gno.land

Building a Decentralized App on gno.land

https://github.com/gnolang/gc24-us-workshop

Presented by Dylan Boltz, Sr. Golang Developer @ Gno.land

Who am I?

Dylan Boltz

Sr. Golang Engineer @ gno.land

Go maximalist

Working in web3 for the past three years

https://github.com/deelawn

Today’s Agenda

- Introduction to gno.land and blockchain in general
- Gno virtual machine architecture
- Develop a stackoverflow-like application on gno.land

Hands On: Phase 1

https://github.com/gnolang/gc24-us-workshop
Clone this repository and follow the instructions in P1 to set up your local gno.land development environment

What is gno and gno.land?

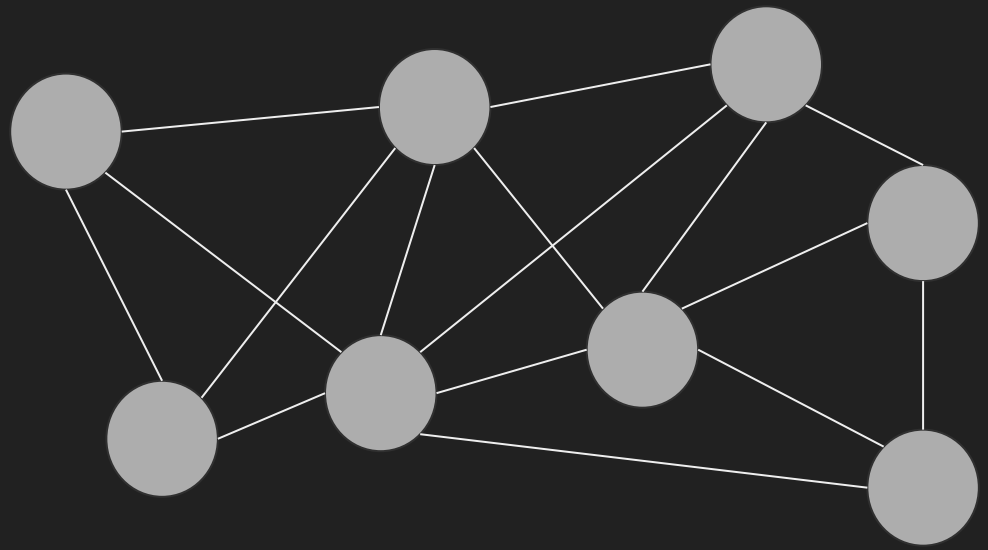
- gno is an interpreted, deterministic version of Go with a persistent data store
- gno incorporates a select list of Go features and standard libraries from Go version 1.17
- gno.land is soon to be released blockchain which uses gno as its smart contract programming language

What is a Blockchain?

- Distributed database
- Immutable transaction history
- Decentralized control
- Verifiable state
- Trustless
- State transitions: func(currentState, txs) newState

Blockchain Network

- Nodes P2P communication
- Clients submit transactions to a node
- Node propagates transactions to the network
- A node proposes a block
- Other nodes verify the block



gno - features and limitations

- Determinism guaranteed
- Generics not supported (yet)
- No concurrency
- Built in application state persistence
- Import select standard libraries and other packages and applications deployed to a VM instance
- Safe execution in a sandboxed environment

https://docs.gno.land/reference/go-gno-compatibility

Example gno.land realm

Globals variables are persisted

Exported functions are callable

Unexported functions only callable from within this realm

```
1  package counter
2
3  import "std"
4
5  var (
6      Value int64
7      LastUpdatedBy std.Address
8  )
9
10 func IncrementValue() int64 {
11     return addToValue(1)
12 }
13
14 func IncreaseValue(amount int64) int64 {
15     return addToValue(amount)
16 }
17
18 func addToValue(amount int64) int64 {
19     Value += amount
20     LastUpdatedBy = std.PrevRealm().Addr()
21     return Value
22 }
```

Why use gno.land?

