justin

alvarius

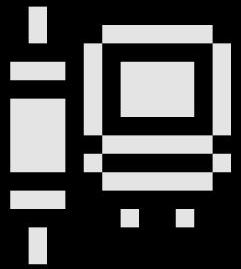
kelvin

LATTICE‡‡‡

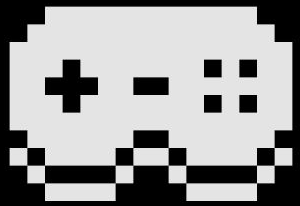
* * * * * *

HOW TO BUILD VERY CRAZY THINGS ON ETHEREUM

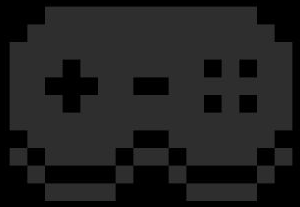
THINGS LIKE VIRTUAL WORLDS



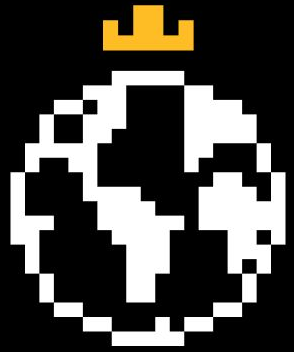
WORLDS WHOSE RULES RUNS ON THE EVM, AND WHOSE STATE IS SECURED BY ETHEREUM



ON-CHAIN GAMES



ON-CHAIN GAMES

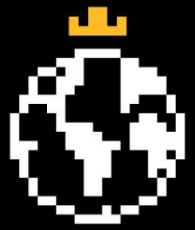


AUTONOMOUS
WORLDS

PROBLEM: BUILDING LARGE ON-CHAIN PROJECTS IS HARD

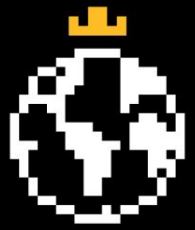
TODAY: TWO NEW TECHNOLOGIES FROM  AND 

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

STATE
SYNC

ADDING
CONTENT

INTER
OPERABILITY

STATE
SYNC

ADDING
CONTENT

INTER
OPERABILITY

OLD APPROACH
STATE SYNC

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

custom struct per entity type

Contracts

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

Client

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

Contracts

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


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


event MonsterHealth(
  uint256 id,
  uint32 health,
);
```

Client

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


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

OLD APPROACH
ADDING CONTENT

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

modify entire network stack to handle new content

Contracts

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


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


event PlantHealth(
  uint256 id,
  uint32 health,
);
```

