Building a Lending Protocol that Powers a Yield-bearing Stablecoin Using Stylus

A workshop.

bayge
CTO, Fluidity Labs (Superposition, Fluidity Money)

Disclaimers

1. None of this is financial advice
2. I am not convincing you to use any products
3. This is a purely technical workshop

The format of this workshop

1. Quick setup: pulling down repo
2. Overview of Stylus: how does it work? High level overview (10 mins)
3. Overview of lending protocols: how do they work? How do some compare? (10 mins)
4. Overview of stablecoins: very high level stablecoin comparison (5 mins?)
5. Building the lending protocol

3

Quick setup (repo download)

Clone with Git

github.com/af-afk/talk-arbiverse-13-11-24 (https://github.com/af-afk/talk-arbiverse-13-11-24)

Quick setup (installation)

If you don't have Rust	Visit https://rustup.rs/ (https://rustup.rs/)
If you don't have Cargo Stylus	`cargo install cargo-stylus`
If you don't have `wasm32-unknown-unknown`	`rustup target add wasm32-unknown-unknown`

5

Quick setup (overview of Rust 1)

```
// Variable bindings (immutable)
let something = 123;
// Variable bindings (mutable) (also shadows!)
let mut something = 456;
// Slices (fixed-size arrays known at compile time)
let a = [123, 456];
// Vectors (arrays with size known at runtime)
let v = vec![123, 456];
// Importing packages ("crates")
use std::collections::HashMap;
// Hashmaps (objects/maps)
let mut h = Hashmap::new();
// Inserting into the hashmap.
h.insert("Hello", "World");
// Function that returns a number.
fn number(x: i32) -> i32 {
  x + 123
}
```

Quick setup (overview of Rust 2)

```
// Structures
struct Person {
    name: String
    age: u32
}
// The Option monad
enum Option<A> {
    Some<A>,
    None
}
let a: Option<i32> = Some(123);
let a: Option<i32> = None;
// The Option in practice.
let mut h = HashMap::new();
h.insert("World", "Hello");
// This would be false!
h.get("Hello").is_some();
// Another enum exists, the Result monad.
enum Result<T, E> {
    Ok(T),
    Err(E)
}
let a = Ok(123);
```