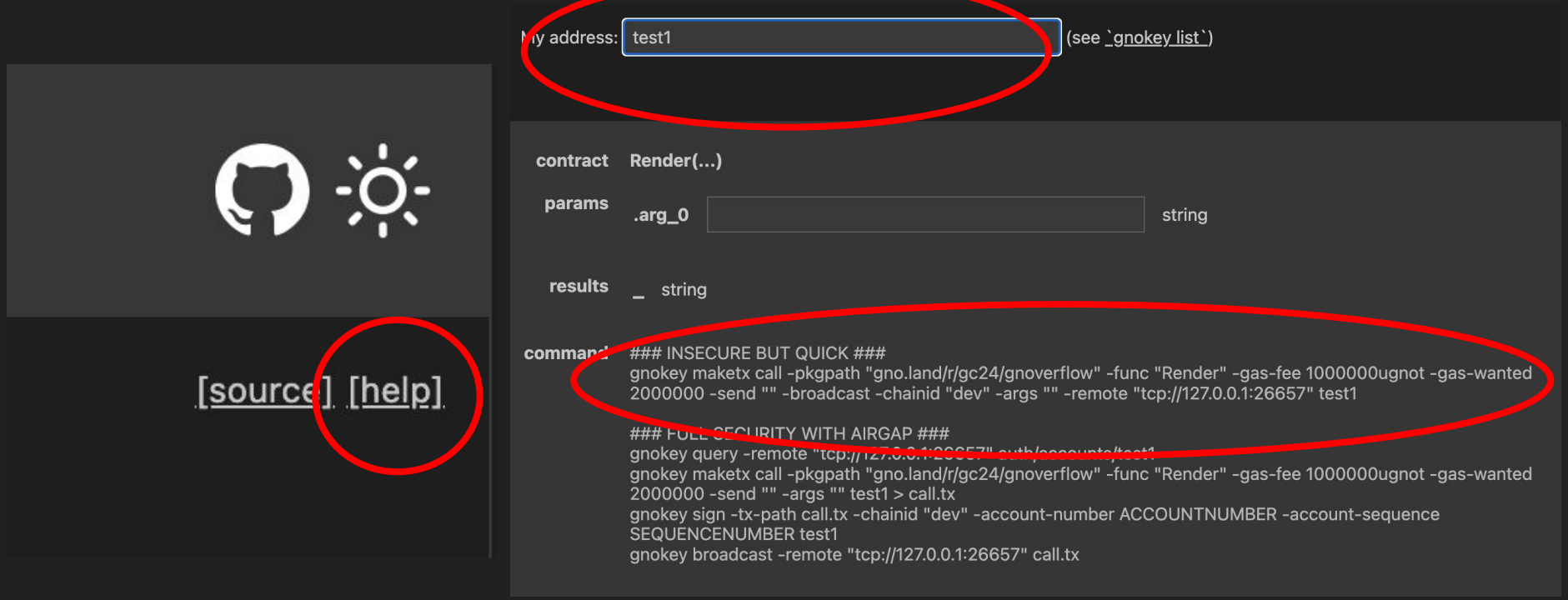
- Enables developers to write transparent and composable smart contracts using Gno, a language chosen for its simplicity
- Abstract away storage and variable lifecycle management by storing any value defined as a global realm variable
- Give developers the choice for if and how they'd like to render application data that is stored on the blockchain, granting users an accessible view into the application's state
- Enable the development of decentralized social applications that need not solely focus on finance or cryptographic assets

Hands On: Phase 2

- Any issues with setting up the gno.land dev environment?
- Open README.md in the phase2 directory
- Stackoverflow-like gno.land realm
- Use `gnodev` and `gnokey` to develop and test
- Define a Question data structure
- Write functions to add and modify questions
- Look for gno.land team members if you need help or have questions

Why type out gnokey commands manually?

Once `gnodev` is started, you can navigate to the gnoverflow realm in your browser at localhost:8888/r/gc24/gnoverflow.



P2: Checkin

- The `Render` function is what gets called when visiting localhost:8888/r/gc24/gnoverflow
- It returns a string, that can be markdown, and is rendered in the browser
- We use AVL trees in place of maps; due to gno's deterministic nature, a map implementation over an ordered range is slightly less efficient
- An AVL tree is a key-value store. Using a Question pointer (*Question) type as the value means that values retrieved from the tree can be modified directly without having to use the trees `Set()` method
- Use `std.PrevRealm().Addr()` to retrieve the `std.Address` that called the function being executed

P2: What have we accomplished?

