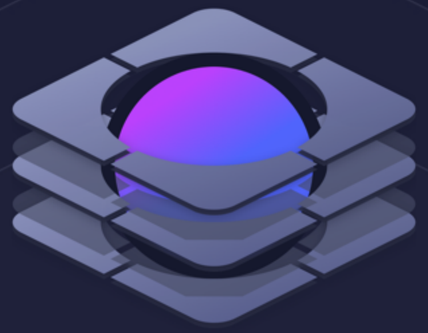


COSMOS SDK

Scalable Blockchain Development

Sunny Aggarwal
Researcher & Core Developer at Cosmos

Why the SDK?

COSMOS SDK

Why the SDK?

Build your own DappChain

Write in Go, not Solidity

Modular blockchain development

COSMOS SDK

KVStore

COSMOS SDK

KVStore

```
type KVStore interface {
    Store
    Get(key []byte) []byte
    Has(key []byte) bool
    Set(key, value []byte)
    Delete(key []byte)
    Iterator(start, end []byte) Iterator
    ReverseIterator(start, end []byte) Iterator
}
```

Multiple KVStores

- Each module can use multiple stores for separate resources
- Each is accessible only using its “key”
- Object Capabilities Security

Keepers

COSMOS SDK

What is a Keeper?

- “Keep”s all the module’s functionality in a stateless manner
- Contains:
  - StoreKeys
  - Methods
  - Keepers from other modules

StoreKeys: Hold a key not a reference

```
type NameKeeper struct {
    storeKey sdk.StoreKey  // The (unexposed) keys used to access the store
from the Context.
}

func (k NameKeeper) GetName(ctx sdk.Context, name string) string {
  store := ctx.KVStore(k.storeKey)
  bz := store.Get([]byte(name))
  return string(bz)
}

func (k NameKeeper) setName(ctx sdk.Context, name string, val string) {
  store := ctx.KVStore(k.storeKey)
  store.Set([]byte(name), []byte(val))
}
```

Methods

```
Func (k NameKeeper) Capitalize(ctx sdk.Context, name string) {
    val := k.GetName(ctx, name)
    newVal := capitalize(val)
    k.SetName(ctx, name, newVal)
}

Func (k NameKeeper) delete(ctx sdk.Context, name string) {
    k.SetName(ctx, name, []byte{})
}
```

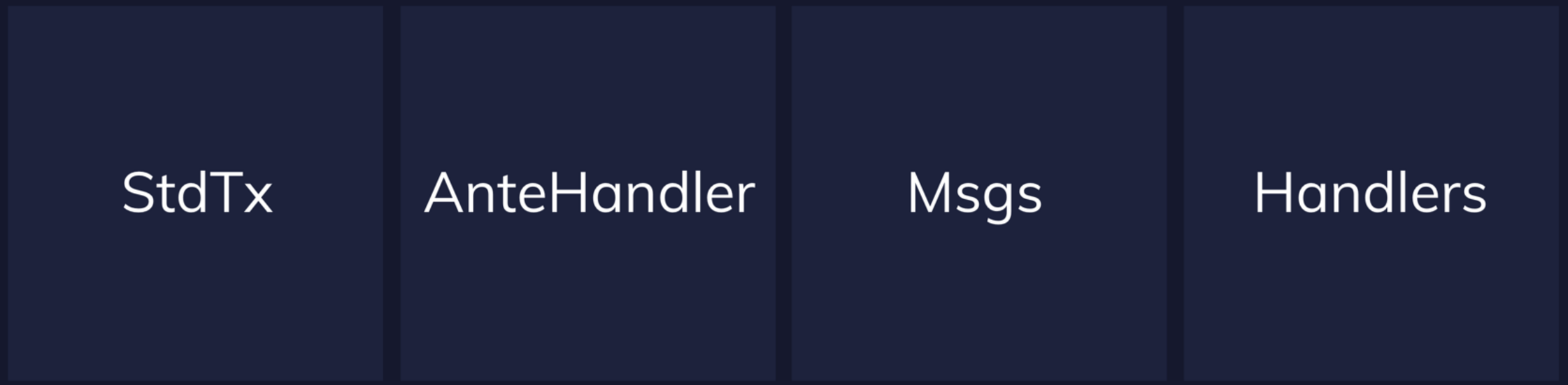
Keepers from Other Modules

```
type NameKeeper struct {
    coinKeeper bank.CoinKeeper
}

Func (k NameKeeper) BuyName(ctx sdk.Context, name
string, val string, buyer sdk.Address, bid sdk.Coins) {
    k.SetName(ctx, name, val)
    k.bankKeeper.SubtractCoins(ctx, buyer, bid)
}
```

Transactions

COSMOS SDK

Users interacting with the chain



StdTx

```
// A standard way to wrap a Msg
// with a Fee and Signatures.
type StdTx struct {
    Msgs          []sdk.Msg        `json:"msgs"`
    Fee           StdFee           `json:"fee"`
    Signatures    []StdSignature   `json:"signatures"`
    Memo          string           `json:"memo"`
}
```

AnteHandler

- Does basic verification of tx validity.
- Checks:
  - Signatures
  - Sequence Numbers
  - Fees are enough for requested Gas
- Passes the Msgs to the proper handler