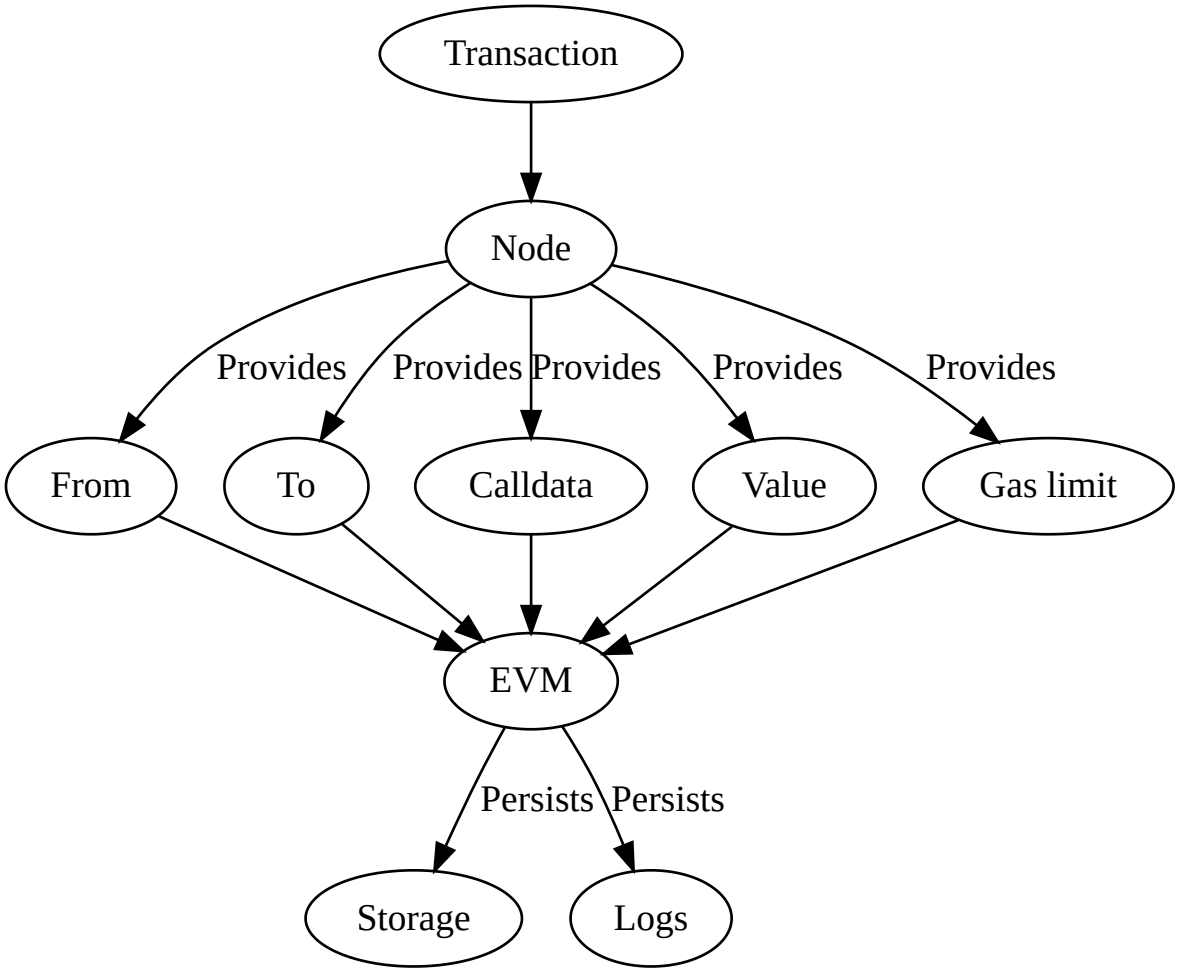
7

Quick setup (overview of Rust 3)

```
// The ? macro is how to shortcircuit control flow in a function.
enum Outcome {
    Fizz(i32),
    Buzz(i32),
    Nothing(i32)
}
fn find_fizzbuzz(x: i32) -> Result<i32, Outcome> {
    match (x % 3 == 0, x % 5 == 0) {
        (true, true) => Ok(x),
        (true, false) => Err(Outcome::Fizz(x)),
        (false, true) => Err(Outcome::Buzz(x)),
        (false, false) => Err(Outcome::Nothing(x)),
    }
}
fn pairs_of_fizzbuzz(x: i32, y: i32) -> Result<(i32, i32), Outcome> {
    let x = find_fizzbuzz(x)?;
    let y = find_fizzbuzz(y)?;
    Ok((x, y))
}
// This would be Ok((15, 20))
pairs_of_fizzbuzz(15, 20)
```

8

Overview of Stylus (intro to the EVM)



9

Overview of Stylus (EVM lifecycle expanded)

1. Node gets from, to, gas limit, and calldata from transaction.
2. Begins executing EVM code at to with the info that's in the transaction.

10

Overview of Stylus (EVM lifecycle expanded)

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

contract HelloWorld {
    uint256 public counter;                 // Generates selector 0x61bc221a.
    function increment() external {         // Generates selector 0xd09de08a.
        counter++;                          // Uses a SLOAD and SSTORE operation.
    }
}
```

Generates bytecode equivalent to (simplified and unoptimised):