- Users can call a realm function to ask a question
- Asked questions are rendered in the browser for all to see
- Questions, being a global realm variable, are persisted on the blockchain automatically
- Modifications to realm storage (Questions) results in automatic persistence
- Data is distributed across the network, redundantly stored on each gno.land blockchain node

Try not to panic but…

DO Panic!

Embrace it

Panicking during realm code execution is normal and is the way to abort a transaction when something unexpected happens

It ensures any realm state changes are rolled back

Determinism

- Necessary
- No network packages are supported
- `time.Now()`
  - Returns the time of the current block
  - The same value is returned for all transactions in the current block

Hands On: Phase 3

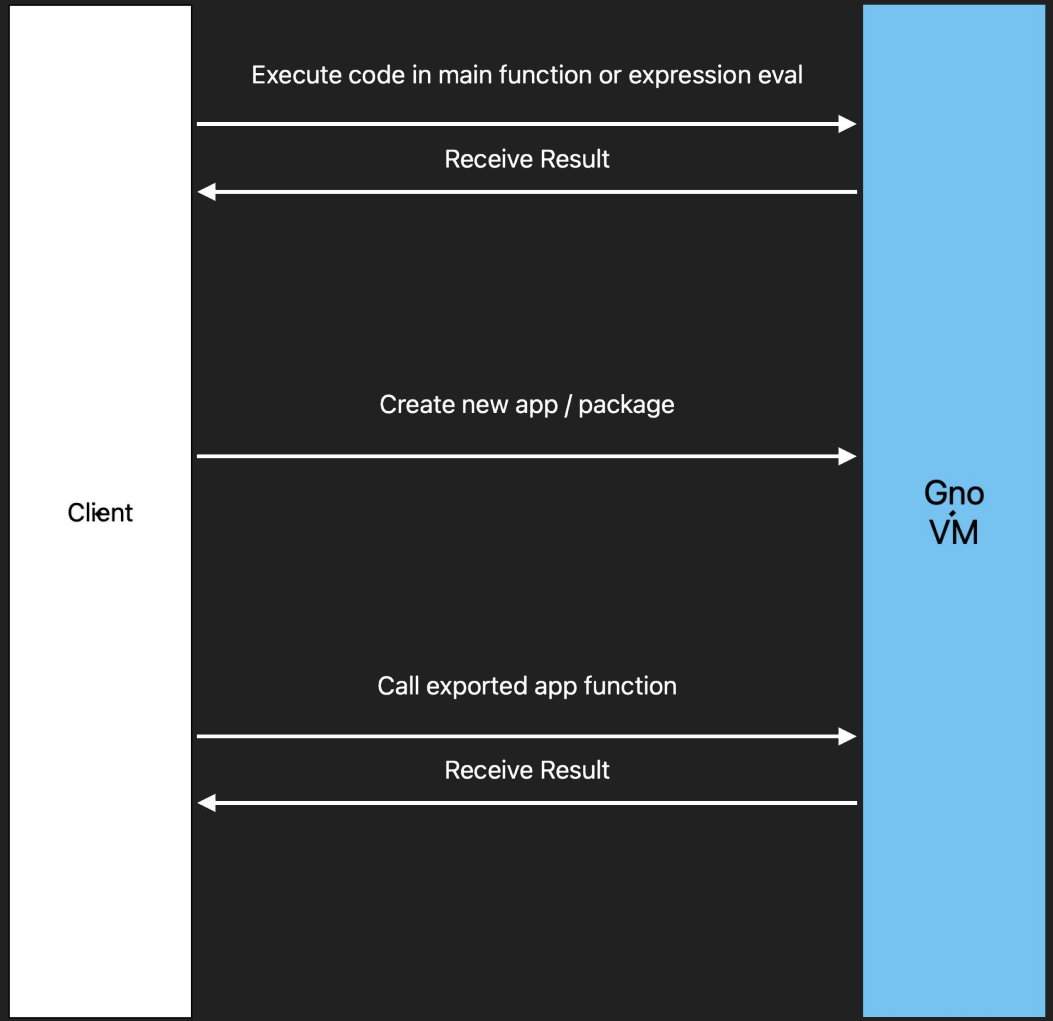
- Define a struct to store an answer
- Answers must be associated with questions
- Where answers are stored is up to you
- Render answers underneath each question
- Should be very similar P2