Msgs

```
// Transactions messages must fulfill the Msg
type Msg interface {
    // Return the message type.
    // Must be alphanumeric or empty.
    Type() string

    // ValidateBasic does a simple validation check that
    // doesn't require access to any other information.
    ValidateBasic() Error

    // Get the canonical byte representation of the Msg.
    GetSignBytes() []byte

    // Signers returns the addrs of signers that must sign.
    // CONTRACT: All signatures must be present to be valid.
    // CONTRACT: Returns addrs in some deterministic order.
    GetSigners() []AccAddress
}
```

Example Msg

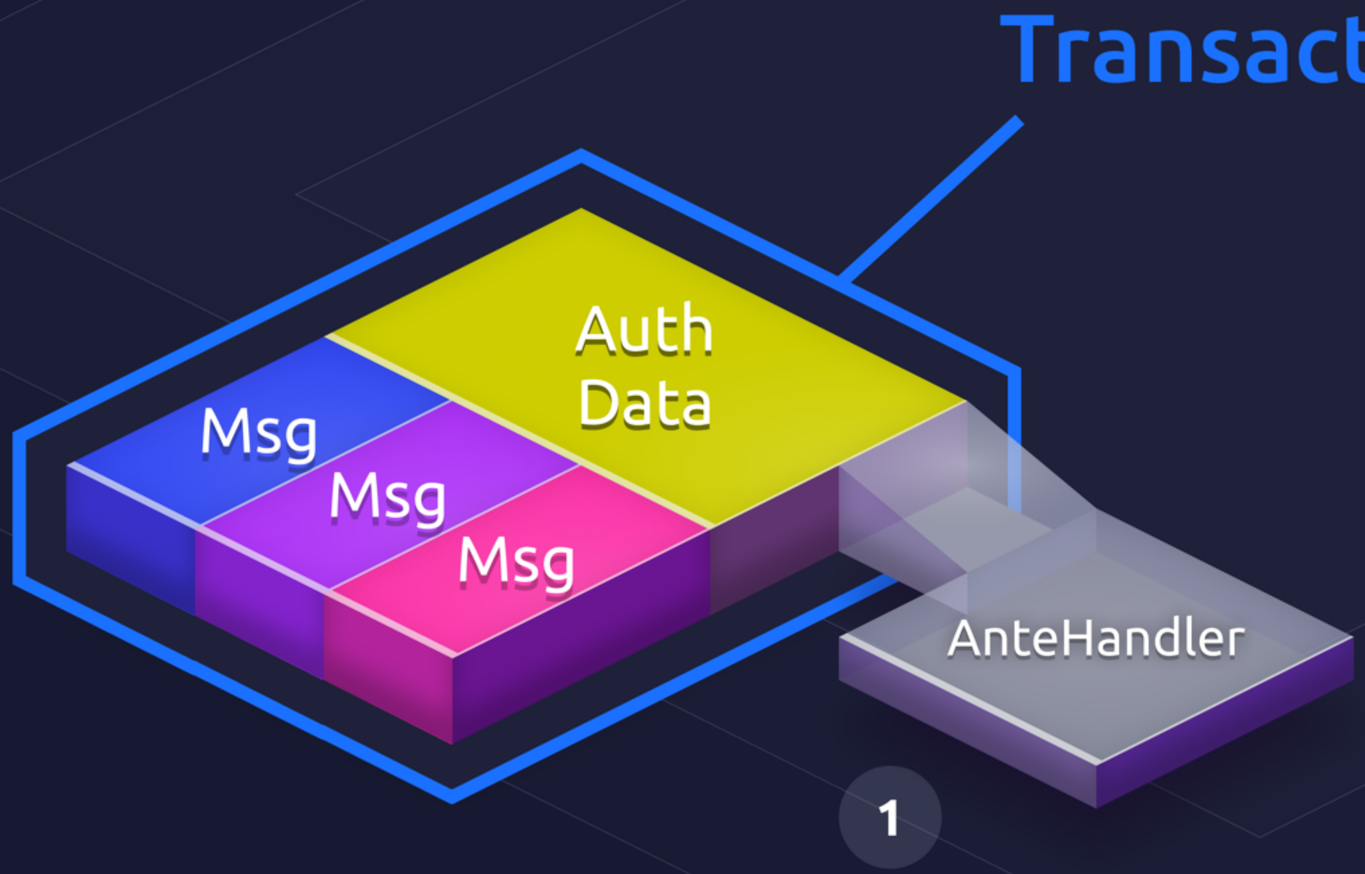
```
type MsgSetName struct {
    Name  string
    Value string
    Owner sdk.AccAddress
}

func (msg MsgSetName) Type() string { return
"nameservice" }

func (msg MsgSetName) GetSignBytes() []byte {
    b, err := json.Marshal(msg)
    if err != nil {
        panic(err)
    }
    return sdk.MustSortJSON(b)
}
```

