Yoichi Hirai (Ethereum Foundation)
@pirapira
Nov. 2, 2017, Cancun

Morphing Smart Contracts with Bamboo

Ethereum is a Heavenly Programming Environment

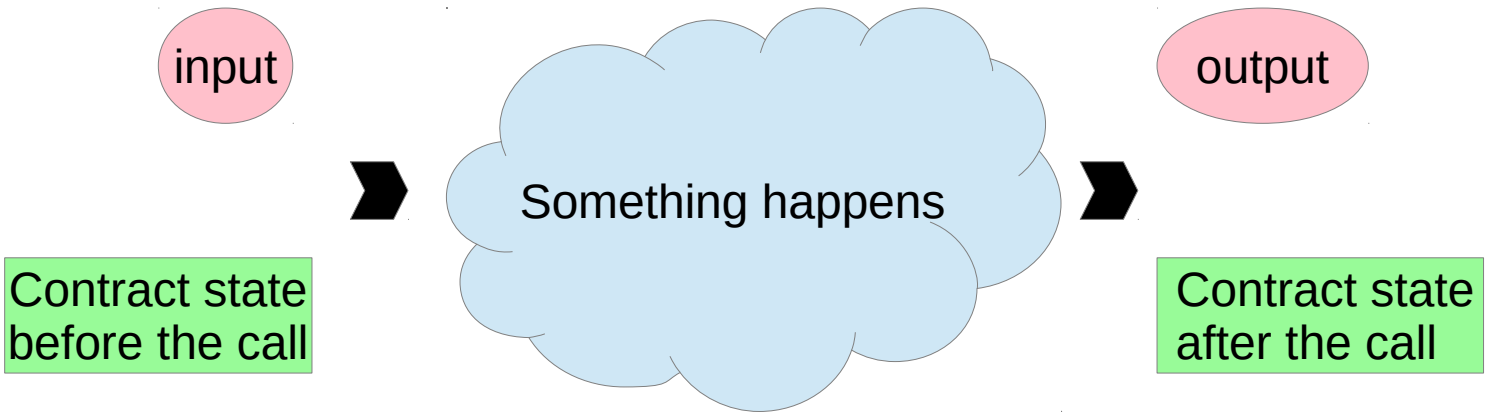
- Cosmic rays
- Malicious admins
- Wrong EVM implementations
- Cats

Bug-free Programming Pays in Ethereum

- Let’s aim there.
- Let’s match what happens and how a program looks.

One Mental Model of Ethereum Contracts

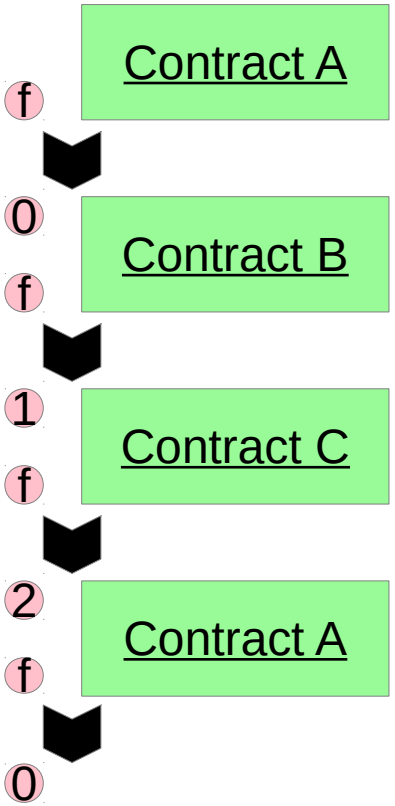
Something calls
the Ethereum contract



The Ethereum contract
waits until it's called again.

