Patterns:

- Repository of sample code

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

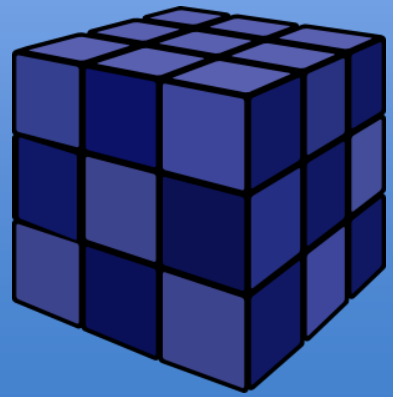
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



https://github.com/acloudfan/Blockchain-Course-Patterns

Patterns:

- Self Destruction

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

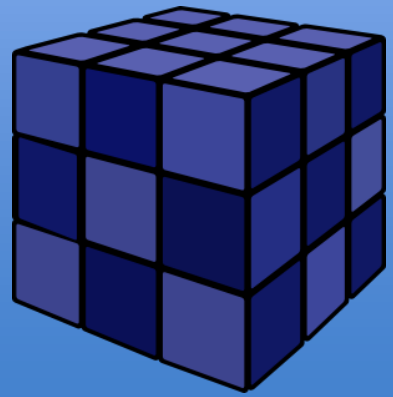
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

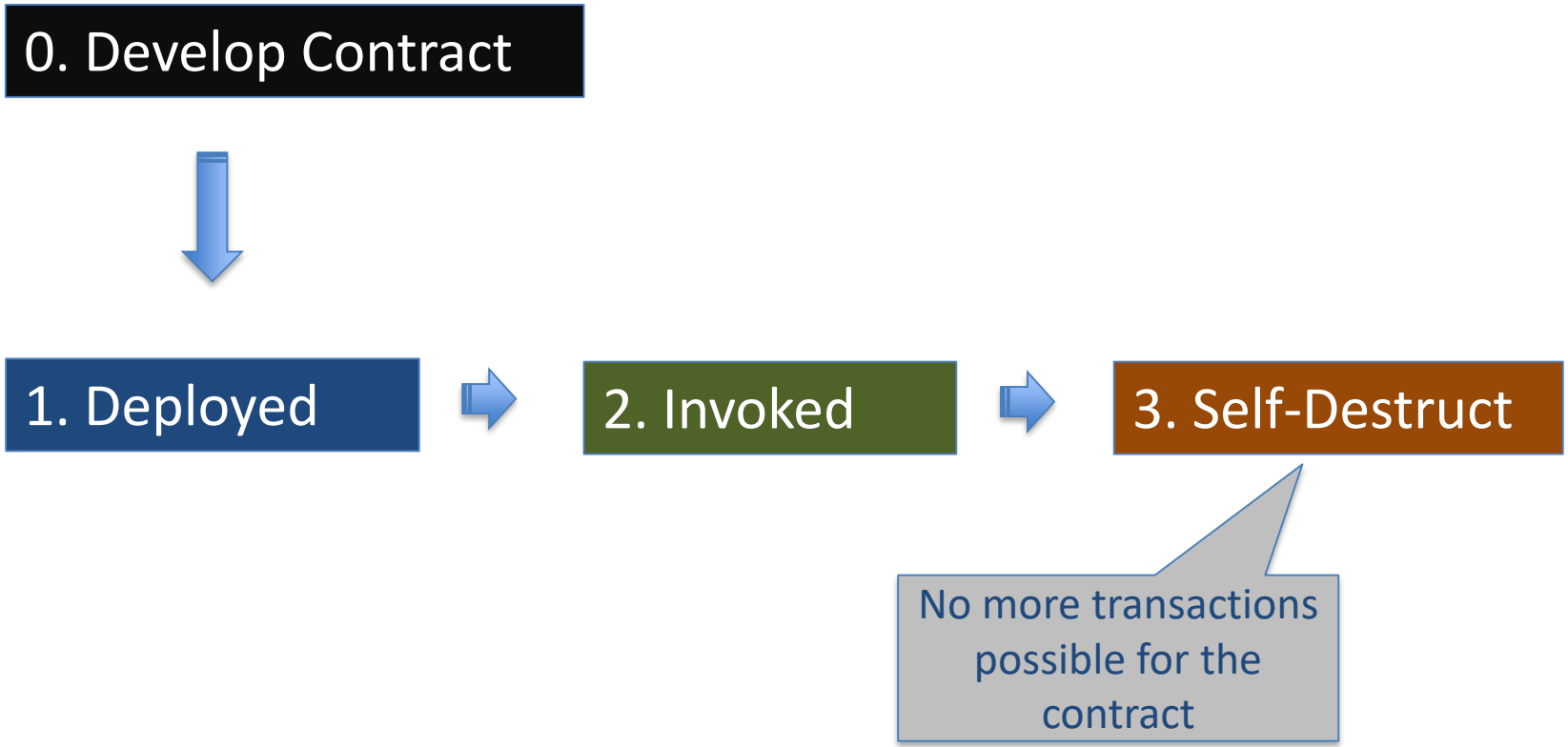
raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Contract Lifecycle



Why self destruct?

