implement on Ethereum**
    - This is how I do the proof that I locked up the tokens

2 LIGHT CLIENTS

PROOF OF WORK LIGHT CLIENTS

BITCOIN

- Let’s say you can’t run a non-mining full node
  - On Bitcoin receiving a block every 10 minutes and applying the latest UTXOs to the local copy of the chain
- In Bitcoin light clients you need to do a hash for each block header to verify (slow)
- Suppose you have a phone and you turned it off or you don’t do this in the background to conserve energy
  - Around 144 blocks come overnight
  - Now you need to download verify the hash of every single block header, which is expensive
- Due to the high barrier to entry to verify your own transactions, users will usually rely on third parties to do this which negates a core advantage of a decentralized protocol in the first place

PROOF OF WORK LIGHT CLIENTS

ISSUES WITH PEACE RELAY

- **Light clients for proof of work are computationally difficult to implement on Ethereum**
  - Truebit (verifiable off-chain computation) was created in efforts to try and do a verifiable Dogecoin light client on Ethereum
- We need a consensus mechanism designed for light client efficiency
  - **TENDERMINT!**

PROOF OF WORK LIGHT CLIENTS

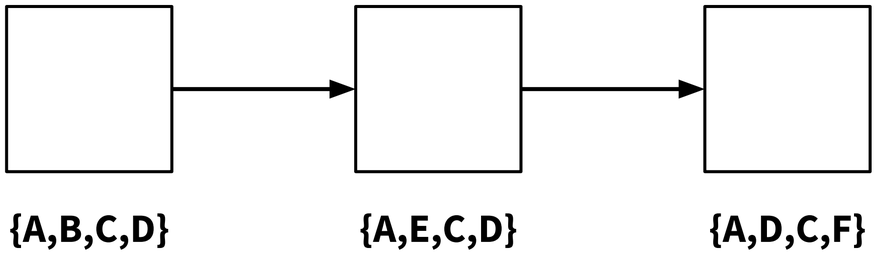
ETHEREUM

- Recall Ethereum is storing the entire state in a Patricia Tree
  - Allowing the construction of Merkle proofs, which trace any key-value pair in the store with only a few KB
  - This root hash is stored in the block header
- Given a proper block header, a client can verify any key-value pair in the corresponding block
- Thus, an Ethereum light client just has to synchronize and verify all headers in the blockchain
- Then a client can query any key in the current state
- Furthermore it is able to verify the result of the query without having to run a single transaction or trust any node

TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- In Tendermint you have a trusted validator set that's checked for each block
- You only need to download the headers for the blocks that have at least 1/3 difference in the validator set
- As long as less than ⅔ of validators are only malicious then a client can trust any header that is signed by a known validator set
    - **So you don't have to download all the blocks headers, you can sync much faster**



TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- On Friday evening your phone has a snapshot of the currently trusted set of validators.
- You decide to turn it off for the night
- On Saturday morning your phone connects again
- In this stage, your client only has to download the most recent header, which is about 2KB
- If the validator set did not change overnight, or if it did change, but only by < ⅓ of the total from the previous set, your phone will immediately trust the new set

TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- At this point, your phone can verify any state from the blockchain as described above (proof of state and transactions) and can interact with the blockchain in a fully secure manner.
- Very notably, you just skipped over Ethereum's requirement of downloading and validating 30,000 intervening block headers.
- If the validator set did change considerably in the time a client was offline, it still does not have to download all headers
    - Instead, it can find a path of intermediate headers
- The requirement for this path for any two consecutive headers is that the validator set changes by < ⅓

TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- At 06:00 the validator set consists of validators A, B, C, D
- At 09:00 the validator set consists of validators A, B, C, E.
- At 12:00 the validator set consists of validators A, B, E, F.
- A light-client (phone) trusts the validator set at 06:00 (A, B, C, D) (for example by manually checking that it is correct).
- It now goes offline and only comes back at 12:00.
- It will receive the new validator set (A, B, E, F).
- Here 50% changed which is an unsafe change, because the light-client cannot verify that >2/3 of the validators it trust (from the set at 06:00) signed off on that change.
    - This means that it might have received a byzantine packet.

TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- The phone however can ask for an earlier header (from 09:00)
- Here the validator set (A, B, C, E) has changed by < ⅓
- This is a trusted change in the validator set and the phone now has an updated trusted validator set(A, B, C, E).
- From here it can update to the most recent validator set (A, B, E, F), because between (A, B, C, E) and (A, B, E, F) is < ⅓ change
- Overall, the change in the validator set can be much larger
- This shows that even if there are large changes in the validator set, a light client still only has to do a minimal amount of work in order to securely update its validator set

TENDERMINT LIGHT CLIENTS

LIGHT CLIENT ISSUE

- Checking a **validator hash** is almost as easy as checking a multisig
- Makes on chain light clients much more feasible than on chain proof of work light clients, which made **Peace Relay** light clients
- If we can build these two chains and have tendermint light clients for each other, we can prove anything about the state of the other chain
    - This is a two way light client system where we can send headers back and forth
    - **Contracts/modules on-chain can keep track of the most recent validator set hash**
        - **There is a mapping between chain-id and the most recent validator hash**
    - This is an **IBC connection** (Inter-Blockchain Communication)

RELAYS WITH OUR NEW LIGHT CLIENTS

LIGHT CLIENT ISSUES

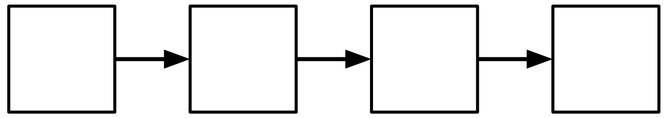
- You can prove the anything about the other state about the chain now with this
- Good enough for transfers, something is locked, etc.
- However this should be platform agnostic
- The light client protocol needs to able to verify (arbitrary depth) merkle proofs and the data that’s contained within them
- No connection between these blockchains though, they are virtualized internet routers

