Lecture 1: Basics of Blockchains

Guillermo Angeris

June 2022

Outline

Administrative stuff

Blockchains

Where's the money?

Outline

Administrative stuff

Blockchains

Where's the money?

Some quick notes

- The lectures will be recorded but not publicly posted (yet!)
- There will be 8 lectures in all
- Requires some mathematical maturity, familiarity with programming concepts
- No knowledge of blockchains is assumed
- May or may not lead to a future (full) course

Topics

- Lectures are ~ 1 hour long
- Will cover a number of topics including:
  - Stablecoins
  - Staking
  - Automated market making
  - Atomicity and MEV
  - And a few others...
- Focus is on simple models, many applications

With that...

- Let's talk about (smart?) blockchains

Outline

Administrative stuff

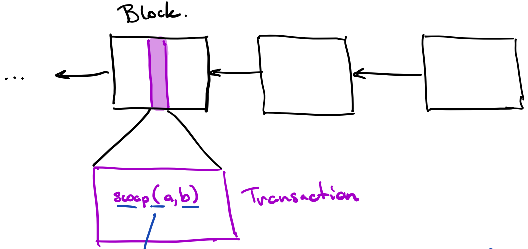
Blockchains

Where's the money?

Blockchain basics

- A blockchain is a collection of *blocks* making up an append-only, immutable ledger
- This ledger records *transactions* in some order
- A transaction is a set of function calls to *contracts*
- Contracts are a collection of function calls to other contracts or standardized primitives

Blockchain basics (and pictures)



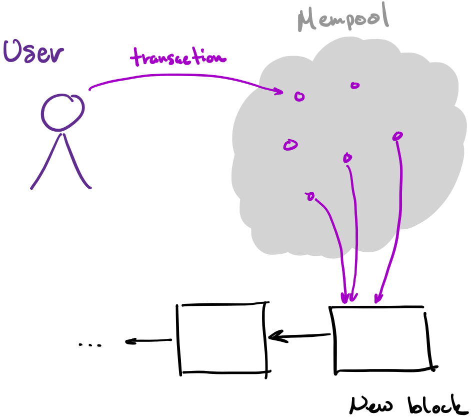
```
fn swap (in, out; user) {
    c_out = in*price
     ....
    if c_out ≥ out {
        transfer (c_out, user)
    }
}
```

Contract call

Interacting with the blockchain

- Users can submit transactions to the *mempool*
- These transactions sit ‘unexecuted’ in the mempool until accepted
- Once transactions are accepted, they are put in a block
- Execution has an associated cost called *gas*

Interacting with the blockchain (now with pictures!)



Blockchains as state machines

- Can view the blockchain as encoding a state machine's history
- Transactions are the state transitions
- The *current state* follows from applying all transactions up to the present block

Blockchains as state machines

- Can view the blockchain as encoding a state machine's history
- Transactions are the state transitions
- The *current state* follows from applying all transactions up to the present block
- In other words: a blockchain is a (very slow, expensive) server
- But it requires very little trust in anyone!

A note on terminology

- The terminology used in this class is somewhat standard
- Things are still changing and ‘connotations’ also differ
- If there are many common names, we default to Ethereum’s naming conventions

  (For better or worse)

Outline

Administrative stuff

Blockchains

Where's the money?

Accounting as an application

- Given this new platform, let's ‘write’ some applications!
- The simplest one: an accounting platform
- One of the major use-cases of trust minimization

The ‘idea’

- Accounts will have some *balances*

- We can read balances using

  `balanceOf(acc)`

- We can send tokens by calling

  `transfer(acc, amount)`

- And that’s it! (For now)

The ‘implementation’

- Balances are kept in a hashmap

```
balances: map[address => unsigned int]
```

- The `balanceOf` method is easy:

```
fn balanceOf(acc) { return balances[acc]; }
```

The ‘implementation’ (continued)

- Balances are kept in a hashmap

```
      balances: map[address => unsigned int]
```

- The `transfer` method is a little more complicated:

```
fn transfer(acc, amount) {
    if (balanceOf(txn.sender) < amount) return;
    balances[txn.sender] -= amount;
    balances[acc] += amount;
}
```

And finally

- We now have a (very simple) financial ecosystem!
- As a direct consequence of having a programmable chain

And finally

- We now have a (very simple) financial ecosystem!
- As a direct consequence of having a programmable chain
- Note that the details are unimportant
- The fact that we can do it is !

What does this look like?

- Board time!

Next lecture

- We will see more ‘complicated’/interesting applications
- (No implementations, though)
- And we will get to the first analyses of applications!