```
0x0    calldataload            // Loads calldata to the stack.
0xE0   shr                     // Shifts calldata to get the selector.
dup0                           // Dupe the selector part of the calldata.
0x61bc221a eq COUNTER jumpi    // Jump to the counter code.
0xd09de08a eq INCREMENT jumpi  // Jump to the increment code.
0x0 0x0 revert                 // We didn't match a selector! Revert!
COUNTER:                       // Jump table offset for the counter view.
    0x0 0x0 sload              // Load the first slot in storage to the stack.
    mstore                     // Store in memory to return.
    return                     // Actually return the current value.
```

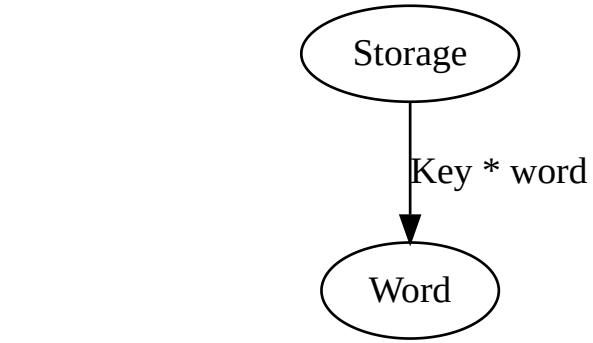
Overview of Stylus (EVM lifecycle expanded)

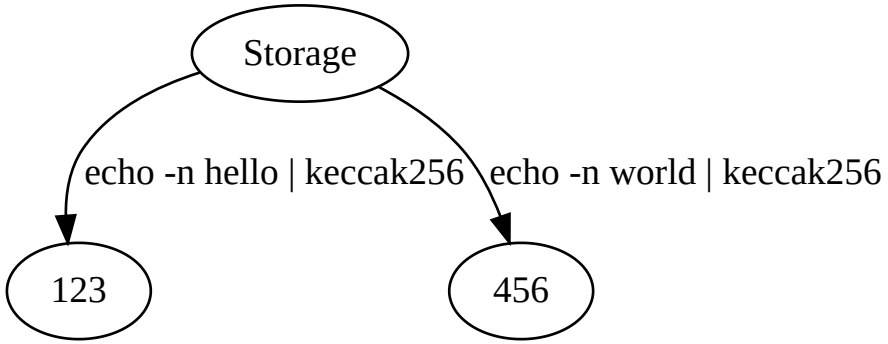
```
#define macro MAIN() = takes(0) returns(0) {
    0x0    calldataload            // Loads calldata to the stack.
    0xE0   shr                     // Shifts calldata to get the selector.
    dup0                           // Dupe the selector part of the calldata.
    0x61bc221a eq COUNTER jumpi    // Jump to the counter code.
    0xd09de08a eq INCREMENT jumpi  // Jump to the increment code.
    0x0 0x0 revert                 // We didn't match a selector! Revert!
    COUNTER:                       // Jump table offset for the counter view.
        0x0 0x0 sload              // Load the first slot in storage to the stack.
        mstore                     // Store in memory to return.
        return                     // Actually return the current value.
    INCREMENT:                     // Selector that matches the increment function.
        0x0 0x0 0x0 sload          // Load the counter.
        0x01 add                   // Increment the counter by 1.
        sstore                     // Store the incremented count.
        0x0 0x0 return             // Return to the caller! We're finished.
}
```

12

Overview of Stylus (EVM lifecycle expanded)

The storage trie:





13

Overview of Stylus (EVM lifecycle expanded)

The classic EVM lifeycle:

1. The node takes the to, from, calldata, gas limit from the transaction.
2. The node emulates the EVM.
3. The node persists the outcome of certain operations (including SSTORE).

Overview of Stylus (MultiVM introduction)

15

Overview of Stylus (MultiVM introduction)

Why WASM?

1. Open specification that's constantly evolving.
2. Other programming language and optimisation pipelines produce WASM code.
3. Access a broader ecosystem of thirdparty tools.

16

Overview of lending protocols (how does Compound V2 work?)

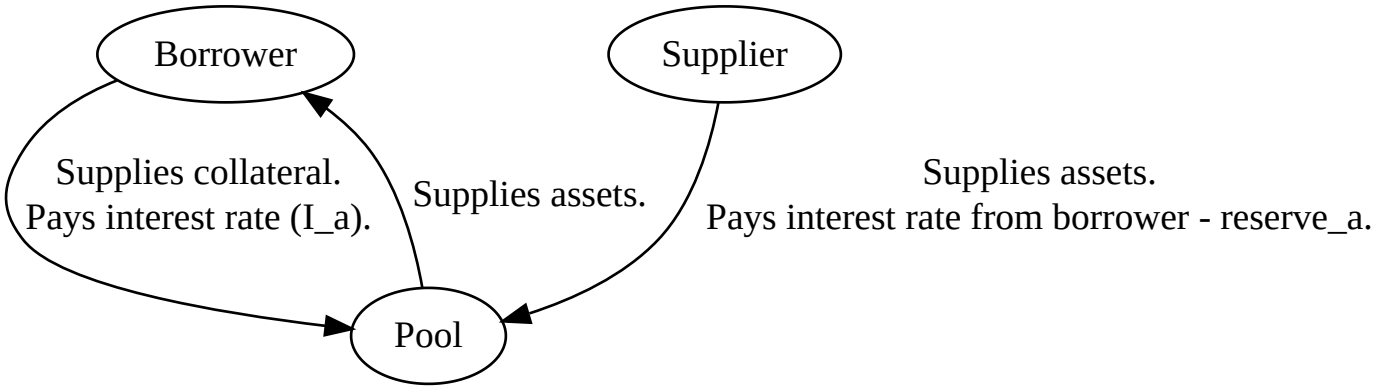
Some math

Only the borrower volatile math without the curve, no fees

