CS 168: Blockchain and Cryptocurrencies

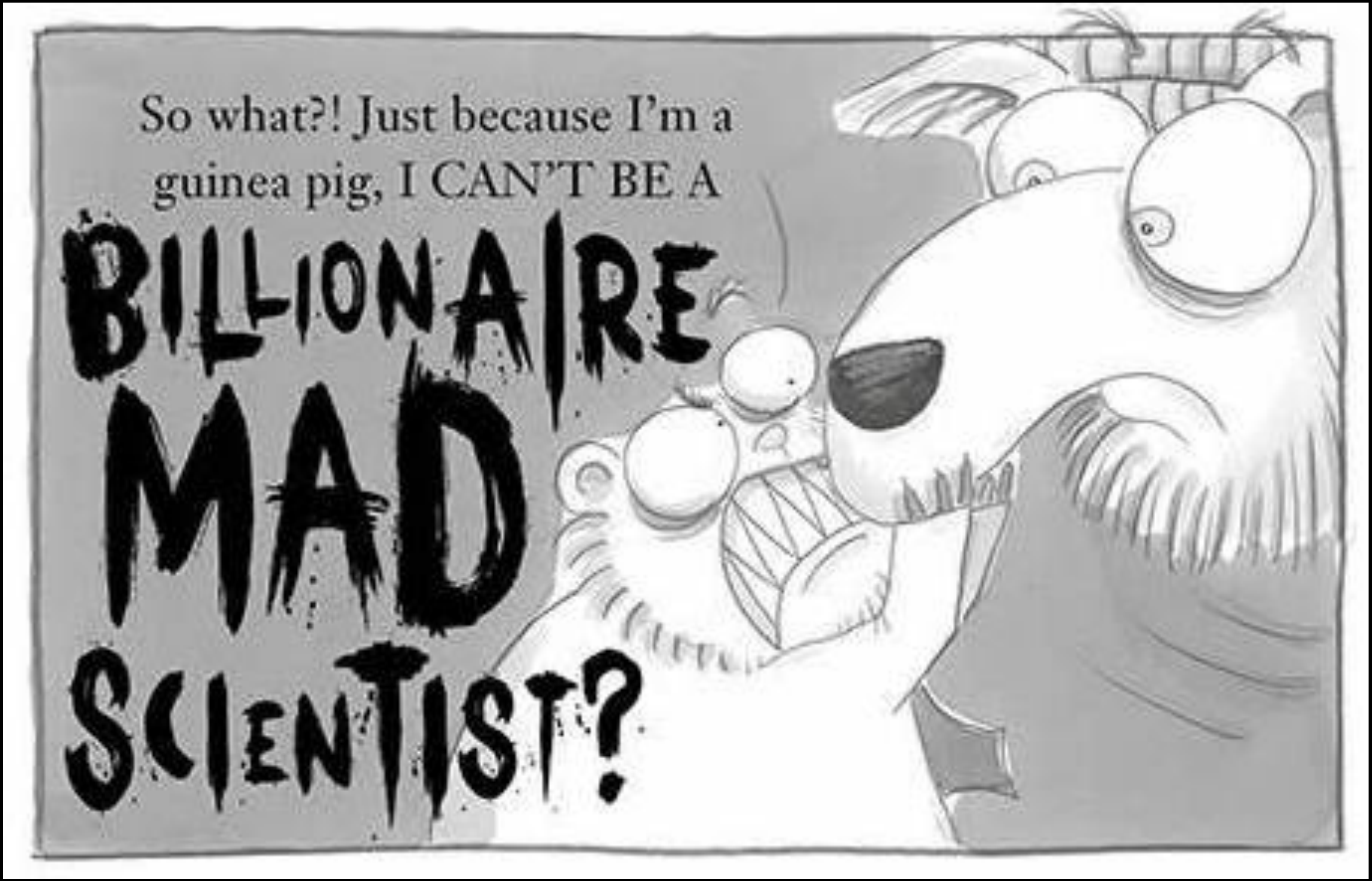
Introduction to SpartanGold

Prof. Tom Austin
San José State University

SpartanGold npm module

- Simplified Bitcoin-like blockchain
- Designed for rapid prototyping
- Currency is called "gold"
- Created by me
- https://github.com/taustin/spartan-gold/

Warning: you are my guinea pigs


So what?! Just because I'm a guinea pig, I CAN'T BE A BILLIONAIRE MAD SCIENTIST?

