Blockchain AT BERKELEY

Workshop 2:

Ethereum & Smart Contracts: Enabling a Decentralized Future

Mission & Vision

Why did we start Blockchain at Berkeley?

Mission:
Help companies benefit from blockchain technology by identifying use cases, building out prototypes, and integrating solutions.

Vision:
Become the leaders of blockchain innovation in the Bay Area.

Overview

1. What's Ethereum
2. Smart Contracts
3. Basic Use Cases
4. Advanced Use Cases
5. Does it make sense to use blockchain?
6. Team formations
7. Brainstorm projects
8. Conclusion

ethereum

HOMESTEAD RELEASE

BLOCKCHAIN APP PLATFORM

Ethereum

1. Blockchain
2. Trustless
3. Decentralized apps
4. Smart Contracts
5. DAOs
6. Bitcoin 2.0

What is Ethereum?

Ethereum is a **decentralized** platform that runs **smart contracts**.
Ethereum is an **account-based blockchain**.
Ethereum is a **distributed** state machine that relies on transactions to move between states.

Smart Contracts: more sophisticated scripting that allows for (nearly) arbitrary computation

Blockchain: state is built from a series of blocks, composed of transactions, created by *accounts*, + network consensus

Decentralized: no single point of control/failure; censorship resistant

Smart Contracts: more sophisticated scripting that allows for (nearly) arbitrary computation

Transaction-based: state transitions occur on new transactions, which transfer value and information between *accounts*

Account-based: state is made up of *accounts*, with each *account* having an address, some balance of ether, and optionally some contract code and storage

What is Ethereum?

Ethereum is a **decentralized** platform designed to run **smart contracts.**

Though Ethereum and Bitcoin have similar features, both are billed quite differently:

- **Ethereum:** Smart Contract Platform
    - Complex and feature-rich
- **Bitcoin:** Decentralized Asset
    - Simple and robust

Ethereum has a **Turing complete scripting language** that is significantly more powerful than Bitcoin Script.

Ethereum is a **distributed** state machine that relies on **transactions** to move between states and execute contract code.

Ethereum is **trustless.**

Ethereum maintains a native asset called **Ether** that is the basis of value in the Ethereum ecosystem, allows for aligning incentives

Transactions in Ethereum execute bits of code

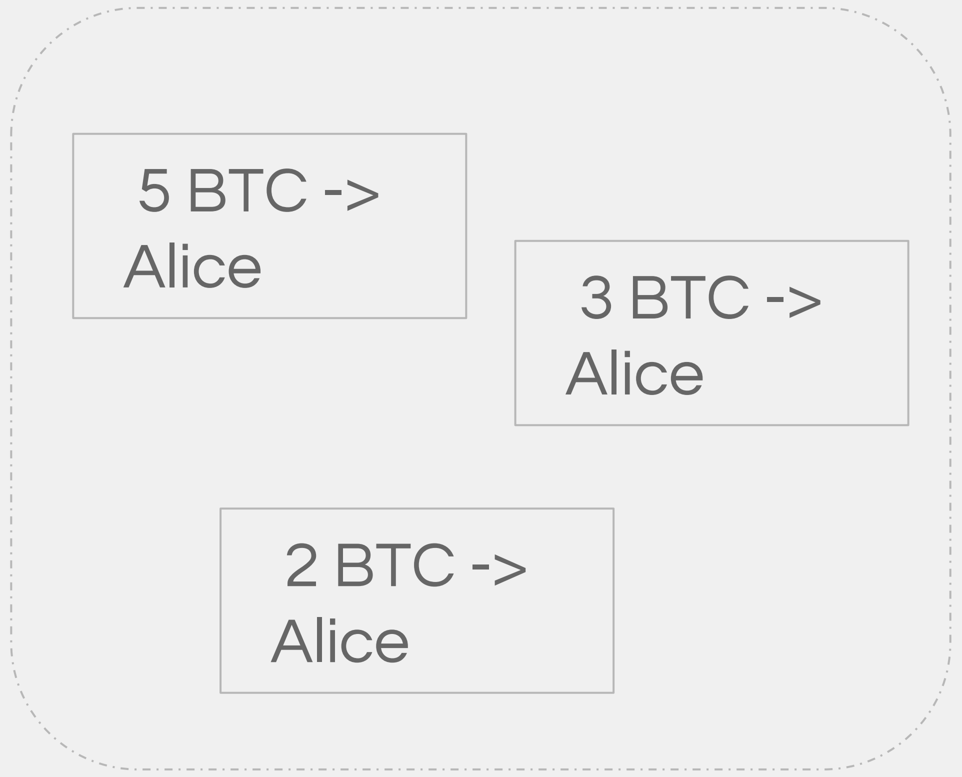
Smart Contracts: more sophisticated scripting that allows for (nearly) arbitrary computations

Accounts vs. UTXOs

Recall: A Bitcoin user's available balance is the sum of unspent transaction outputs for which they own the private keys to the output addresses.
Instead Ethereum uses a different concept, called **Accounts**.