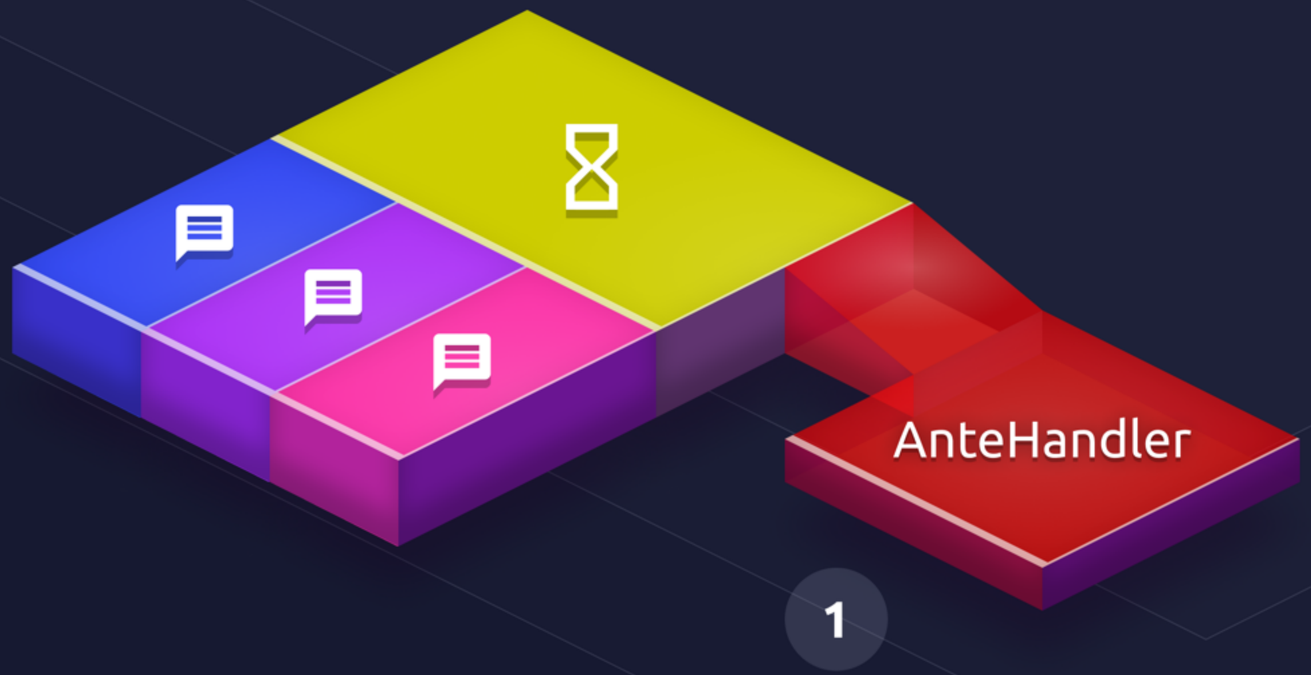
```
func (msg MsgSetName) ValidateBasic() sdk.Error {
    if msg.Owner.Empty() {
      return sdk.ErrInvalidAddress(msg.Owner.String())
    }
      return sdk.ErrUnknownRequest(“Bad“)
    }
    return nil
}

func (msg MsgSetName) GetSigners()
[]sdk.AccAddress {
    return []sdk.AccAddress{msg.Owner}
}
```