Client

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


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

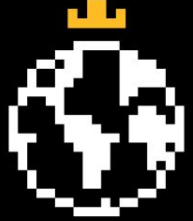
OLD APPROACH
INTEROPERABILITY

manual or via existing (DeFi) interfaces

NEW APPROACH

YOU

FUN
CREATIVITY
NOVELTY
CONTENT
DESIGN



MUD

COMMON
HARD
PROBLEMS

NEW APPROACH

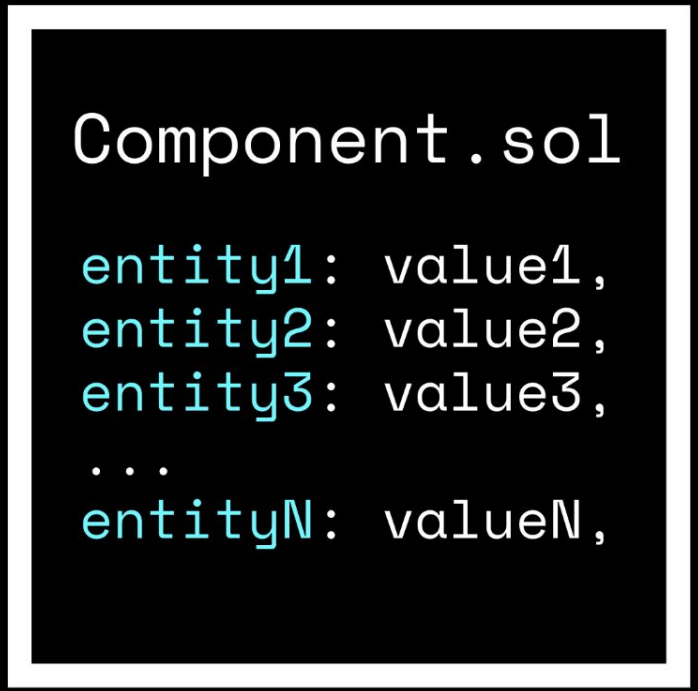
ENTITY COMPONENT SYSTEM

NEW APPROACH

```
uint256 entity;
```

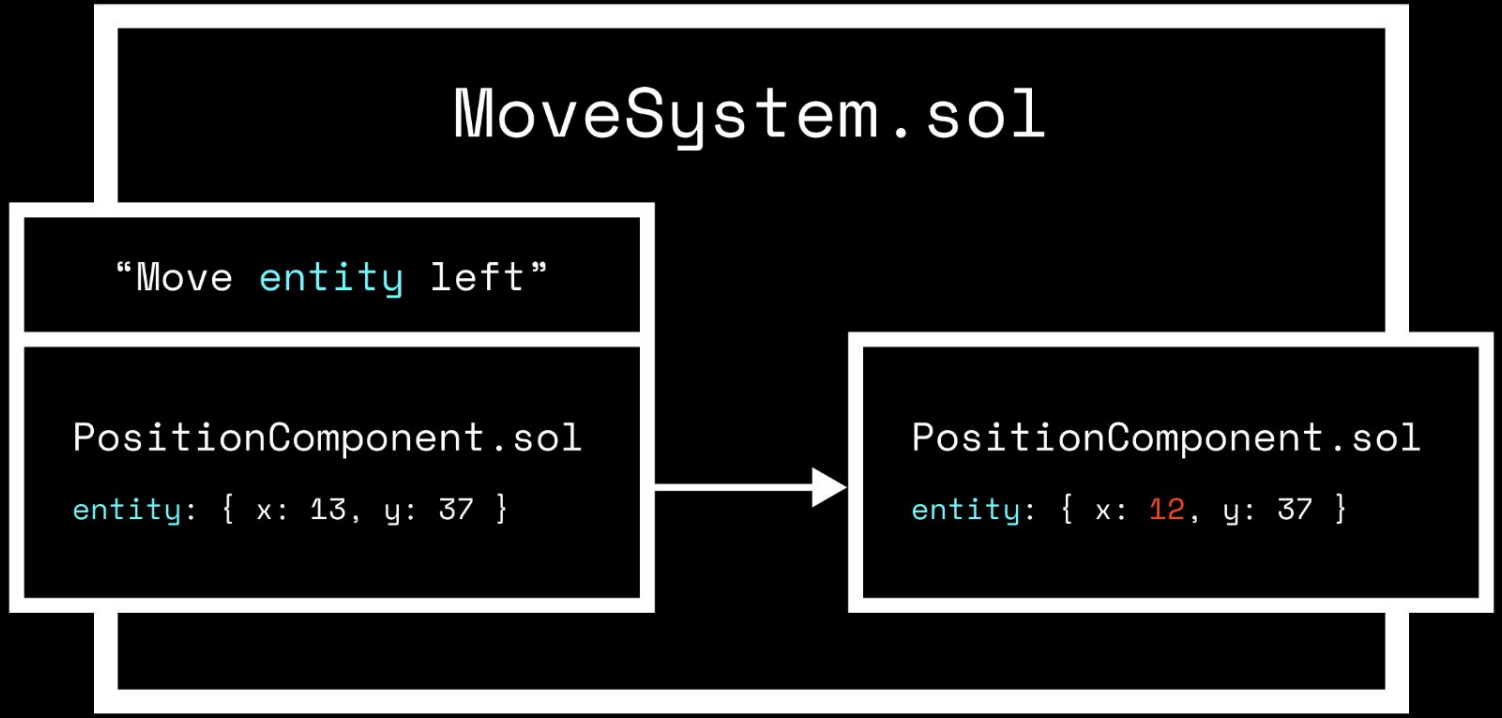
an entity is just a uint256 id

NEW APPROACH



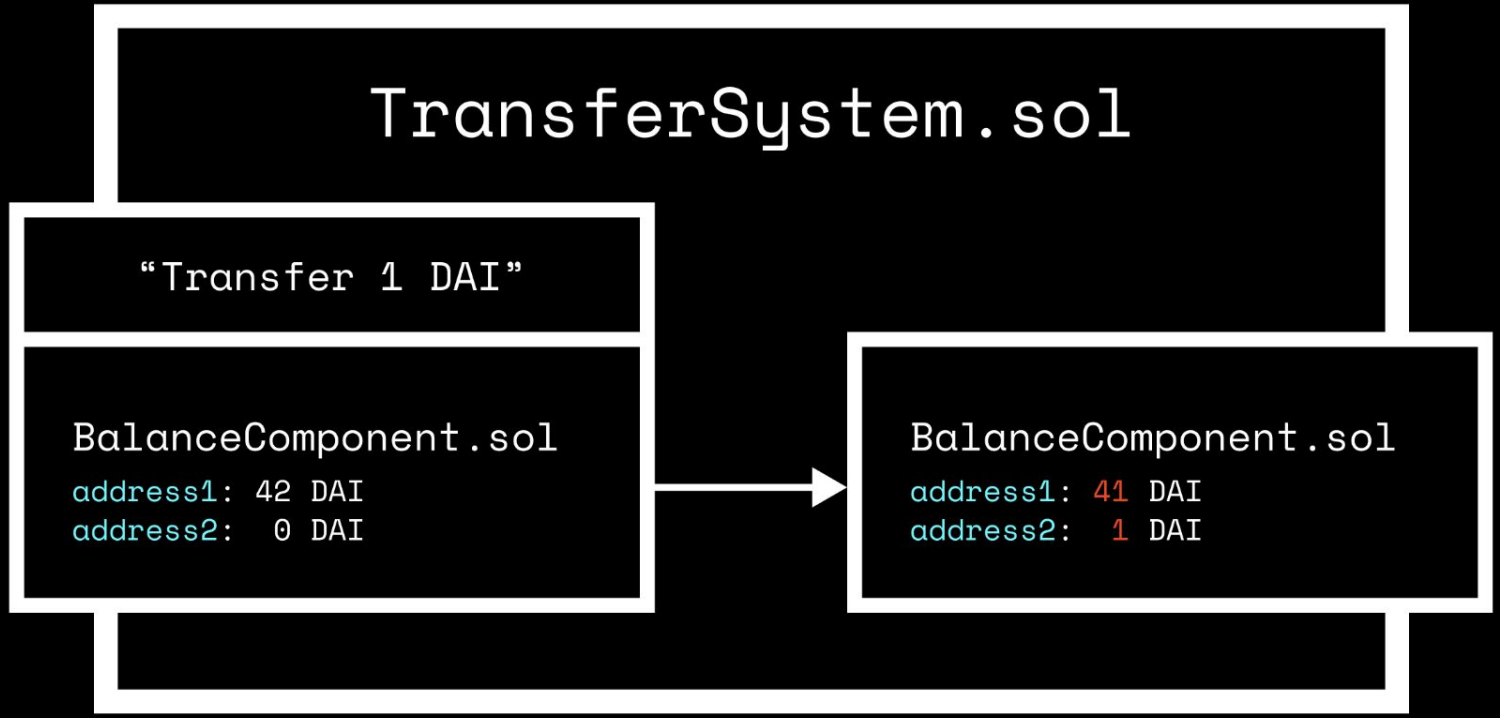
components link entities to values

NEW APPROACH



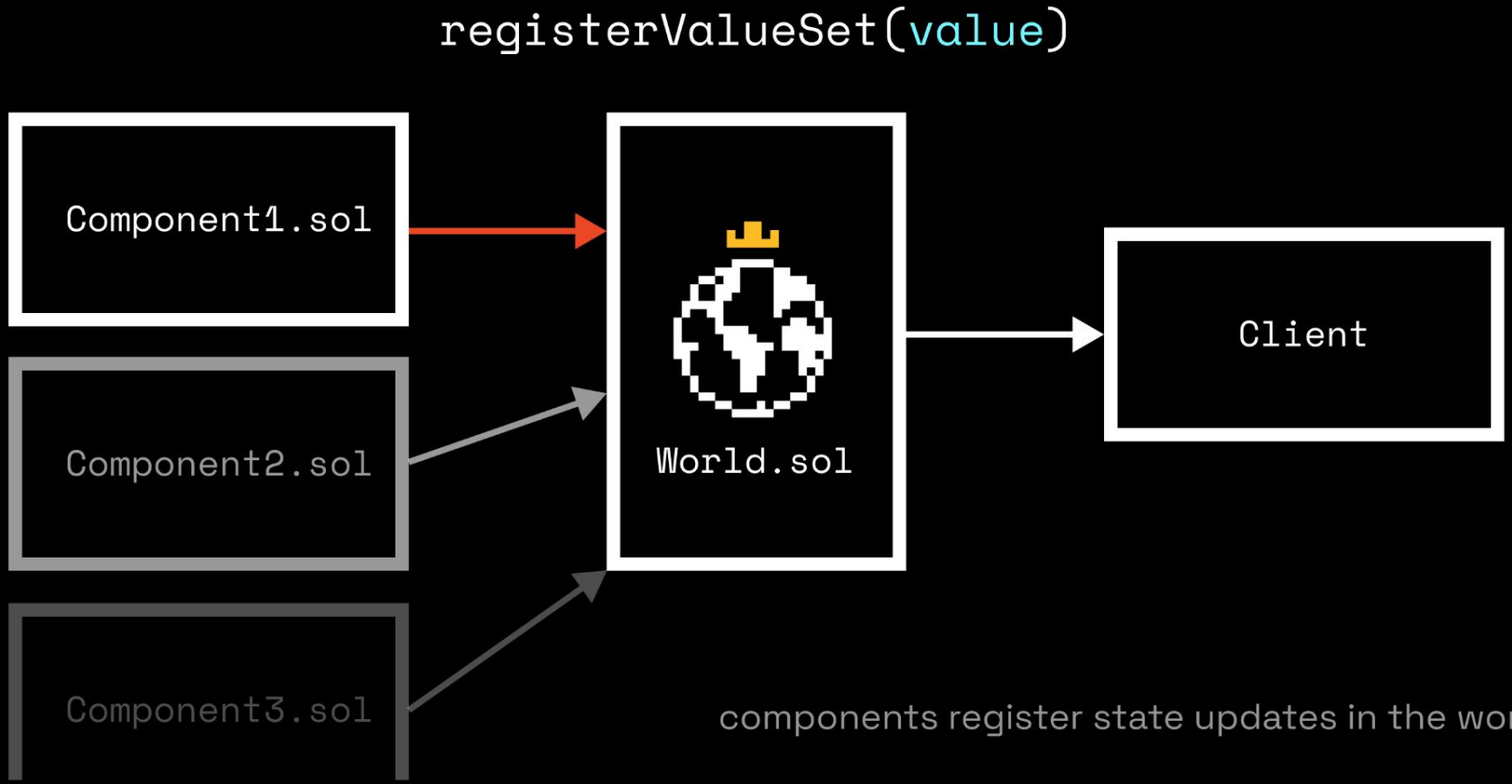
systems execute logic based on components

(NEW) APPROACH



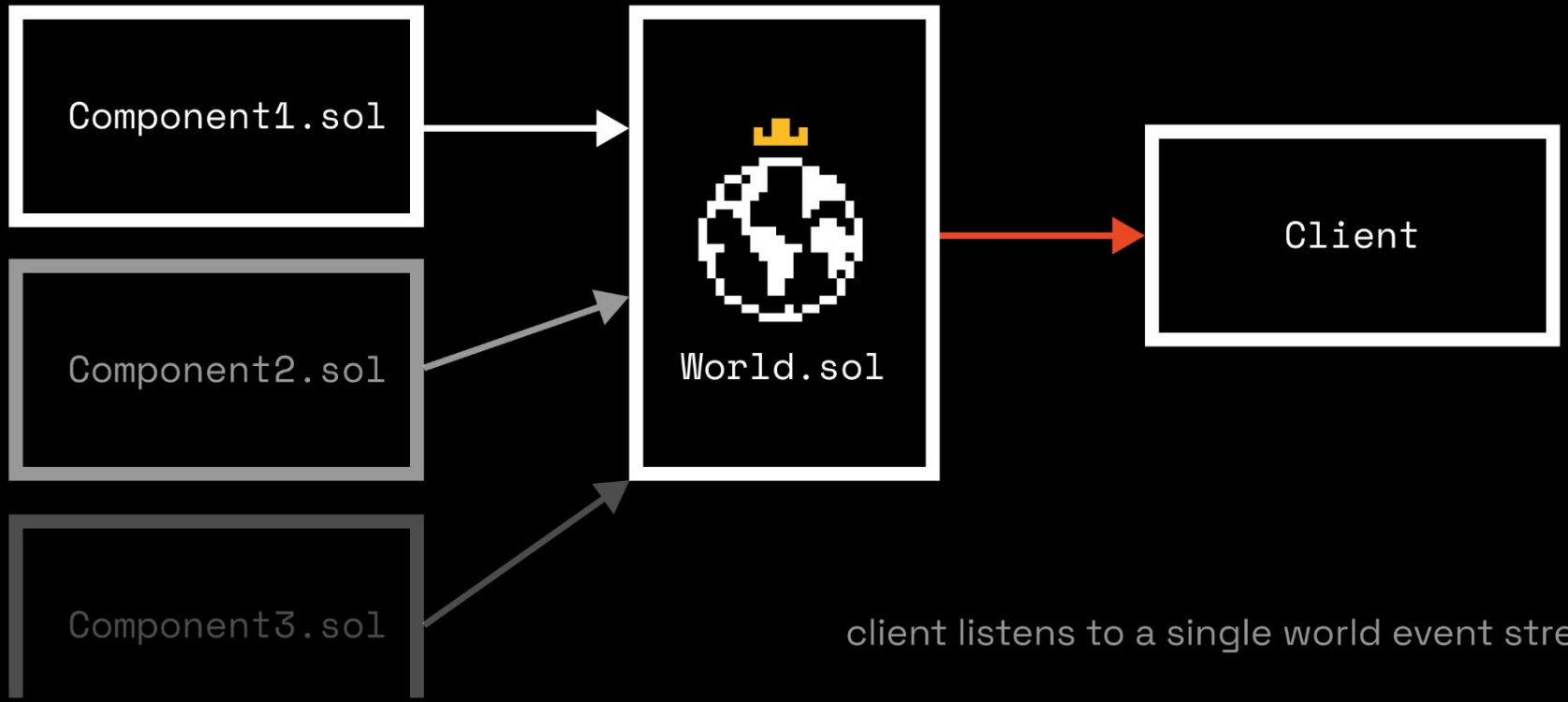
existing standards (ERC20/721/...) could almost be called ECS