Similarities to Bitcoin

- *Proof-of-work* (PoW) blockchain
  - Miners validate blocks by finding a valid PoW
- Blocks collect transactions
- Mining rewards:
  - Coinbase reward: newly minted gold
  - Transaction fees

Key differences from Bitcoin

Bitcoin:

- Transactions stored in a Merkle tree
- Bitcoin script
- UTXO-based model
- Proof-of-work target adjusts over time
- Fixed block size

SpartanGold:

- Transactions stored in a map
- No scripting language
- Account-based model
- Proof-of-work target fixed
- No block size limit

Running in single-threaded mode

```
$ node driver.js
Starting simulation.  This may take a moment...
Initial balances:
Alice has 233 gold.
Bob has 99 gold.
Charlie has 67 gold.
Minnie has 400 gold.
Mickey has 300 gold.
Donald has 0 gold.
Alice is transfering 40 gold to zy2sIPBlf9PeM36D/
j0SyTznb8c3ESsDekNlZtSi06s=
Minnie: found proof for block 1: 5660
Minnie: cutting over to new chain.
Mickey: cutting over to new chain.
…
```

```
Balances:
  hDDXlpBFlnKViXVhbpJbf+tua7F8yMPIYtjJ+8KbWbk=: 675

Funds: 675
Address: hDDXlpBFlnKViXVhbpJbf+tua7F8yMPIYtjJ+8KbWbk=
Pending transactions:

What would you like to do?
*(c)onnect to miner?
*(t)ransfer funds?
*(r)esend pending transactions?
*show (b)alances?
*show blocks for (d)ebugging and exit?
*(s)ave your state?
*e(x)it without saving?

Your choice:
```

Running in multi-process mode

Lab, part 1: Experiment with SpartanGold

- Install with:
  `npm install spartan-gold`
- Download `singleThreadedExample.js`, `tcpMiner.js`, `minnie.json`, and `mickey.json` from the course website.
- Experiment with single-threaded and multi-process mode.
- Details in Canvas.

SpartanGold Design

Key concepts

- All classes can be extended
- Override methods if you want to change behavior

`Transaction` class fields

- `from`: Address of the payer
  - derived from public key
- `nonce`: orders transactions from payer
- `pubKey`
- `sig`: Signature for the transaction
- `fee`: Transaction fee paid to miner
- `data`: Generic JSON object (for extensibility)
- `outputs`: Discussed next slide…

Transaction outputs

- One transaction may pay multiple recipients
- The `outputs` field: array of JSON objects
  - Object keys: { address, amount }
  - Address: The recipient
  - Amount: Gold given to recipient

JSON for sample transaction
(parts elided with "…")

```
{ from: '4HWTOR8cgvejeMd…',
  nonce: 0,
  pubKey: '-----BEGIN PUBLIC KEY-----\n' …,
  sig: '83adb439…',
  fee: 1,
  outputs: [
   { amount: 25, address: 'vAy8w7bavN9…' }
  ],
  data: {}
}
```

`Transaction` methods

- `sign(privKey)`
- `validSignature()`
- `sufficientFunds(block)`
  - Pass in most recently confirmed block
  - Returns true if client has enough gold for transaction
- `totalOutput()`
  - Sum of all outputs plus the transaction fee

Determining transaction validity

- The `from` field matches `pubKey`
- Signature is valid
- The `nonce` is valid
  - Greater than last received nonce
- Payer has enough gold for the transaction

Block class

- Stores transactions
- Tracks balances
- Contains rules for validating transactions and blocks

`Block` class fields

- `rewardAddr`
  - Address of miner for coinbase reward
- `prevBlockHash`
  - First block (genesis) does not have previous block
- `target`: Maximum accepted PoW value
- `proof`
- `coinbaseReward`
- `chainLength`
- `timestamp`
- `transactions`: transaction ID -> transaction
- `balances`