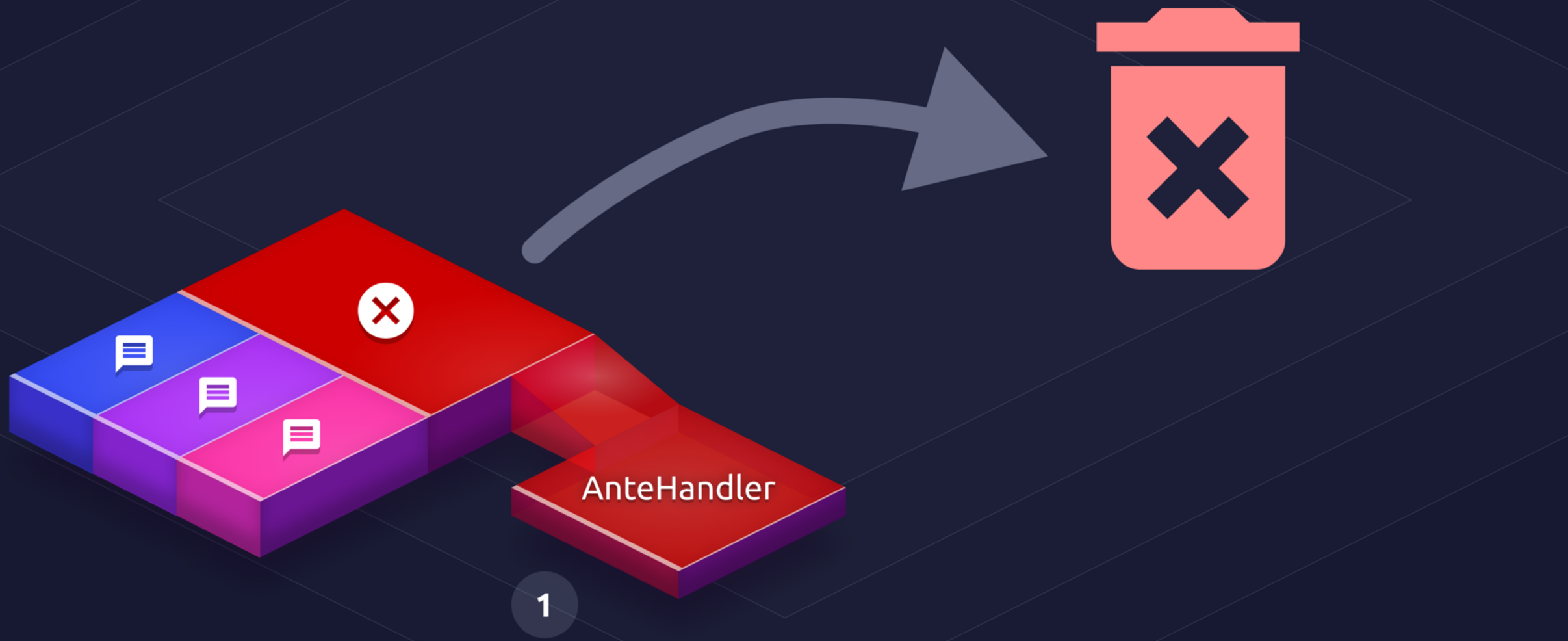
Handler

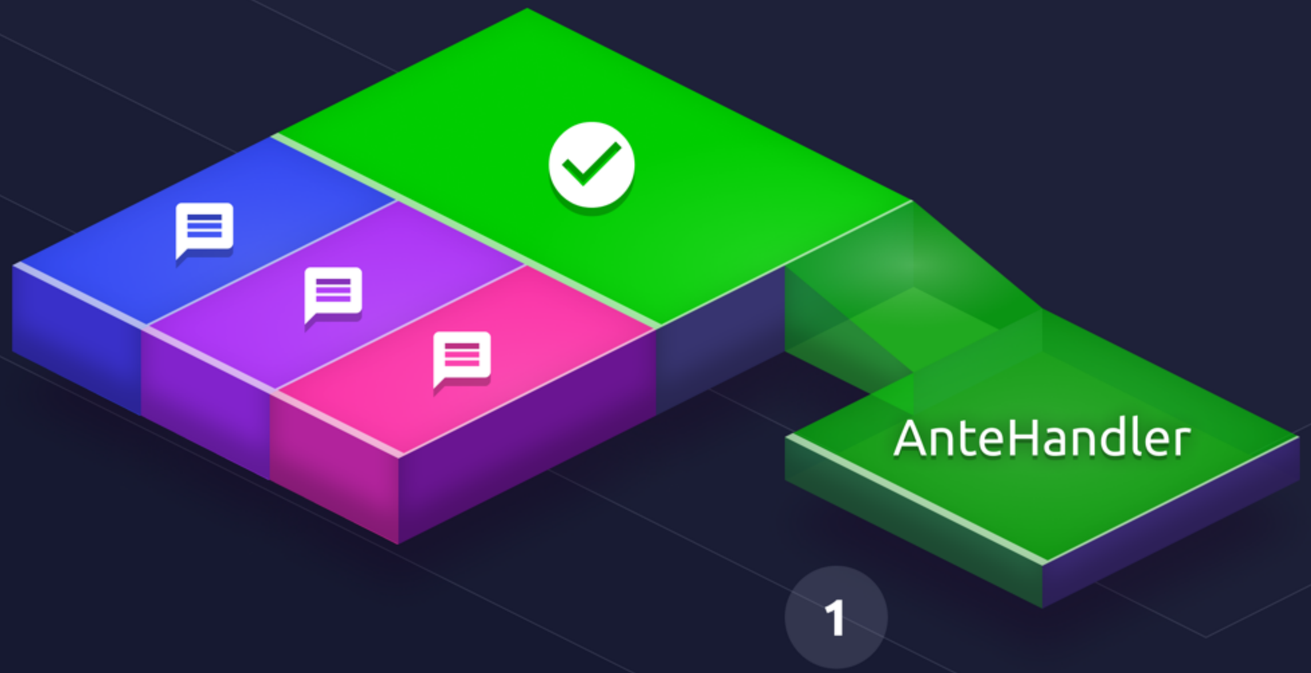
- Defined for a specific Msg Type
- Takes in a Msg and executes the appropriate Keeper methods
- Returns an sdk.Result
  - Data
  - ABCI Code (Errors)
  - Logs
  - Tags (Events)

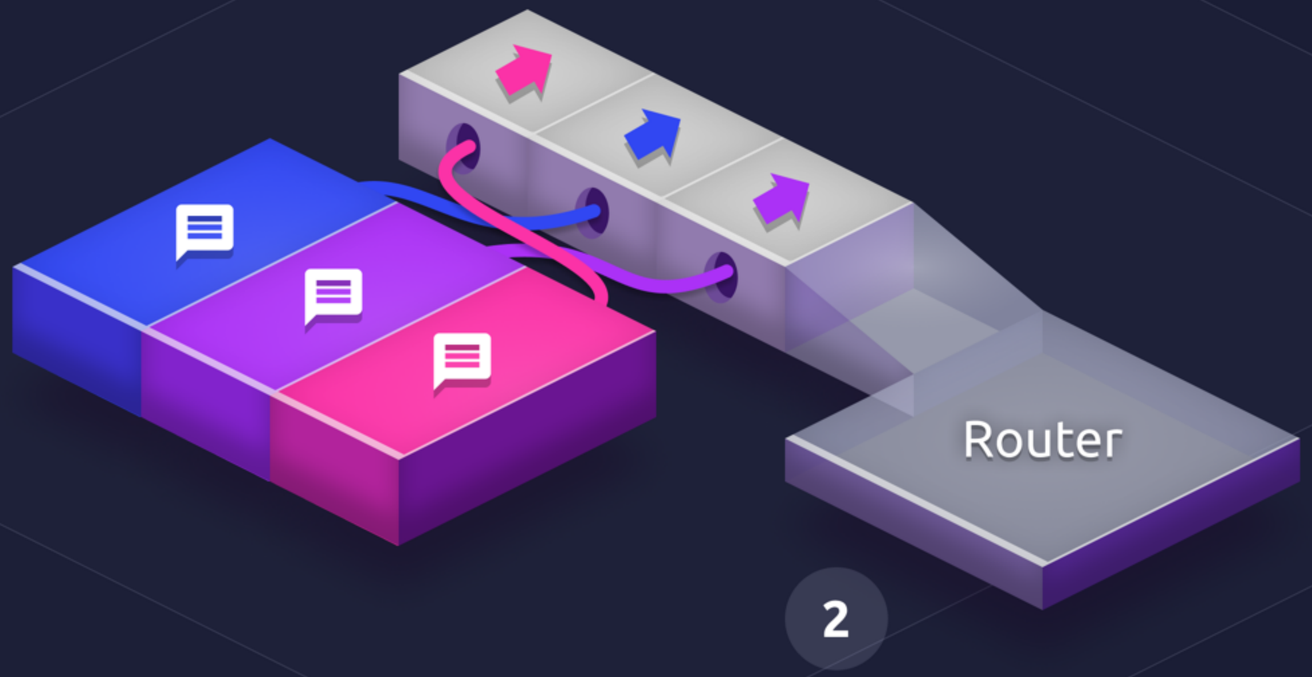
```
// Result is the union of ResponseDeliverTx and
ResponseCheckTx.
type Result struct {

    // Code is the response code, is stored back on the
chain.
    Code ABCICodeType

    // Data is any data returned from the app.
    Data []byte

    // Log is just debug information. NOTE:
nondeterministic.
    Log string

    // GasWanted is the maximum units of work we allow
this tx to perform.
    GasWanted int64

    // GasUsed is the amount of gas actually consumed.
NOTE: unimplemented
    GasUsed int64

    // Tx fee amount and denom.
    FeeAmount int64
    FeeDenom  string

    // Tags are used for transaction indexing and
pubsub.
    Tags Tags
}
```