```
# Utilisation rate for a.
U_a = Borrowed_a / (Cash_Reserves_a + Borrowed_a)
# Interest rate for the market a. Set by the DAO.
I_a = 0.025 + U_a * 0.2
# Exchange rate for the ctoken of the asset.
E_a = (underlying_bal + total_borrow_bal_a + reserves_a) / ctoken_supply_a
```

17

Overview of lending protocols (how does Compound V2 work?)



- Borrowers supply collateralisation, and receive assets in exchange from suppliers.
- Interest rate is calculated by the borrowing interest rate, which is calculated by the utilisation rate.
- Borrowers pay this interest rate.
- Suppliers receive the accumulated interest.

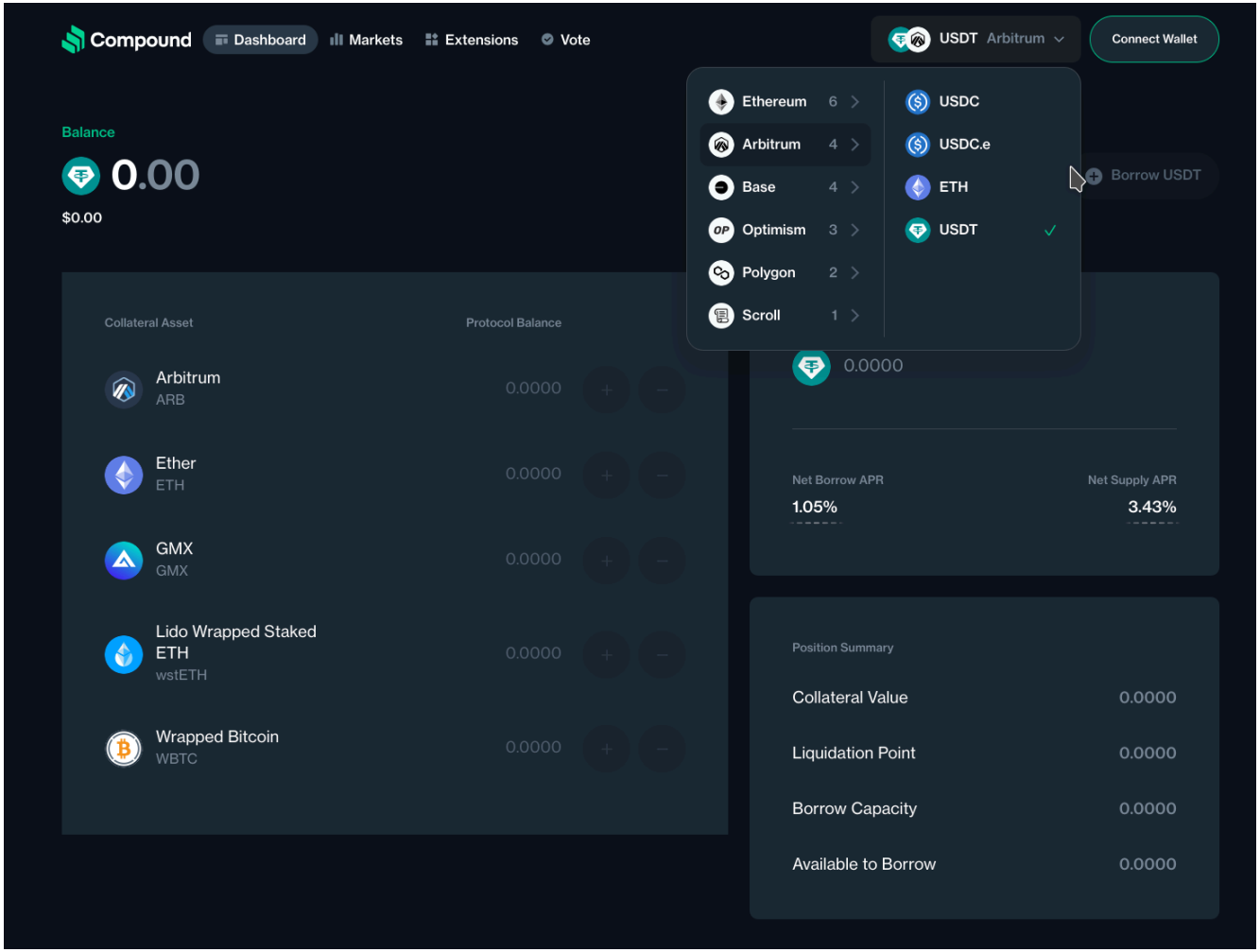
18

Overview of lending protocols (how does Compound V3 work?)

1. Single asset can be borrowed. Different markets UX.
2. Different interest rate for borrowing and supplying, governed by the DAO.
3. UX improvements.
4. Conversion of assets when liquidated to USDC.

19