`Block` methods

- `isGenesisBlock()`
  - Genesis block has special rules.
- `hasValidProof()`
- `balanceOf(addr)`: Gold available for specified client
- `totalRewards()`: Transaction fees + coinbase reward
- `contains(tx)`: True if transaction is in ***current*** block
- `addTransaction(tx)`
  - **Overridden in `BuggyClient` for lab**
- `serialize()`: Converts block to string form
  - some fields are omitted
- `rerun(prevBlock)`: Described next slide…

`rerun` method

- Clients and miners do not trust other's blocks.
  - Exception: The genesis block is trusted.
- The rerun method:
  - Resets balances and nonces to match previous block
  - Replays all transactions contained in the block
- Returns true if all transactions are re-added successfully

`Client` class

- Posts transactions
- Stores public/private keys
- Tracks blocks
  - Listens for new blocks
  - Verifies block validity
  - Requests missing blocks
  - Tracks last confirmed block

When is a block "confirmed"

- In Bitcoin, a block is confirmed:
  - After a chain of 6 blocks has been produced building on this block.
  - Takes about an hour in Bitcoin.
- SpartanGold uses the same approach.
- Probabilistic
  - could still roll back (though unlikely)

`Client` methods

- `availableGold`: **getter** for the amount of gold the client can currently spend
- `postTransaction(outputs, fee)`
- `showAllBalances()`
- `showBlockchain()`
- `log(msg)`
- `receiveBlock(block)`
  - Invoked on `Blockchain.PROOF_FOUND` message
  - Verifies block's validity
  - Stores block
  - If better than current block, updates current block

`Miner` class

- Extends `Client` class
- Collects transactions into a block
- Finds proof for a block

`Miner` methods

- `initialize()`:
  - set up listeners and begin mining
- `findProof()`:
  - searches for a valid PoW
- `addTransaction(tx)`:
  - Invoked on `Blockchain.POST_TRANSACTION` message

`Blockchain` class

- Contains settings for blockchain
- Stores constants
- Makes new blocks or transactions *as appropriate for current blockchain*
  - Might be `Transaction` or `Block` subclasses
  - Helps with SpartanGold's extensibility

`Blockchain` static methods

- `deserializeBlock(s)`:
  - converts string to instance of `Block` class
- `makeBlock(...)`:
  - Equivalent to `new Block(...)`, except that appropriate subclass is above.
- `makeTransaction(...)`:
  - Equivalent to `new Transaction(...)`, except …
- `makeGenesis(cfg)`: next slide…

`makeGenesis`

- Configures settings for blockchain
- Takes in JSON configuration
- Mandatory parameters:
  - `transactionClass`: Transaction (sub)class
  - `blockClass`: Block (sub)class

`makeGenesis` optional parameters

- Blockchain configuration details:
  - `powLeadingZeroes`
  - `coinbaseAmount`
- Genesis block balances (choose at most one):
  - `clientBalanceMap`: client -> amount `Map`
  - `startingBalances`: address -> amount JS object
- Client configuration details:
  - `defaultTxFee`
  - `confirmedDepth`

`FakeNet` class

- Simulates network connection
- Override to:
  - provide more realistic connection
    - See this approach in `tcpMiner.js`
  - Simulate different types of behaviors
    - Delayed messages
    - Dropped messages

`FakeNet` methods

- `register(…clientList)`
  - Adds clients to list of known clients
- `recognizes(client)`
- `sendMessage(address, msg, o)`
  - `address`: client to send message to
  - `msg`: name of the event
  - `o`: payload of the message
- `broadcast(msg, o)`
  - Calls `sendMessage` to all known clients

Lab, part 2: Replay attack

Download `replayAttack.js` and `buggyBlock.js` from the course website.

Details in Canvas

Lab, part 3: Explain replay attack

Contrast `buggyBlock.js` with the overridden methods from `block.js` in [`https://github.com/taustin/spartan-gold/`](https://github.com/taustin/spartan-gold/).

What differences do you notice?

Why did this attack work?

Write 2-3 sentences explaining what you think.