3

IBC

IBC PROTOCOL

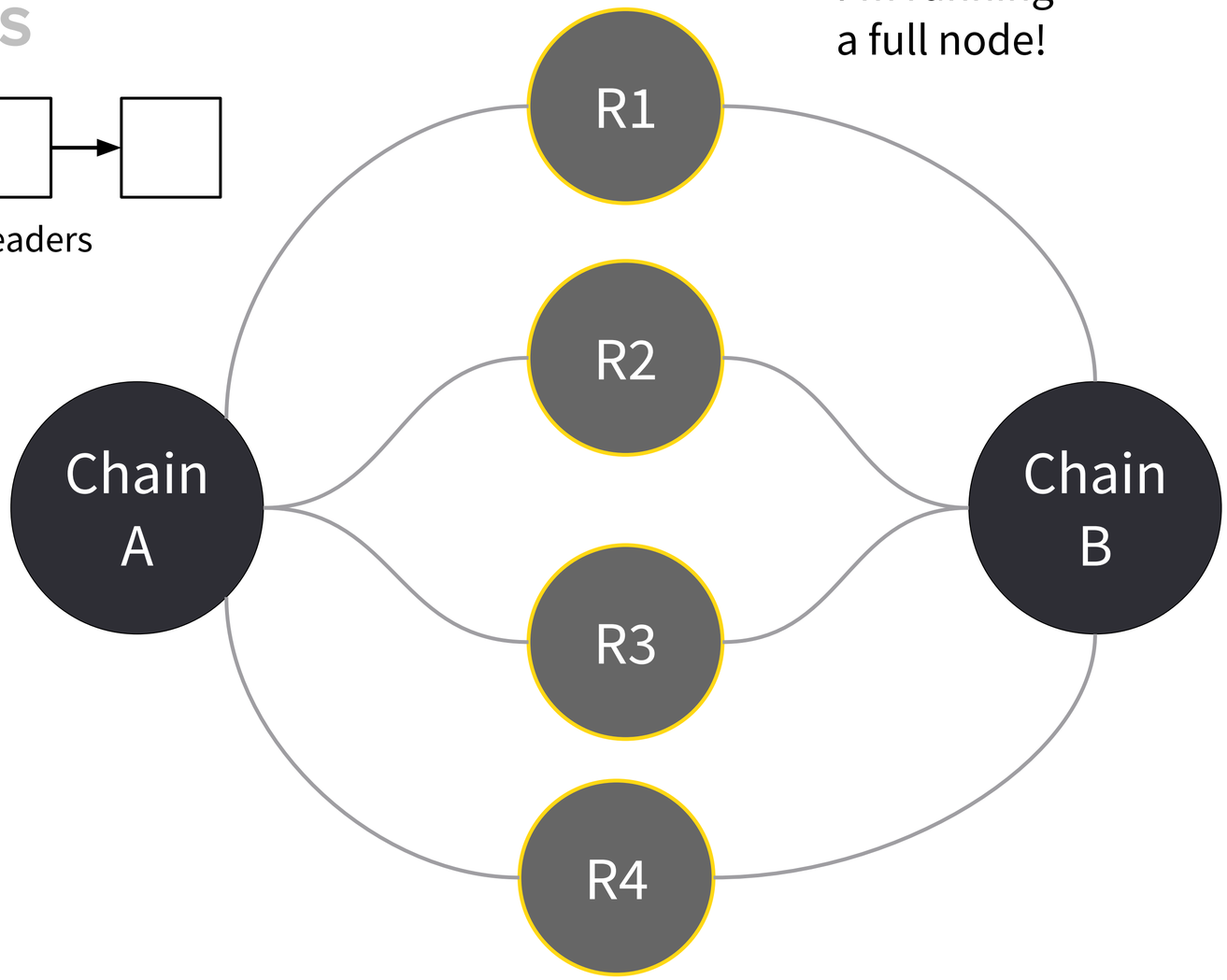
RELAYERS



Chain A Block Headers

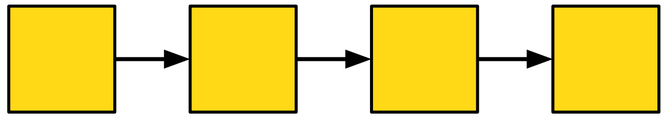
Assume only one relayer running a full node

I'm running a full node!



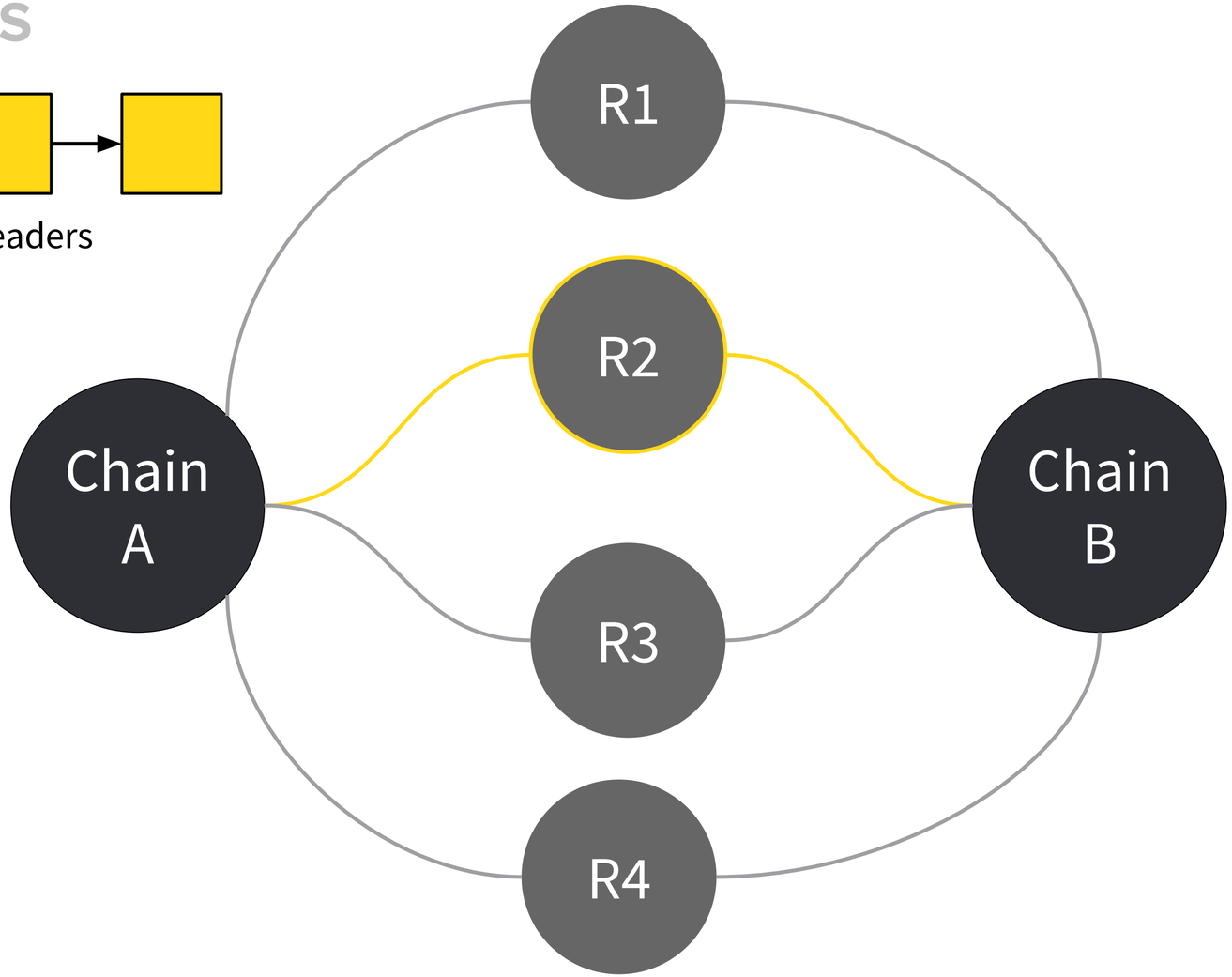
IBC PROTOCOL

RELAYERS



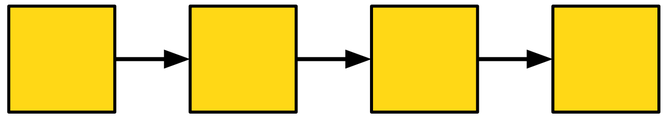
Chain A Block Headers

Some relayer sees every header, he submits those headers to chain B.