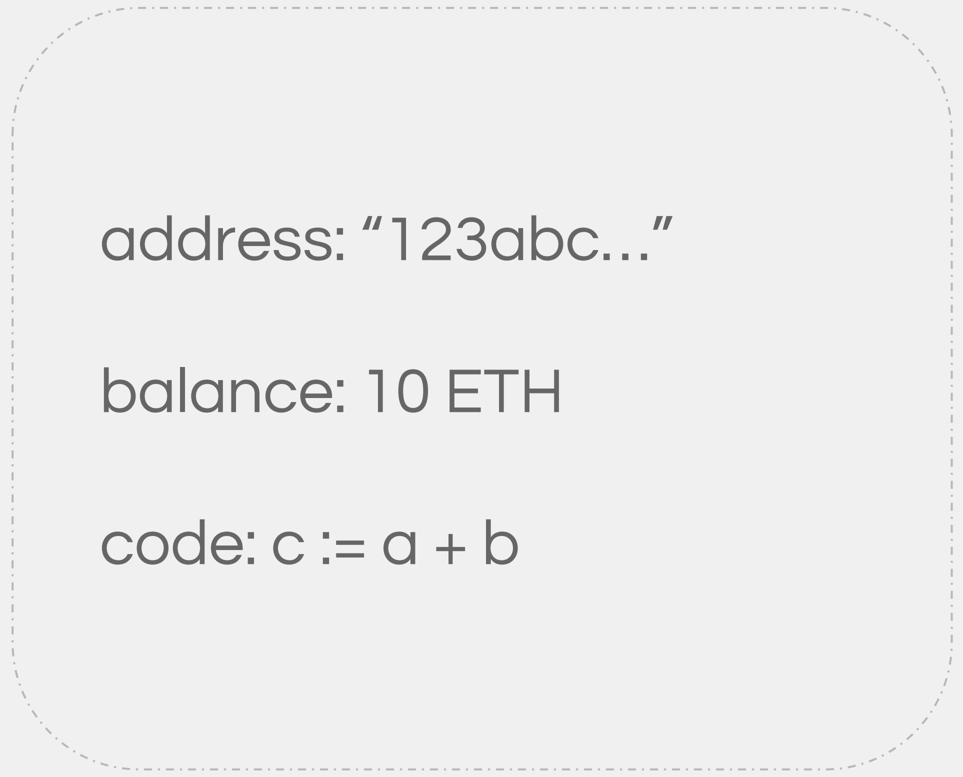
Bitcoin:

Bob owns private keys to set of UTXOs



Ethereum:

Evan owns private keys to an account



Externally Owned Accounts (EOAs):

- Generally owned by some external entity
- Identified by an *address*
- Holds some balance of *ether* (unit of Ethereum currency)
- Can send transactions (transfer ether to other accounts, trigger contract code)

Contract Accounts (Contracts):

- Identified by an *address*
- Has some *ether* balance
- Has associated contract code
- Code execution is triggered by transactions or messages (function calls) received from other contracts
- Contracts have persistent storage

All Accounts == Network State

The state of all accounts is the state of the Ethereum network, i.e., the *entire* Ethereum network agrees on the current balance, storage state, contract code, etc… of *every single account*.

The network state is updated with every block. You can think of the block as the state transition function; it takes the previous state and produces a new network state, which every node has to agree upon.

} Accounts interact with the network, other accounts, other contracts, and contract state through transactions.

Ethereum Virtual Machine

The Ethereum contract code that actually gets executed on every node is so-called EVM code, a low-level, stack-based byte-code language.
Every Ethereum node runs the EVM as part of its block verification procedure.
EVM as a state transition mechanism:

$$\begin{tikzcd}
\text{(block\_state, gas, memory, transaction, message, code, stack, pc)} \arrow[d] \\
\text{(block\_state', gas')}
\end{tikzcd}$$

where block_state is the global state containing all accounts and includes balances and long-term storage

EVM Design Goals:

Simplicity: op-codes should be as low-level as possible. The number of op-codes should be minimized.
Determinism: The execution of EVM code should be deterministic; the same input state should always yield the same output state.
Space Efficiency: EVM assembly should be as compact as possible
Specialization: easily handle 20-byte addresses and custom cryptography with 32-byte values, modular arithmetic used in custom cryptography, read block and transaction data, interact with state, etc
Security: it should be easy to come up with a gas cost model for operations that makes the VM non-exploitable