NEW APPROACH
STATE SYNC



NEW APPROACH
STATE SYNC

```
emit ValueSet(value)
```



client listens to a single world event stream

NEW APPROACH
STATE SYNC

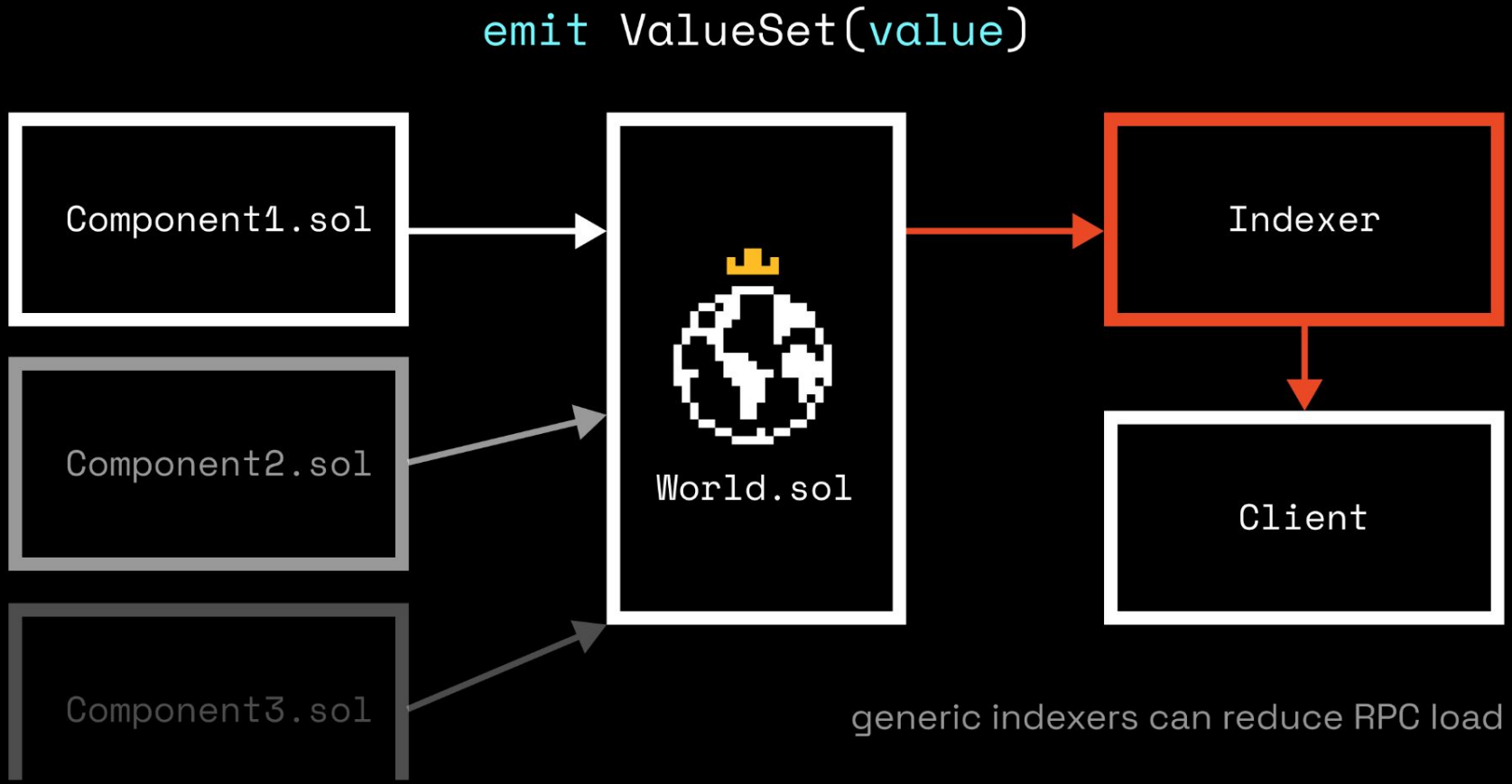
```
new Component(id);
```

CONTRACTS

```
defineComponent(id);
```

CLIENT

NEW APPROACH
STATE SYNC



NEW APPROACH
ADDING CONTENT

100

30

true

Fighter

entities are collections of component values

NEW APPROACH
ADDING CONTENT

Health Component — 800

Attack Component — 200

Movable Component — true

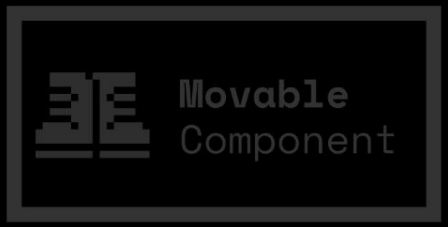
Dragon

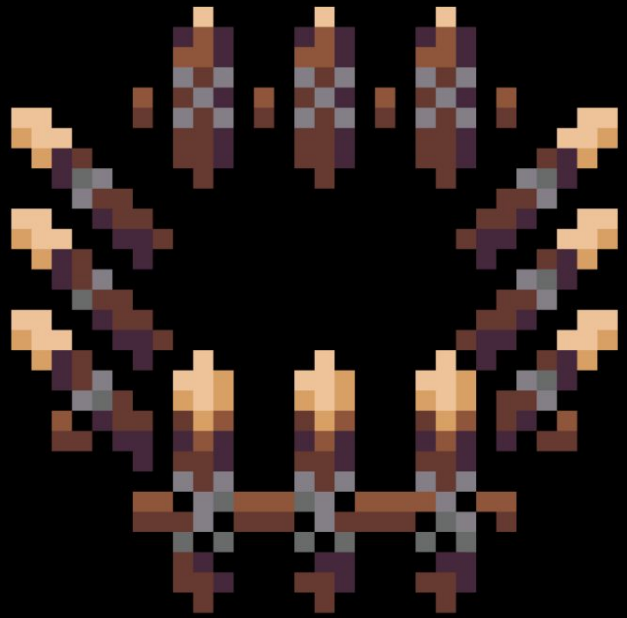
add entities by setting new component values