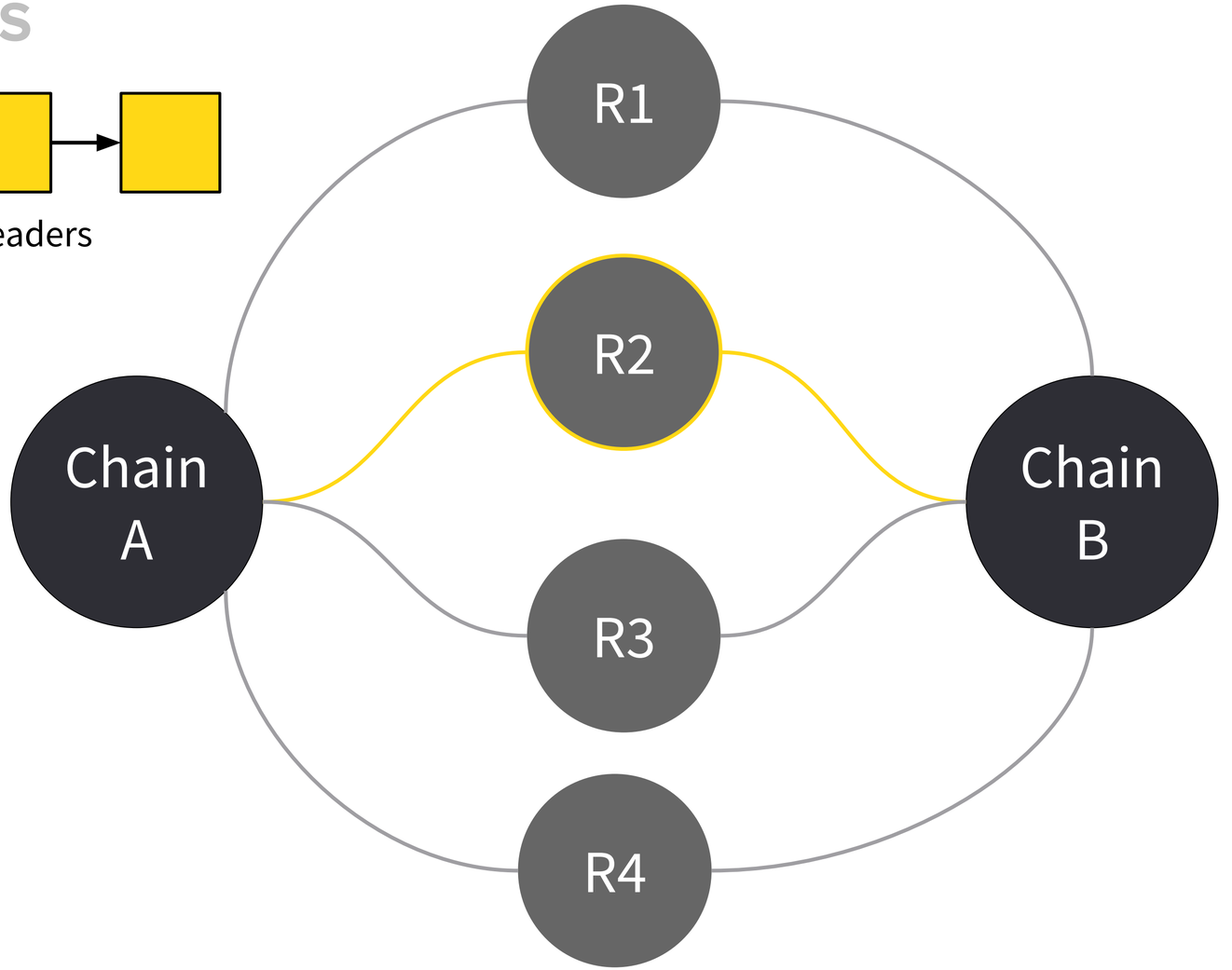


IBC PROTOCOL

RELAYERS



Chain A Block Headers



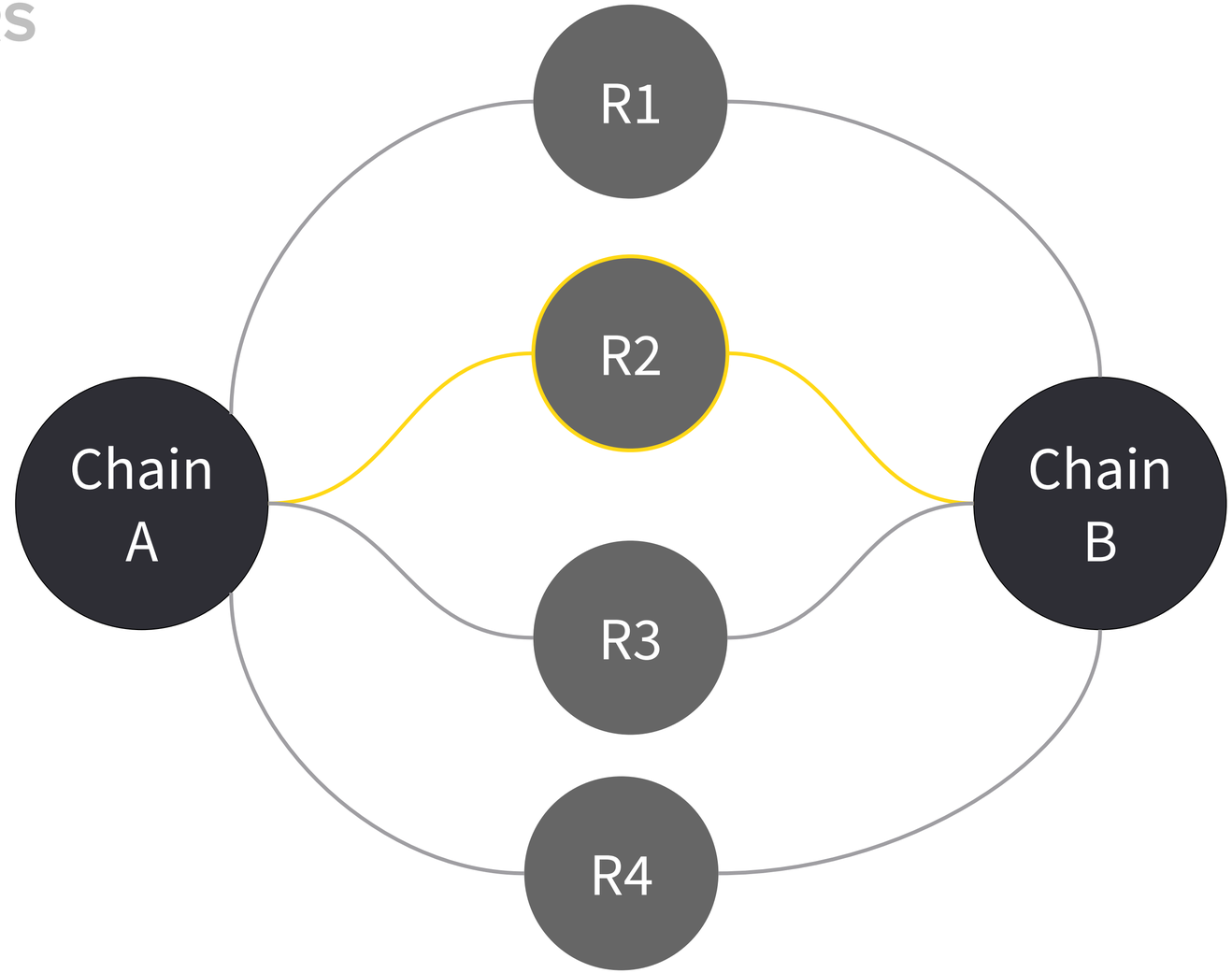
B can verify A's headers now

IBC PROTOCOL

RELAYERS

This is the base layer. We want ways to guarantee reliable transfer over IBC. We need **channels**!

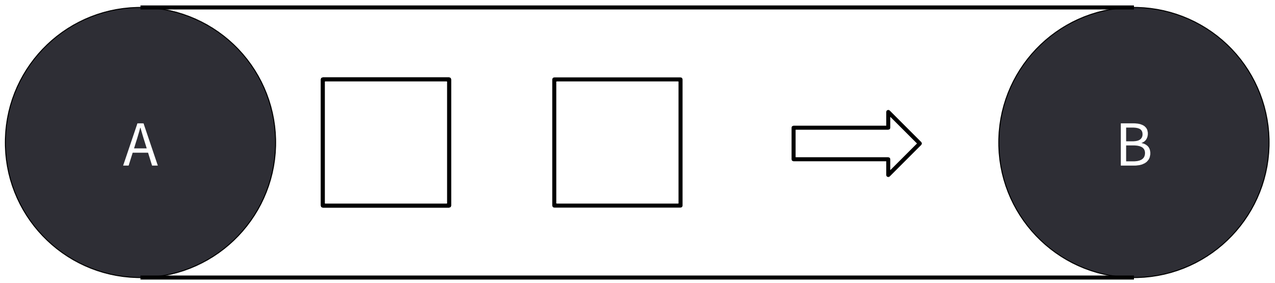
What we just described is an abstraction of a **connection**



IBC PROTOCOL

OVERVIEW

Connections are two way block header transmission buffers where you send block headers between chains

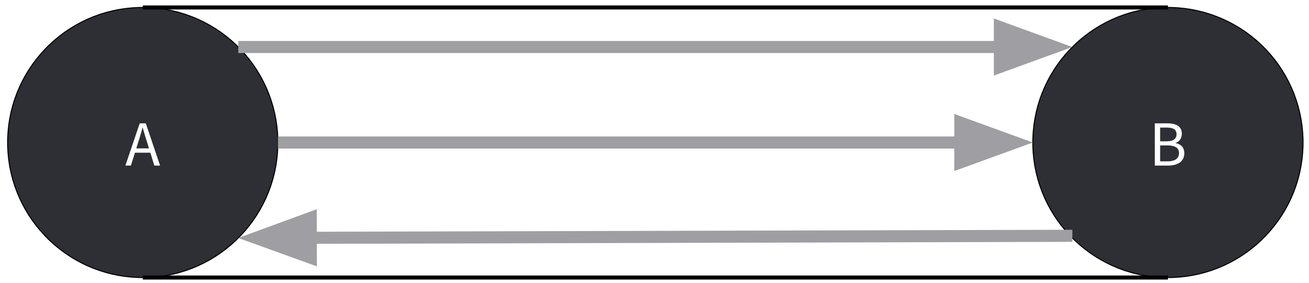


IBC PROTOCOL

CHANNELS, CONNECTIONS

Proof Carrying Packet: Contains some data that's in the sender's state with some proof that it is in the sender's chain state
i.e. Address X has 10 ETH on it) transfers for a specific application/module