P3: Checkin

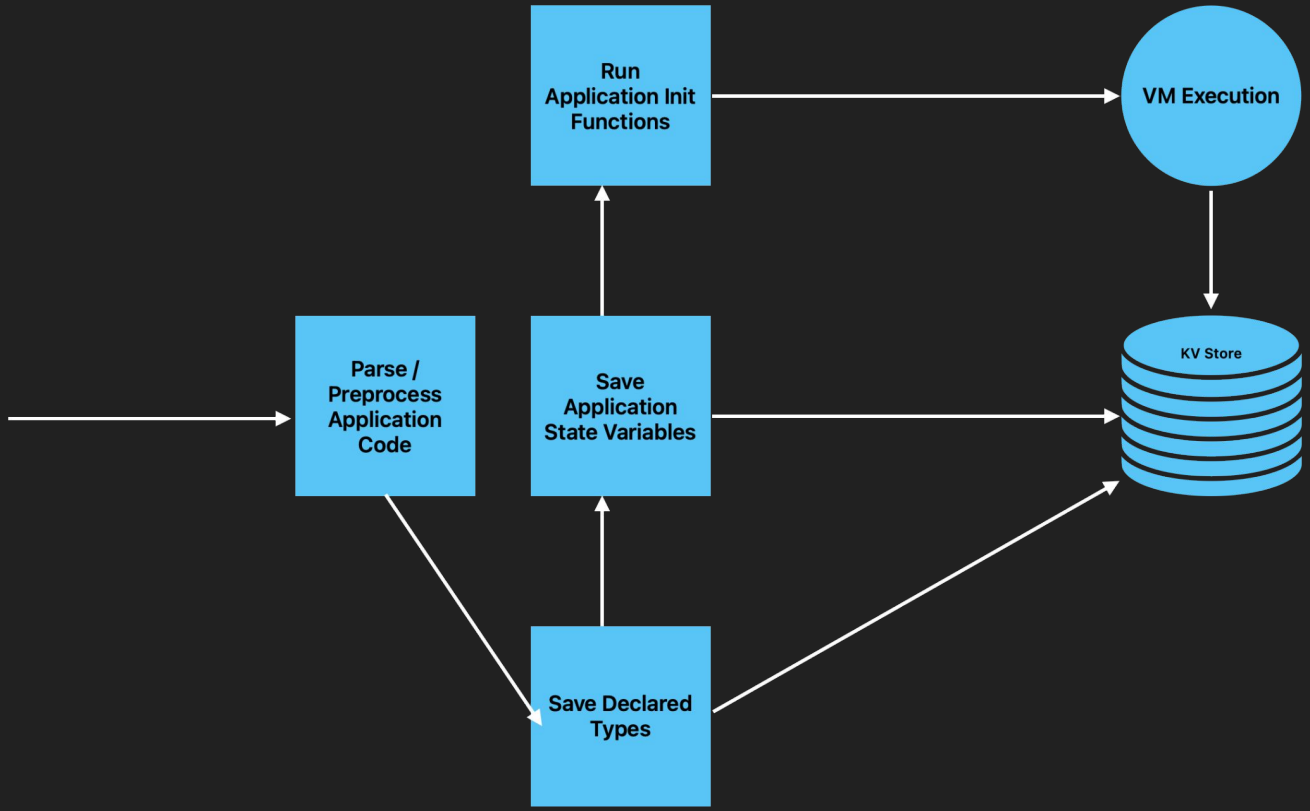
- Answers may want to have a field to store the ID of the question it corresponds to
- How to store Answers?
  - As a slice field of a Question
  - In a separate AVL tree
- How answers are stored will dictate how they need to be iterated over in the Render function
- Should the same address be able to answer a question multiple times or should they be forced to update the existing answer? Your call
- Answer IDs can be global or question specific depending on planned features

P3: What have we accomplished?