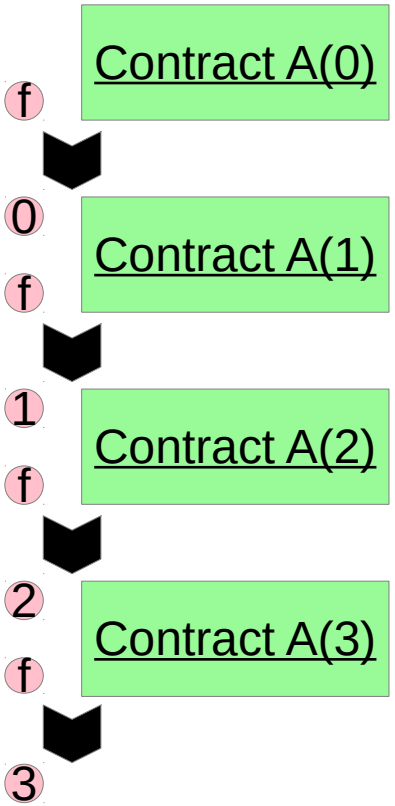
- (Haskellers or category theorists?)

The first Bamboo program




```
contract A()
{
    case (uint256 f()) {
        return 0 then become B();
    }
}
contract B()
{
    case (uint256 f()) {
        return 1 then become C();
    }
}
contract C()
{
    case (uint256 f()) {
        return 2 then become A();
    }
}
```

Look, Ma, no State Variables




```
contract A(uint256 counter)
{
    case (uint256 f()) {
        return counter then become A(counter + 1);
    }
}
```

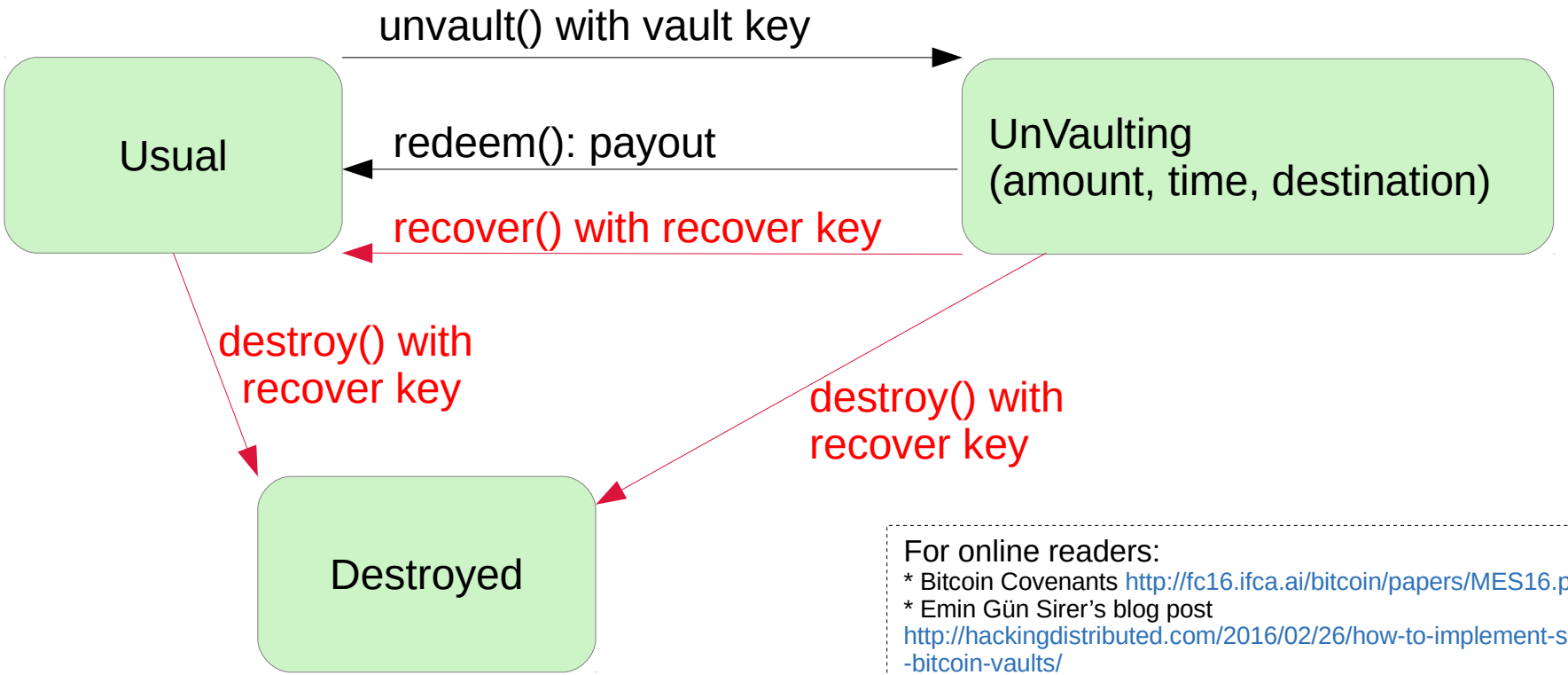
It‘s kind of similar to Erlang.