NEW APPROACH
ADDING CONTENT

400

30





Defense tower

add entities by recombining existing components

NEW APPROACH
ADDING CONTENT

Healing Shrine

- Healing
- Health
- Position

Healer

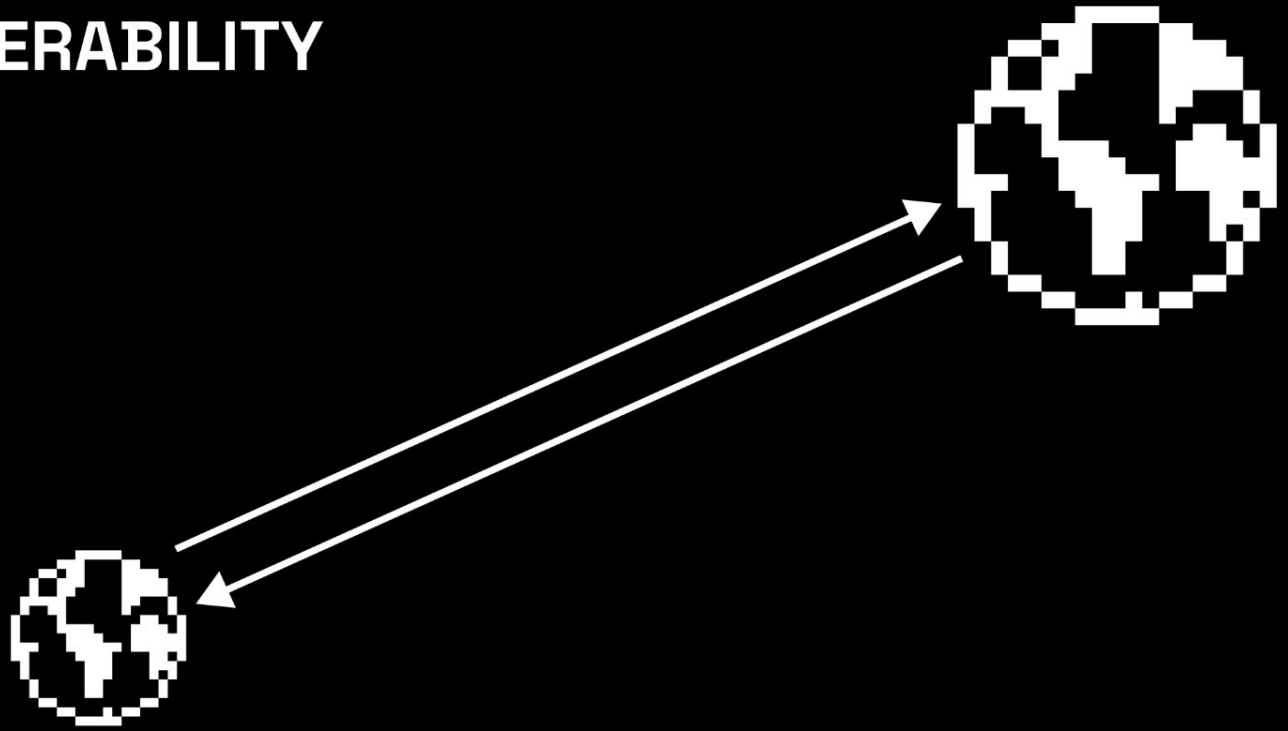
- Healing
- Health
- Position
- Movable

Healing Potion

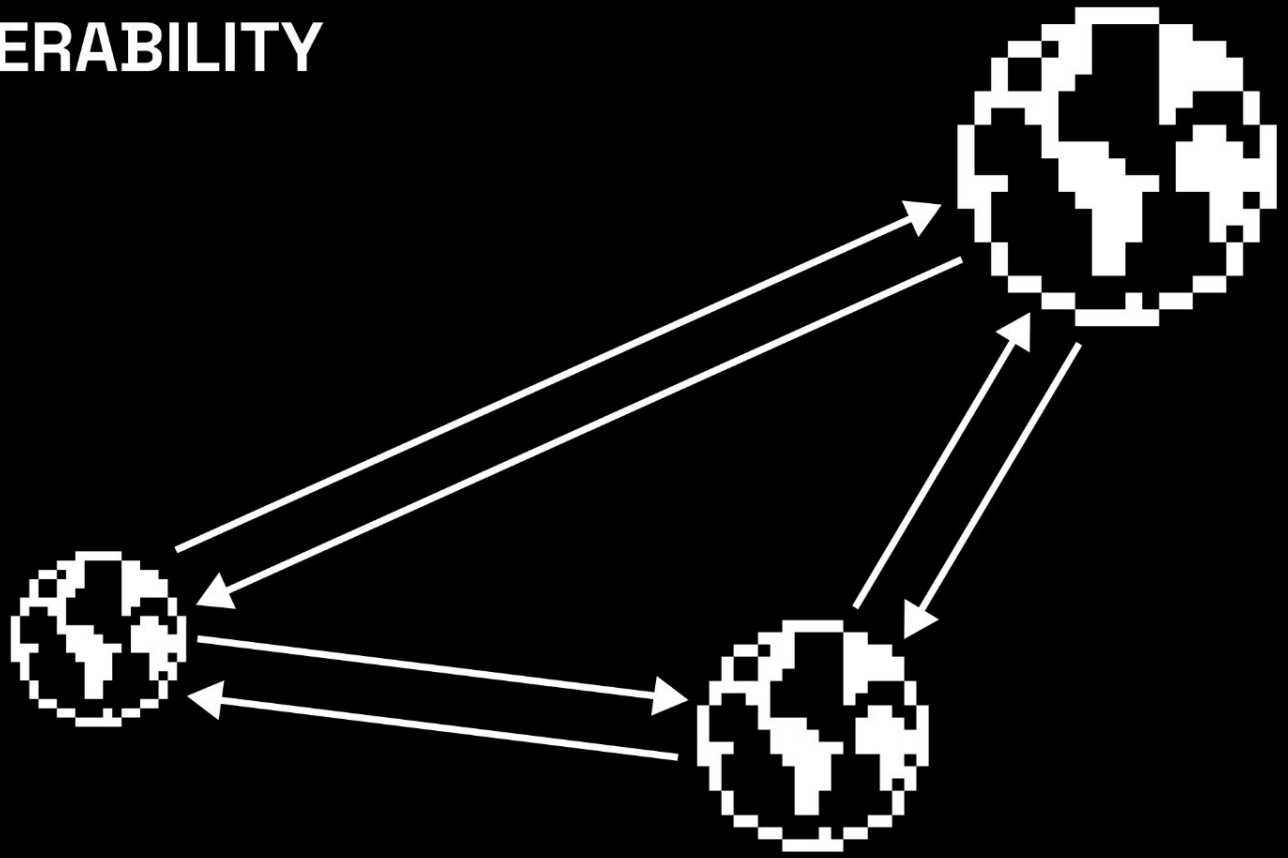
- Healing
- Consumable

add entities by adding new components

NEW APPROACH
INTEROPERABILITY



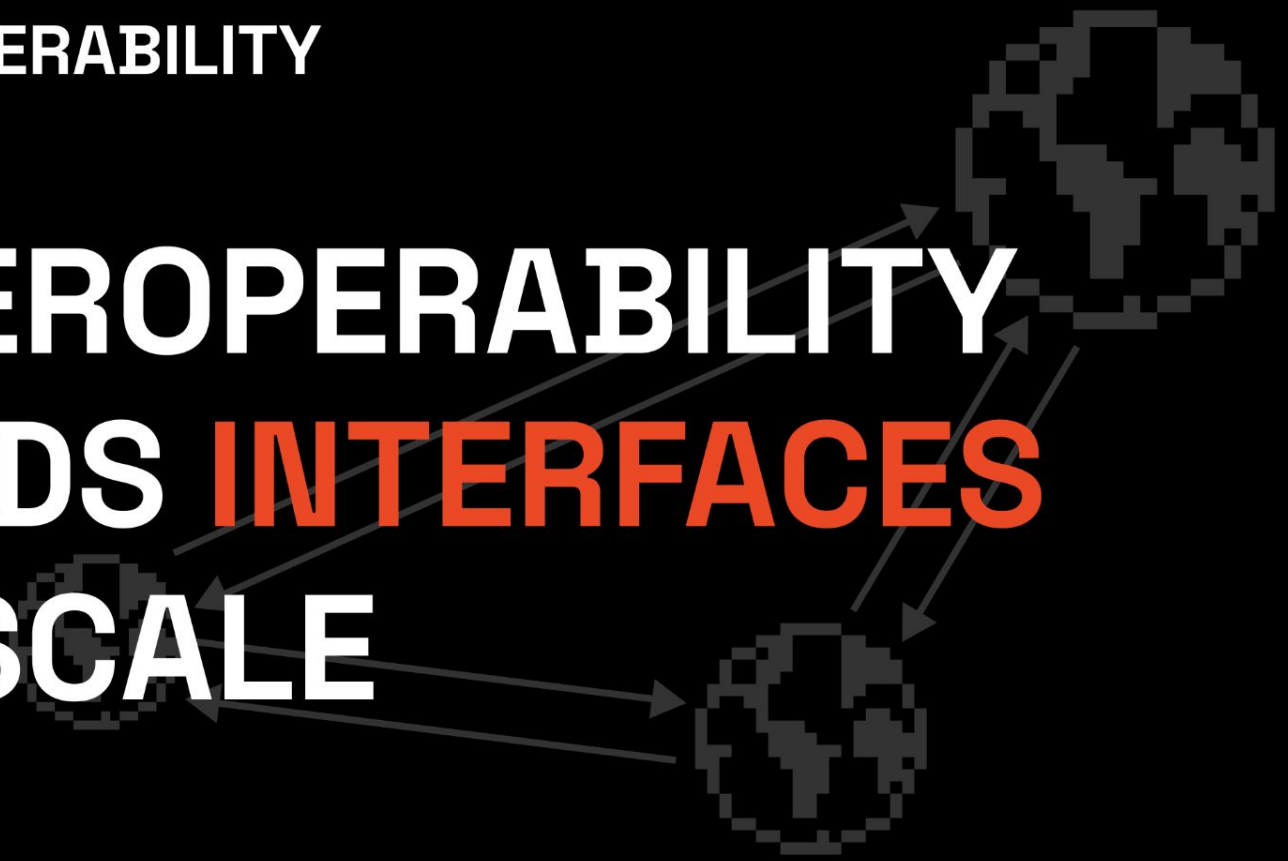
NEW APPROACH
INTEROPERABILITY



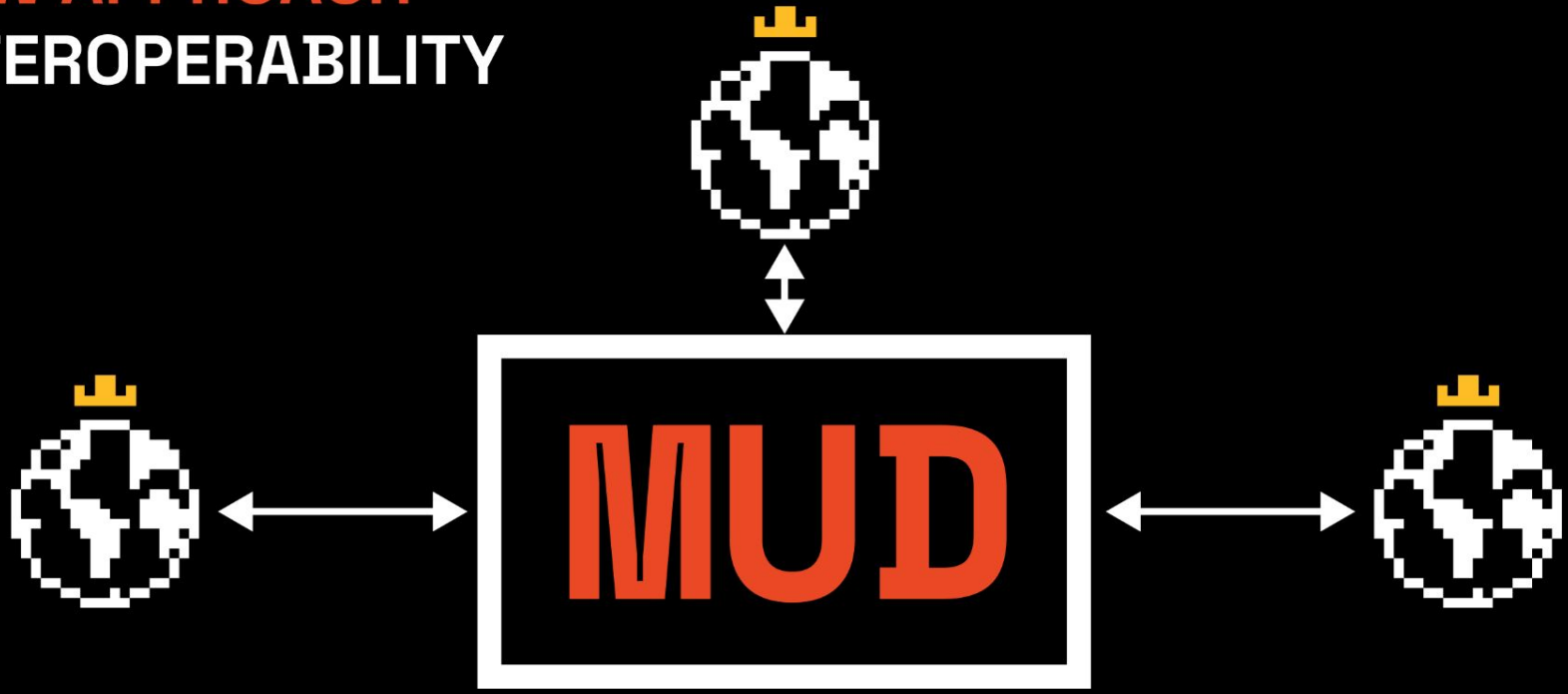
NEW APPROACH

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);
```

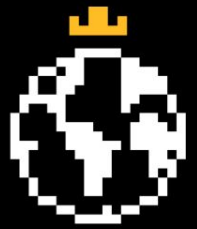
NEW APPROACH
INTEROPERABILITY

MUD

interface
for **anything**

"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

UNANOUNCED

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

Golem

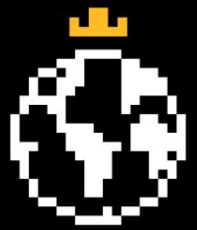
- Position
- Movable
- Stamina
- Combat
- Inventory

SKY STRIFE

VOXEL GAME

Grass

- Position
- Item



MUD

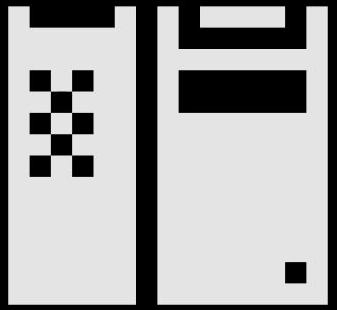
STATE SYNC

ADDING CONTENT

INTEROPERABILITY

ALL PROBS SOLVED

FULL NODES ARE GREAT

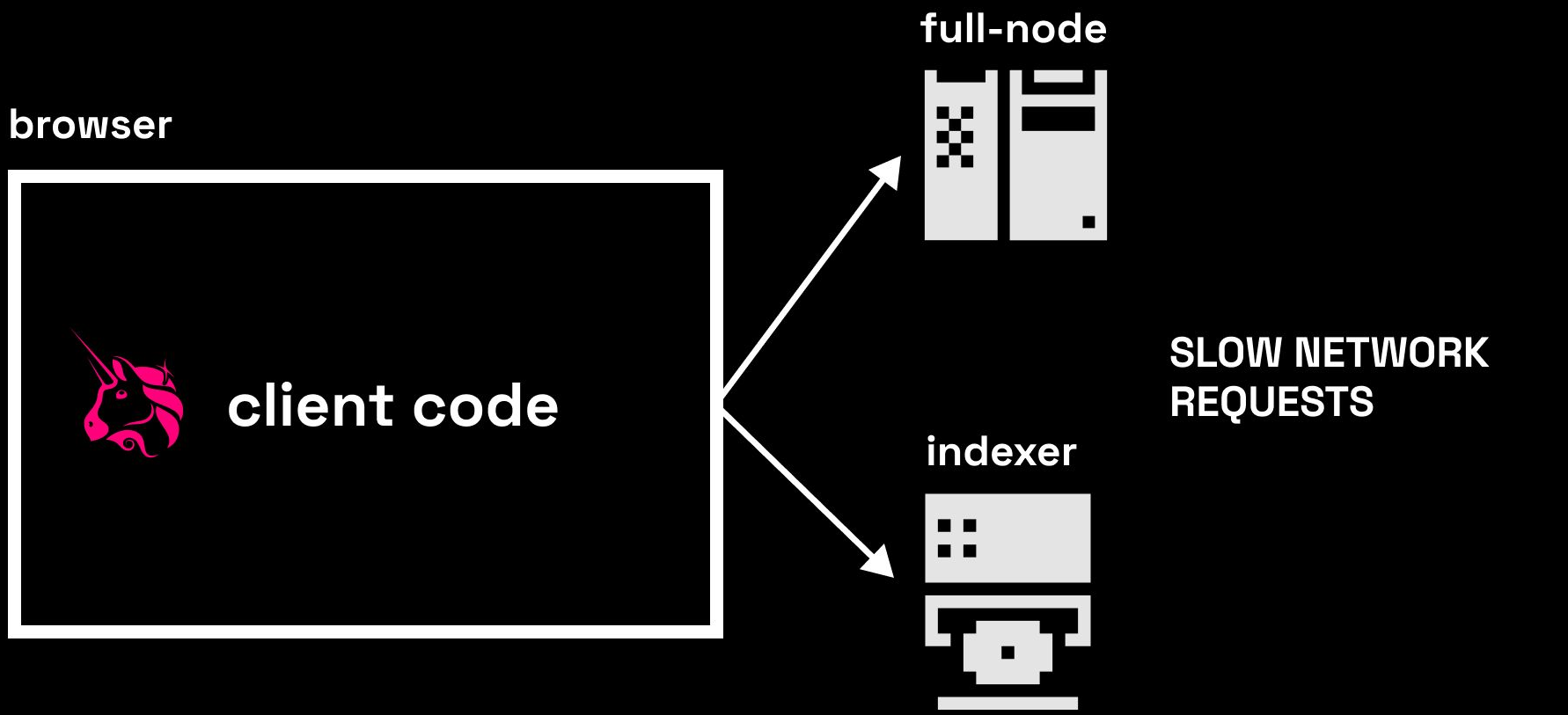


WITH A FULL NODE YOU CAN:

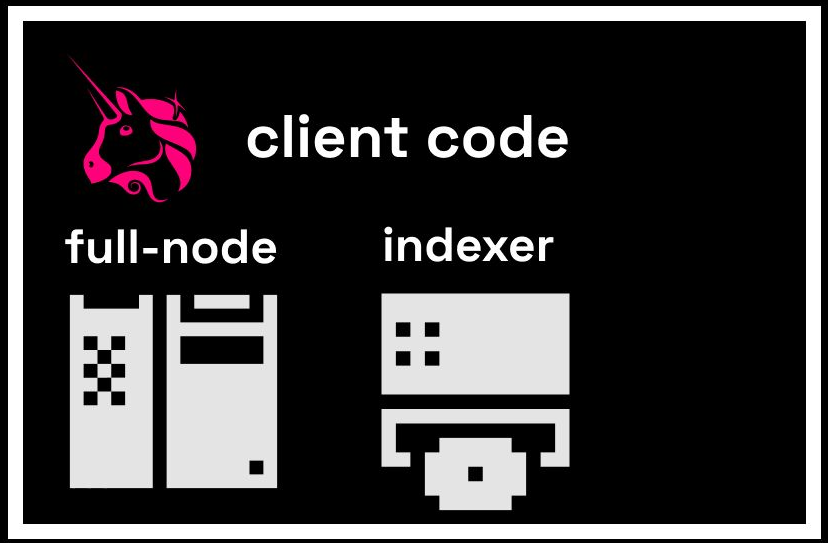
- ACCESS STATE OF THE CHAIN DIRECTLY FROM THE NODE DB
- SIMULATE TRANSACTIONS

- 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

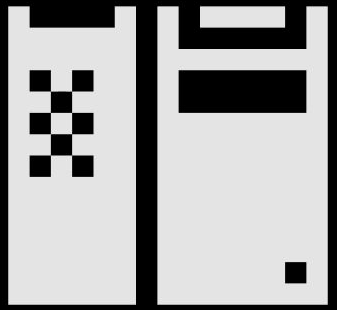

full-node
browser
client code
SLOW NETWORK
REQUESTS
indexer

browser