You can have multiple **channels** inside an IBC **connection**



You can then create a **channel** within the **connection** which is used **for a one way "proof carrying" packet**

```
{
   transfer data
   {
      addr_from:
      addr_to:
      amnt:
      denom:
   }
      metadata
      proof
}
```

IBC PROTOCOL

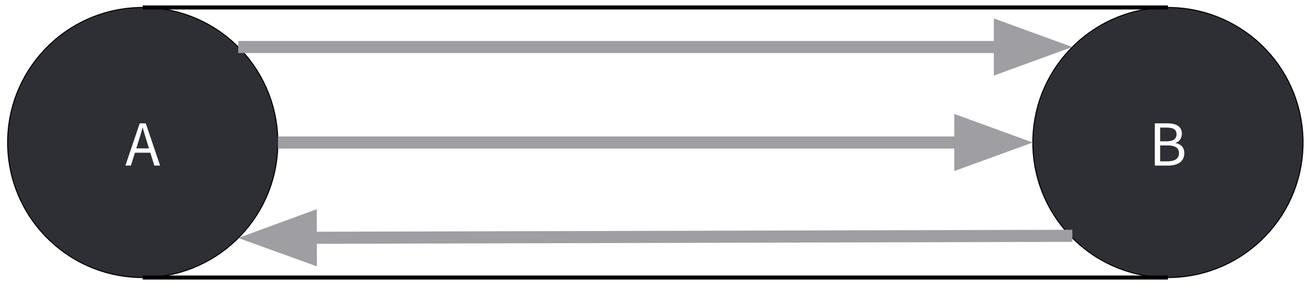
CHANNELS, CONNECTIONS

Proof Carrying Packet: Contains some data that's in the sender's state with some proof that it is in the sender's chain state
i.e. Address X has 10 ETH on it) transfers for a specific application/module

Transfer Data

Metadata

Proof




```
{
   transfer data
   {
      addr_from:
      addr_to:
      amnt:
      denom:
   }
      metadata
      proof
}
```

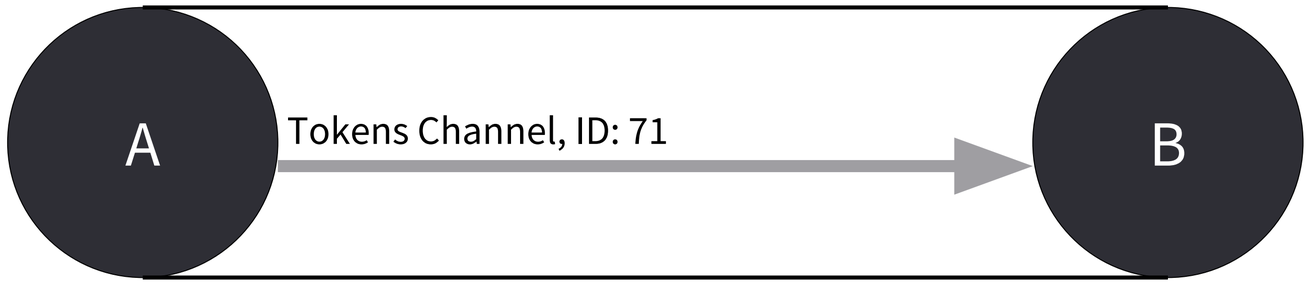
IBC PROTOCOL

CHANNELS, CONNECTIONS

Similar to Network Ports!

Packets over 1 channel get sent sequentially.

Can use multiple channels for parallelization



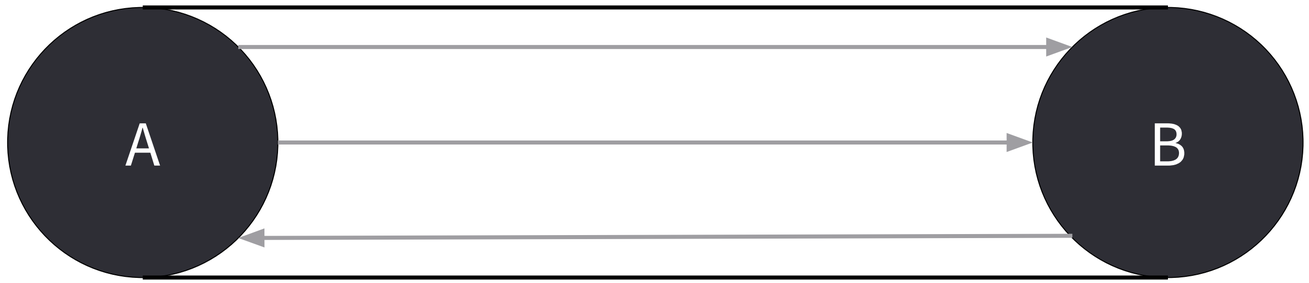
IBC PROTOCOL

CHANNELS, CONNECTIONS

Each of these channels can have different operations modes

Case 1: UDP Style
Just sending data over the channel and directly.

