justin
alvarius
kelvin
LATTICE

***
HOW TO BUILD VERY CRAZY THINGS ON ETHEREUM
THINGS LIKE VIRTUAL WORLDS
WORLDS WHOSE RULES RUNS ON THE EVM, AND WHOSE STATE IS SECURED BY ETHEREUM
ON-CHAIN GAMES

AUTONOMOUS WORLDS
PROBLEM: BUILDING LARGE ON-CHAIN PROJECTS IS HARD
TODAY: TWO NEW TECHNOLOGIES FROM 🔄 AND 🔴 TO ENABLE AUTONOMOUS WORLDS
MUD: AN ENGINE FOR AUTONOMOUS WORLDS
SOLVING ALL THE HARD PROBLEMS OF BUILDING ON-CHAIN GAMES
STATE SYNC

ADDING CONTENT

INTER OPERABILITY
struct Monster {
  uint8 type;
  uint32 health;
  uint32 attack;
}

custom struct per entity type
OLD APPROACH
STATE SYNC

interface Monster {
  type: number;
  health: number;
  attack: number;
}

duplicate structs on the client

Contracts

struct Monster {
  uint8 type;
  uint32 health;
  uint32 attack;
}

Client

interface Monster {
  type: number;
  health: number;
  attack: number;
}
OLD APPROACH
STATE SYNC

function getMonsters() public view returns (Monster[] memory);

load initial state via custom getter functions

Contracts

struct Monster {
  uint8 type;
  uint32 health;
  uint32 attack;
}

function getMonsters() public view returns (Monster[] memory);

Client

interface Monster {
  type: number;
  health: number;
  attack: number;
}
OLD APPROACH
STATE SYNC

event MonsterHealth(
  uint256 id,
  uint32 health,
);

update client state via custom events
OLD APPROACH
ADDING CONTENT

```solidity
struct Plant {
  uint8 type;
  uint32 health;
}

function getPlants() public view returns (Plant[] memory);

event PlantHealth(
  uint256 id,
  uint32 health,
);

interface Plant {
  type: number;
  health: number;
}

contract.on(
  "PlantHealth", 
  (id, health) => {
    plants[id].health = health;
  }
);
```

modify entire network stack to handle new content
OLD APPROACH
INTEROPERABILITY

manual or via existing (DeFi) interfaces
NEW APPROACH

ENTITY COMPONENT SYSTEM
NEW APPROACH

`uint256 entity;`

an entity is just a `uint256` id
NEW APPROACH

Component.sol

entity1: value1,
entity2: value2,
entity3: value3,
...
entityN: valueN,

components link entities to values
NEW APPROACH

MoveSystem.sol

“Move entity left”

PositionComponent.sol
entity: { x: 13, y: 37 }

PositionComponent.sol
entity: { x: 12, y: 37 }

systems execute logic based on components
(NEW) APPROACH

TransferSystem.sol

"Transfer 1 DAI"

BalanceComponent.sol

- address1: 42 DAI
- address2: 0 DAI

BalanceComponent.sol

- address1: 41 DAI
- address2: 1 DAI

existing standards (ERC20/721/...) could almost be called ECS
NEW APPROACH
STATE SYNC

registerValueSet(value)

Component1.sol
Component2.sol
Component3.sol

World.sol

Client

components register state updates in the world
NEW APPROACH
STATE SYNC

Client

emit ValueSet(value)

Component1.sol

Component2.sol

World.sol

Component3.sol

client listens to a single world event stream
NEW APPROACH
STATE SYNC

new Component(id);
defineComponent(id);

CONTRACTS

CLIENT
NEW APPROACH
STATE SYNC

emit ValueSet(value)

Component1.sol

Component2.sol

Component3.sol

World.sol

Indexer

Client
generic indexers can reduce RPC load
NEW APPROACH
ADDING CONTENT

Health Component
100

Attack Component
30

Movable Component
true

entities are collections of component values

Fighter
NEW APPROACH
ADDING CONTENT

Health Component: 800
Attack Component: 200
Movable Component: true

Dragon

add entities by setting new component values
NEW APPROACH

ADDING CONTENT

Health Component

400

Attack Component

30

Defense tower

add entities by recombining existing components
NEW APPROACH
ADDING CONTENT

Healing Shrine
- Healing
- Position

Healer
- Healing
- Position
- Movable

Healing Potion
- Healing
- Consumable

add entities by adding new components
NEW APPROACH
INTEROPERABILITY
NEW APPROACH
INTEROPERABILITY
NEW APPROACH
INTEROPERABILITY