Overview of lending protocols (how does Compound V3 work?)



20

Overview of lending protocols (how does AAVE work?)

- Stable and variable rate loans
- Siloed mode
- Efficiency mode
- Safety module
- Utilisation function
- Single asset pool

Overview of lending protocols (how does Morpho Blue work?)

- Liquidation Incentive Layer
- Permissionless creation of markets
- Collateral is not lent out
- Losses are distributed amongst lenders with bad debt

Overview of lending protocols (how does Agilerate work?)

- Algorithmic interest rate controller

AgileRate: Bringing Adaptivity and Robustness to DeFi Lending Markets

Mahsa Bastankhah[1], Viraj Nadkarni[1], Xuechao Wang[2], and Pramod Viswanath[1]

[1] Princeton University, Princeton, NJ, USA
mhs.bastankhah@princeton.edu, viraj@princeton.edu, pramodv@princeton.edu
[2] Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China
xuechaowang@hkust-gz.edu.cn

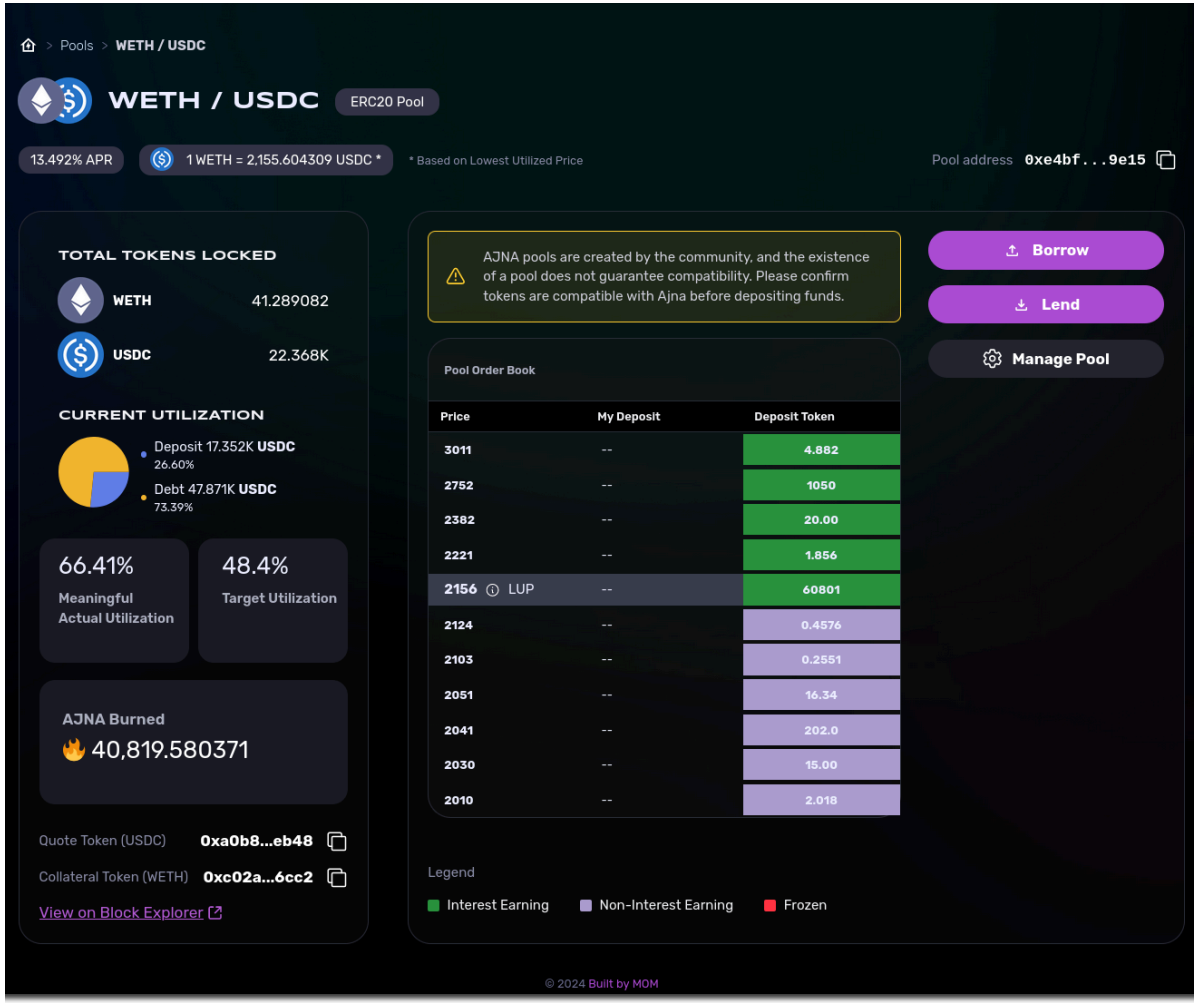
2024

23

Overview of lending protocols (how does Ajna work?)

- No governance (no managers)
- Create markets instantly. Interest and risk priced with AMM model
- Onus of lending risk is extra on supplier

Overview of lending protocols (how does Ajna work?)

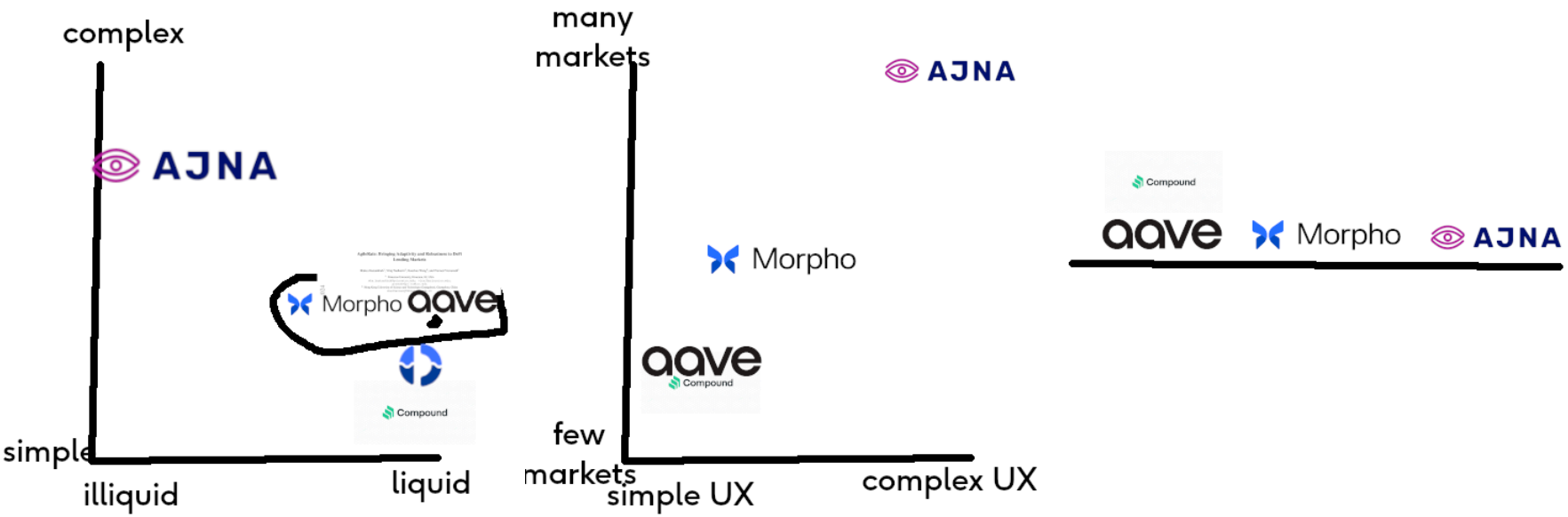


Overview of lending protocols (Instadapp Fluid)

- Loan to Value based on liquidations

26

Overview of lending protocols (how do they all stack up?)



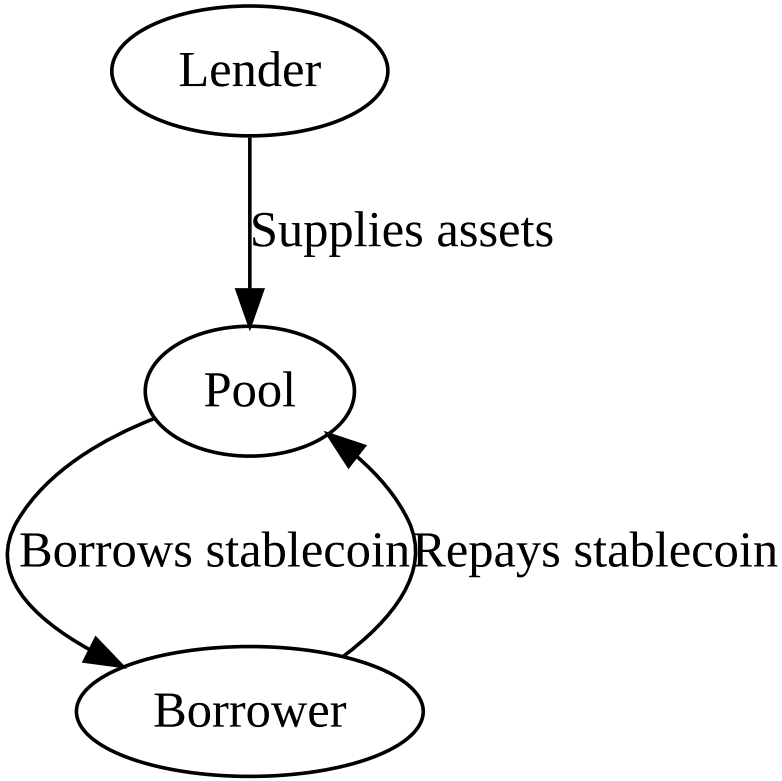
27

Overview of stablecoins (the different types)

1. Overcollateralised (backed)
2. Undercollateralised (algorithmic)

28

Overview of stablecoins (how do stablecoins work with lending protocols?)



29

Overview of stablecoins (how does LUSD work?)

1. 0% interest on loans
2. Loans repaid normally

30

Overview of stablecoins (how does LUSD work?)

1. User supplies $200 in collateral, priced by a oracle.