WHY NOT??

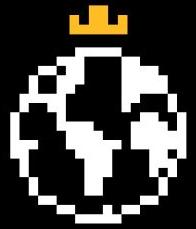
- RUN INDEXER CLIENT SIDE
- SIMULATE TXS
- NO NETWORK DELAY AFTER SYNC!



FULL NODES ARE EXPENSIVE!
BANDWITH, STORAGE

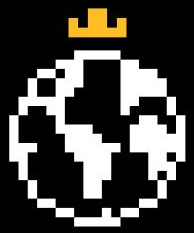
1. UX-HURTING NETWORK CALLS
2. WAIT FOR MINED TX TO SHOW SIDE EFFECTS
3. REMOTE INDEXERS

CAN WE DO BETTER?



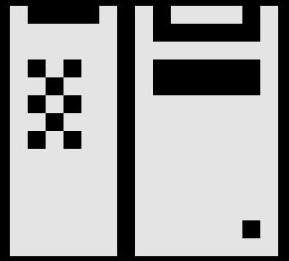
AUTONOMOUS WORLDS ARE MOSTLY STANDALONE, UNLIKE TRADITIONAL DAPPS

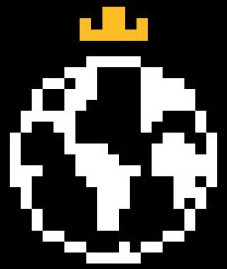
SIMULATING TX REQUIRES KNOWING STATE OF OTHER SMART CONTRACTS, LIKE ERC-20s ON BOTH SIDES OF THE POOL



MUD =

namespaced
full-node

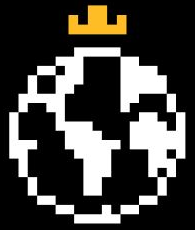




MUD SYNCS A WORLD, A NAMESPACE FOR DATA AND LOGIC

DATA = COMPONENTS

LOGIC = SYSTEMS



MUD

- 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

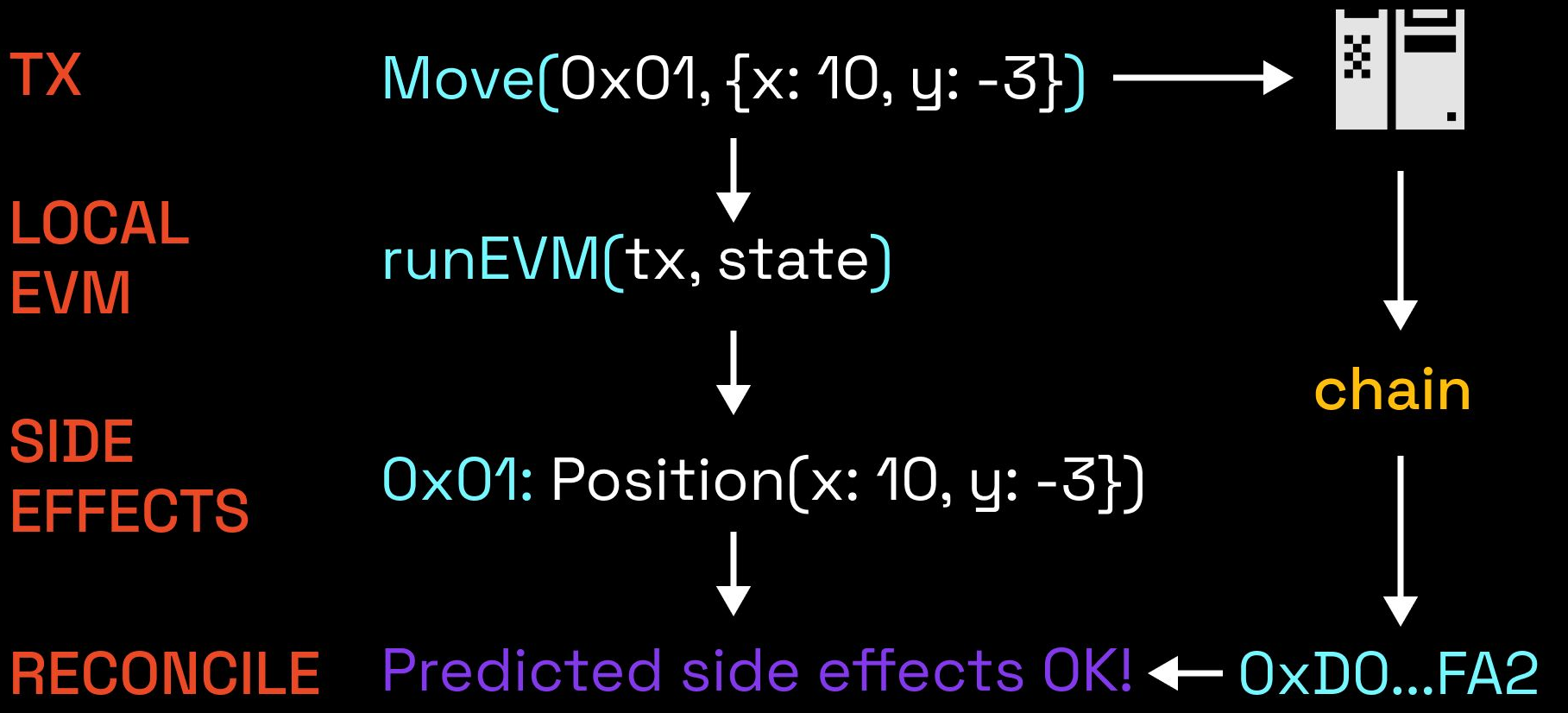
```
contract 0xA2F..1
0x0: 0xFAB6....81
0x1: 0x1AF0....D1
contract 0xAE1..4
0x0: 0x0013....6A
```

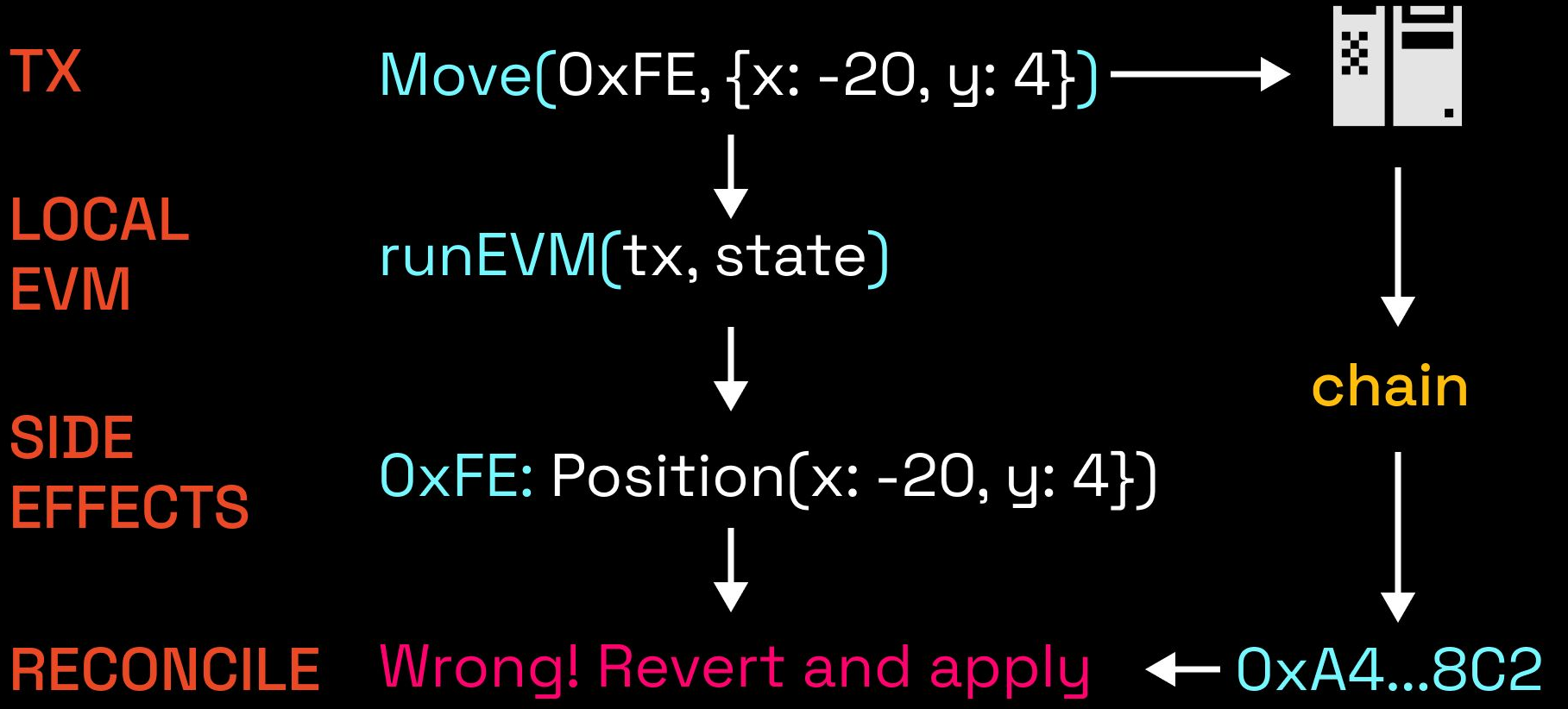
MUD

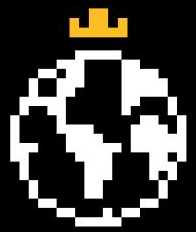
```
0x0: [Position(12,45), Health(200)]
0x1: [CanFly(), Health(10)]
0x2: [Position(1, -4), Balance(100)]
```

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
0xA4...8C2
chain



MUD

- READ / INDEX COMPONENTS WITHOUT NETWORK DELAY
- SIMULATE TX WITHOUT NETWORK DELAY

EXTENDING WORLDS WITH MUD

>df

DEVS CAN EXTEND PROTOCOLS VIA SMART CONTRACTS AND NEW CLIENT — THAT'S THE POWER OF FULLY ON-CHAIN PERMISIONLESS 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