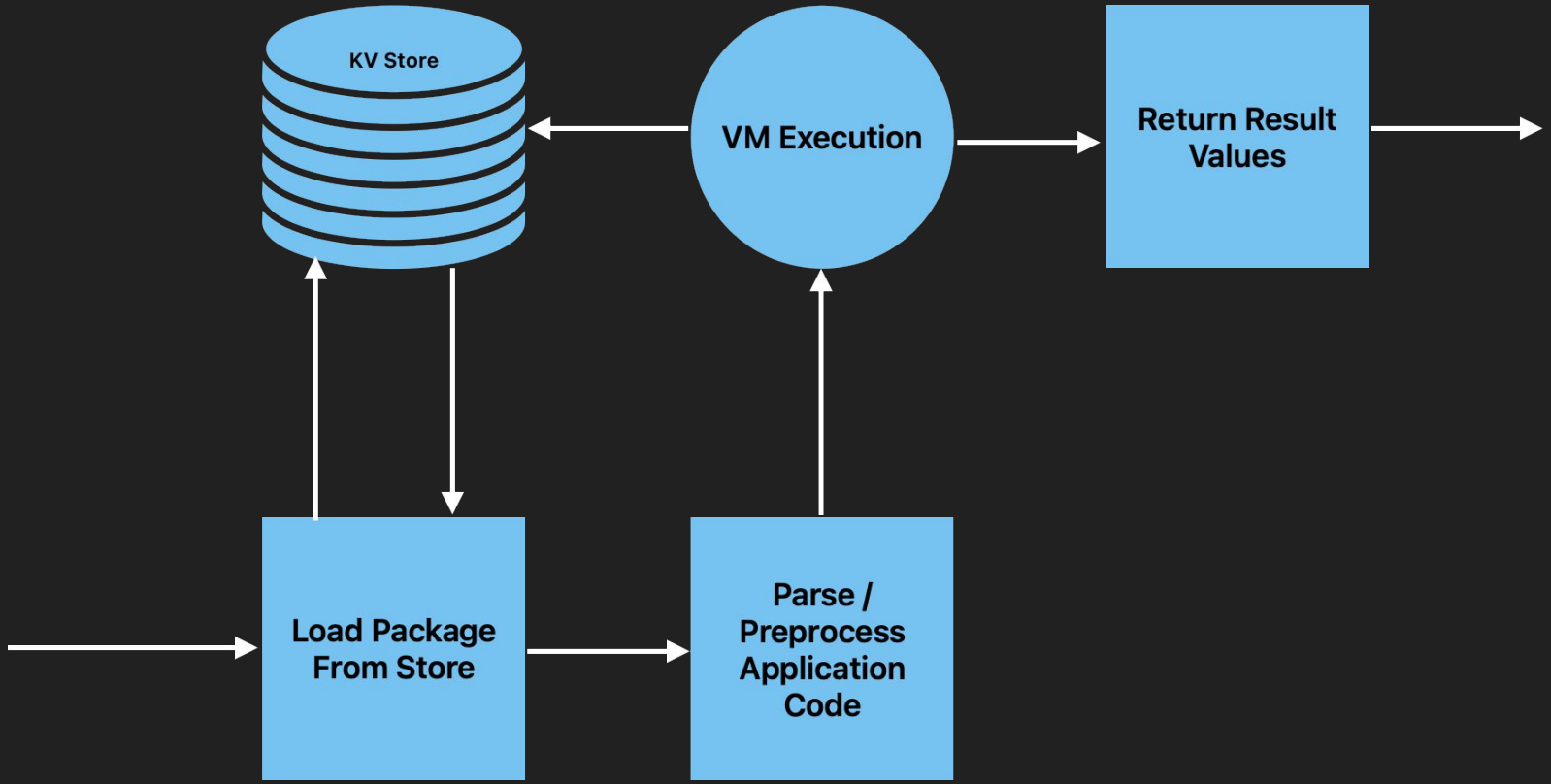
- Questions can now be answered
- Answers can be modified
- Answers are stored on the gno.land blockchain
- It’s starting to look a bit more useful


