Rearchitecting the **Blockchain Ecosystem**

Billy Rennekamp

Developer Relations - Tendermint Inc

Evolution of Blockchain Development

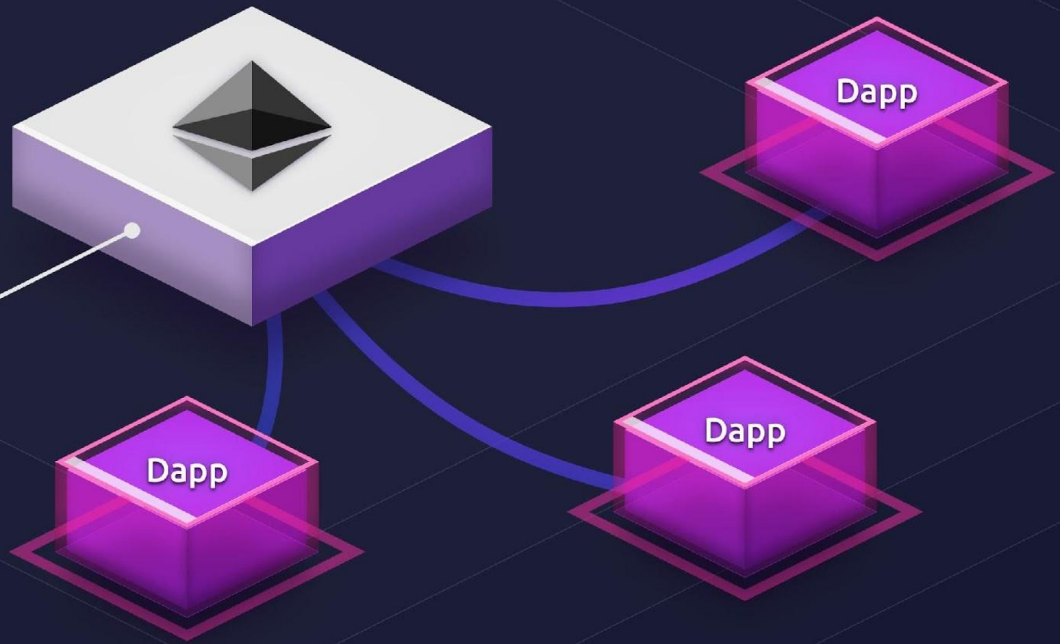
Gen 1: Forked Bitcoin Codebase

Payments
UTXO
Fee by tx size
Bitcoin script
Proof of work

App Developer's
"Zone of Control"

Gen 2: Ethereum Smart Contracts

Ether fee coin
Account model
Patricia tries
EVM
Proof of work



App Developer's
"Zone of Control"

Gen 3: Cosmos SDK

Least authority
Modular
Extensible
Golang
Proof of stake

App Developer's
"Zone of Control"

Application Specific: Specialization

Application Specific: Self Sovereign

Core Module: Tendermint

- Mature, state-of-the-art BFT consensus & networking protocol
- Vertical scaling solution for your base-layer chain
- Instant confirmation: 1 block finality
- 170+ validators on latest testnet
- Formally proven