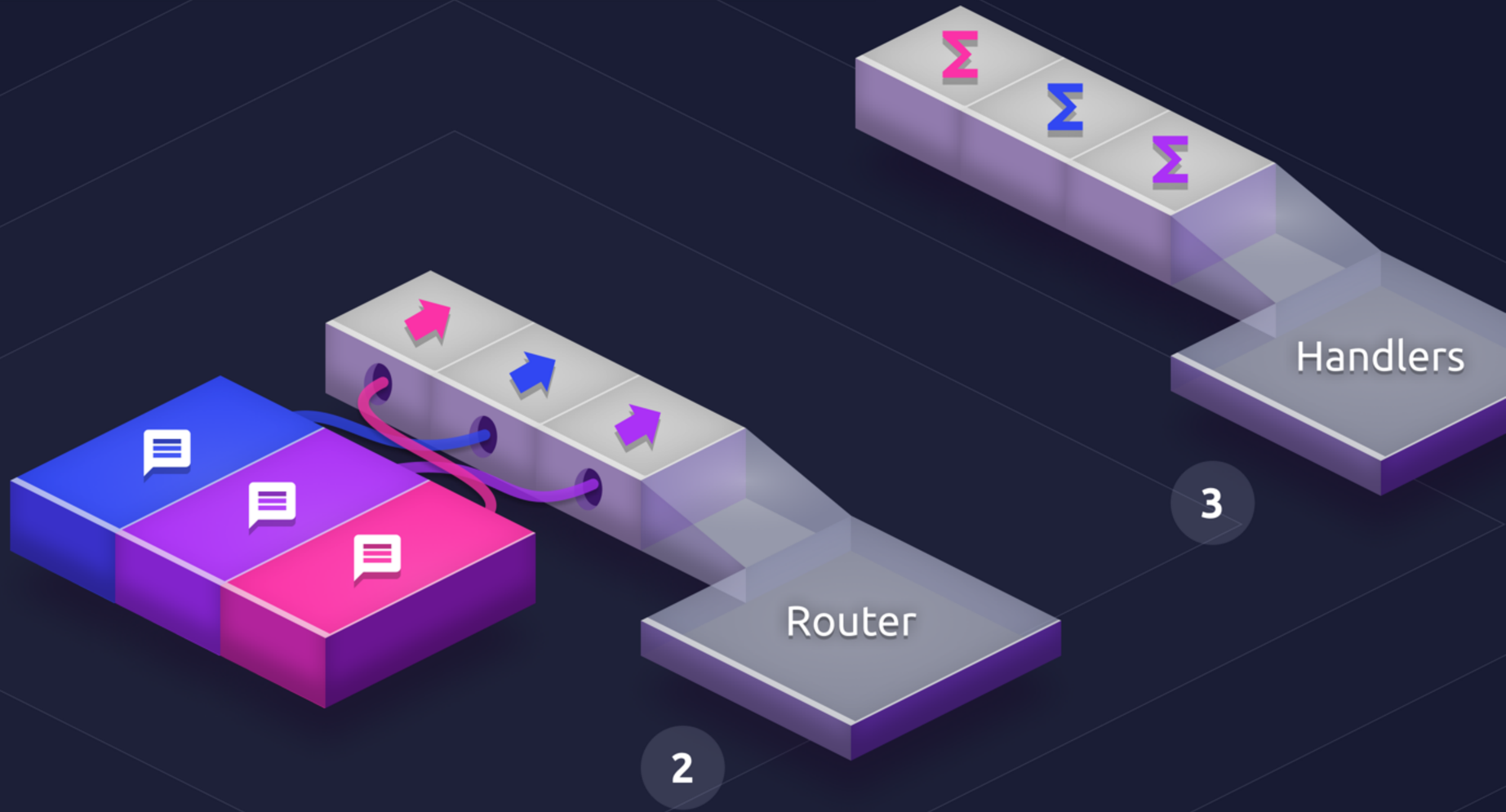

Transaction
Auth
Data
Msg
Msg
Msg
AnteHandler

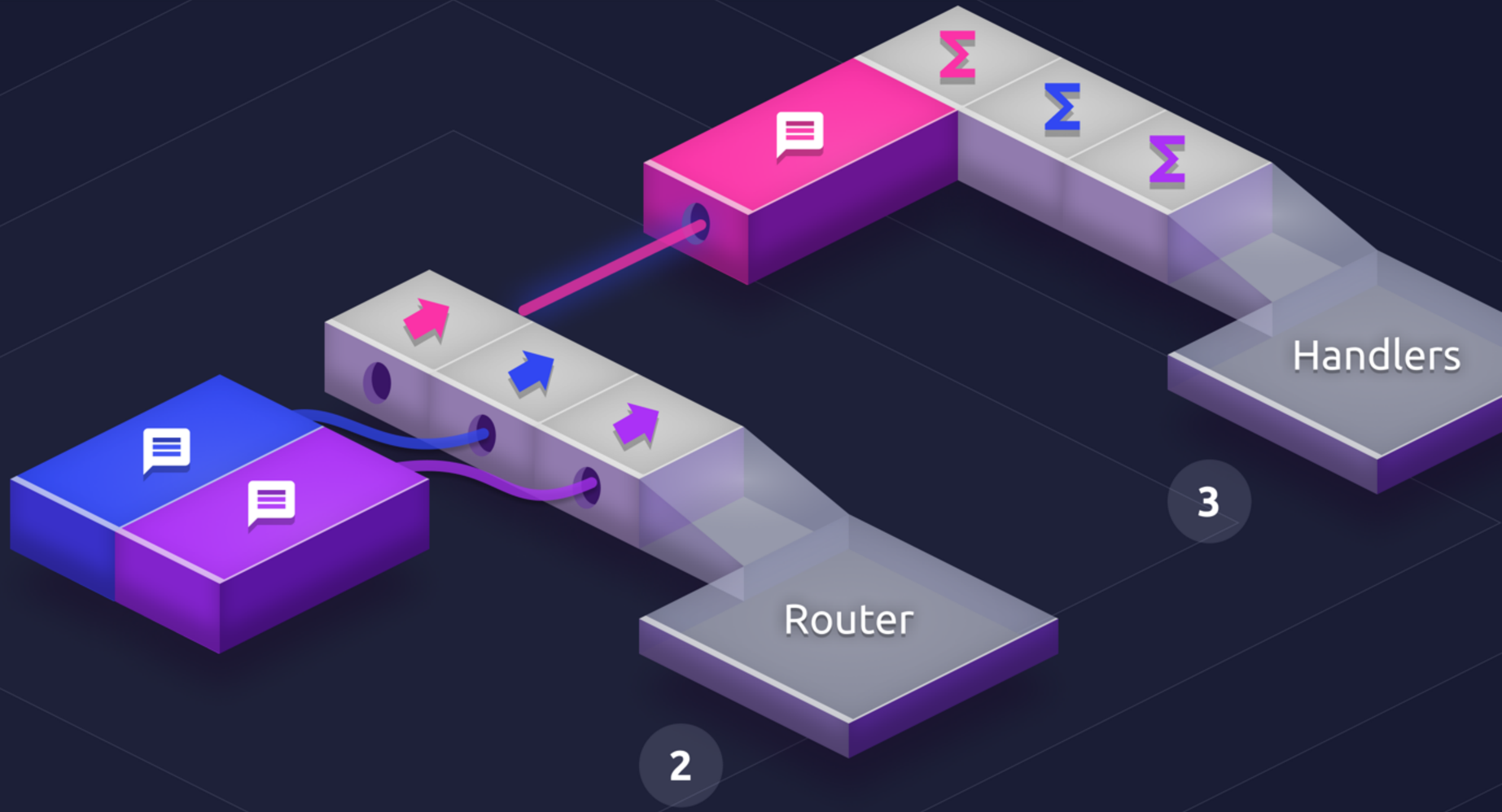

AnteHandler

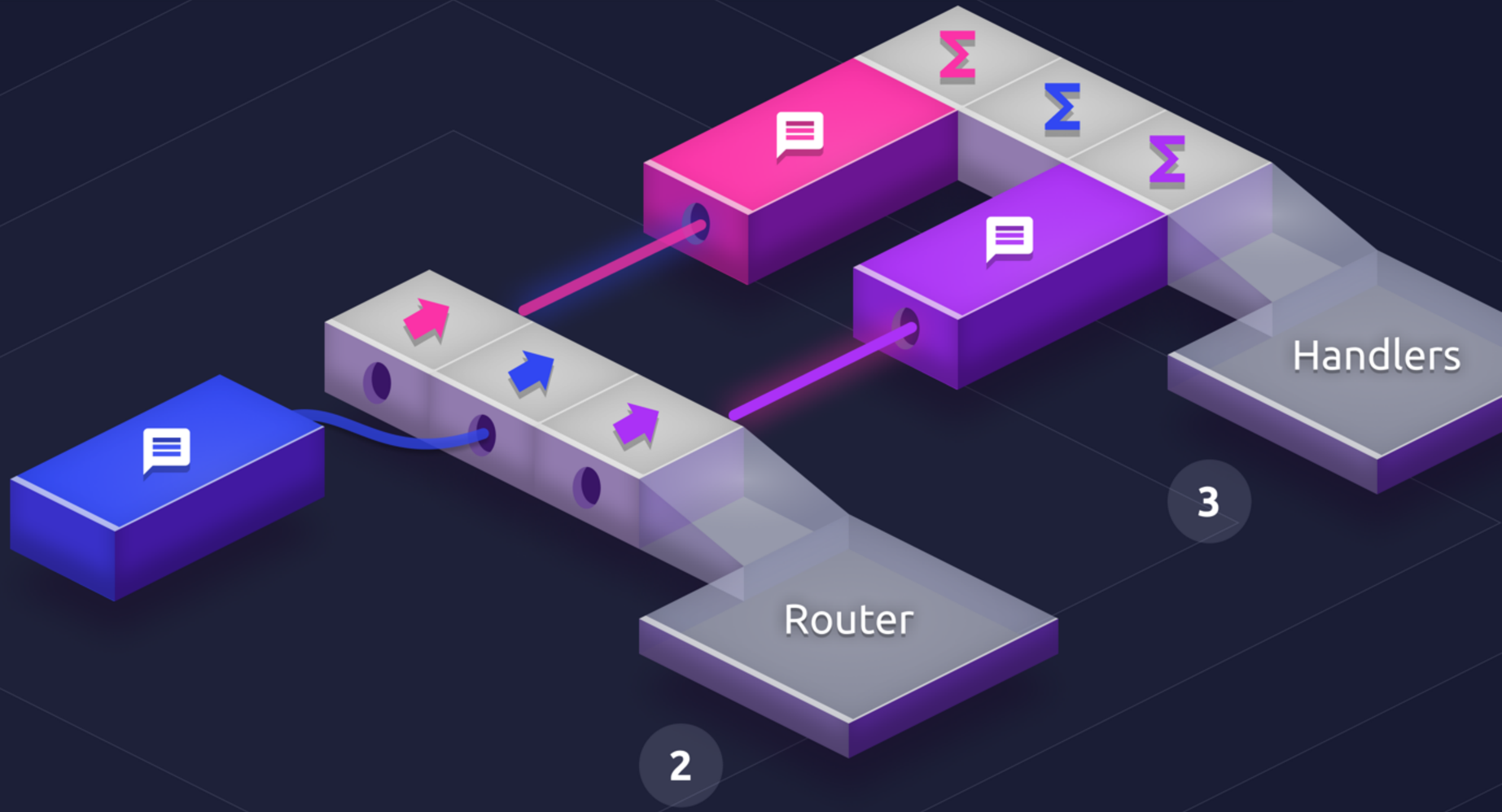

AnteHandler

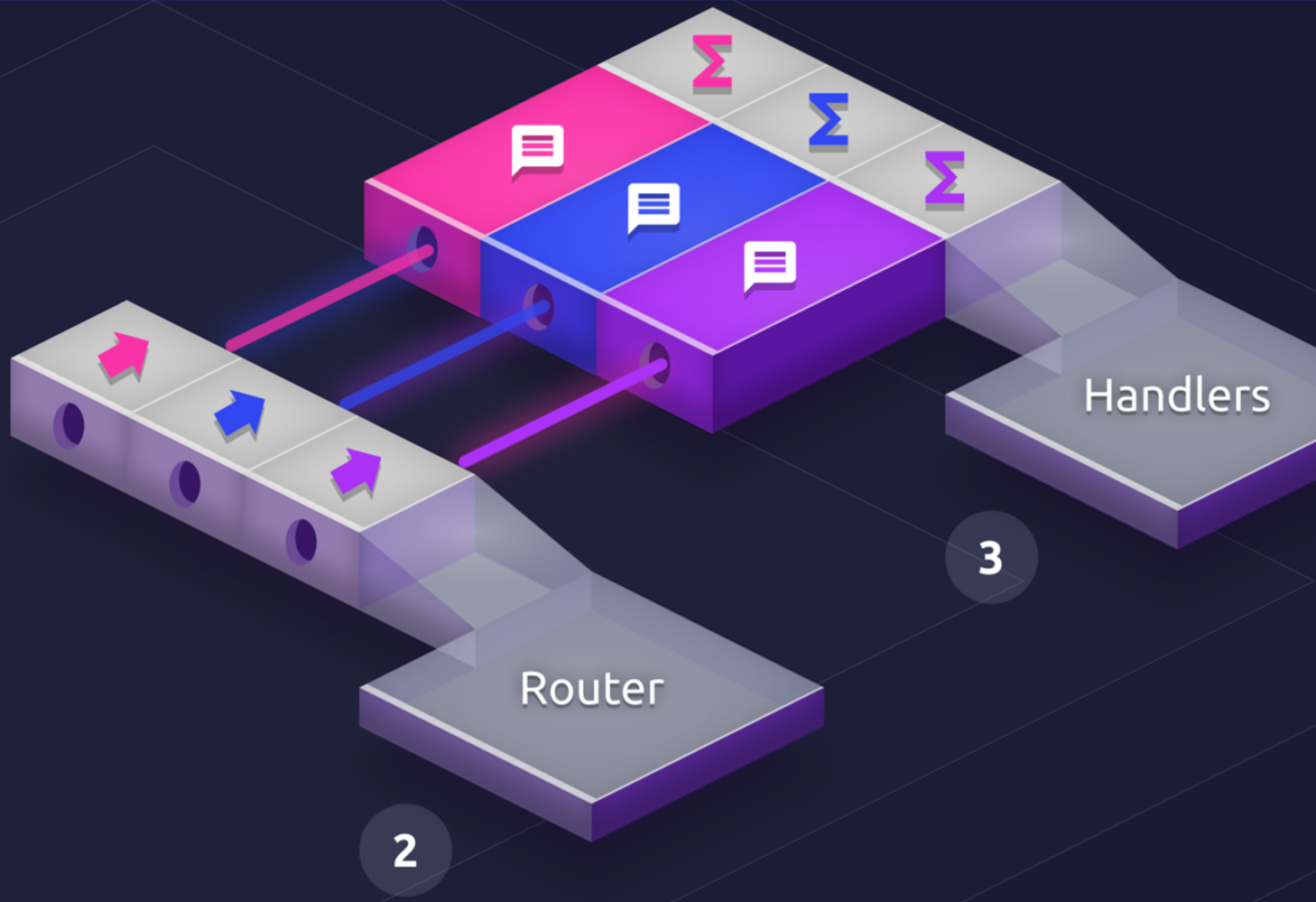

AnteHandler

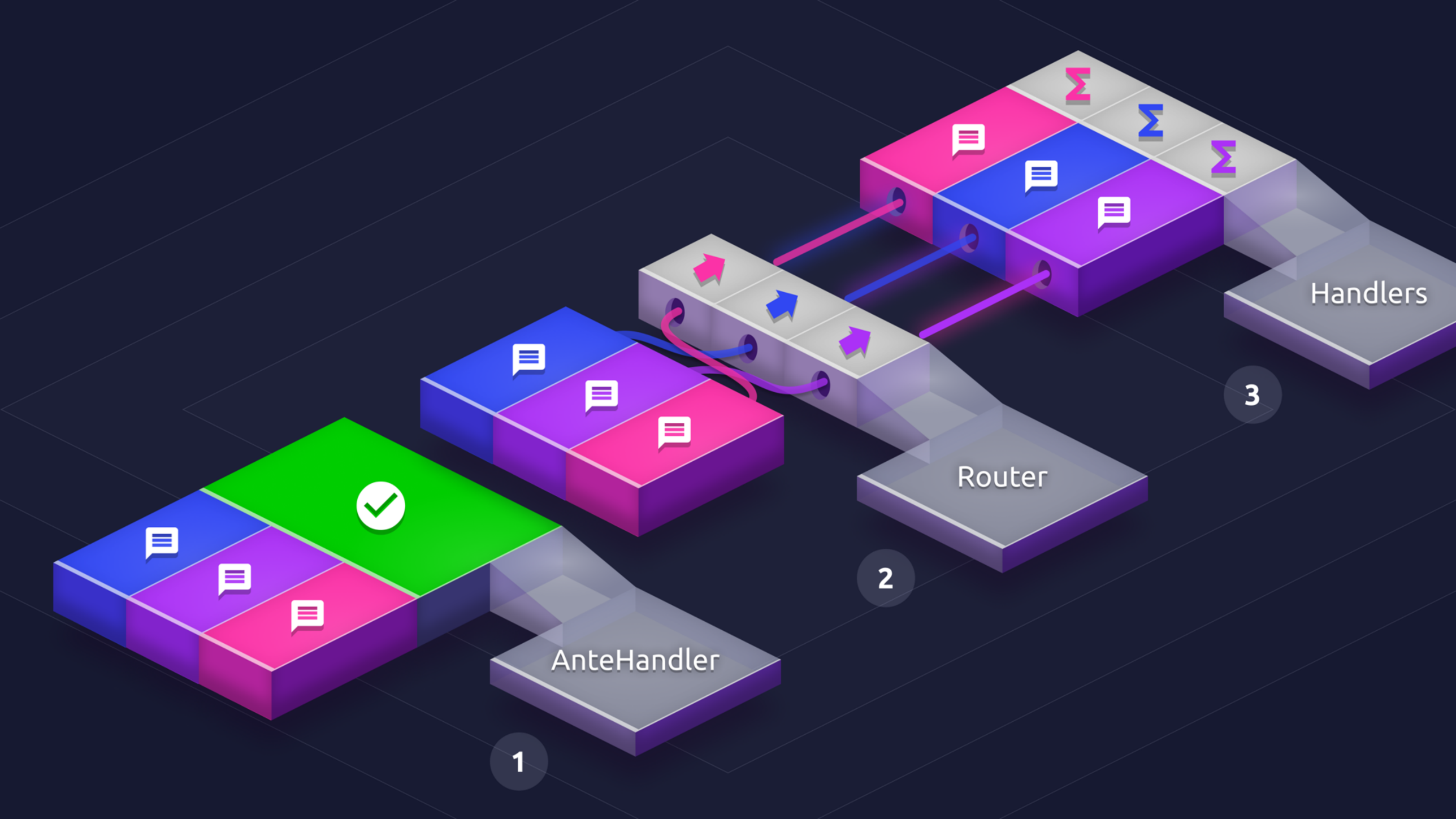

Router
2


Handlers
3
Router
2


Handlers
3
Router
2


Handlers
3
Router
2


Handlers
3
Router
2


Handlers
3
Router
2
AnteHandler
1

Modules Review

Well-constrained and easy-to-reason-about business logic defining *your* app's functionality

Keepers

- Core functionality
- Getters and Setters into the store
- Limits how modules interact with each other

Transactions

- Defines how users interact with blockchain

Follow the Full Tutorial

github.com/sunnya97/sdk-nameservice-example

Check Out Jae Kwon's Talk

Tomorrow
9:50am