EVM Gas and Fees

EVM Gas and Fees

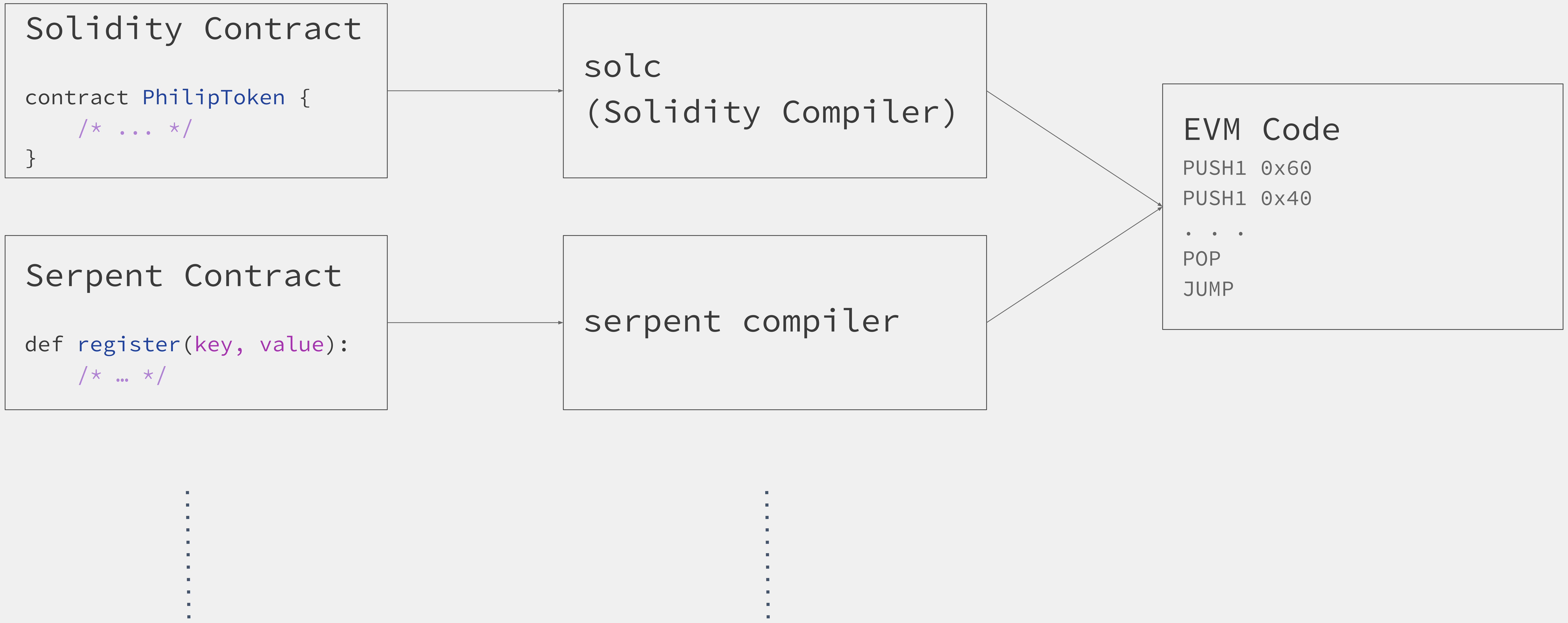
Immediate Issue:
What if our contract has an infinite loop?

```
function foo()
{
    while (true) {
        /* Loop forever! */
    }
}
```

Every node on the network will get stuck executing the loop forever! By the halting problem, it is impossible to determine ahead of time whether the contract will ever terminate ⇒ Denial of Service Attack!

Ethereum's Solution:
Every contract requires "gas", which "fuels" contract execution.
Specifically, every EVM op code requires a certain amount of gas in order to execute.
Every transaction specifies the startgas, or maximum quantity of gas it is willing to consume, and the gasprice, or the fee in ether it is willing to pay per unit gas.

EVM Code Compilation



```
contract PhilipToken {
    /* ... */
}
```

```
def register(key, value):
    /* … */
```

```
PUSH1 0x60
PUSH1 0x40
. . .
POP
JUMP
```