- Needs driven by business

Example: Timed contracts

E.g., Bidding, after the win no more bids allowed

Example: Nature of business

E.g., Loan contract destroyed after the loan is paid off

3. Self-Destruct

- The contract becomes unavailable after it self-destruct's

Existing transactions on contract stay forever but no new transaction

```
// This is where the contract is destroyed
function  killContract()  OwnerOnly {
  suicide(owner);
}
```

Restrict who can cause self destruction

All funds held in the contract sent to this address

Dead Contract Transactions

- Transactions will fail
- Fund's sent to a self destructed contract will be LOST

To prevent fund loss:

1. Remove all references to dead contracts
2. *OR* Call get before send to ensure that contract is NOT dead

Patterns:

- Contract Factory

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

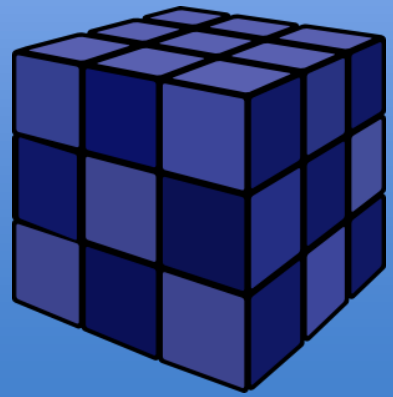
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