2. User gets back $200 LUSD.
3. User must maintain the debt.

31

Building the lending protocol and stablecoin (the goals)

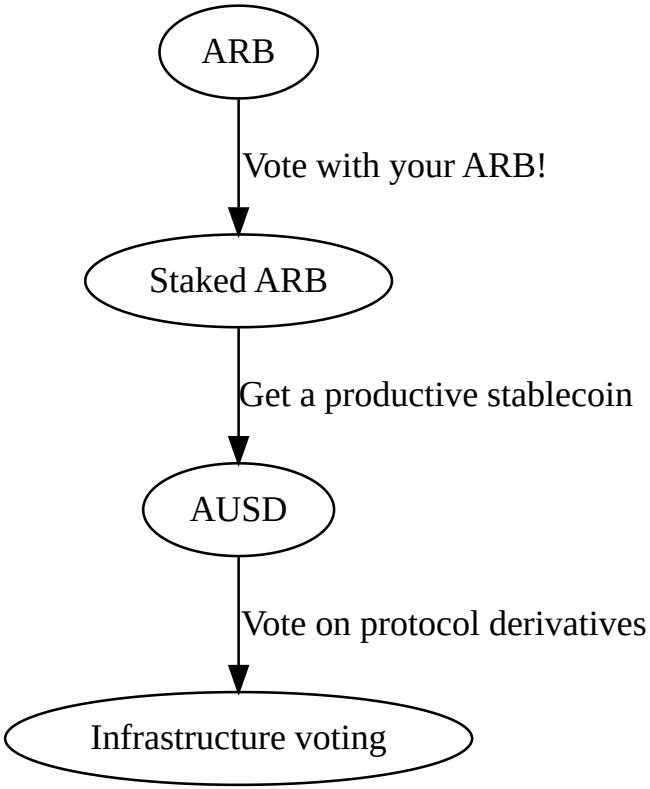
1. Lend Staked ARB
2. Receive AUSD (Arb USD)
3. We don't pay the upfront fee, instead we pay ongoing interest

Infrastructure

1. A stablecoin with exposure to ARB means native yield for infra providers
2. Ongoing interest will make the stablecoin productive

Building the lending protocol and stablecoin (the goals)

1. Staker infra is needed
2. We want exposure to ARB
3. This is not a consumer-facing stablecoin

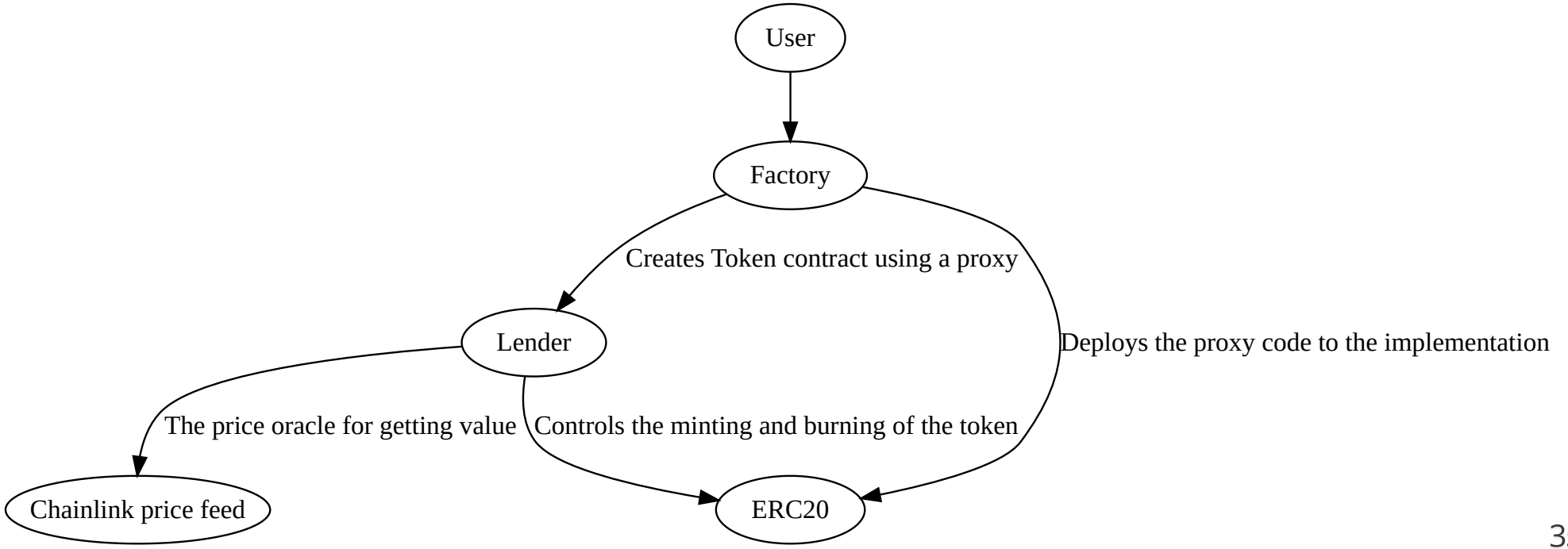


Building the lending protocol and stablecoin (the primitives)

1. Same mechanism as Liquity without the upfront fee (except security fee)
2. Compound inspired

34

Building the lending protocol and stablecoin (high level)



35

Building the lending protocol and stablecoin (part 1)

Build the "hello-world" crate.