Smart Contracts - Introduction

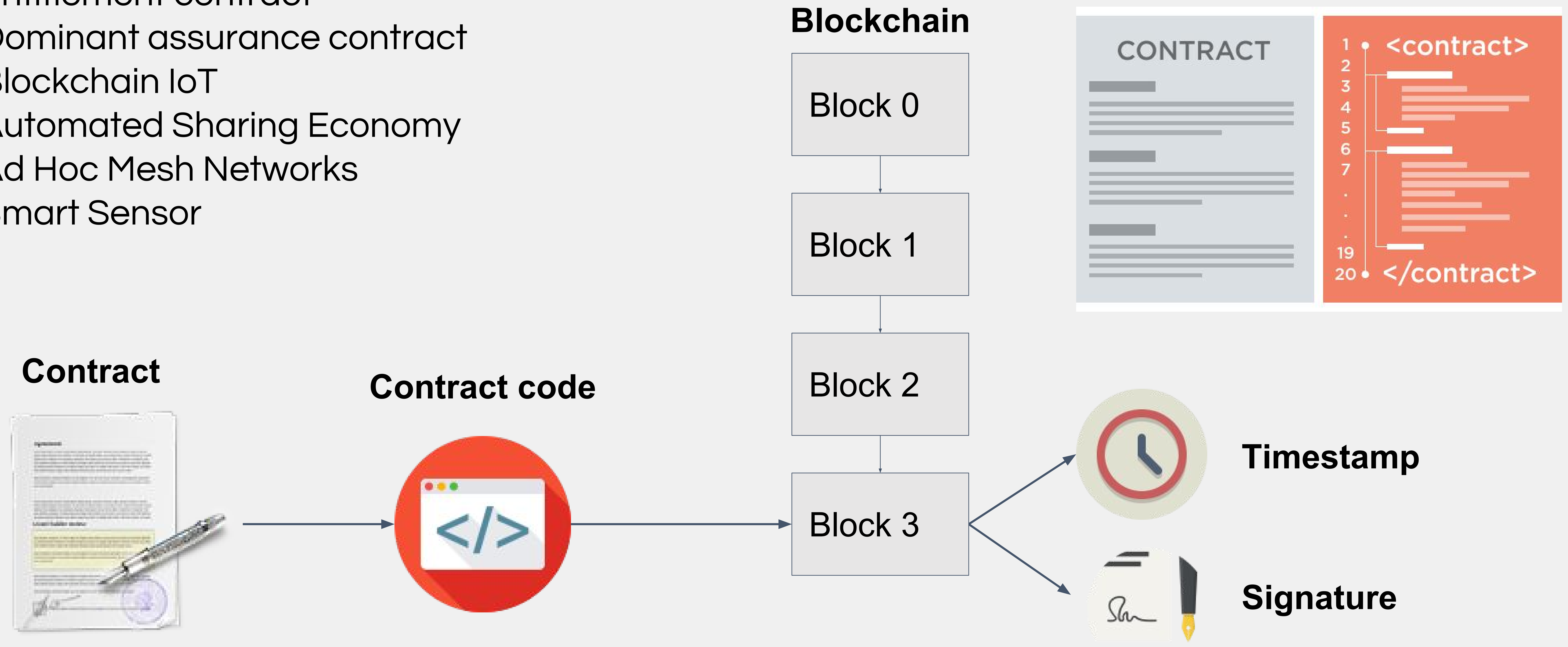
con·tract
(noun) /ˈkäntrakt/
1. a written or spoken agreement, especially one concerning employment, sales, or tenancy, that is intended to be enforceable by law.

smart con·tract
(noun) /smärt ˈkäntrakt/
1. code that facilitates, verifies, or enforces the negotiation or execution of a digital contract.

In the case of Ethereum, a smart contract is just an account with code.
Contracts in Ethereum are like autonomous agents that live inside of the Ethereum execution environment, **always executing a specific piece of code when "poked" by a transaction or message**, and having direct control over their own ether balance and their own permanent state.

Smart Contracts

1. Disintermediation
1. Entitlement contract
2. Dominant assurance contract
3. Blockchain IoT
4. Automated Sharing Economy
5. Ad Hoc Mesh Networks
6. Smart Sensor



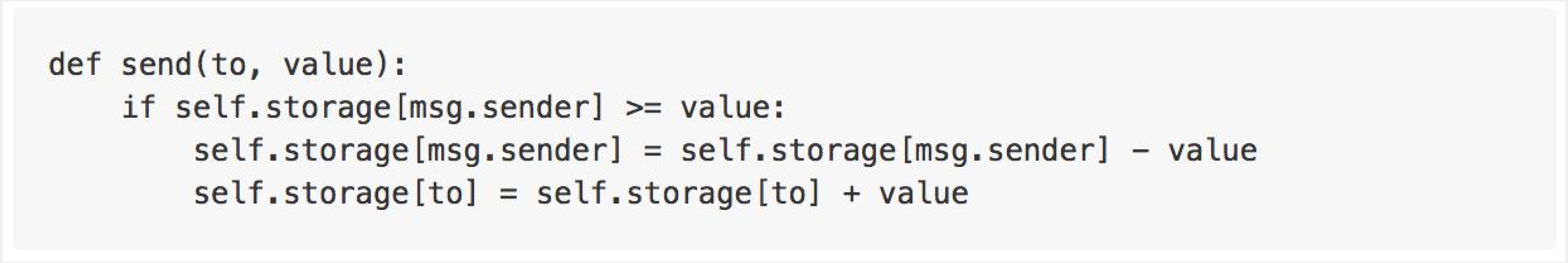
Blockchain at Berkeley

Basic Use Cases

Token Systems

- Very easy to implement in Ethereum
- Database with one operation
  - Ensure Alice has enough money and that she initiated the transaction
  - Subtract X from Alice, give X to Bob