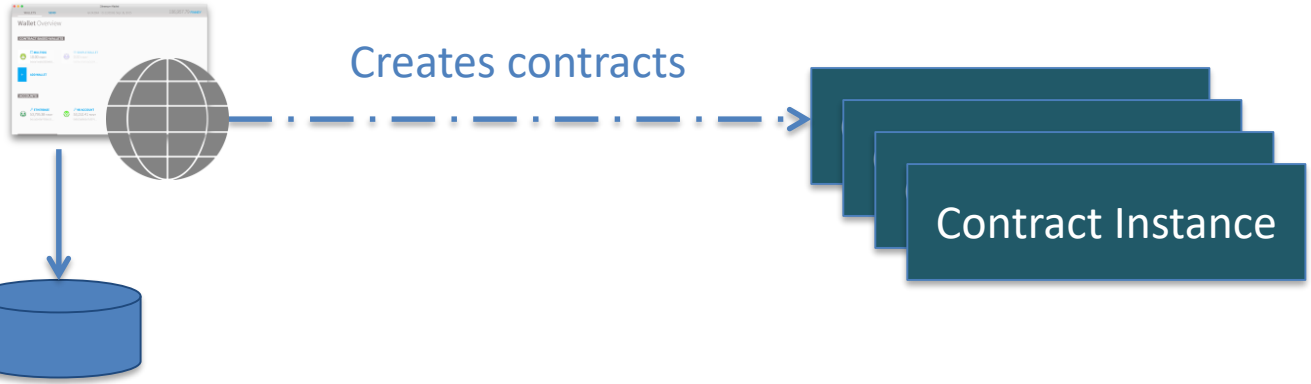
raj@acloudfan.com

@acloudfan

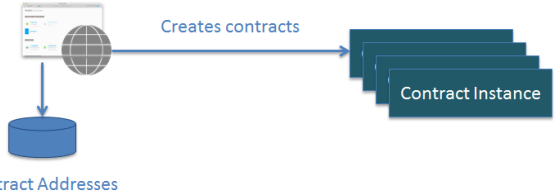
http://ACloudFan.com

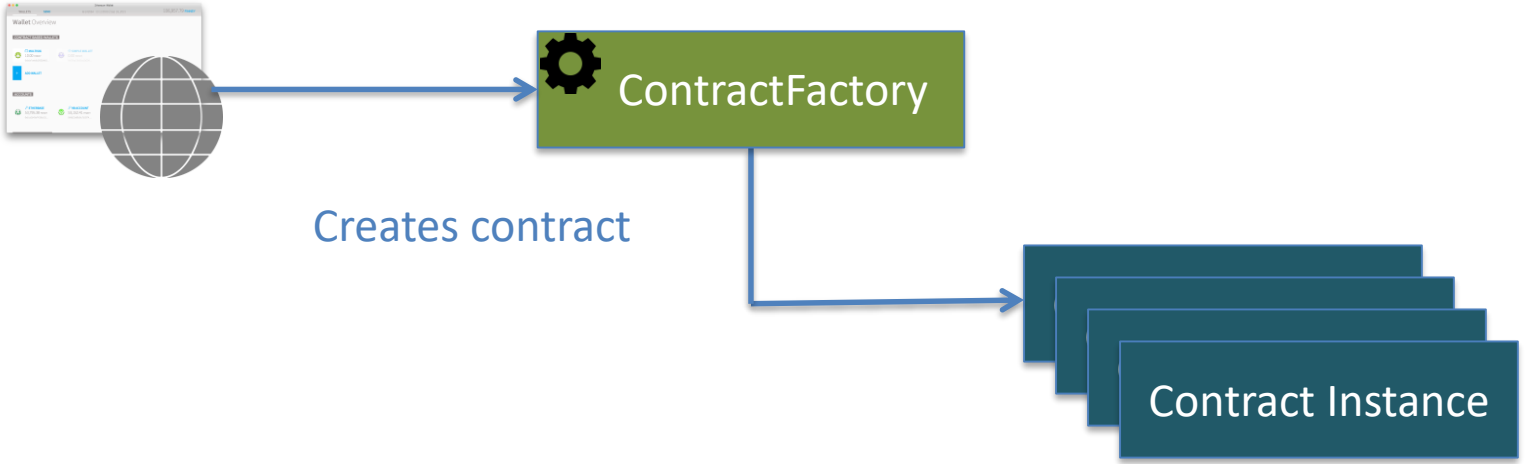


Contract Creation



Factory Pattern





- External persistent storage not needed
- Storage on EVM will cost gas

Benefits

- Hides the complexity & encapsulates business rules
- May manage the contracts in a collection
- Insulates the Dapp from contract changes & additions

Common Scenarios

- Digital tokens or assets
- Multi-contract management
- Rules for contract creation

Example

AccountFactory

SavingAccount

SilverAccount

GoldAccount

PlatinumAccount

Avg Balance >= $1000

Interest Rate= 1%

Transaction/month = 5

Avg Balance >= $10,000

Interest Rate= 3%

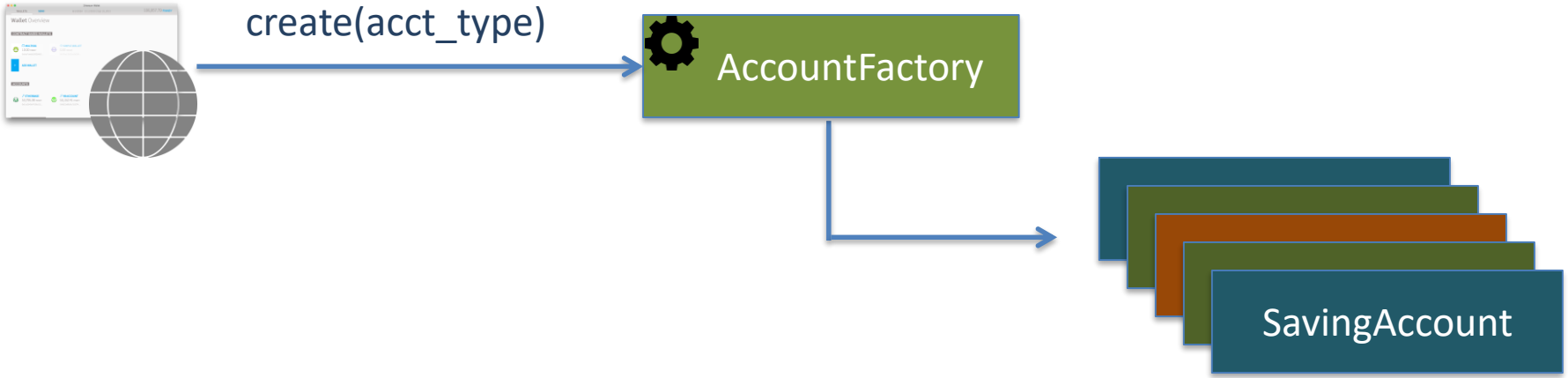
Transaction/month = 15

Avg Balance >= $100,000

Interest Rate= 6%

Transaction/month = 100

Factory Pattern



- Based on type a new contract is created
- New account types may be added without needing change in Dapp
- Factory contract instance may manage contracts in a collection

