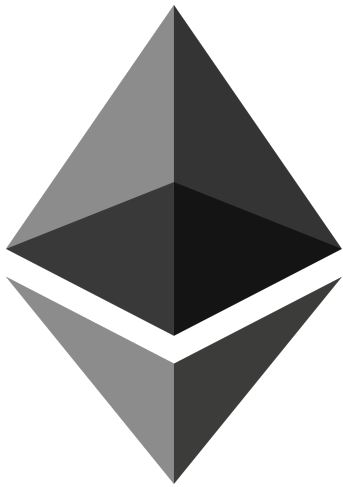




Ethereum and Smart contracts

Module 2 (part 3)

Part 3

Gas and transaction limits

Ethereum Gas – is the lifeblood of the Ethereum ecosystem



- Gas is a unit that measures the amount of computational effort that it will take to execute certain operations on EVM.
- Every **transaction** (either smart contract or non-smart contract transaction) requires at least **21,000 gas** (it is the gas limit for standard transactions)
- The total cost of a transaction (the "TX fee") is the **Gas Limit * Gas Price**.

*payment is calculated in Gas and gas is paid in ETH.

Gas in Ethereum is a necessary evil

- All miners and full nodes must evaluate all transactions (including running associated code if it is a smart contract tx)
 - limit computation cost
- All miners must store all state
 - limit storage use
- Short-cut the halting problem
 - There is an upper GAS_LIMIT, so all programs will halt

Check out [Etherscan](https://etherscan.io).

Every EVM operation has a fixed gas cost

	opcodes	gas cost
Basic operations	ADD, MUL, PUSH, JUMP	2-10
Storage read	SLOAD	200
Storage write	SSTORE	5000
Storage write (from zero)	SSTORE	20000
Storage zeroize	SSTORE	-10000
Contract call	CALL, CODECALL, etc.	700
Transaction overhead	n/a	21000
Contract creation	n/a	32000
Contract destruction	SELFDESTRUCT	-19000

Analogy for gas limit and gas price

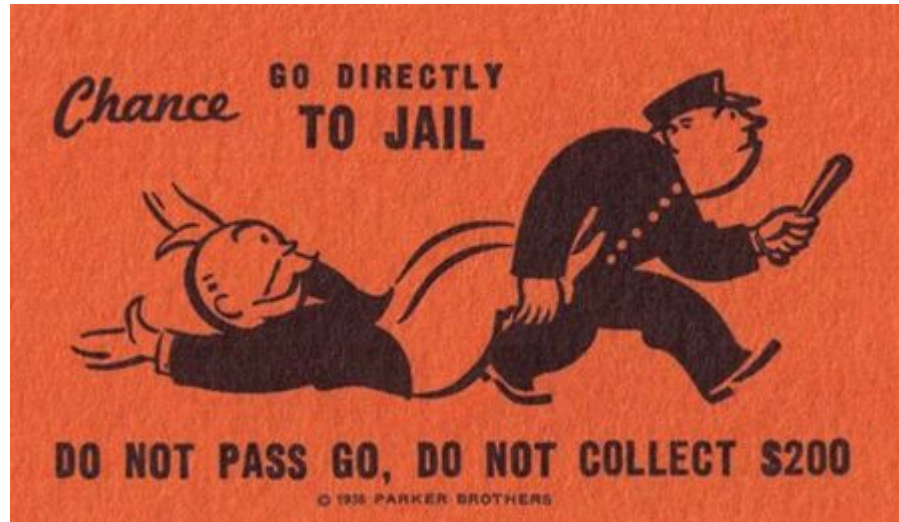
- You can think of the **gas limit** like the amount of liters/gallons/units of gas for a car. You can think of the **gas price** as the cost of that liter/gallon/unit of gas.
 - **With a car**, it's \$2.50 (price) per gallon (unit).
 - **With Ethereum**, it's 20 GWEI (price) per gas (unit).
- To fill up your "tank", it takes...
 - 10 gallons at \$2.50 = \$25
 - 21000 units of gas at 20 GWEI = 0.00042 ETH.

Therefore, the total TX fee will be 0.00042 Ether.

Gas limit

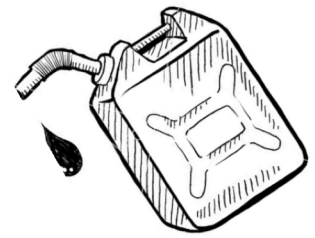
- Why is it called limit?
- Because it's the maximum amount of units of gas you are willing to spend on a transaction. This avoids situations where there is an error somewhere in the contract, and you spend 1 ETH....10 ETH....1000 ETH..... going in circles but arriving no where (see this [slide](#))
- If you do not want to spend as much on gas, lowering the gas limit won't help much, **why?**

Out-of-gas exceptions are bad news



- State reverts to previous value
 - Except that $\text{START_GAS} * \text{GAS_PRICE}$ is still deducted

Infinite loops?