INTEROPERABILITY NEEDS INTERFACES TO SCALE
NEW APPROACH
INTEROPERABILITY

MUD is an interface for on-chain worlds
OLD APPROACH
INTEROPERABILITY

ERC 721
interface
for ownership

“How many entities does this address own?”
balanceOf(address);

“Who owns this entity?”
ownerOf(uint256);
NEW APPROACH INTEROPERABILITY

“Give me all movable attack entities owned by this address”

Query(
    HasValue(Owner, address),
    Has(Attack),
    Has(Movable)
);
MUD IS GENRE AGNOSTIC
MUD IS GENRE AGNOSTIC
SKY STRIFE

ON-CHAIN RTS
39 COMPONENTS
22 SYSTEMS
0 NETWORKING CODE
UNANNOUNCED

ON-CHAIN VOXEL GAME
8 COMPONENTS
7 SYSTEMS
0 NETWORKING CODE
MUD

STATE SYNC
ADDING CONTENT
INTEROPERABILITY
ALL PROBS SOLVED
FULL NODES ARE GREAT
WITH A FULL NODE YOU CAN:

- **ACCESS STATE OF THE CHAIN DIRECTLY FROM THE NODE DB**
- **SIMULATE TRANSACTIONS**
• TRADITIONAL DAPP CLIENTS ARE NOT FULL NODE!
• THEY RELY ON INFURA/ALCHEMY TO SERVE THEIR DATA
• KEEP A COPY OF THE STATE CLIENT SIDE, OFTEN WITH LOTS OF CODE
• CLIENT CONNECTS TO FULL NODE
• CACHE STATE IT IS INTERESTED IN
• KEEPS IT IN SYNC
• WITH LOTS OF CUSTOM CODE AND INDEXERS
browser

client code

full-node

indexer

WHY NOT??

- RUN INDEXER CLIENT SIDE
- SIMULATE TXS
- NO NETWORK DELAY AFTER SYNC!
FULL NODES ARE EXPENSIVE!
BANDWITH, STORAGE
1. UX-HURTING NETWORK CALLS
2. WAIT FOR MINED TX TO SHOW SIDE EFFECTS
3. REMOTE INDEXERS
CAN WE DO BETTER?
AUTONOMOUS WORLDS ARE MOSTLY STANDALONE, UNLIKE TRADITIONAL DAPPS
SIMULATING TX REQUIRES KNOWING STATE OF OTHER SMART CONTRACTS, LIKE ERC-20s ON BOTH SIDES OF THE POOL
MUD SYNCS A WORLD, A NAMESPACE FOR DATA AND LOGIC

DATA = COMPONENTS
LOGIC = SYSTEMS
- Initial sync via MUD Indexer or Full Node
- Keep state up to date via Full Node or MUD Stream Service
struct Position {
    int64  x;
    int64  y;
}

COMPONENTS ARE SELF-DESCRIPTIVE. MUD READS THEIR ON-CHAIN SCHEMA
**FULL-NODE**

```plaintext
contract 0xA2F..1
0x0: 0xFAB6....81
0x1: 0x1AF0....D1
contract 0xAE1..4
0x0: 0x0013....6A
```

**MUD**

```plaintext
0x0: [Position(12,45), Health(200)]
0x1: [CanFly(), Health(10)]
0x2: [Position(1, -4), Balance(100)]
```
RUN COMPLEX QUERIES ON COMPONENTS WITHOUT NETWORK DELAY

runQuery(Has(Position), HasValue(Health, { balance: 10 })))
TX

Move(0x01, {x: 10, y: -3})

LOCAL EVM

runEVM(tx, state)

SIDE EFFECTS

0x01: Position(x: 10, y: -3)

RECONCILE

Predicted side effects OK! 0xD0...FA2

chain
TX

Move(0xFE, {x: -20, y: 4})

LOCAL EVM

runEVM(tx, state)

SIDE EFFECTS

0xFE: Position(x: -20, y: 4})

RECONCILE

Wrong! Revert and apply

chain

0xA4...8C2
MUD

- Read / Index Components without Network Delay
- Simulate TX without Network Delay
EXTENDING WORLDS WITH MUD
DEVS CAN EXTEND PROTOCOLS VIA SMART CONTRACTS AND NEW CLIENT — THAT’S THE POWER OF FULLY ON-CHAIN PERMISSIONLESS APPS
HOWEVER, DEVELOPERS NEED TO SHIP NEW CLIENTS AND INDEXERS