Patterns:

- Registry Pattern

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

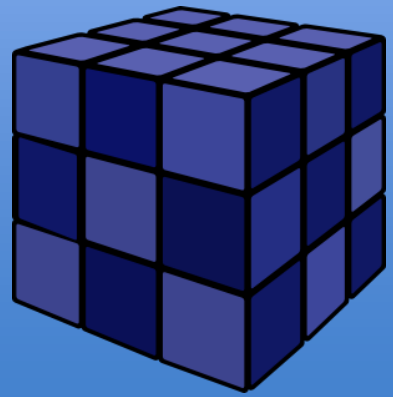
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

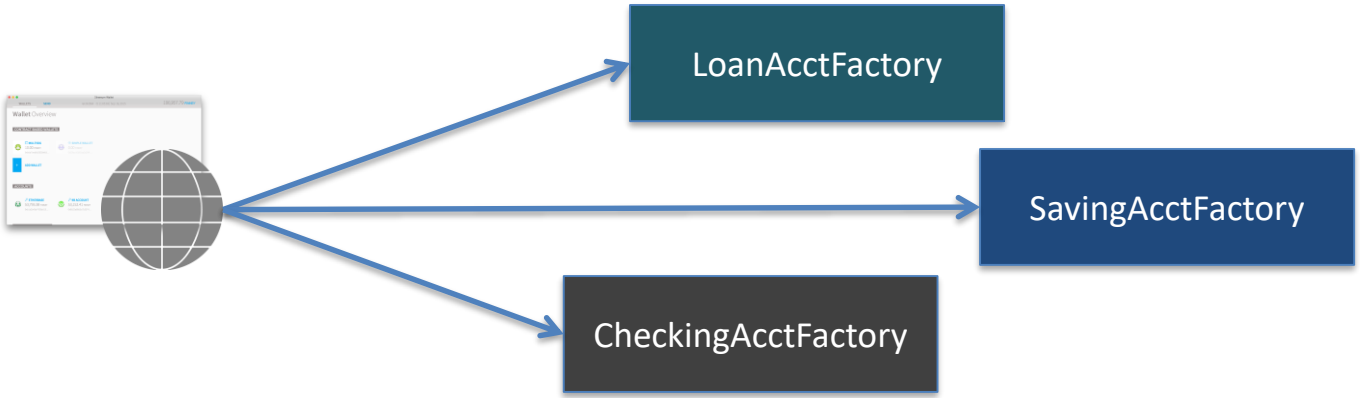
@acloudfan

http://ACloudFan.com



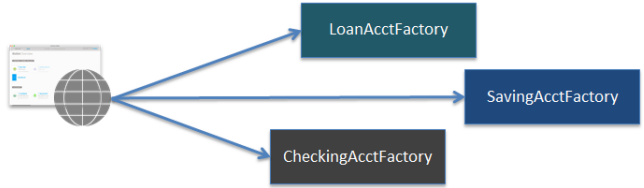
Contract References

- Complex apps may have dependency on multiple contracts



- App needs to maintain addresses for these contracts
- The contracts may need to be change/switched over time

Name Registry



Invoke Contract

getAddress('LoanFactory');