Example (from Ethereum white paper):



```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

```
contract PhilipToken {

    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply
       tokens to the creator of the contract */
    function PhilipToken(uint256 initialSupply)
    {
        // Give the creator all initial tokens
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send tokens to a recipient address */
    function transfer(address to, uint256 value)
    {
        if (balanceOf[msg.sender] < value) throw;        // Check if the sender has enough
        if (balanceOf[to] + value < balanceOf[to]) throw; // Check for overflows
        balanceOf[msg.sender] -= value;                  // Subtract from the sender
        balanceOf[to] += value;                          // Add the same to the recipient
    }
}
```

Minimum Viable Token

PhilipToken is a stripped down version of a digital token contract, written in Solidity.

You can instantiate it with some initial supply of tokens which can be transferred between different accounts.

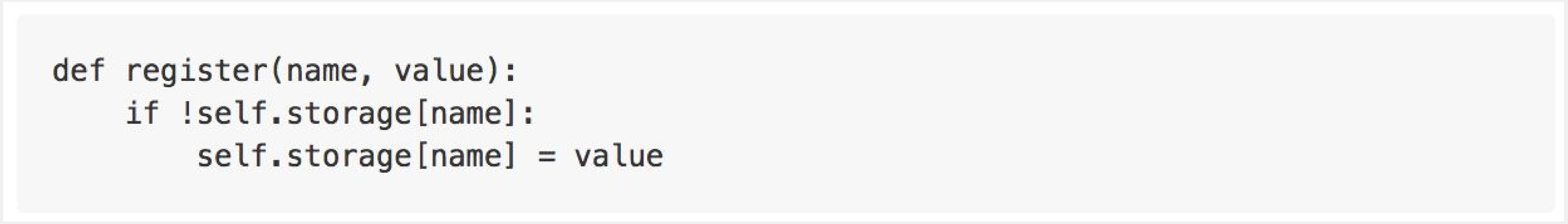
Remember, the contract is an account! It has its own ether balance, address, storage, etc...

Public Registry / Public Database

Example: **Namecoin**

- DNS system
  - Maps domain name to IP address
  - "maxfa.ng" => "69.69.69.69"
- Immutable
- Easy implementation in Ethereum

Example (from Ethereum white paper):



```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

Crowdfunding and Incentivization

Simple Example: "**Ether-on-a-stick**"

- Allows you to put a bounty on the completion of arbitrary tasks
- Contributors pool money into a smart contract that pays out to a specified recipient iff contributors vote that the task was indeed complete

Example use case

- A company is polluting a local river and nearby residents bear a negative externality
  - Local gov't slow/unresponsive but residents are willing to pay
- Residents pool money together to incentivize the company to clean it up

Implements a Dominant Assurance Contract: solves the free rider problem

Blockchain at Berkeley

Advanced Use Cases

Decentralized File Storage

"Decentralized Dropbox Contract": Pay individuals small amounts of Ether to rent out extra hard drive space

Example contract specification:

- Split cat picture into blocks, encrypt each block for privacy
- Create Merkle tree from blocks, save Merkle root in contract
- Every N blocks, the contract will:
  - Using previous block header (source of randomness), pick a random block in Merkle tree
  - First entity to provide proof of storage of block (Merkle branch) receives small ETH reward

Recovering the file:

- Query node storing file and pay a small fee (via micropayment channels) to retrieve it
- >Decrypt data, obtain furry kitten
- >Profit

IPFS
IPFS is The Permanent Web
A new peer-to-peer hypermedia protocol

Decentralized Prediction Markets

Prediction markets draws on the wisdom of the crowd to **forecast the future**

- Market makers create event
  - Ex: "Who will win the 2016 US Presidential election?"
  - Events must be public and easily verifiable, with set due date.
- Participants buy **shares** of Trump or Hillary and pay a small fee
- On election day, random **oracles** on the network vote on who won.
  - Oracles who voted with the majority collect a fee, they are otherwise penalized
- Shareholders who voted correctly cash out on their bet

The share price for each market accurately represents the best predicted probability of event occurring

- Someone has extra information => arbitrage opportunity

Decentralized Prediction Markets

GNOSIS augur

Use cases

- Cost efficient way to buy information on a future event
  - Instead of hiring pundits and experts, create a market for your event
  - "Will this movie be a flop?"
  - Bet for and against your event to incentivize people who have information about this event (in this case, Hollywood insiders)
- Hedging and insurance
  - Fire insurance is a bet that your house will burn down
  - Create market "Will my house burn down?" and vote yes
  - => receive compensation if your house burns down
  - Possible to implement an entire insurance liquidity pool
  - Potential for extremely thin margins since no central intermediary is required