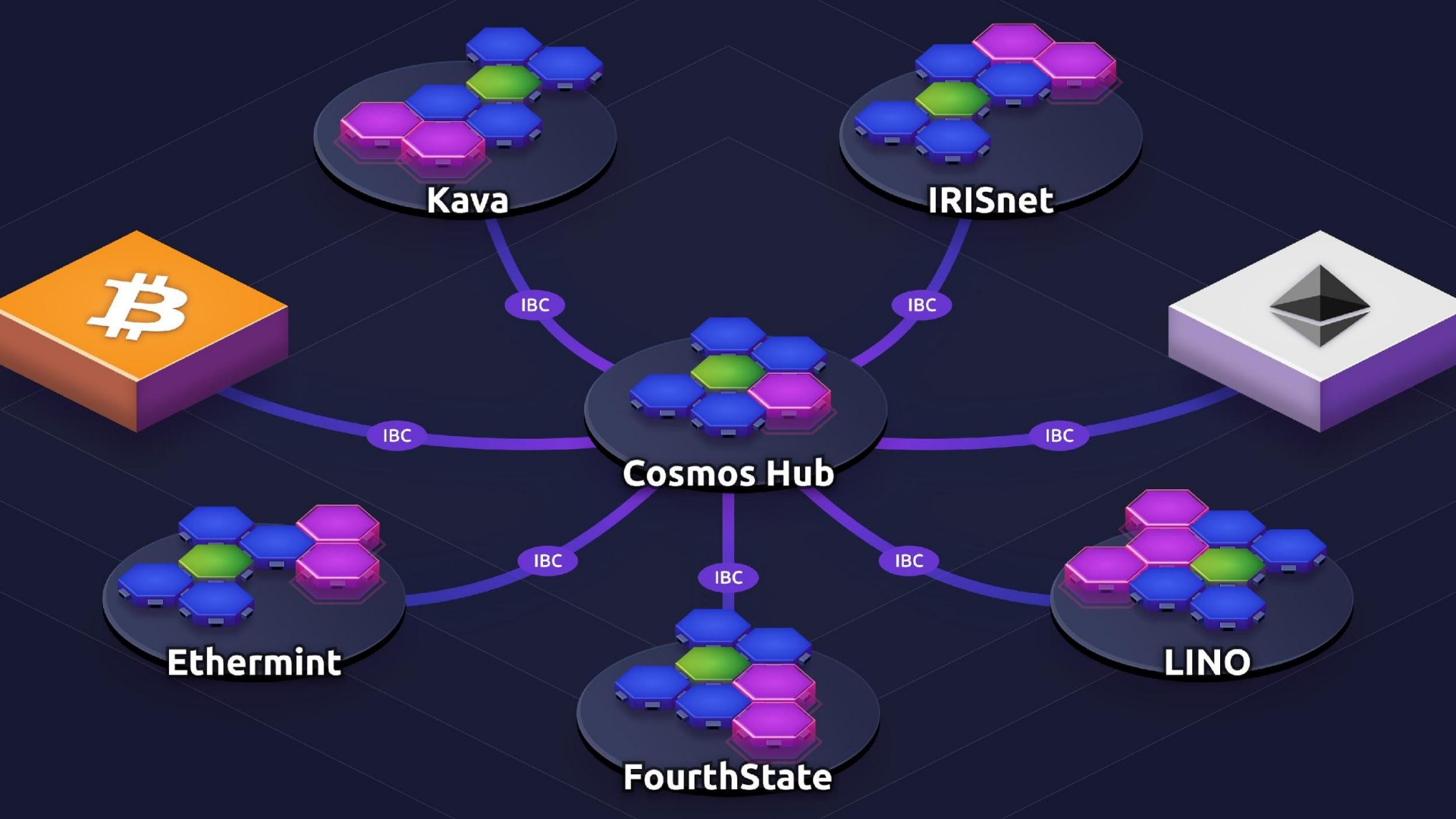
Core Module: IBC

- The Inter-Blockchain Communication (IBC) Protocol enables the Internet of Blockchains

- Move coins & data between different blockchains

- Basis for horizontal scaling and interoperability

- Analogous to TCP/IP for the Internet


Kava
IRISnet
IBC
IBC
IBC
IBC
Cosmos Hub
IBC
IBC
IBC
Ethermint
FourthState
LINO

The Cosmos-SDK is a Modular Framework

- Modular = Modules
- Modules ≈ Contracts
- Tightly coupled logic and storage for your app
- Modules might interact with each other / import state from each other
- Modules can be re-used across different chains

Each module has Messages

- Messages are like functions in Solidity
- Messages are included in transactions
- They modify state
- Baseapp receives Messages from Tendermint/ABCI and routes to Modules