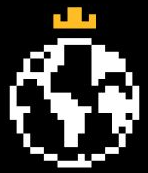
3RD PARTY

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?

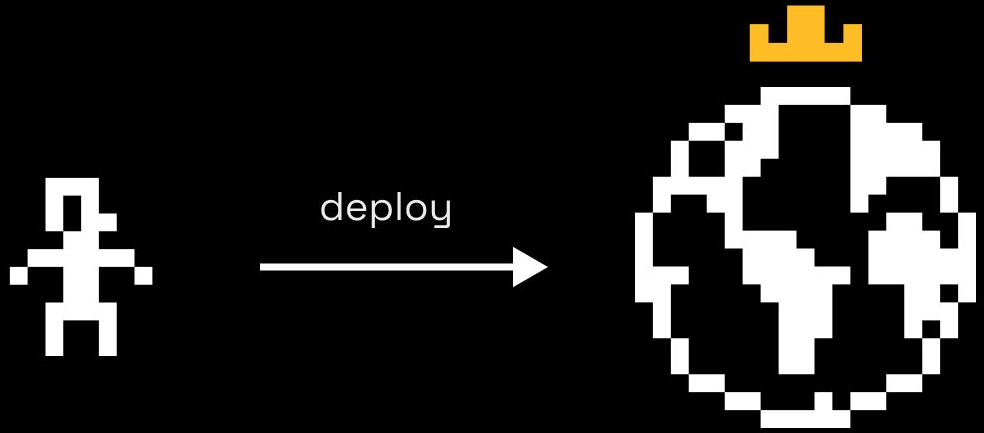


World.sol

PERMISIONLESS

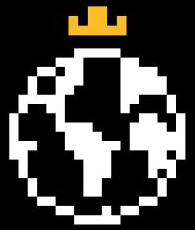
NO OWNER

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, PERMISIONLESSLY

LET’S ILLUSTRATE

COMPONENTS

- Position
- Movable
- Resource

Core Team

0x01

0x02

0x03

0x04

0x05

0x06

0x07

0x08

SYSTEMS

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

Core Team

COMPONENTS

- Position
- Movable
- Resource

Core Team

0x02

Move()

0x03

0x04

0x06

0x07

SYSTEMS

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

Core Team

0x05

0x08

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT


0x01

0x02

0x03


0x07

0x04

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team


0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

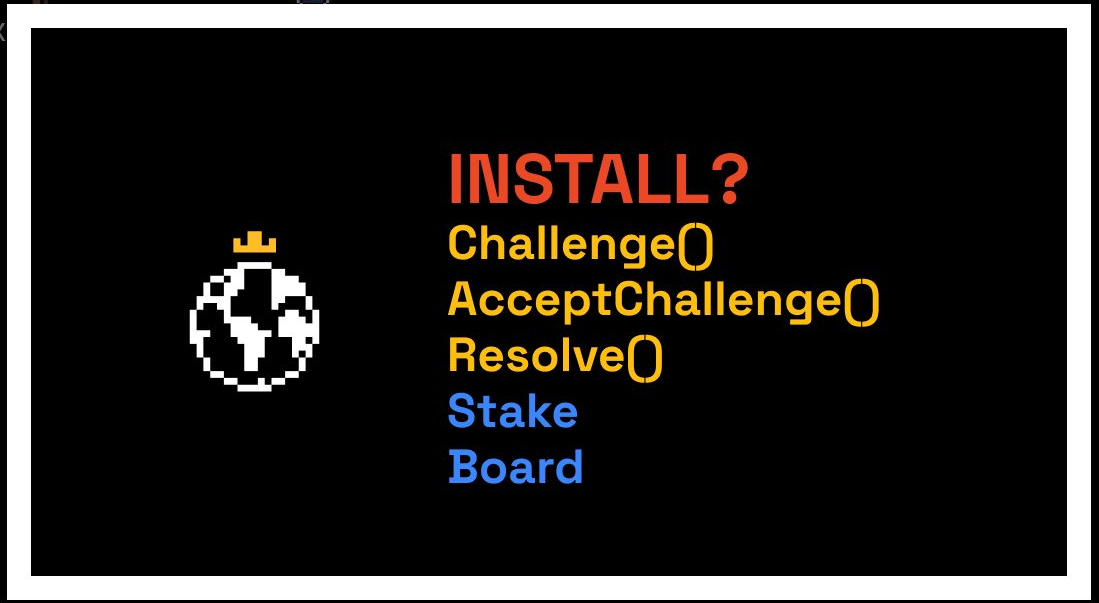
- Stake
- Board

Team TTT

0x04

0x

0x02



SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

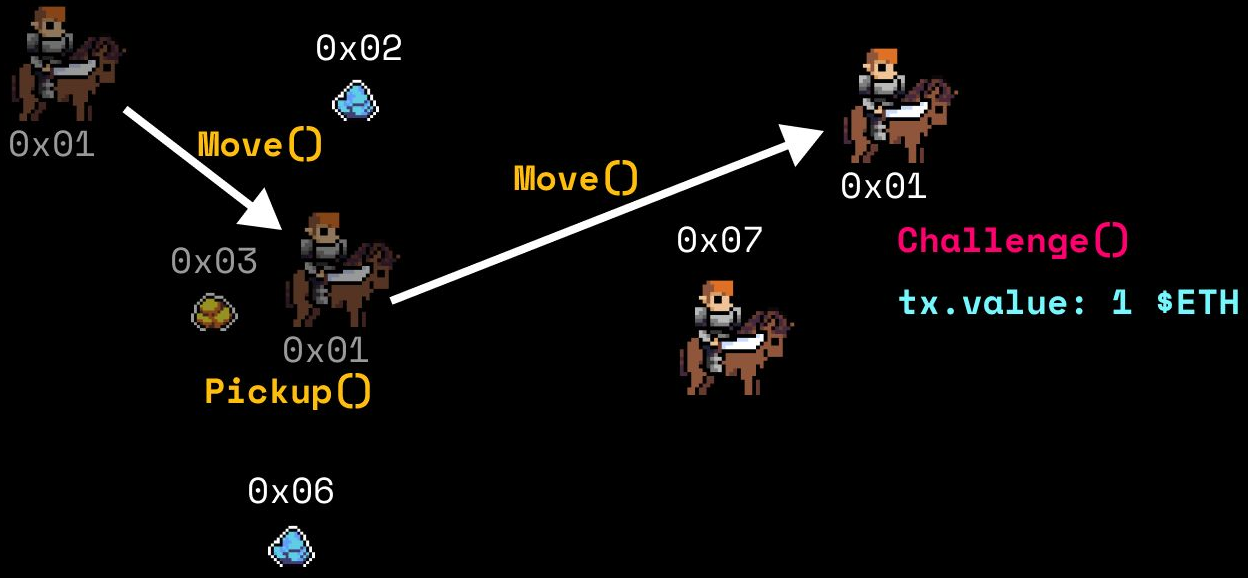
COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT



SYSTEMS

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

Core Team

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

Position
Movable
Resource

Core Team

Stake
Board

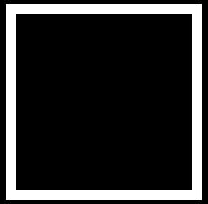
Team TTT

0x04

0x02

0x01

0x07



0x09

Stake(1)
Board()

AcceptChallenge()

tx.value: 1 $ETH

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x04