Decentralized Prediction Markets

Use cases

- Set up a security bug bounty
    - "Will my company be hacked?" Bet heavily against it to create a financial incentive
    - Someone who finds vulnerability will buy affirmative shares, then perform their hack
        - > Profit
    - Augur secures their own code this way
        - "Will someone be able to steal the money in this prediction market?"
- Signaling: "Put your money where your mouth is"
    - Demonstrate your commitment to something by showing you will take a large financial loss if you miss your commitment
    - Ex. Kickstarter campaign; investors are worried you will delay launch date
        - "Will my Kickstarter campaign launch on time?"
        - Bet heavily that you WILL launch your produce on time.

Decentralized Prediction Markets

Benefits to being decentralized

- No restrictions on market creation
  - But raises ethical questions
- Shared liquidity pool
  - No reason why the same market should exist in multiple countries
  - Allows for more advanced markets;
    e.g. combinatorial prediction markets
- Censorship-resistant
- Automatic, trustless payments

Decentralized IoT

FILAMENT

Filament

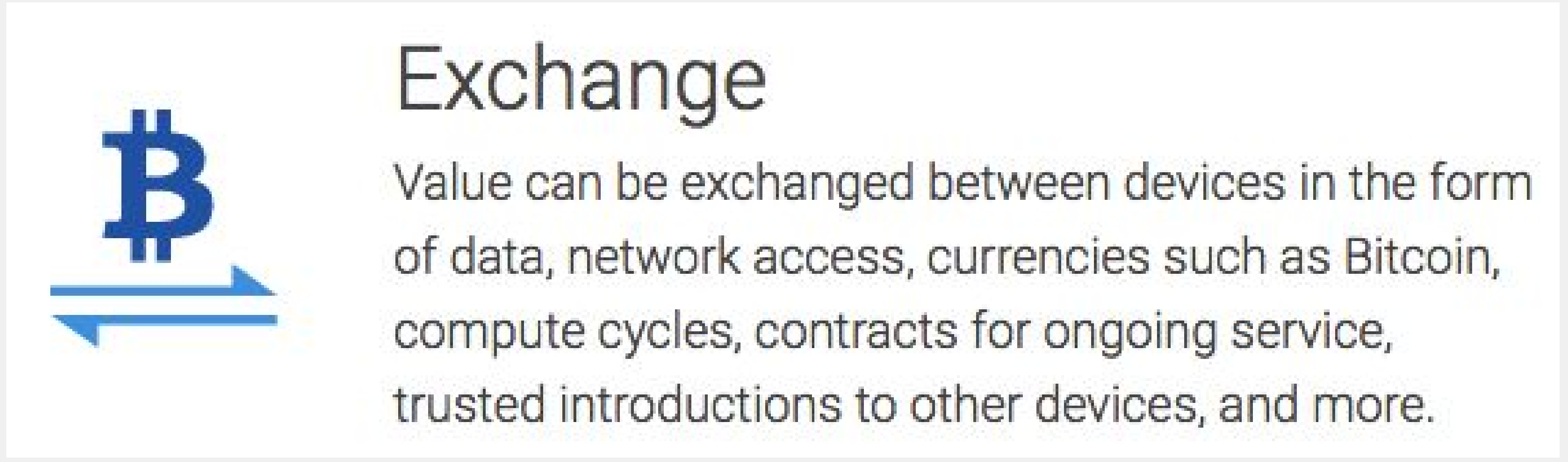
- "Blockchain-based decentralized Internet of Things"
- "Ad hoc mesh networks of smart sensors"
- Intended for industrial IoT applications

Product

- Sensors with 10 mile range
- battery lasts years
- no internet connection needed - uses mesh networking

Technologies used:

- **Telehash** - end-to-end message encryption
- **TMesh** - self-forming radio mesh networks
- **Blockname** - private device discovery
  - Uses Bitcoin blockchain + public notaries to verify authenticity of name/address bindings
- **Blocklet** - smart contracts and microtransactions



Exchange

Value can be exchanged between devices in the form of data, network access, currencies such as Bitcoin, compute cycles, contracts for ongoing service, trusted introductions to other devices, and more.