Each module has Handlers

- When a module receives a Message, it is handled by the Handler
- Handlers apply logic to a Message before changing state in the Keeper
- Similar to actions in Redux/Vuex

Each module has a Keeper

- The Keeper manages state.
  - Specifies how an applications state changes
  - Similar to reducers/redux in React or mutations/vuex in Vue
- Contains Getters
- Contains Setters
- Can be shared across modules

Msgs → Handlers → Keeper

- Msgs Trigger Handlers
  - Msgs are similar to public/external contract functions in Ethereum
- Handlers call the Keeper
  - Handlers send payloads of information that carry your data to your store
  - Similar to actions from redux/vuex
- Keeper changes state
  - Specify how an applications state changes
  - Similar to reducers/redux or mutations/vuex

NFT Msgs

```
type MsgTransferNFT struct {
    Sender    sdk.AccAddress
    Recipient sdk.AccAddress
    Denom     string
    ID        string
}
```

```
type MsgEditNFTMetadata struct {
    Sender   sdk.AccAddress
    ID       string
    Denom    string
    TokenURI string
}
```

```
type MsgMintNFT struct {
    Sender    sdk.AccAddress
    Recipient sdk.AccAddress
    ID        string
    Denom     string
    TokenURI  string
}
```

```
type MsgBurnNFT struct {
    Sender sdk.AccAddress
    ID     string
    Denom  string
}
```

NFT Handler

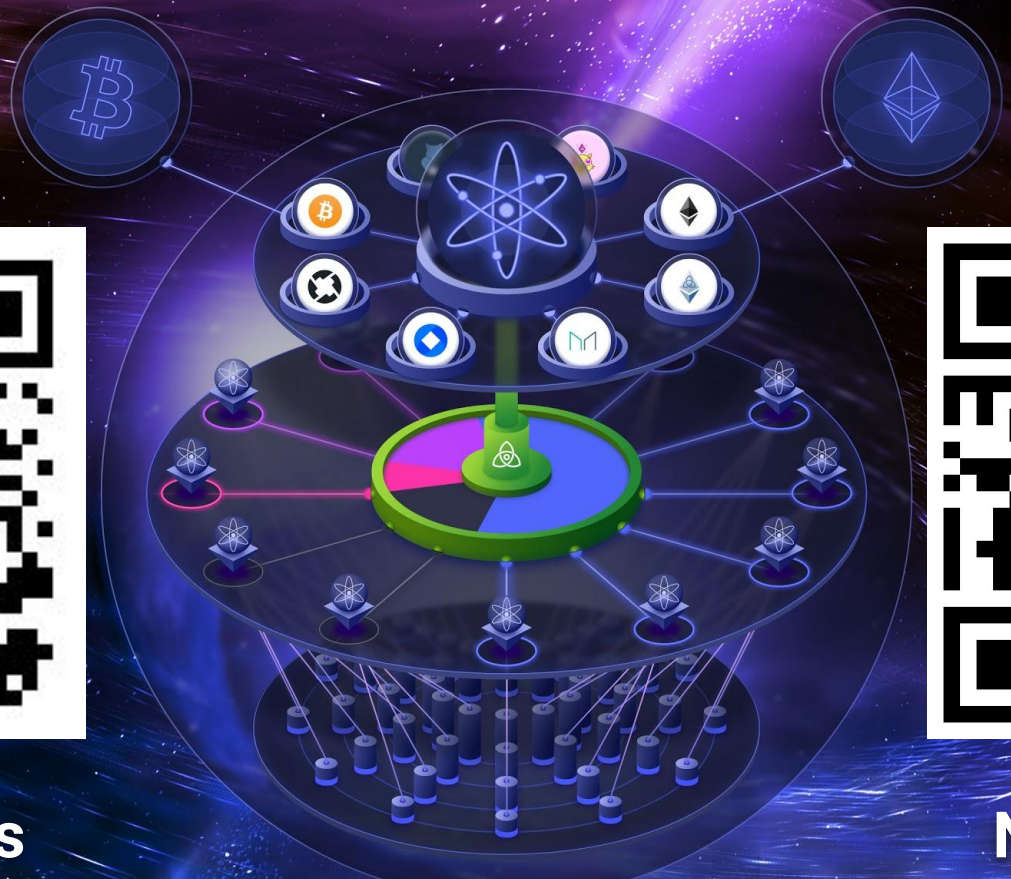
```
// NFTHandler routes the messages to the handlers
func NFTHandler(k nft.Keeper) sdk.Handler {
    return func(ctx sdk.Context, msg sdk.Msg) sdk.Result {
        switch msg := msg.(type) {
        case nft.MsgTransferNFT:
            return nft.HandleMsgTransferNFT(ctx, msg, k)
        case nft.MsgEditNFTMetadata:
            return nft.HandleMsgEditNFTMetadata(ctx, msg, k)
        case nft.MsgMintNFT:
            return HandleMsgMintNFT(ctx, msg, k)
        case nft.MsgBurnNFT:
            return nft.HandleMsgBurnNFT(ctx, msg, k)
        default:
            errMsg := fmt.Sprintf("unrecognized nft message type: %T", msg)
            return sdk.ErrUnknownRequest(errMsg).Result()
        }
    }
}
```