0x02

Drop()

0x01

0x07

0x09

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x04

0x02

0x01

Drop()

0x07

0x09

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x04

0x02

Drop()

0x01

0x07

0x09

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x04

0x02

Drop()

0x07

0x01

0x09

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x02

Drop() Resolve()

0x01

0x07

0x09

0x04

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

COMPONENTS

- Position
- Movable
- Resource

Core Team

- Stake
- Board

Team TTT

0x04

0x02

transfer 1$ETH

0x01

0x07

0x09

0x06

SYSTEMS

Move()
PickupResource()
Drop()

Core Team

0x05

0x08

Challenge() AcceptChallenge() Resolve()

Team TTT

TTT IS JUST LIKE TENNIS! IT'S AN AUGMENTED REALITY

OTHER PLAYERS

COMPONENTS

- Position
- Movable
- Resource

Core Team

WTF??

SYSTEMS

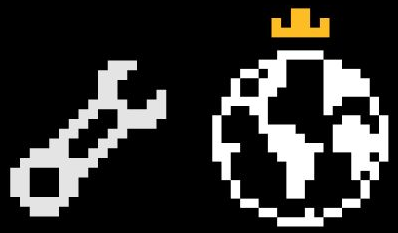
- Move()
- PickupResource()
- Drop()

Core Team

AUGMENTED REALITIES:
CAPITALISM
COMPETITION
MINI-GAMES

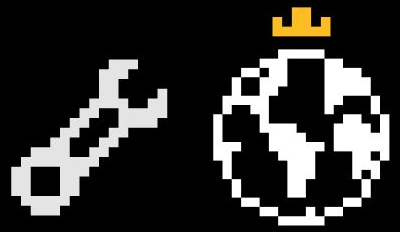
ALL PERMISIONLESSLY

WORLD IS OWNERLESS

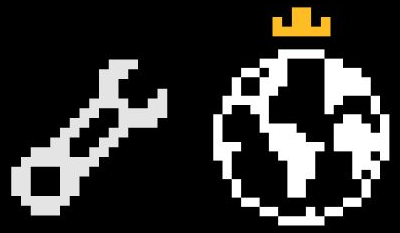


SO YOU WANT TO BUILD AN AUTONOMOUS WORLD

@KELVINFICHTER
BUILDING THE OPTIMISM COLLECTIVE



GOOD LUCK



JUST KIDDING

SOMETHING NEW

INTRODUCING THE OP STACK

SOMETHING NEW

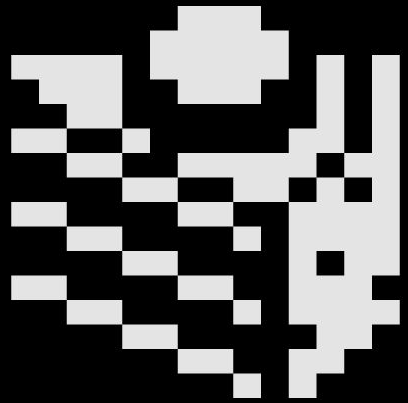
INTRODUCING THE OP STACK*

*we need like three months to write the docs

THE OP STACK

OVERVIEW

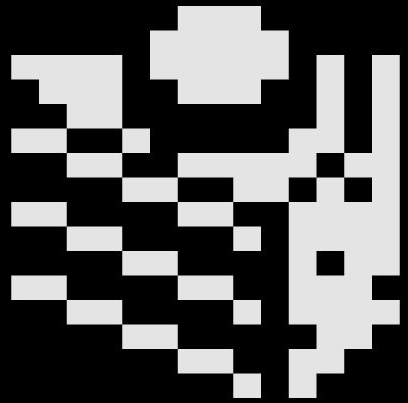