Registry

LoanAcctFactory

SavingAcctFactory

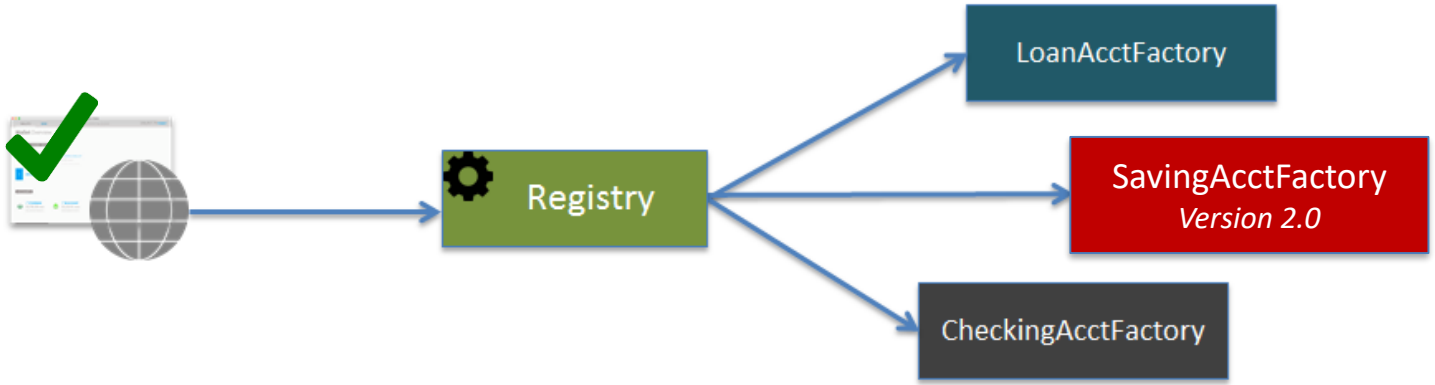
CheckingAcctFactory

Manages mapping

LoanFactory=0x….
SavingFactory=0x…
CheckingFactory=0x…

Benefits

- Simpler dependency management
  - Application uses names instead of addresses
  - Newer versions of dependency will not impact the Dapp



Sample

Invoke contract .at(**address**)

address

```
getContractInfo("CheckingAccountFactory")
```

NameRegistry

CheckingAccountFactory
CA_Address

SavingAccountFactory
NEW_SA_Address

```
nameRegistry.registerName ("CheckingAccountFactory", CA_Address,1)
nameRegistry.registerName ("SavingAccountFactory", SA_Address, 1)
```


```
nameRegistry.registerName("CheckingAccountFactory", NEW_SA_Address,2)
```

Patterns:

- Mapping Iterator

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

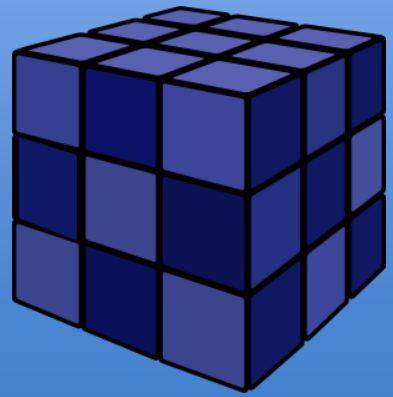
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Mapping Iterator

- No out of the box feature available

```
mapping(address => bytes32)  addressMap;
```

```
address[] keys = addressMap.keys()
```

// Not possible

Iteration Pattern

- Use a separate array for maintaining the keys

```
mapping(address => bytes32)  addressMap;

address[]   addresses;
```

- All operations carried out on both the mapping & the array

Implementation

Insert
Address = Name

Function
addKeyValue(…)

```
mapping(address => bytes32)  addressMap;

address[]   addresses;
```

Iterator

Get Key
count

Function
getKeyCount()

```
return addresses.length;
```

Loop
(0..count-1)

Function
getValueAtIndex()

```
return addressMap(addresses[index]);
```

Iterator Caveat

- As the number of keys will grow the cost of iteration will go up
  - Storage costs will go up
  - Optimization possible – avoid iteration
- Iteration in a constant function is OK

https://github.com/acloudfan/Blockchain-Course-Patterns

```
contract UserAddressRegistry
```

Patterns:

- Withdrawal Pattern

Discount Coupon Links to UDEMY courses:

https://www.udemy.com/hyperledger/?couponCode=DKHLF1099

https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099

https://www.udemy.com/rest-api/?couponCode=DKRST1099

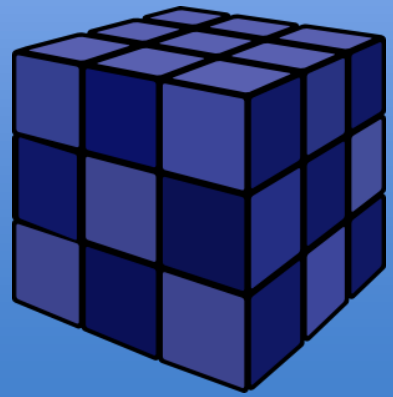
mentoring, seeking Blockchain part time work, project guidance, advice … …
http://www.bcmentors.com

This deck is part of a online course on "Ethereum: Design and Development of Decentralized Apps.