But I am trying to trap Solidity users with sugarly syntax.

Code from Learn You Some Erlang for Great Good
.

```
fridge1() ->
    receive
        {From, {store, _Food}} ->
            From ! {self(), ok},
            fridge1();
        {From, {take, _Food}} ->
            %% uh....
            From ! {self(), not_found},
            fridge1();
        terminate ->
            ok
end.
```

Ethereum contracts are games, like a vault



For online readers:

* Bitcoin Covenants http://fc16.ifca.ai/bitcoin/papers/MES16.pdf
* Emin Gün Sirer's blog post http://hackingdistributed.com/2016/02/26/how-to-implement-secure-bitcoin-vaults/
* Nick Johnson's implementation https://www.reddit.com/r/ethereum/comments/4wy6t9/ether_vault_store_your_ether_timelocked_for_easy/
* Dennis Peterson's http://www.blunderingcode.com/ether-vaults/

How to check vault.sol

Solidity code taken from http://www.blunderingcode.com/ether-vaults/

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey not_destroyed {
    <snip>
    }
    function redeem() only_vaultkey not_destroyed {
    <snip>
    }
    function recover(address _newHotwallet) only_recoverykey
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_destroyed {
        destroyed = true;
    }
}
```

How to check vault.sol – identify states

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey not_destroyed {
    <snip>
    }
    function redeem() only_vaultkey not_destroyed {
    <snip>
    }
    function recover(address _newHotwallet) only_recoverykey
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_destroyed {
        destroyed = true;
    }
}
```

Non-zero means UnVaulting.

True means Destroyed

How to check vault.sol—Check the Constructor

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey not_destroyed {
    <snip>
    }
    function redeem() only_vaultkey not_destroyed {
    <snip>
    }
    function recover(address _newHotwallet) only_recoverykey
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_destroyed {
        destroyed = true;
    }
}
```

…. results in Usual state.

How to check vault.sol-- Check Transitions from Usual

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey
    <snip>
    }
    function redeem() only_vaultkey not_destroye
    <snip>
    }
    function recover(address _newHotwallet) only
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_dest
        destroyed = true;
    }
}
```

From Usual	From UnVaulting	From Destroyed
Usual		
Usual(?) or UnVaulting		
Usual(?)		
Usual		
Destroyed		