Client
Execute code in main function or expression eval
Receive Result
Create new app / package
Call exported app function
Receive Result
Gno
VM

Realm Creation Flow



Realm Call Flow



Hands On: Phase 4

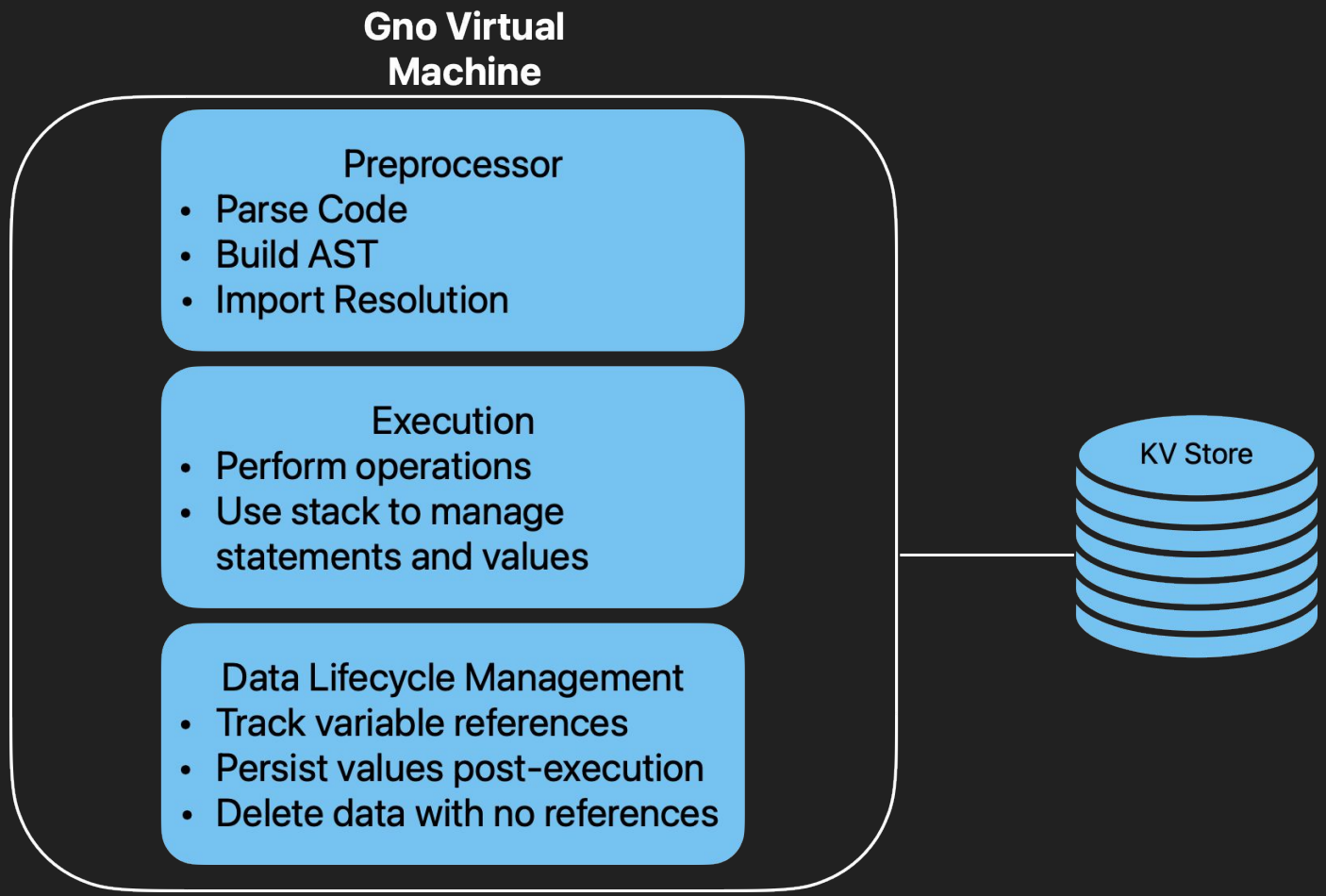
- Define a moderator that can lock questions and answers
  - Hint: global variable assignments and `init()` functions are only execution at the time of realm creation
- Upvotes
  - Questions and answers can both be upvoted
  - Only one upvote per address is allowed
- Update the Render function to display
  - If something is locked
  - The number of upvotes