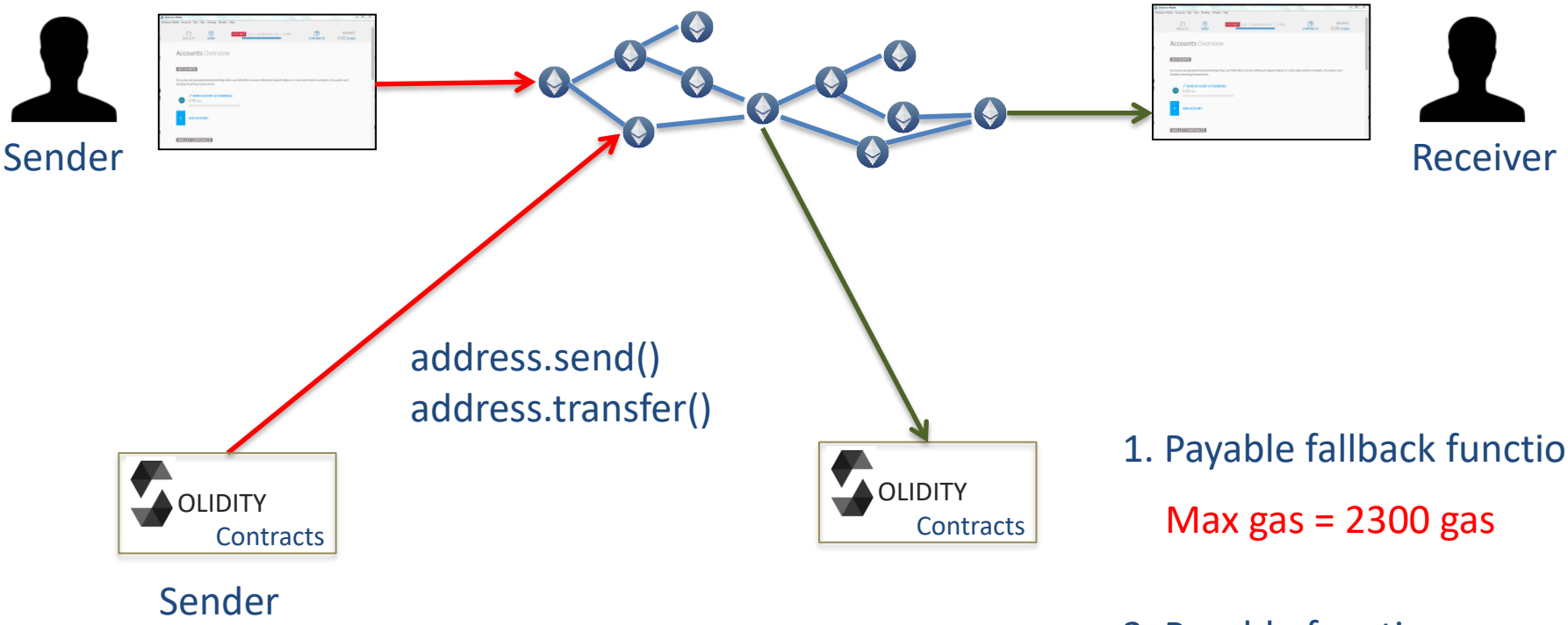
raj@acloudfan.com

@acloudfan

http://ACloudFan.com



Send Pattern

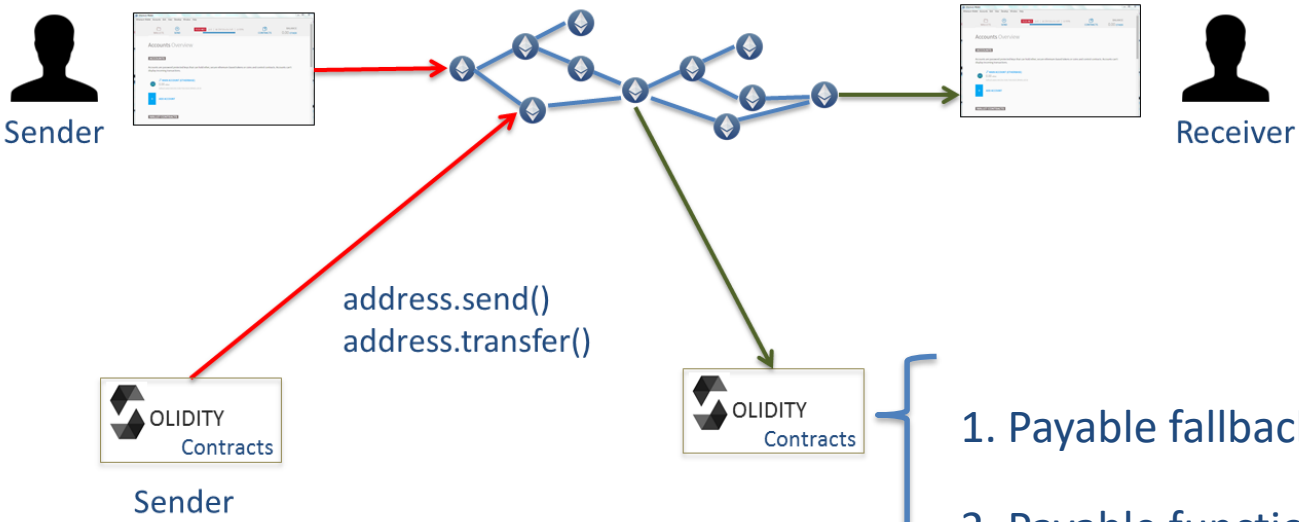


1. Payable fallback function

   Max gas = 2300 gas

2. Payable function

Send/Transfer can Fail



1. Payable fallback function runs out of gas
2. Payable function runs out of gas