```
cd pkg/hello-world
make build
```

^ If this exists with status 0, success!

36

Building the lending protocol and stablecoin (part 2)

```
cd pkg/hello-world
```

Get the "is_pet" function passing:

```
pub fn is_pet(c: Creature) -> bool {
    match c {
        Creature::Donkey => true,
        Creature::Dog | Creature::Human => false,
    }
}
```

Success:

```
./tests.sh part-1
```

Building the lending protocol and stablecoin (how to test contracts)

(This is in pkg/fizzbuzzer)

```
#[storage]
#[entrypoint]
struct Fizzbuzzer {
    pub counter: StorageU256,
}

#[public]
impl Fizzbuzzer {
    pub fn ctor(&mut self, c: U256) -> Result<(), Vec<u8>> {
        self.counter.set(c);
        Ok(())
    }

    pub fn is_fizzbuzz(&self) -> Result<bool, Vec<u8>> {
        let c = self.counter.get();
        Ok((c % U256::from(3)).is_zero() && (c % U256::from(5)).is_zero())
    }
}
```

38

Building the lending protocol and stablecoin (how to test contracts)

```
#[cfg(test)]
proptest! {
    #[motsu::test]
    fn test_contract_fizzbuzzer(num in any::<u128>()) {
        let mut c = unsafe {
            <Fizzbuzzer as stylus_sdk::storage::StorageType>::new(U256::ZERO, 0)
        };
        let num = U256::from(num);
        c.ctor(num).unwrap();
        let is_fizz = num % U256::from(3) == U256::ZERO;
        let is_buzz = num % U256::from(5) == U256::ZERO;
        assert_eq!(c.is_fizzbuzz().unwrap(), is_fizz && is_buzz);
    }
}
```

39

Building the lending protocol and stablecoin (factory design)

```
// Deploy the ERC20 proxy, and set it's proxy later to point to the Lending impl.
let erc20_addr = unsafe {
    RawDeploy::new()
        .deploy(&create_proxy_bytecode(erc20_impl), U256::ZERO)
        .map_err(Error::DeployFailure)?
};
let lending_addr = unsafe {
    RawDeploy::new()
        .deploy(&create_proxy_bytecode(lending_impl), U256::ZERO)
        .map_err(Error::DeployFailure)?
};
erc20_call::initialise(erc20_addr, lending_addr)?;
lending_call::ctor(lending_addr, erc20_addr)?;
Ok((lending_addr, erc20_addr))
```

```
// Minimal viable proxy bytecode.
pub const NORMAL_PROXY_BYTECODE_1: [u8; 18] = ...;
pub const NORMAL_PROXY_BYTECODE_2: [u8; 16] = ...;
pub fn create_proxy_bytecode(addr: Address) -> [u8; 54] {
    concat_arrays!(...)
}
```

Building the lending protocol and stablecoin (the oracle)

```
pub fn get_latest_price() -> Result<U256, Error> {
    let w = unwrap_second_word(
        &RawCall::new()
            .call(CHAINLINK_FEED_ADDR, &latestRoundDataCall {}.abi_encode())
            .map_err(Error::ChainlinkError)?,
    )?;
    if w.is_negative() {
        return Err(Error::ChainlinkNegativeFeed);
    }
    Ok(w.into_raw())
}
```

41

Building the lending protocol and stablecoin (the reference)

```
def borrow(self, ticket, cur_time, ausd_amt, token_collateral):
    usd_collateral = self.oracle.value_of_asset(token_collateral)
    if self.utilisation_rate(ausd_amt, usd_collateral) > self.COLLATERAL_REQ:
        raise BadDebt
    redemption_amt = token_collateral * (ausd_amt / usd_collateral)
    security_deposit = redemption_amt * self.SECURITY_DEPOSIT_RATE
    token_collateral -= security_deposit
    redemption_amt -= security_deposit
    self.token_for_redemptions += redemption_amt
    self.security_deposits += security_deposit
    self.cash_supply += ausd_amt
    self.borrows[ticket] = [Timepoint(cur_time, 0)]
    self.collateral[ticket] = token_collateral
    self.debt[ticket] = ausd_amt
    return ausd_amt
```

1. Prevent them from going over the collateral rate
2. Begin keeping of information

42

Building the lending protocol and stablecoin (part 3)

Get the tests passing for the timepoint code!

43

Building the lending protocol and stablecoin (part 4)

Get the tests passing for the borrow math!

44

Building the lending protocol and stablecoin (part 5)

Get the tests passing for the liquidate code!

45

Building the lending protocol and stablecoin (part 6)

Get the tests passing for the

Follow my frens

47

PS

Superposition mainnet the week of the 18th. 48

Thank you

A workshop.

bayge
CTO, Fluidity Labs (Superposition, Fluidity Money)
alex@superposition.so (mailto:alex@superposition.so)
https://superposition.so (https://superposition.so)
@baygeeth (http://twitter.com/baygeeth)
https://github.com/af-afk/arbiverse-talk (https://github.com/af-afk/arbiverse-talk)