NFT Handler

```
// HandleMsgMintNFT handles MsgMintNFT
func HandleMsgMintNFT(ctx sdk.Context, msg types.MsgMintNFT, k keeper.Keeper,
) sdk.Result {

    nft := types.NewBaseNFT(msg.ID, msg.Recipient, msg.TokenURI)
    err := k.MintNFT(ctx, msg.Denom, &nft)
    if err != nil {
        return err.Result()
    }

    // emit events here

}
```

NFT Keeper

```
func (k Keeper) MintNFT(ctx sdk.Context,
    denom string, nft exported.NFT) (err sdk.Error) {
    collection, found := k.GetCollection(ctx, denom)
    if found {
        collection, err = collection.AddNFT(nft)
        if err != nil {
            return err
        }
    } else {
        collection = types.NewCollection(denom, types.NewNFTs(nft))
    }
    k.SetCollection(ctx, denom, collection)

    ownerIDCollection, _ := k.GetOwnerByDenom(ctx, nft.GetOwner(), denom)
    ownerIDCollection = ownerIDCollection.AddID(nft.GetID())
    k.SetOwnerByDenom(ctx, nft.GetOwner(), denom, ownerIDCollection.IDs)
    return
}
```



SDK Tutorials
(tutorial.cosmos.network)

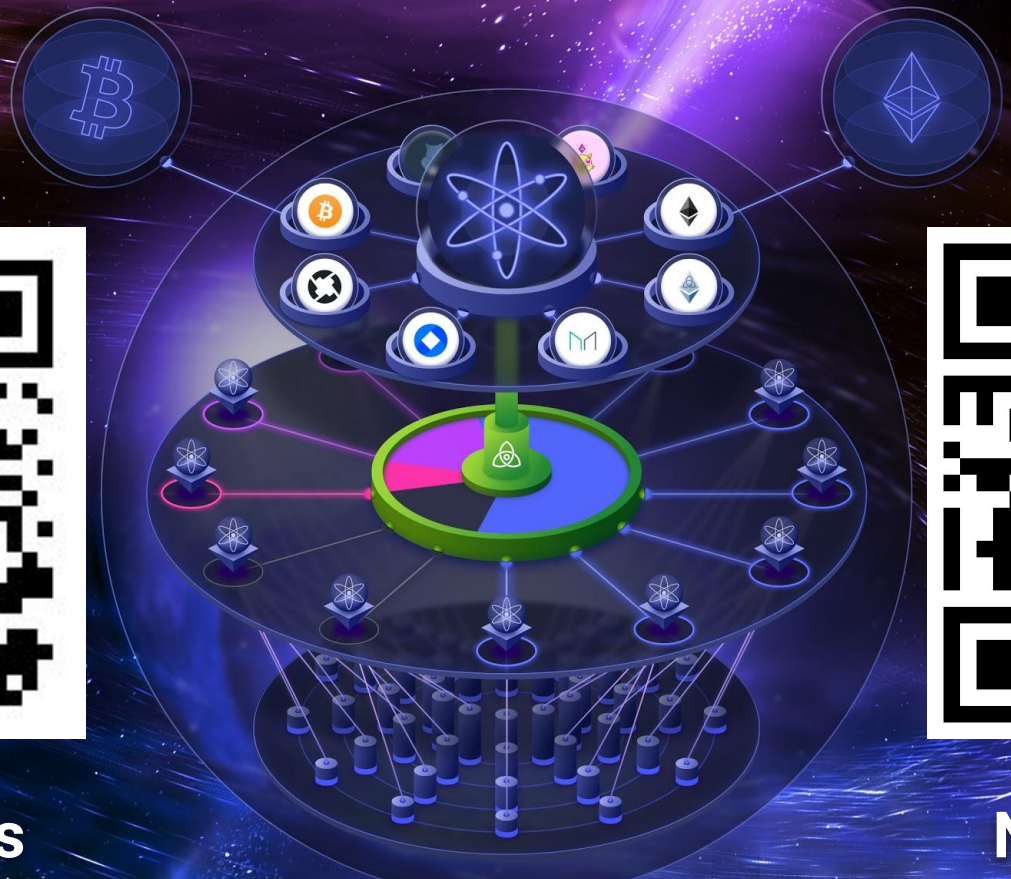
NFT Example
(@okwme/cosmos-nft)

IBC + NFT outgoing

```
    packetData := types.PacketData{
        ID:       id,
        Denom:    denom,
        TokenURI: tokenURI,
        Sender:   sender,
        Receiver: receiver,
        Source:   isSourceChain,
    }
    packetDataBz, err := packetData.MarshalJSON()