3. Payable function throws exception

send() Vs. transfer()

address.send()

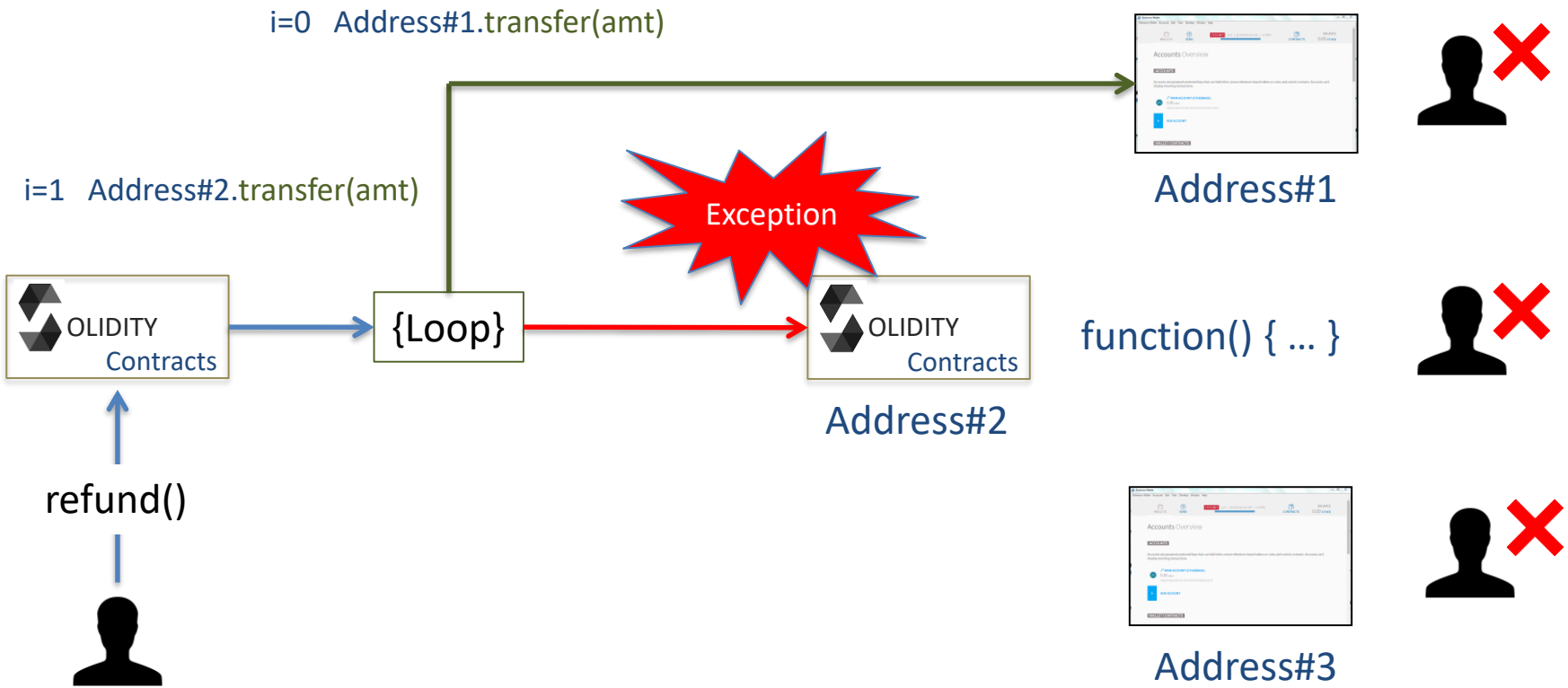
- Returns false on failure *BUT* it does not HALT contract execution

address.transfer()

- Throws exception on failure; HALTs the execution

Best Practice: Check return from send() Better to use transfer()