Filament is a great application of decentralized tech especially because of its emphasis on **resilience** and **dependability.**

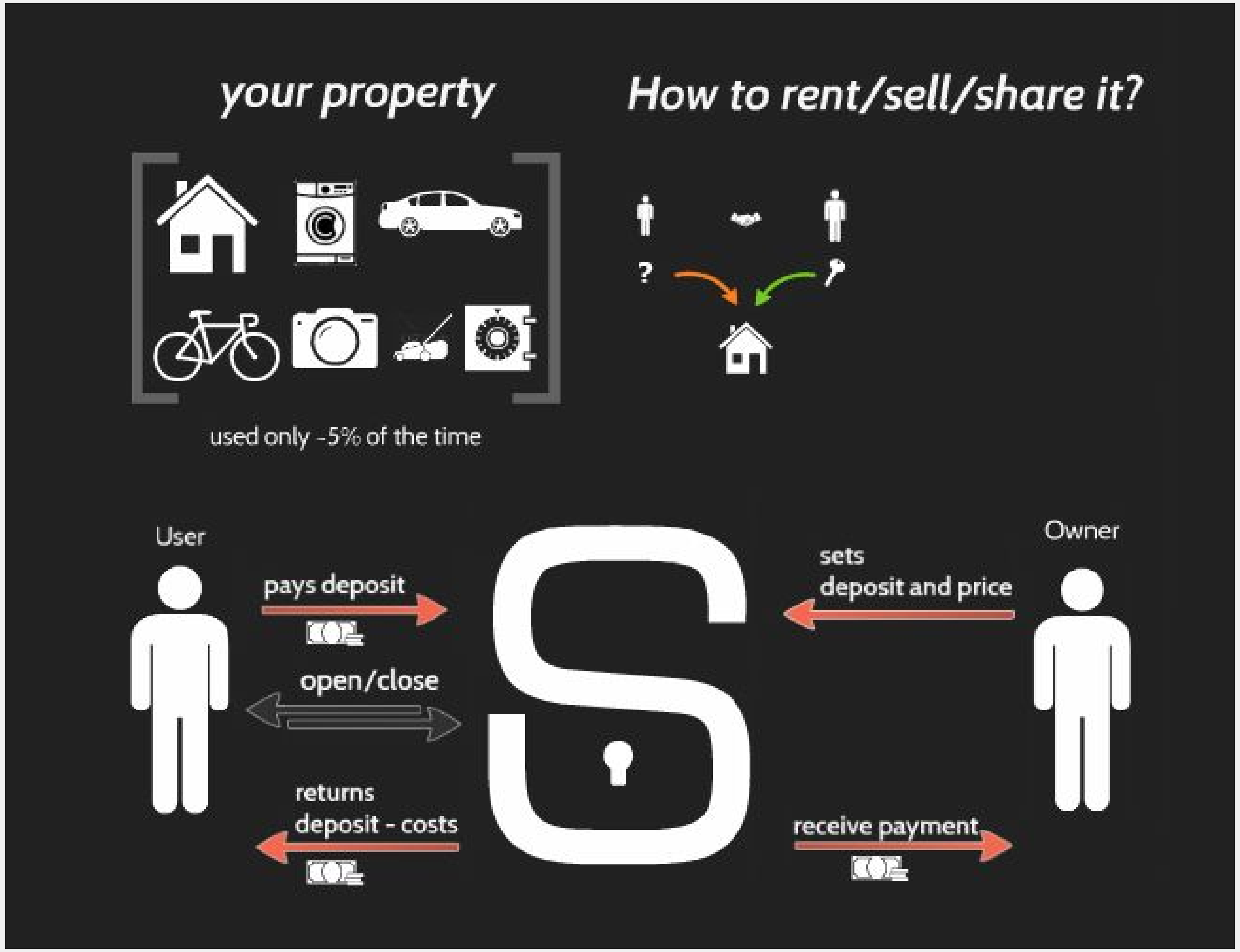
Decentralized Sharing Economy

Slock.it: A lock that can be directly opened by paying it

- Owner sets a deposit + price
- Renter pays deposit + price into lock connected to Ethereum node
- Lock detects payment and unlocks itself

Use Cases (Slock.it):

- Fully automated Airbnb apartments
  - no need to meet with owner for key
- Wifi routers rented on demand
- Fully automated shop
  - Purchase goods by sending the price of the good to the lock that holds it
- Automated bike rentals



Decentralized Sharing Economy

Benefits of being decentralized (Slock.it):

- ...Trustless?
  - Decentralized reputation is very hard to implement
  - Centralized solutions probably better
- Programmable money
  - Centralized solutions still programmable
  - Public blockchains (Bitcoin/Ethereum etc) have easier technological integration
  - No personal information needed to conduct transactions
- IoT: Device autonomy
  - Devices can act independently of a central management system
  - Modular

Decentralized Autonomous Organization

A **Decentralized Autonomous Organization**, or **DAO,** is an organization governed entirely by code (smart contracts)

- Create and vote on proposals
- Businesses can theoretically exist entirely on a blockchain
  - Uncertain legal status
- "Code is law"

Issues:

- Hard to edit the governing laws (the code) once deployed
  - Possible solution: Child DAO
- Inactive participants: not enough people vote on proposals

DASH

- Privacy-centric cryptocurrency
- % of mining reward

TheDAO

- Crowd venture fund
- Largest crowdfunded project in history
  - Raised >$150 million in Ether
- Hacked in June 2016, >$60 million worth of Ether stolen (10% of Ethereum market cap)

Other DAOs on Ethereum:

- Slock.it
- Digix.io
  - Store gold on the Ethereum blockchain

Blockchain at Berkeley

Generalizations:
Does it make sense to use
a blockchain?