    packet := channeltypes.NewPacket(
        seq,
        uint64(ctx.BlockHeight())+DefaultPacketTimeout,
        sourcePort,
        sourceChannel,
        destinationPort,
        destinationChannel,
        packetDataBz,
    )
    // generate the capability key
    key := sdk.NewKVStoreKey(types.BoundPortID)
    return k.channelKeeper.SendPacket(ctx, packet, key)
```

IBC + NFT incoming

```
func (k Keeper) ReceiveTransfer(...) error {
    if data.Source {
        prefix := types.GetDenomPrefix(destinationPort, destinationChannel)
        if !strings.HasPrefix(data.Denom, prefix) {
            return sdk.NewError("incorrect denomination")
        }
        nft := NewBaseNFT(data.ID, data.Receiver, data.TokenURI)
        return k.nftKeeper.MintNFT(ctx, data.Denom, &nft)
    }
    prefix := types.GetDenomPrefix(sourcePort, sourceChannel)
    if !strings.HasPrefix(data.Denom, prefix) {
        return sdk.NewError("incorrect denomination")
    }
    escrowAddress := types.GetEscrowAddress(destinationPort, destinationChannel)
    denom := data.Denom[len(prefix):]
    nft, err := k.nftKeeper.GetNFT(ctx, denom, data.ID)
    if !nft.GetOwner().Equals(escrowAddress) {
        return sdk.NewError("cant nft_transfer un-owned NFT")
    }
    nft.SetOwner(data.Receiver)
    return k.nftKeeper.UpdateNFT(ctx, denom, nft)
}
```



SDK Tutorials
(tutorial.cosmos.network)

NFT Example
(@okwme/cosmos-nft)