ROLLUPS GONE MODULAR



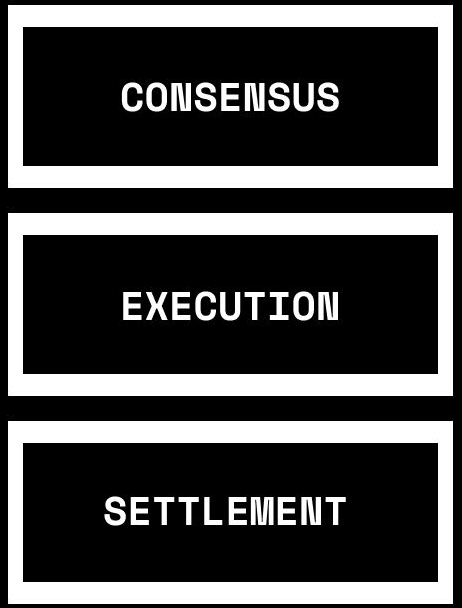
THE OP STACK

OVERVIEW

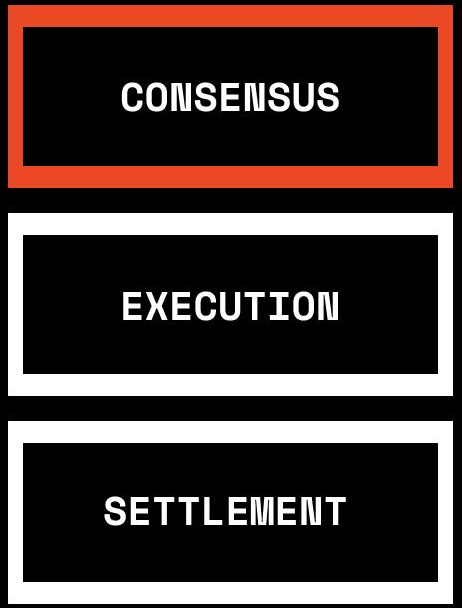
THREE SIMPLE LAYERS



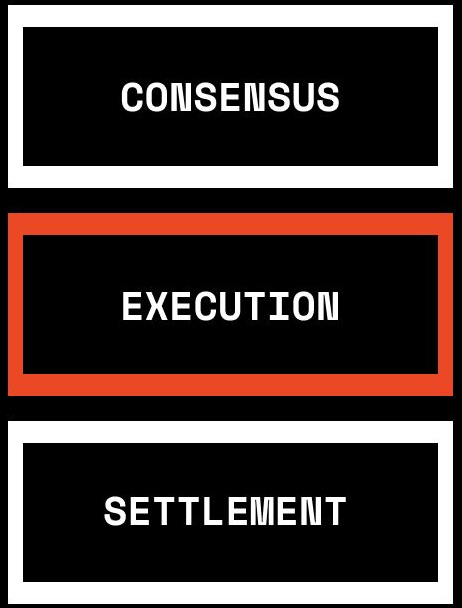
THE OP STACK
OVERVIEW



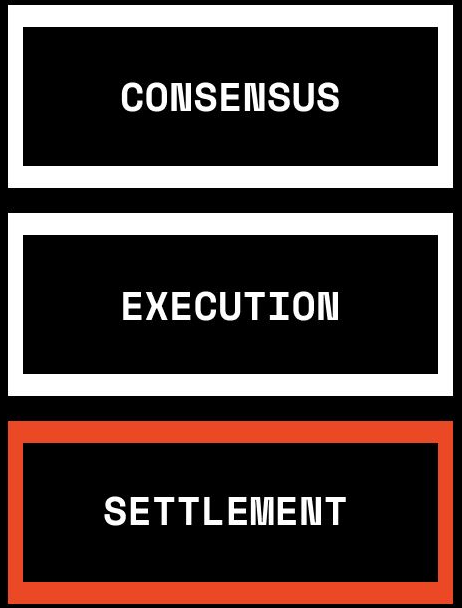
THE OP STACK
OVERVIEW



THE OP STACK
OVERVIEW



THE OP STACK
OVERVIEW



THE OP STACK
CORE CONCEPTS

MODULAR THEORY
IN PRACTICE

THE OP STACK

CORE CONCEPTS

DATA AVAILABILITY

PUBLISH DATA ANYWHERE

THE OP STACK

CORE CONCEPTS

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.*

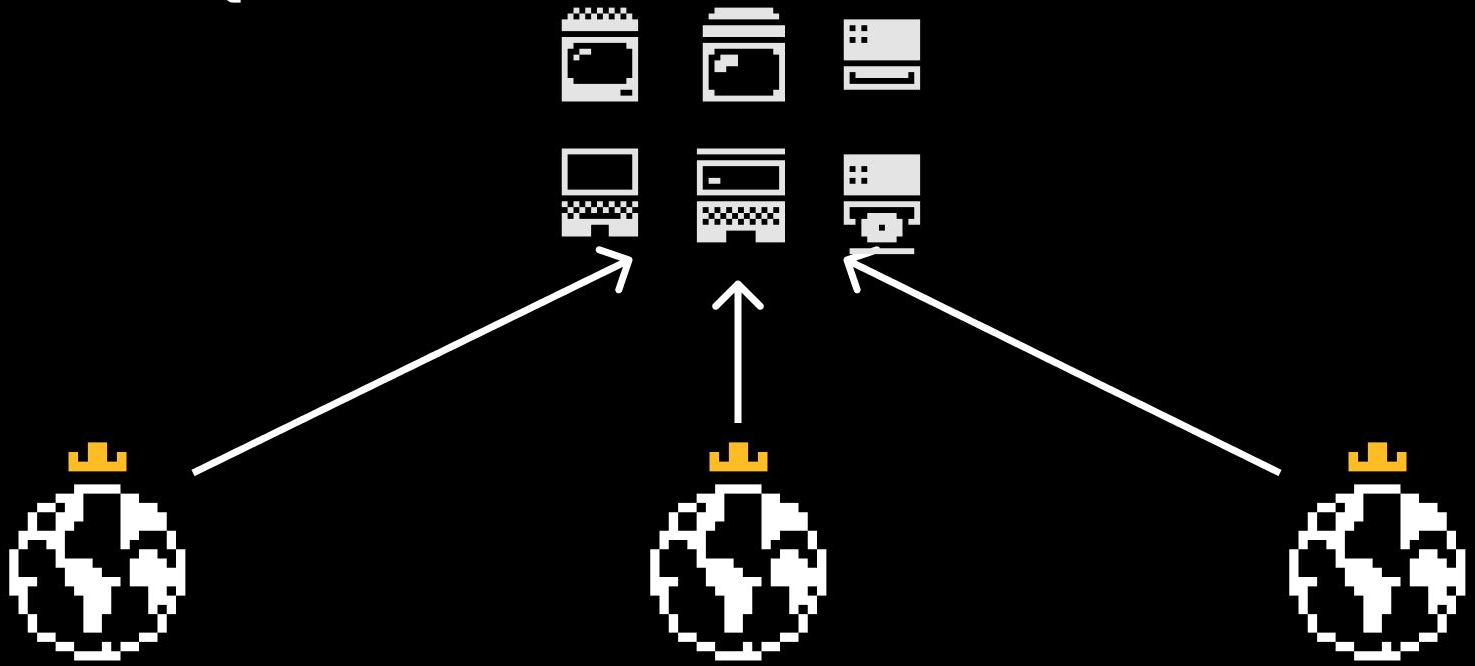
*currently a very big problem

THE OP STACK
SHARED SEQUENCING

THE OP STACK
SHARED SEQUENCING

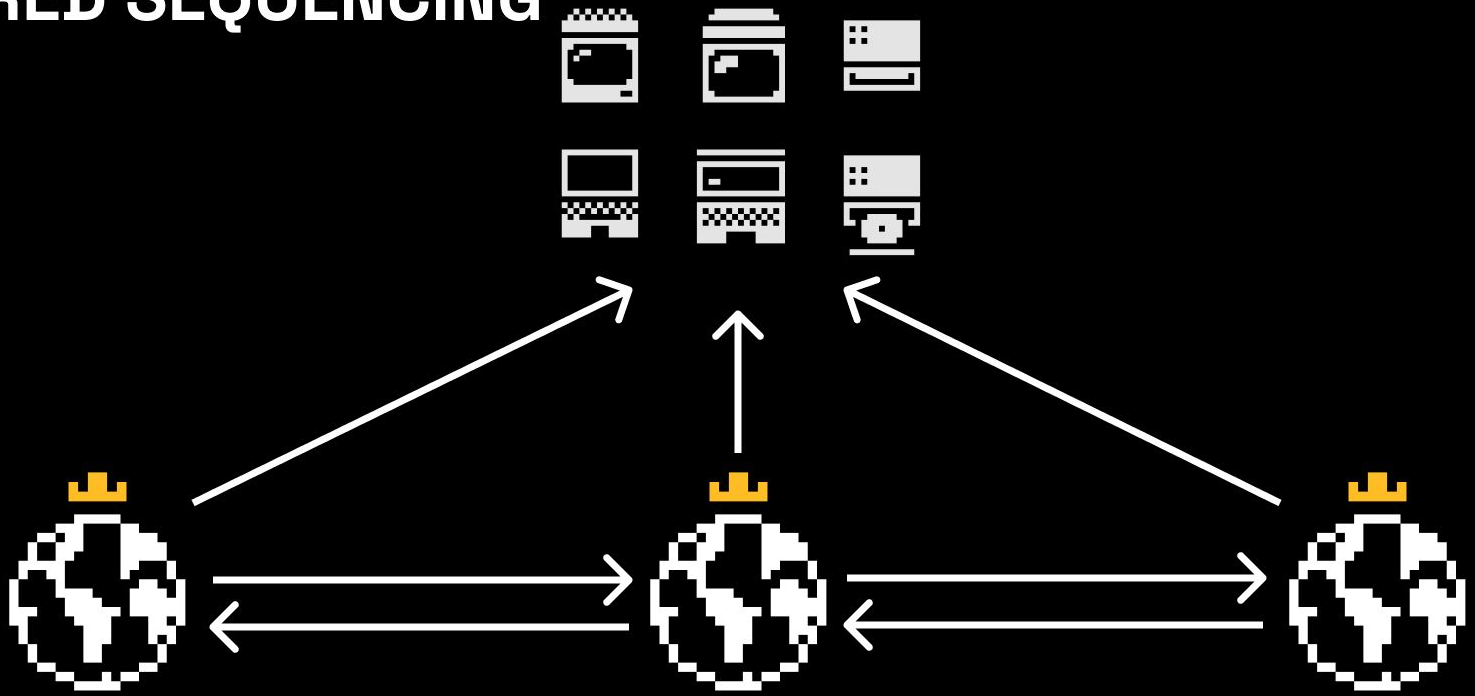
THE OP STACK

SHARED SEQUENCING



THE OP STACK

SHARED SEQUENCING

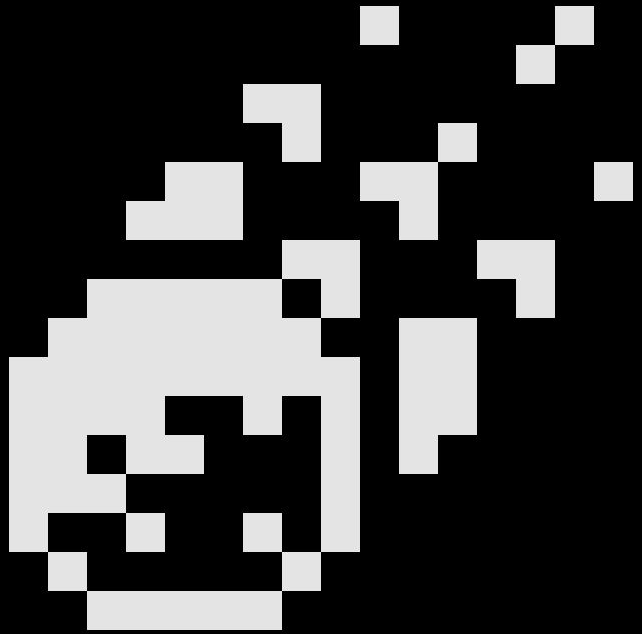


WHY MAKE IT FOSS?

THE OP STACK
MOTIVATION

BECAUSE
IT HAS TO BE.

THE OP STACK
MOTIVATION



THE OP STACK
MOTIVATION

THE OP STACK
MOTIVATION

THE OP STACK
MOTIVATION


S
AT

THE OP STACK

CLOSING REMARKS

GO NUTS,
BUILD SOMETHING CRAZY.

THE OP STACK

CLOSING REMARKS

AND THANKS FOR COMING TO MY TED TALK

+

OP

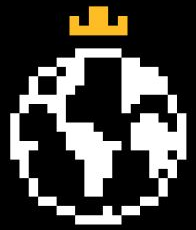
OPCRAFT

POWERED BY MUD
RUNNING ON OP STACK
PROCEDURAL WORLD
RELEASED TODAY!

AW ARCADE: 4PM TODAY

HACKER BASEMENT

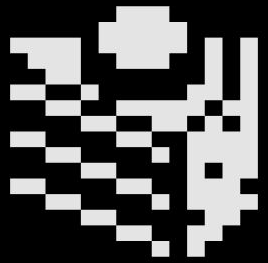

SKY STRIFE
4PM HACKER BASEMENT



MUD

AN ENGINE FOR AUTONOMOUS WORLDS

MUD.DEV



OP STACK

SOON

LATTICE ‡‡‡

* * *

* * *