What if a contract has an infinite loop?



A pragmatic solution:

- A transaction requires “**gas**” to fuel contract execution
- Each EVM opcode requires a fixed amount of gas to execute
- Every transaction specifies the maximum ether the sender is willing to spent on the transaction (max gas + gas price)
- If the contract successfully executes, the unspent ether is refunded to the sender
- If execution runs out of gas, the execution reverts without refunding ether to the transaction sender

Gas price

- If you want to **spend less** on a transaction, you can do so by lowering the amount you pay per unit of gas. The price you pay for each unit increases or decreases how quickly your transaction will be mined.
- **During normal times:**
 - 40 GWEI Gas Price will almost always get you into the next block.
 - 20 GWEI will usually get you within the next few blocks.
 - 2 GWEI will usually get you within the next few minutes.

Miners choose transactions based on GAS_PRICE

Send funds

FROM

 Account 1 - 1.00 ETHER

TO

0x000000..

AMOUNT

0.0

ETHER

1.00 ETHER

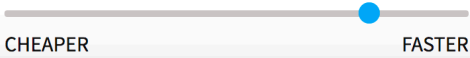
☐ Send everything

You want to send **0 ETHER**.

SHOW MORE OPTIONS

SELECT FEE

0.000053 ETHER



This is the most amount of money that might be used to process this transaction. Your transaction will be mined **probably within 30 seconds**.

Can you customize Gas?

 METAMASK

Send ETH

Only send ETH to an Ethereum address.

From:  AtlantaWolf
21.999043 ETH
\$2,372.24 USD

To: 0x5b7f69B1452068cD60810: 

Amount: 1 ETH
[Max](#) \$107.83 USD

Transaction Fee:

Fastest 0.00063 ETH \$0.07	Fast 0.0001 ETH \$0.01	Slow 0.00004 ETH \$0.00
---	-------------------------------------	--------------------------------------

[Advanced Options](#)

CANCEL NEXT

Send ETH

Customize Gas

Close

Advanced

New Transaction Fee

0.000105 ETH

~Transaction Time

~40 sec

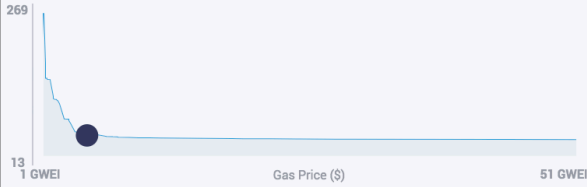
Gas Price (GWEI)

5

Gas Limit

21000

Live Gas Price Predictions



Slower

Faster

Send Amount

Transaction Fee

New Total

1 ETH

0.000105 ETH

1.000105 ETH

\$107.85

CANCEL

SAVE

NEXT



MyEtherWallet Behind-The-Scenes

Made by @veenspace

START HERE

1. You **sign** your transaction
2. You **send** your transaction

WITH MEW!



TX



3. Your transaction goes to a **MEW node...**

5 GETH, 5 PARITY NODES

infura /
etherscan /
MEW node...

*ABOVE THIS LINE IS WHAT MEW IS RESPONSIBLE FOR
BELOW IS JUST HOW THE BLOCKCHAIN WORKS.*

5. Miners **pick transactions** from the pool
HIGH GAS PRICES PICKED FIRST!

4. ...who put **it in the pool** of all signed transactions
IF YOUR TX IS PENDING, IT'S IN THE POOL



6. Miners then **put transactions in a block and add it to the chain**



ONCE IN THE BLOCKCHAIN, YOUR TX IS PERMANENT!

Will increasing the gas price get it mined faster? Does setting a low gas price mean it won't ever be mined?

Where can I see what miners are accepting?