USERS ALSO NEED TO KNOW WHERE THOSE CONTRACTS AND NEW CLIENTS ARE
1ST PARTY VS 3RD PARTY
EXAMPLE:
DARK FOREST EXTENSION: PLANETS CAN BE “REWARDING”, CAPTURING THEM GIVES YOU $ETH
PROBLEM:

HOW DO USERS KNOW THIS EXISTS?
WHERE ARE THE CONTRACTS?
HOW DOES THE CLIENT KNOW WHAT TO DO WITH THE DATA?
INDEXERS?
CAN WE DO BETTER?
World.sol

PERMISIONLESS

NO OWNER
NO 1ST PARTY OR 3RD PARTY
CREATORS OF THE WORLD HAVE NO POWER
ANYONE CAN CREATE COMPONENTS AND SYSTEMS
MUD

ANYONE CAN CREATE NEW COMPONENTS (DATA) AND SYSTEMS (LOGIC) THAT:

- ARE ACCESSIBLE IN THE CLIENT
- ARE INDEXED
- ARE IN THE DEBUGGER
- CAN BE EXECUTED IN THE LOCAL EVM
ALL SYSTEMS CAN READ ANY COMPONENT
ONLY RULE: COMPONENTS WHITELIST SYSTEMS THAT CAN WRITE TO THEIR STATE
VERY IMPORTANT IDEA: AUGMENTED REALITY
Beyond the core components and systems all players believe in, it possible to create augmented reality layers that a subset of players will engage with, permissionlessly.
LET’S ILLUSTRATE
COMPONENTS
- Position
- Movable
- Resource
- Stake
- Board

Core Team
- Team TTT

INSTALL?
- Challenge()
- AcceptChallenge()
- Resolve()
- Stake
- Board

SYSTEMS
- Move()
- PickupResource()
- Drop()

Core Team
- Team TTT
COMPONENTS

- Position
- Movable
- Resource
- Stake
- Board

Core Team

Team TTT

SYSTEMS

Move() PickupResource() Drop()

Challenge() AcceptChallenge() Resolve() Team TTT

transfer 1$ETH
TTTT IS JUST LIKE TENNIS! IT’S AN AUGMENTED REALITY
OTHER PLAYERS
AUGMENTED REALITIES: CAPITALISM COMPETITION MINI-GAMES
ALL PERMISIONLESSLY
WORLD IS OWNERLESS
SO YOU WANT TO BUILD AN AUTONOMOUS WORLD

@KELVINFIGHTER
BUILDING THE OPTIMISM COLLECTIVE
SOMETHING NEW

INTRODUCING THE OP STACK
SOMETHING NEW

INTRODUCING THE OP STACK*

*we need like three months to write the docs
THE OP STACK
OVERVIEW

ROLLUPS
GONE
MODULAR
THE OP STACK
OVERVIEW

CONSENSUS

EXECUTION

SETTLEMENT
THE OP STACK
OVERVIEW

CONSENSUS

EXECUTION

SETTLEMENT
THE OP STACK
OVERVIEW

CONSENSUS

EXECUTION

SETTLEMENT
THE OP STACK
CORE CONCEPTS

MODULAR THEORY IN PRACTICE
THE OP STACK
CORE CONCEPTS

DATA AVAILABILITY
PUBLISH DATA ANYWHERE
DERIVATION
TRANSACTIONS FROM ANYTHING
THE OP STACK
CORE CONCEPTS

EXECUTION
RUN EVERYTHING
THE OP STACK
CORE CONCEPTS

SETTLEMENT
SEND ASSETS EVERYWHERE
THE OP STACK
SHARED SEQUENCING

NO SEQUENCER?
NO PROBLEM.*

*currently a very big problem
THE OP STACK
SHARED SEQUENCING
THE OP STACK
SHARED SEQUENCING
THE OP STACK

SHARED SEQUENCING
THE OP STACK
SHARED SEQUENCING
WHY MAKE IT FOSS?
THE OP STACK
MOTIVATION

BECAUSE
IT HAS TO BE.
THE OP STACK
MOTIVATION
THE OP STACK
MOTIVATION
GO NUTS,
BUILD SOMETHING CRAZY.
AND THANKS FOR COMING TO MY TED TALK
POWERED BY MUD
RUNNING ON OP STACK
PROCEDURAL WORLD
RELEASED TODAY!
AW ARCADE: 4PM TODAY
HACKER BASEMENT
SKY STRIFE
4PM HACKER BASEMENT
MUD
AN ENGINE FOR AUTONOMOUS WORLDS
MUD.DEV
SOON

OP STACK