Challenge

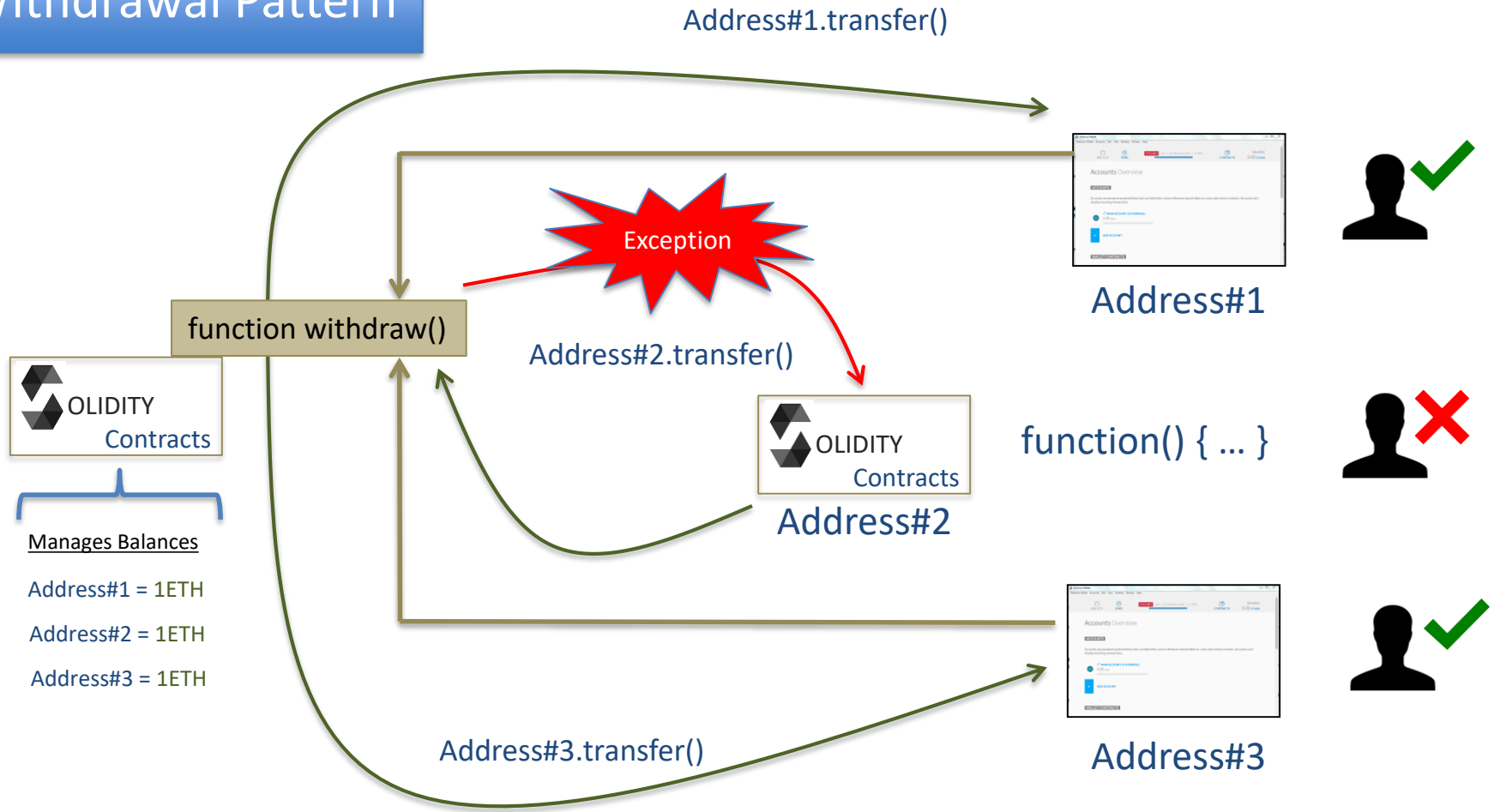


Funds stuck in the contract as exception will cause all state changes to Roll back

Withdrawal Pattern

- Sender contract exposes a withdraw function
  - The receivers invoke the withdrawal function for getting ethers

Withdrawal Pattern



Sender Versus Withdrawal Pattern

- Sender pattern
  - Sending contracts uses send() or transfer()
  - Send() returns false on failure ; transfer() throws an exception
  - Misbehaving code in fallback or payable function can cause issues
- Withdrawal pattern
  - Receiver calls the withdraw() function to initiate send/transfer