Case 2: **TCP Style**



IBC PROTOCOL

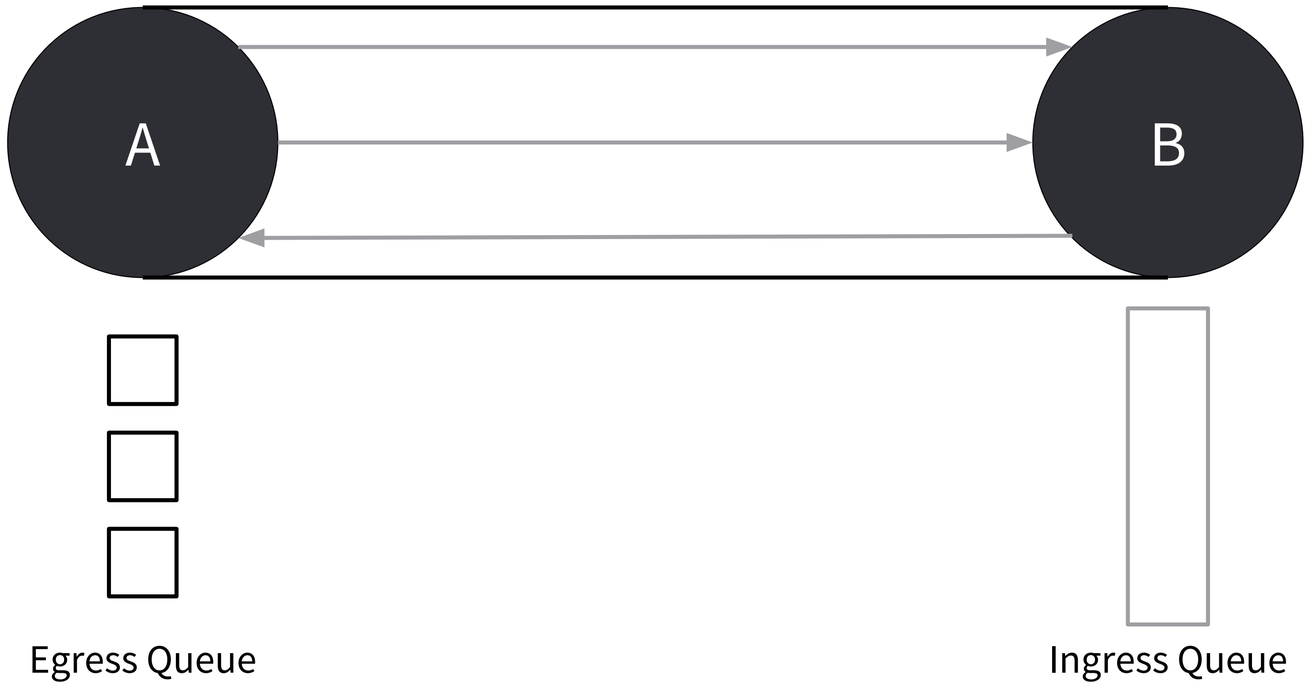
CHANNELS

TCP Style

Egress Queue = Outgoing Queue

Ingress Queue = Incoming Queue

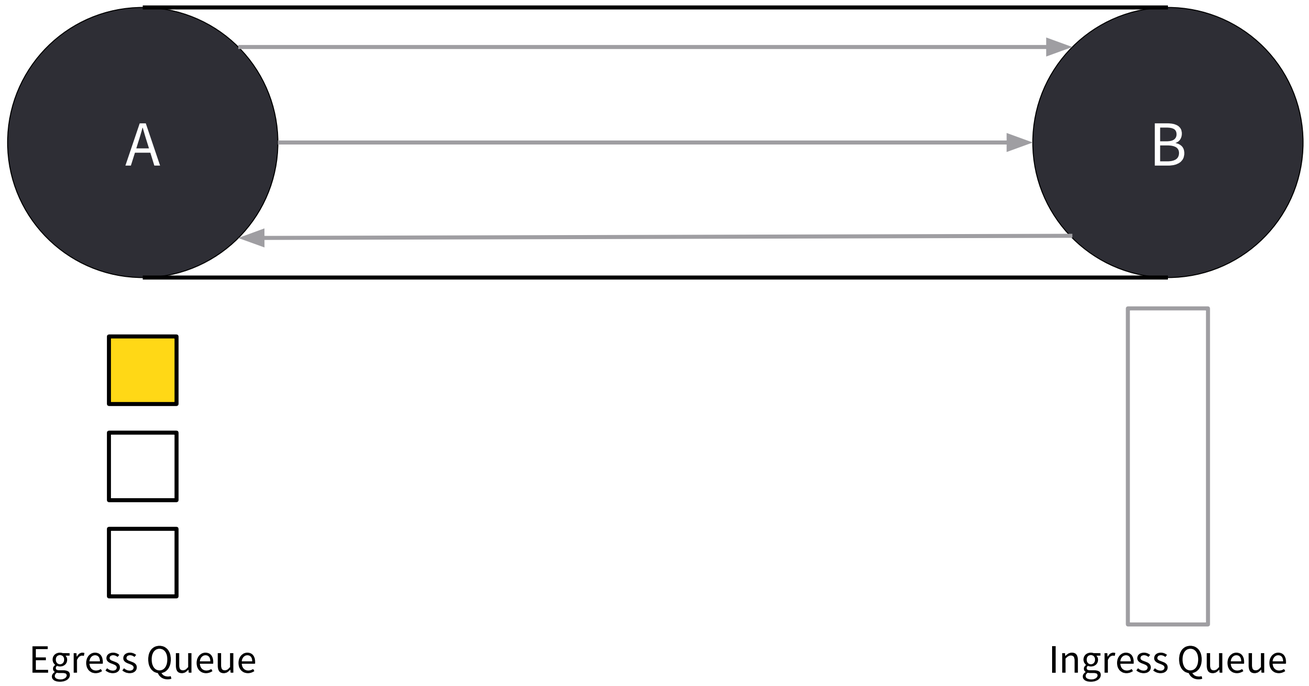
Chain A has a queue of packets (submitted by users) it's going to send to Chain B



IBC PROTOCOL

CHANNEL EXAMPLE

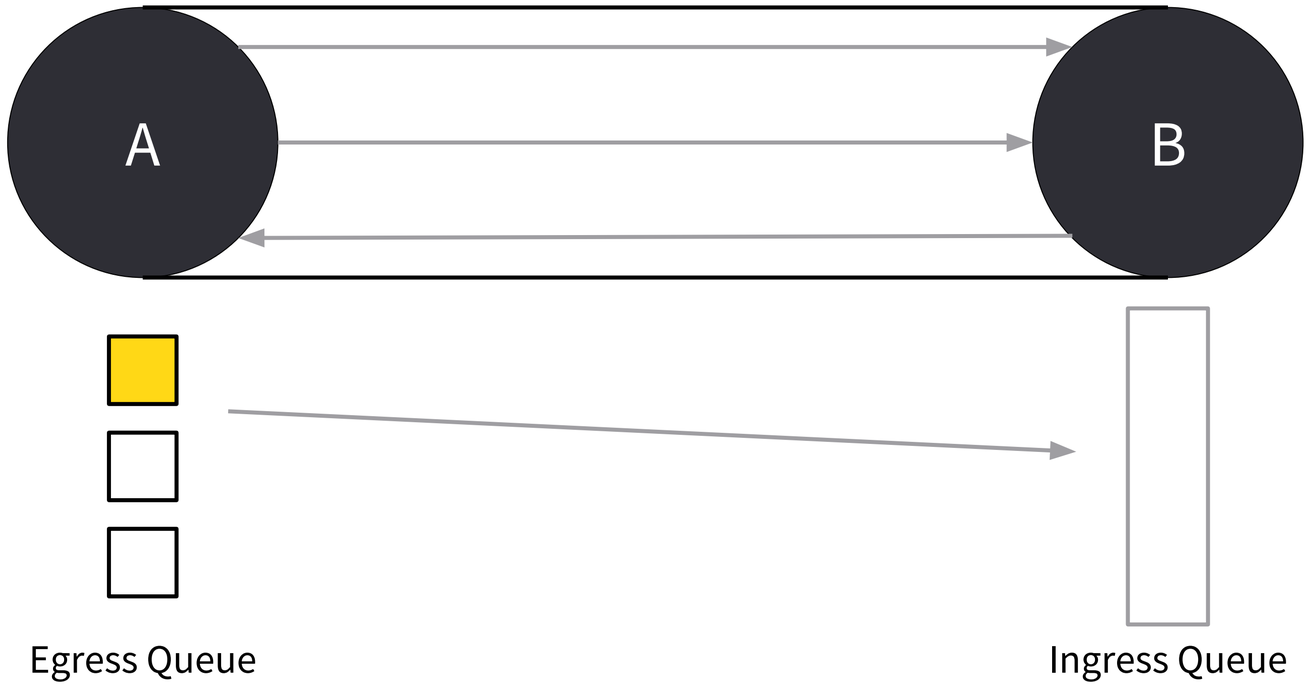
User puts packet in egress queue of A.



IBC PROTOCOL

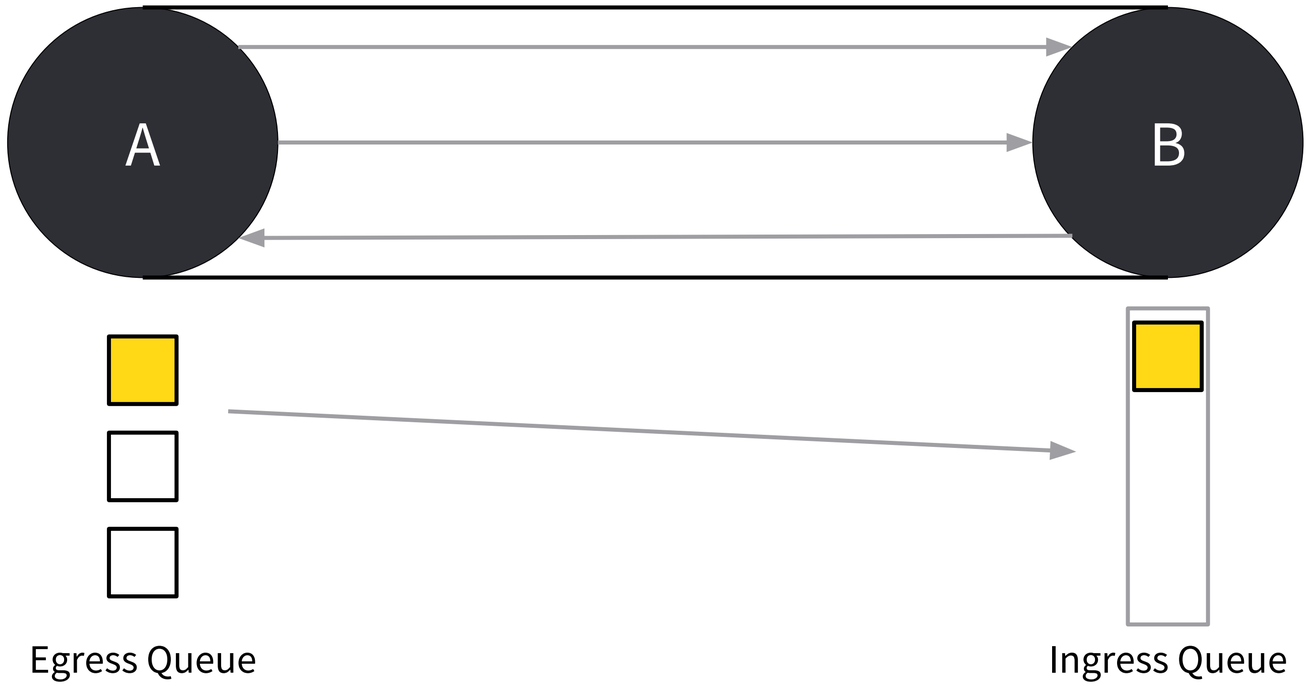
CHANNEL EXAMPLE

Any relayer submits a proof to B that A has that packet in its queue.