How to check vault.sol-- Check Transitions from UnVaulting

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey
    <snip>
    }
    function redeem() only_vaultkey not_destroye
    <snip>
    }
    function recover(address _newHotwallet) only
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_dest
        destroyed = true;
    }
}
```

From Usual	From UnVaulting	From Destroyed
Usual	UnVaulting	
Usual(?) or UnVaulting	UnVaulting(?)	
Usual(?)	Usual	
Usual	Usual	
Destroyed	Destroyed	

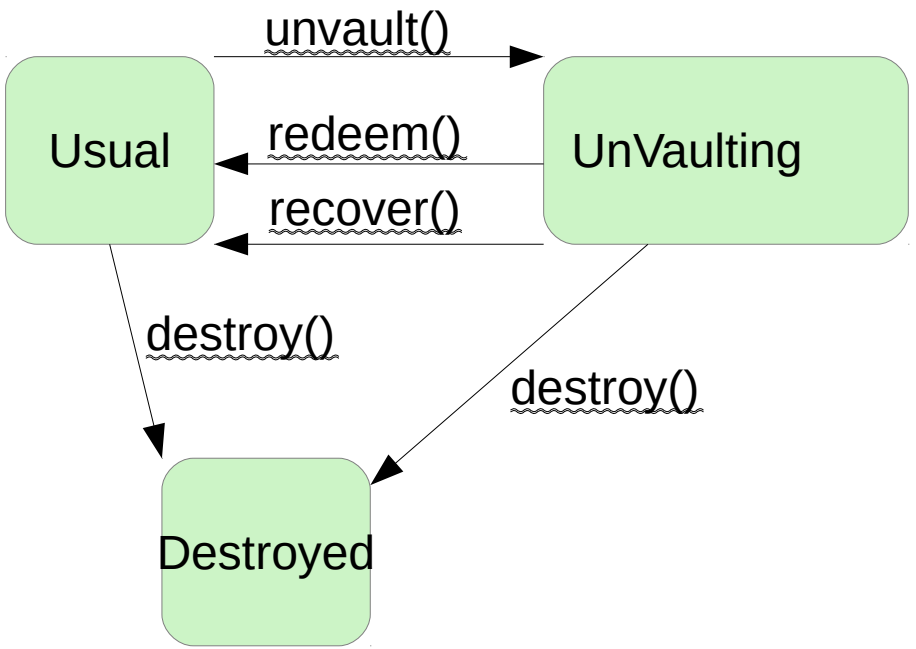
How to check vault.sol-- Check Transitions from Destroyed

```
contract Vault {
    uint public unvaultedAmount;
    bool public destroyed;
    <snip>
    function Vault(<snip>) {}
    function () not_destroyed {}
    function unvault(uint _amount) only_vaultkey
    <snip>
    }
    function redeem() only_vaultkey not_destroye
    <snip>
    }
    function recover(address _newHotwallet) only
not_destroyed {
    <snip>
    }
    function destroy() only_recoverykey not_dest
        destroyed = true;
    }
}
```

From Usual	From UnVaulting	From Destroyed
Usual	UnVaulting	Abort
Usual(?) or UnVaulting	UnVaulting(?)	Abort
Usual(?)	Usual	Abort
Usual	Usual	Abort
Destroyed	Destroyed	Abort

- You had to read the program three times!
- Reviewing a program takes at least #states × #lines

How to check vault.bbo




```
contract Vault(address vaultKey, address recoveryKey) {
  case(void unvault(uint256 _amount, address _hotWallet)) {
    <snip> return then become UnVaulting(<snip>);
  }
  case(void destroy()) {
    <snip> return then become Destroyed();
  }
}
contract UnVaulting(uint256 redeemtime, uint256 amount,
address hotWallet, address vaultKey, address recoveryKey) {
  case(void redeem()) {
    <snip> return then become Vault(vaultKey, recoveryKey);
  }
  case(void recover()) {
    <snip> return then become Vault(vaultKey, recoveryKey);
  }
  case(void destroy()) {
    <snip> return then become Destroyed();
  }
}
contract Destroyed() {
  // any call just throws;
}
```

More Language Features

Reentrancy Guard

```
void = hotWallet.default() with amount reentrance { abort; };
```

Creating a Contract

```
bid new_bid =
  deploy bid(sender(msg), value(msg), this) with value(msg)
  reentrance { abort; }; // failure throws.
```

Arrays

```
contract Token
(address => uint256 balances)
{
    case(bool transfer(address _to, uint256 _amount))
    {
        <snip: various checks>
        balances[_to] = balances[_to] + _amount;
        return true then become Token(balances);
    }
    <snip>
}
```

What‘s missing & Priorities

- Language Specification
- An independent interpreter

- Nicer error message.
- Integrate with truffle, embark etc.

- Detect unused local variable.
- Detect too much stack usage.
- Detect aliasing of mappings.

- Calling externally defined contracts.

- No functions
- No loops
- “Avoid success at all costs”

How you can help

Can OCaml	Have a look at https://github.com/pirapira/bamboo and tell me what you think
Know Linden Scripting Language or Erlang	Tell me your favorite features in these langs.
Can LaTeX	spec.tex!
Can draw	Logo!

The compiler probably has bugs. Lots of eyes needed.