P4: Checkin

- `var Moderator = std.PrevRealm().Addr()` will set Moderator equal to the address creating the realm. When using gnodev, this is the address of test1
- How should upvotes be stored? Does the fact that only one upvote is allowed per address influence this decision?
- We have been using the uintavl package for uint64 -> value mappings. There is also the gno.land/p/demo/avl package for string -> value mappings.
- When using a new non-stdlib package, don’t forget to add it to the gno.mod file
- The std.Address type has a String() method that returns the address’s string representation

P4: What have we accomplished?

- Moderator is a permissioned role!
  - Only the moderator can lock questions
  - Only the moderator can transfer role ownership
- Imagine how further development could yield tiered and/or elected levels of permission content curation roles
- Adding upvoting functionality will allow question askers and moderators to make more informed decisions when it comes to resolving a question
- Again, all of this data lives on the gno.land blockchain


Gno Virtual Machine
Preprocessor
• Parse Code
• Build AST
• Import Resolution
Execution
• Perform operations
• Use stack to manage statements and values
Data Lifecycle Management
• Track variable references
• Persist values post-execution
• Delete data with no references
KV Store

What is GNOT?

- gno.land’s native token
- Used to pay gas fees
- Can be transferred between addresses
- Can be sent to realm’s during function calls
- Realm code can check if GNOT was sent and act accordingly

Banker

The banker enables realms to interact with GNOT

```
// GetBanker returns a new Banker, with its capabilities matching the given
// [BankerType].
func GetBanker(bt BankerType) Banker {
```

```
type BankerType uint8

// Available types of banker.
const (
	// Can only read state.
	BankerTypeReadonly BankerType = iota
	// Can only send from tx send.
	BankerTypeOrigSend
	// Can send from all realm coins.
	BankerTypeRealmSend
	// Can issue and remove realm coins.
	BankerTypeRealmIssue
```

```
type Banker interface {
	GetCoins(addr Address) (dst Coins)
	SendCoins(from, to Address, amt Coins)
	TotalCoin(denom string) int64
	IssueCoin(addr Address, denom string, amount int64)
	RemoveCoin(addr Address, denom string, amount int64)
}
```

Hands On: Phase 5

- Allow questions to be created with bounties
  - A question is created with a bounty equal to the amount of ugnot specified in the -send flag
  - Ex: `gnokey maketx call -func PostQuestion -send 100000ugnot ...`
- Moderators and question authors can resolve questions
  - No more answers or edits are allowed
  - If a question has a bounty, the bounty is payed out to the chosen answer
- Declutter question rendering
  - Only Render question titles
  - Clicking on a title should load the full question and all answers on a new page
  - Take advantage of the `Render` function's single string argument to specify which question to load

P5: Checkin

- Make use of `std.GetOrigSend` to check if a question should set a bounty
  - The coins sent are automatically deposited in the realm's account
- When a question is resolved, pay out bounties to the address that provided the chosen answer
  - Get a banker of type `BankerTypeRealmSend` to enable transferring of funds
  - The realm's address can be obtained via `std.CurrentRealm().Addr()`
- Use markdown links to link question titles to the full questions
  - The link should be something like /r/gc24/gnoverflow:questions/0
  - The colon separates the realm from the `Render` argument
  - Parse the `Render` argument to determine which question to load
  - Use `strconv.Itoa` to convert the argument to an integer question ID

Gnoverflow Summary

- Decentralized, feature-minimal version of stack overflow supporting
  - Questions
  - Answers
  - Upvotes
  - Moderation
  - Bounties
  - Rendering
- The entire application and its data are stored on the gno.land blockchain
- The code is available for anyone to inspect before interacting with the app
- Bounty payments are built in and governed by the logic of the realm
- No need to worry about:
  - The application author changing the functionality
  - Data being lost

Official website

gno.land

Gno Docs

docs.gno.land

Gno Playground

play.gno.land

Twitter

@_gnoland

Github

gnolang/gno