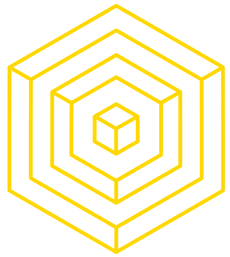


IBC PROTOCOL

CHANNEL EXAMPLE



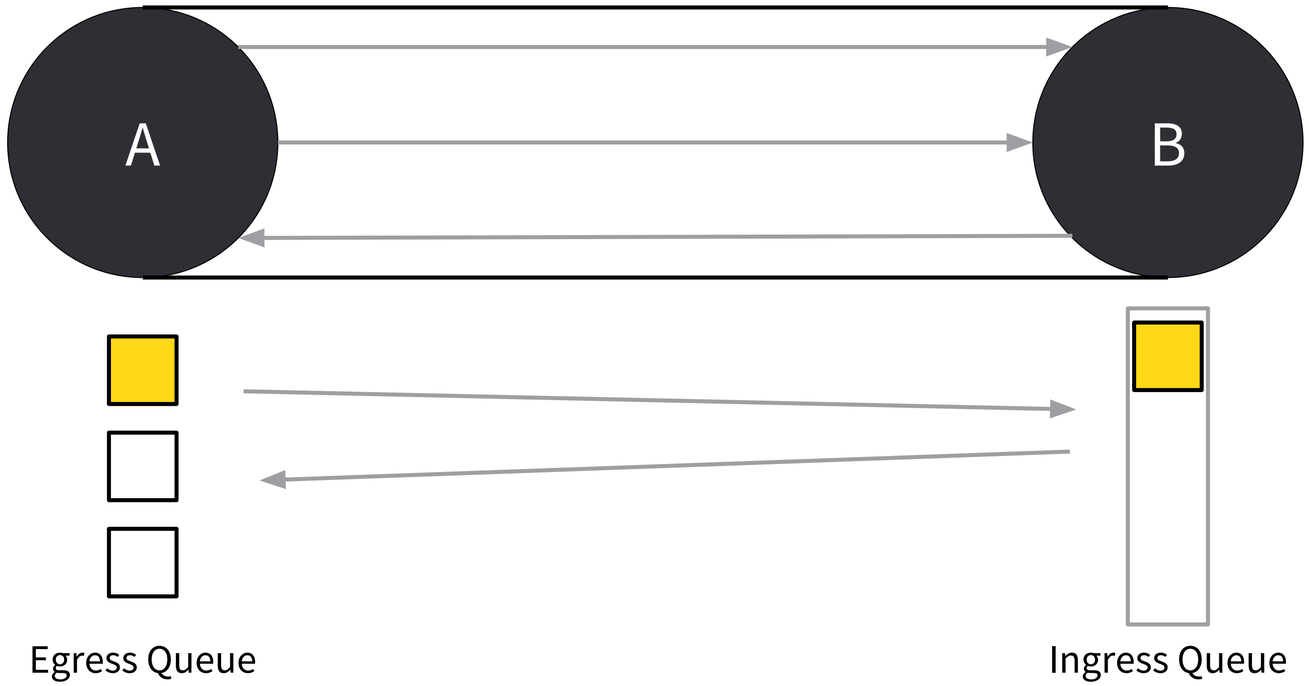
At some point, relayer must insert packet into ingress of Chain B



IBC PROTOCOL

CHANNEL EXAMPLE

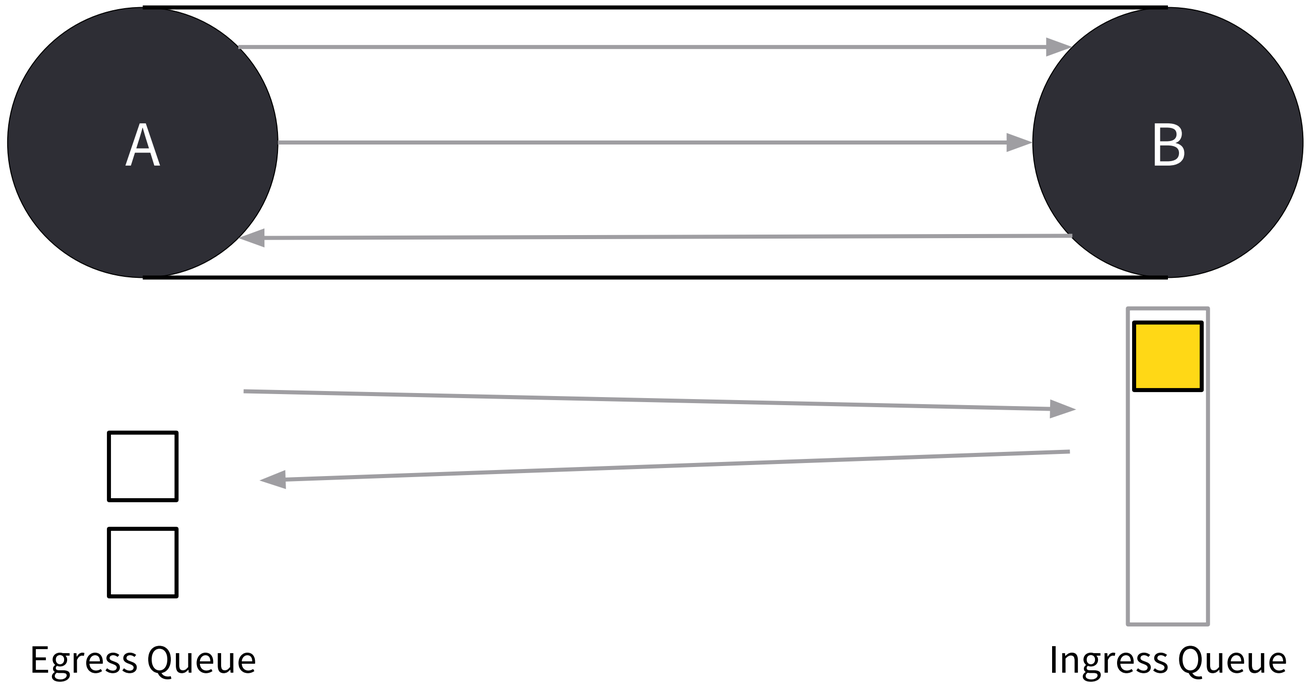
Sending chain needs to know that the receiving chain has received the packet, so B sends a proof to A via relayers

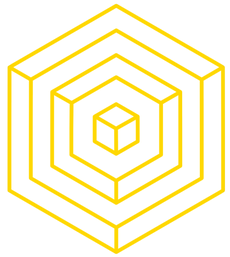


IBC PROTOCOL

CHANNEL EXAMPLE

When it receives this proof, it pops it out of the egress queue of A.

