Rearchitecting the **Blockchain Ecosystem**

Billy Rennekamp

Developer Relations - Tendermint Inc

Evolution of Blockchain Development

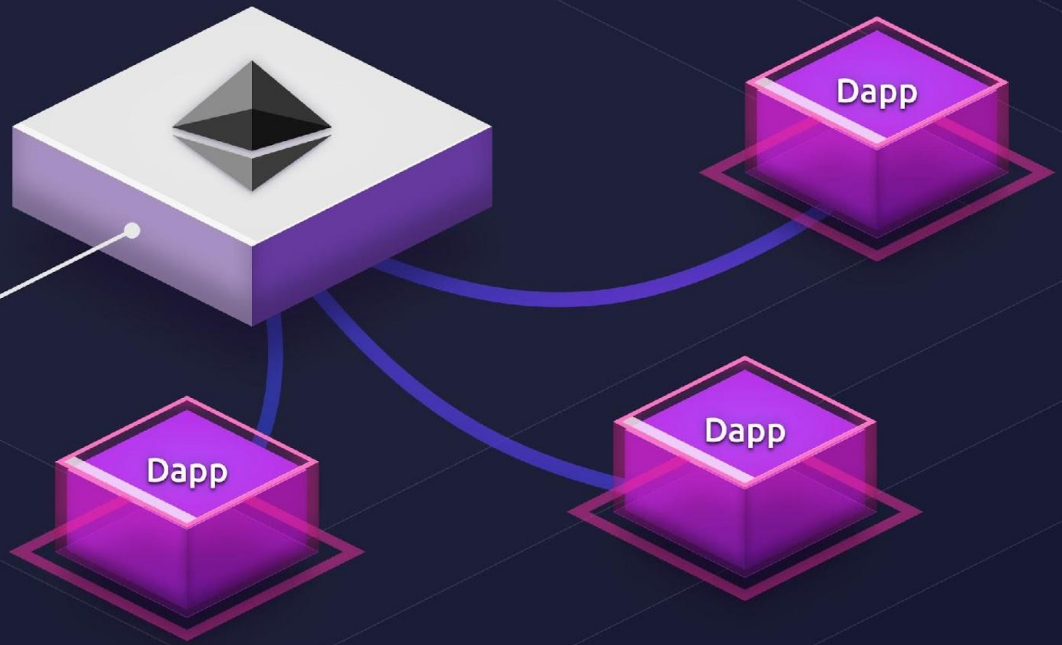
Gen 1: Forked Bitcoin Codebase

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

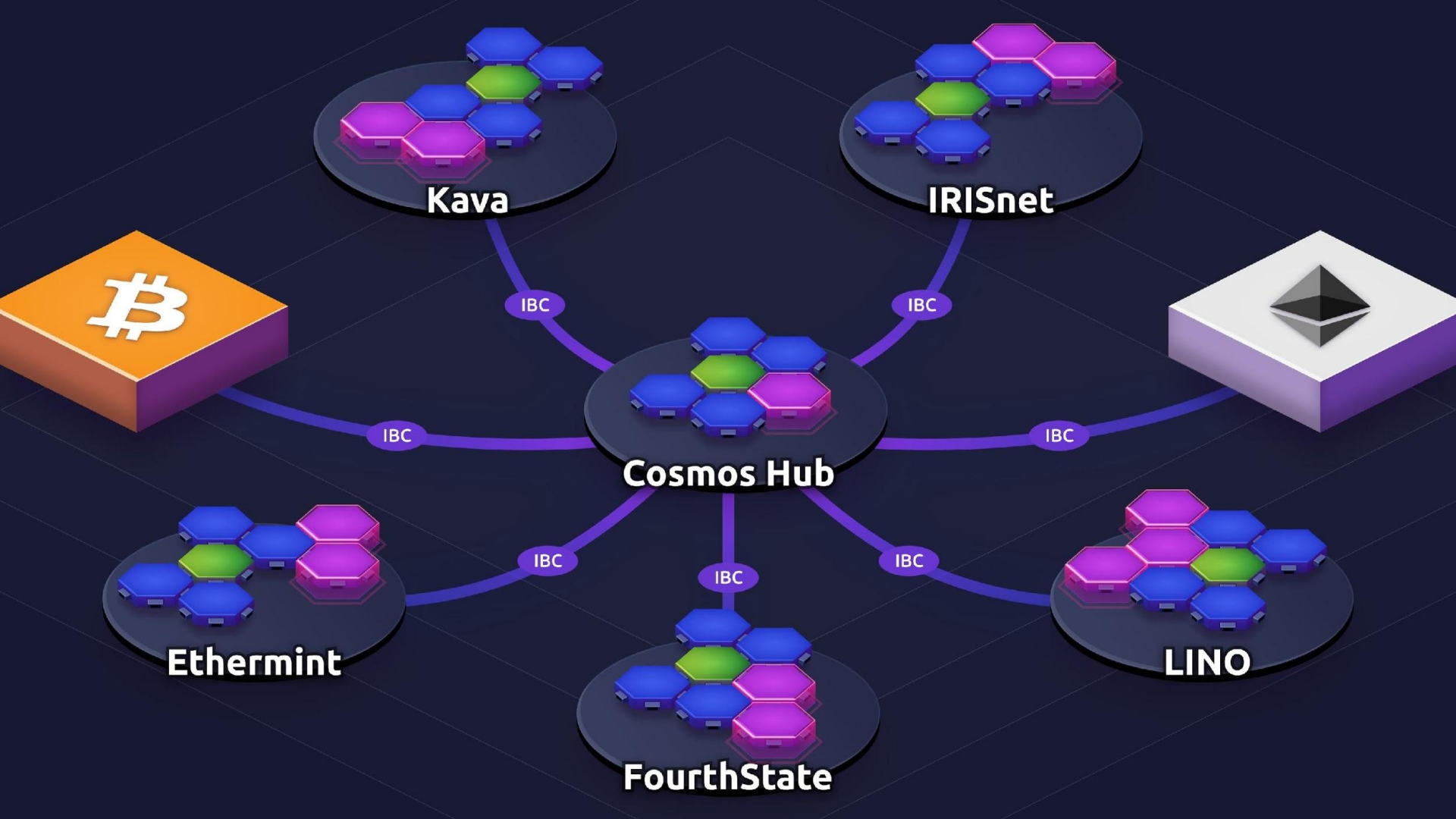
Core Module: IBC

- 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
LINO
FourthState

The Cosmos-SDK is a Modular Framework

- Aggregates a collection of interoperable modules
- Each Module has its own Message processor
- The Baseapp directs each message to the respective module
- A REST server is exposed for interaction
- Msg / Handler / Querier / Keeper

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

Each module has Messages

- Messages are like functions in Solidity
- They are included in transactions
- They modify the state
- The base app receives Messages from Tendermint and directs them to the appropriate Module

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

```go
// 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

```go
// 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

}
```

COSMOS

INTERNET OF BLOCKCHAINS

SDK Tutorial

(cosmos/sdk-application-tutorial)

NFT Example

(okwme/cosmos-nft)