Limitations of Smart Contracts and Blockchain tech

No trustless way to access outside data

- Must rely on **oracles** to provide information from outside the blockchain
  - Problem... Oracles must be trusted
- Potential Solution: **Proven execution** (untrusted oracles)
  - Oraclize.it has a shoddy implementation
    - TLSnotary - modification of TLS protocol to provide cryptographic proof of receiving https page
- Potential Solution: **Oracle network** votes on information
  - Drawback: Consensus protocol on top of a consensus protocol
  - Hard to align incentives/reputation

No way to enforce on-chain payments

- Cannot implement financial products like loans and bonds
  - Money must be held on blockchain to ensure payment
- Intuition: We pay interest on loans partially because of risk of default

Contracts cannot manipulate confidential data

- Confidential data cannot be assembled on someone else's computer
- Very limited access control capabilities
- Can only store encrypted data and decrypt it locally
- Potential solution: **Homomorphic encryption**

Essential Properties of Blockchain Killer Apps

Dapps can be thought of as **client-side software** - no central manager

Trustless environments that need consensus or coordination (ex. Smart grids, energy markets)

Privacy-centric systems (ex. social networks?)

- Although data shouldn't be stored on the blockchain itself

Programmable money with **open integration**

- **IoT**, **M2M** payments, (e.x. IBM ADEPT)
- Easy to send and receive money - no personal information required
- **Micropayments** possible (ex. Brave)

Fault-tolerant, resilient systems (ex. Filament)

- **Autonomous networks and devices**

News ways to creating incentives (ex. Gnosis)

New governance models (ex. DAOs, futarchy)

Disintermediation, censorship-resistance

Trust in math and code, not institutions

Contrast with centralization:

Deep integration, **cohesive user experience**

- **Efficiency** - blockchains are slow in general
- **Full control** over data and read/write permissions

ALWAYS ASK: Why is using a blockchain better than a central database?

Blockchain at Berkeley

Conclusion

Blockchain technology (and distributed tech) is going to lead to a new world of decentralization
Remember that theoretically, almost any app that exists today can be built in a completely decentralized way

Key question: **Why is blockchain better than using a centralized database?**

YOU GET A BLOCKCHAIN! AND YOU GET A BLOCKCHAIN!
EVERYBODY GETS A BLOCKCHAIN!!!

YO DAWG I HERD U LIEK SMART CONTRACTS
SO I PUT A CONTRACT IN YOUR CONTRACT SO YOUR CONTRACT CAN CONTRACT WITH A CONTRACT WHILE YOU CONTRACT
imgflip.com

Blockchain at Berkeley

Minor requests:

- Add yourself to our linkedin page:
  - www.linkedin.com/company/blockchain-at-berkeley
  - Post your position (Consultant or Developer)
- Join Facebook page: general & consulting
  - http://www.facebook.com/BlockchainBerkeley
- Keep checking your Slack for updates
  - We have a least 2-3 updates every day
  - Contribute to discussion, no BS

Team Formation

- Agree on two two-hour work sessions per week.
  - It is mandatory to attend at least one of them.
- Agree on a weekly scrum meeting time
  - Mandatory for all team members.
- Inform us about your agreed meeting times. It will be posted on the Blockchain at Berkeley calendar.
- <u>Consultants, be ready to present your project idea by next week.</u>
  - Ask the community and officers for help if needed.
- Devs, be ready to show us your deployed Ethereum contract by next week.
  - Dev environment setup session this Thursday
  - Use the Geth command line interface and OSX/Linux. Follow the Greeter tutorial at ethereum.org/greeter.

Blockchain at Berkeley

Teams

Irsyad Sjah - Supply Chain	Jon Allen - IoT	Selenne Berthely - Finance	Anthony DiPrinzio - Healthcare
Varun Agarwal	Sameer Reddy	William Park	Griffin Haskins
Richard Liu	Lucas Yum	Nihar Dalal	Brian Goodness
Alex Gao	Surya Duggirala	Griffin Baum	Arman Jabbari
Brett McGinnis	Aparna Krishnan	Justin Mi	Steven Cheng
Irra (Liangyuan) Na	Collin Chin	Yash Bhate	Ali Mousa

Blockchain
AT BERKELEY

Thanks!

We’re the world’s first university-based blockchain consulting firm.

Like us on Facebook:
@Berkeleyblockchain
https://www.facebook.com/BlockchainBerkeley/