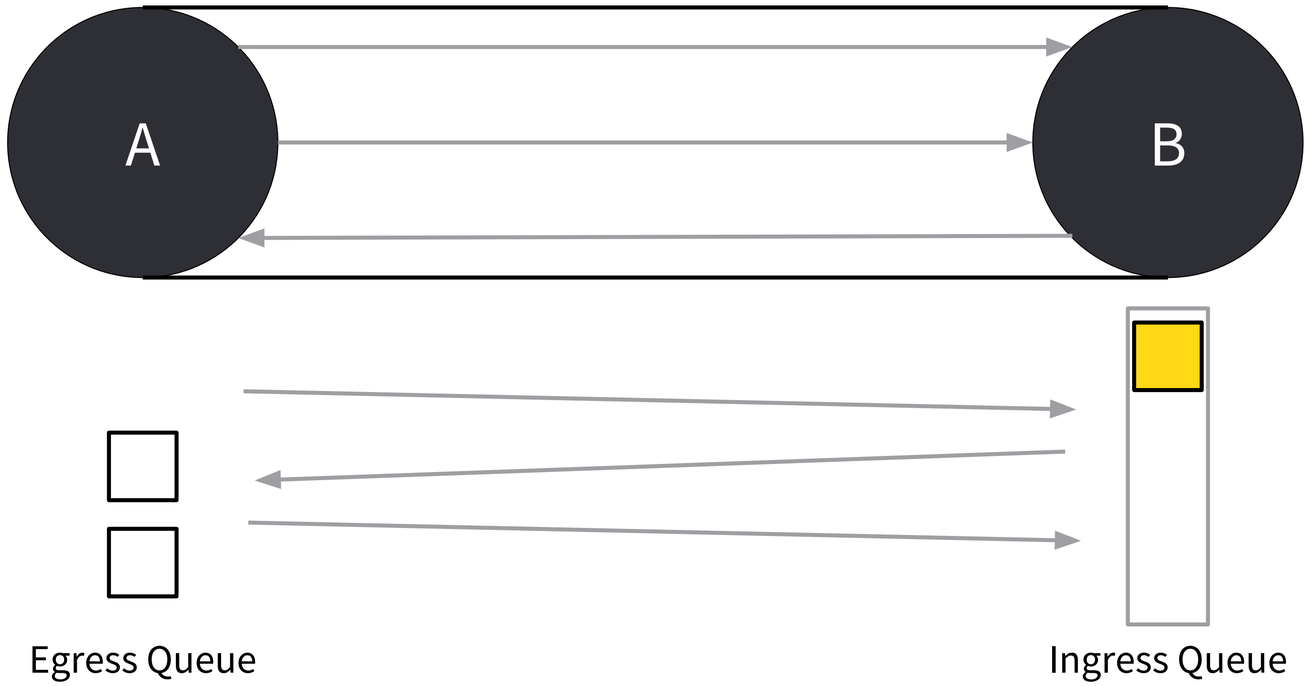




IBC PROTOCOL

CHANNEL EXAMPLE

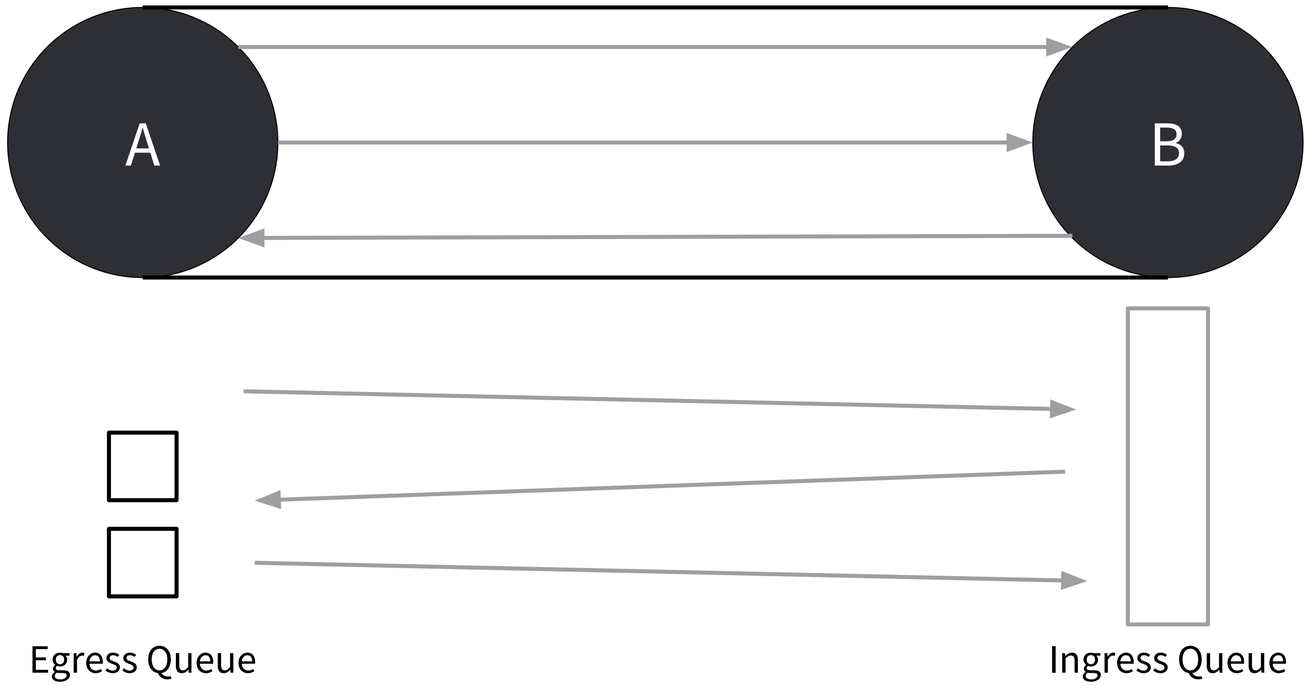
Once popped from A's queue, a relayer transmits this proof to chain B

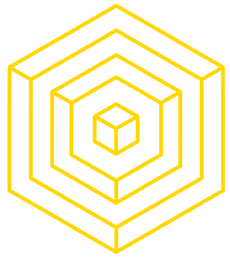


IBC PROTOCOL

CHANNEL EXAMPLE

B verifies that A popped from its egress queue and then B pops its queue.

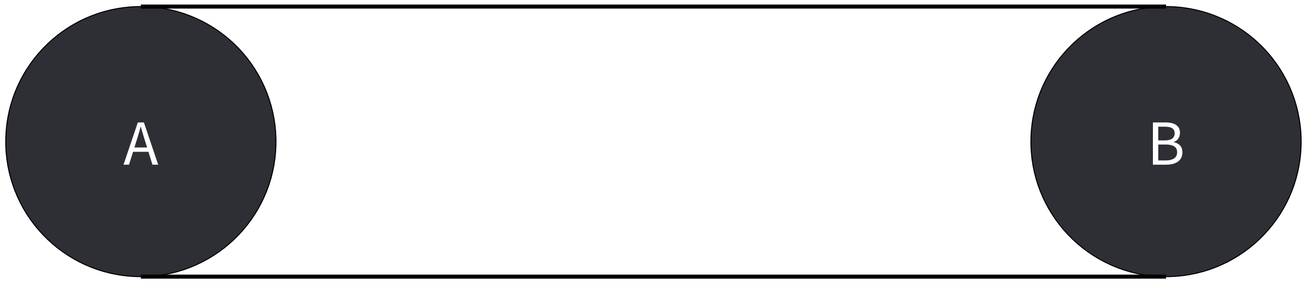




IBC PROTOCOL

APPLICATION LAYER

On top of TCP/IP you have other protocols like HTTP, SMTP, FTP on these TCP ports for a server



What's the analogy for the IBC?

IBC PROTOCOL

PACKET STRUCTURE

- Different type of cross chain things can have **different types of packets**
    - (i.e. token transfer, non-fungible asset transfers, Smart contract calls, etc)
- **Token transfers over TCP/IBC** - Example of a defined protocol:

```
{
  transfer data
  {
    addr_from:
    addr_to:
    amnt:
    denom:
  }
  metadata
  proof
}
```

IBC PROTOCOL

OVERVIEW

- Can have these cross chains steps happen synchronously or asynchronously on a per chain basis
- You can also define rules for scenarios where packets timeout or other handling
- **Timeouts?**
  - Waiting for an ack back but never receive one. Need to set a max time that waiting for. If doesn't go through by timeout, execute recovery function (i.e. refund the money)

IBC PROTOCOL

OVERVIEW

- What to measure timeouts in?
  - One chain itself doesn’t have a good measure of time due to distributed nature.
  - Need something to measure timeout in… Use the block headers
  - Using you own blocks can lead to double spends. So use the other chain as the timeout counter.
- Example:
  - When a packet on A reaches the front of the queue, it is block 11 on chain B.
  - Sets the timeout for 5 blocks.
  - If the packet gets successfully relayed someone can submit proof of it to A..
  - However, if it doesn’t, someone can submit a proof that by block 18, it still hasn’t gotten into chain B

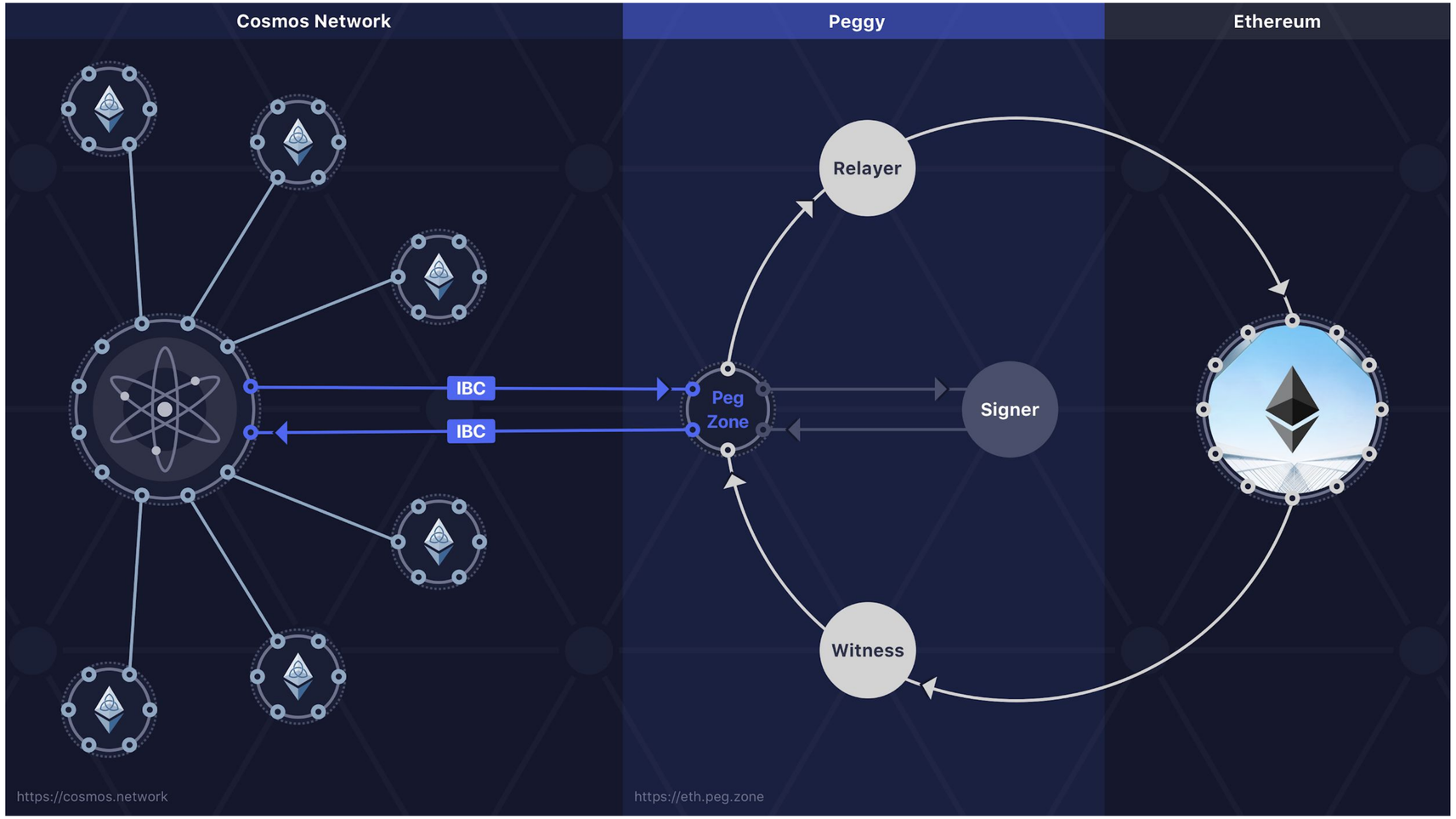
IBC PROTOCOL

HUBS AND PEGS

- A Hub = ISP in this IBC protocol
- It connects the blockchains together than run the IBC protocol, instead of doing P2P
- Hub is better for efficiency and it also maintains token invariance (two chains can’t double spend each other)
- Provides other services too. Cloud Hosting (Shared security model), Mediation (Plasma), DNS, etc
- Peggy = A way of connecting non-tendermint chains
  - Finality gadget over proof of work chains

BONUS: PEGGY

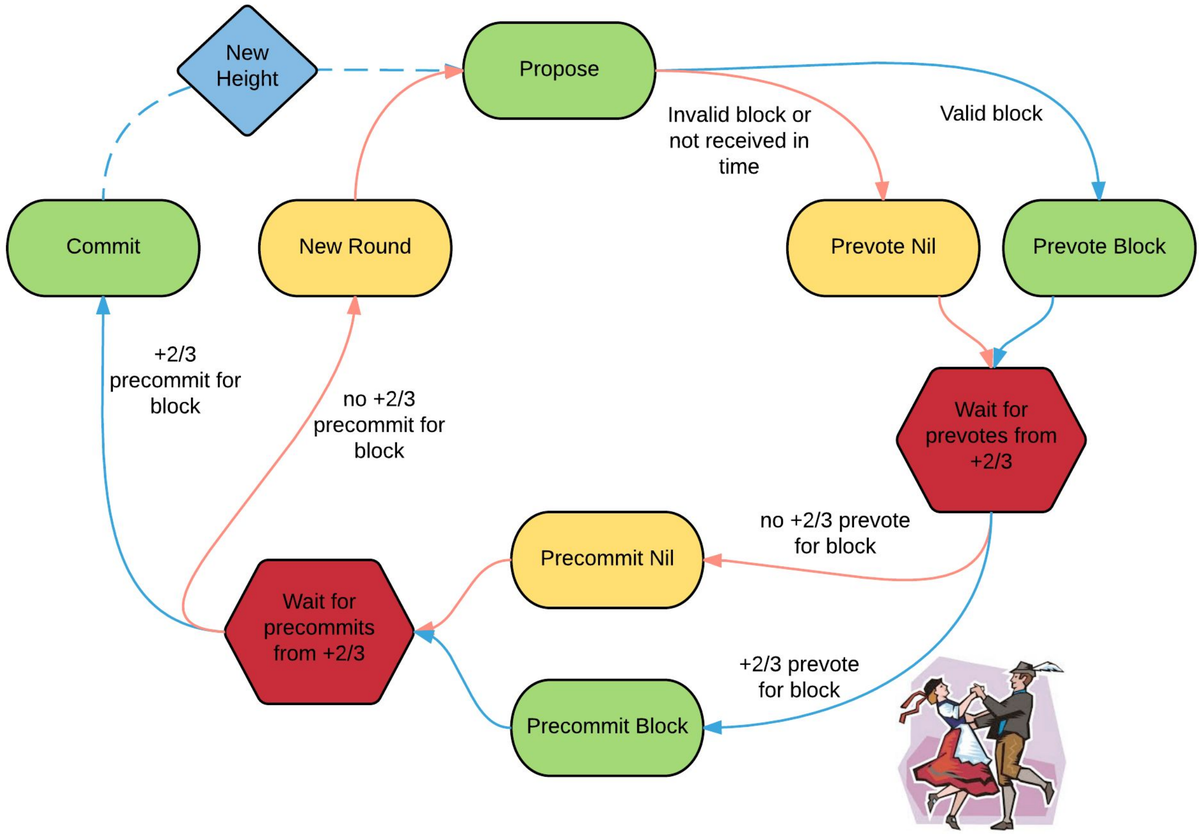
CONSENSUS PROTOCOLS



BONUS: TENDERMINT

CONSENSUS PROTOCOLS

- **TEEENNDDEEERRRRMIIIINTTTTTT**



GOOD LUCK ON FINALS

We hoped you learned something in this class.

BLOCKCHAIN AT BERKELEY