<http://ethgasstation.info/>



Recommended Gas Prices

(based on current network conditions)

Speed	Gas Price (gwei)
SafeLow (<30m)	2
Standard (<5m)	3
Fast (<2m)	20

Sending and receiving Ether

Method	<code>address.send()</code>	<code>address.transfer()</code>	<code>address.call.value()()</code>
Possibility to set gas limit	no	no (but may be in future)	yes
Gas limit	2300	2300	settable via <code>.gas(gasLimit)</code>
Return value when error	false	throws exception	false

send all amount in **Wei** to a given address

Where this gas goes to?

the payable function of the contract/ fallback function

APIs for ether transfer

	Gas provided to the receiver	Error propagation
<code>Transfer(amount)</code>	2300	yes
<code>Send(amount)</code>	2300	no
<code>Call.value(amount)()</code>	All remaining gas	no

Ether transfers

An ether transfer to a contract implicitly calls the receiver contract

```
contract A {  
    address b = 0xbeef;  
    function foo() {  
        b.transfer(10);  
    }  
}
```



Contract address: 0xbeef

```
contract B {  
    function () payable{  
        // log payment  
    }  
}
```

Important for security:
Contract B takes control over the execution

Authorization in smart contracts

Can any user call arbitrary functions in contracts?

- Yes. The contract must explicitly restrict access to sensitive functions

```
contract OwnableWallet {  
    address owner = 0x1234;  
  
    function critical() {  
        msg.sender.transfer(10);  
    }  
}
```

Authorization in smart contracts

Can any user call arbitrary functions in contracts?

- Yes. The contract must explicitly restrict access to sensitive functions

```
contract OwnableWallet {  
    address owner = 0x1234;
```

```
    function critical() {
```

```
        require(msg.sender == owner);
```

```
        msg.sender.transfer(10);
```

```
    }
```

```
}
```

Only the owner can
invoke this function

All transactions specify START_GAS, GAS_PRICE

1. If $\text{START_GAS} \times \text{GAS_PRICE} > \text{caller.balance}$, halt
2. Deduct $\text{START_GAS} \times \text{GAS_PRICE}$ from `caller.balance`
3. Set $\text{GAS} = \text{START_GAS}$
4. Run code, deducting from GAS
5. For negative values, add to GAS_REFUND
 - a. GAS only decreases
6. After termination, add GAS_REFUND to `caller.balance`

Back of envelope numbers you should know

Average gas price (as of March '18): <https://etherscan.io/chart/gasprice>

<http://ethgasstation.info/>

~20gigawei = 0.00000002 ether

Price of Ether (as of March '18):

~\$500 per Ether

Cost per transaction:

21000 gas “base” for a transaction = \$0.21 **(21 cents per transaction)**

Cost of data:

~75 gas per byte of data stored = \$0.77/kB **(77 cents per kilobyte)**

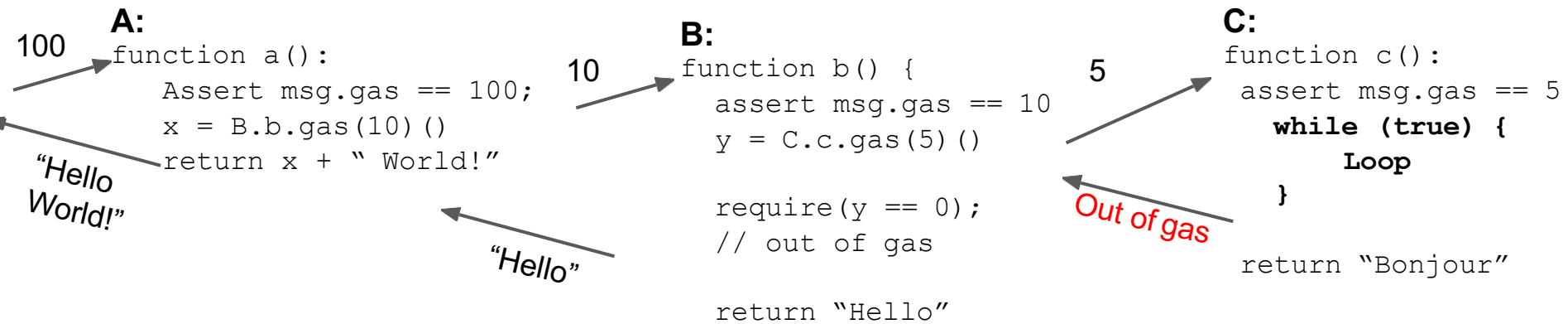
Built-in support for calling other contracts

- `a.transfer(x)` sends `x` to address `a`
 - returns 0 if this fails due to call stack
- `foo.call.value(3).gas(20764)( bytes4(sha3("bar()")));`
 - also `callcode`, `delegatecall`
 - default is 0 value, all available gas
- `new` constructor deploys a new contract
 - Careful, it's expensive!

Remember:

Smart contracts code is fixed *forever*.
Calls required to update functionality

Callers can choose how much gas to send



(pseudocode: exact syntax used here does not work in Solidity)

Where to Get Testnet Ether?

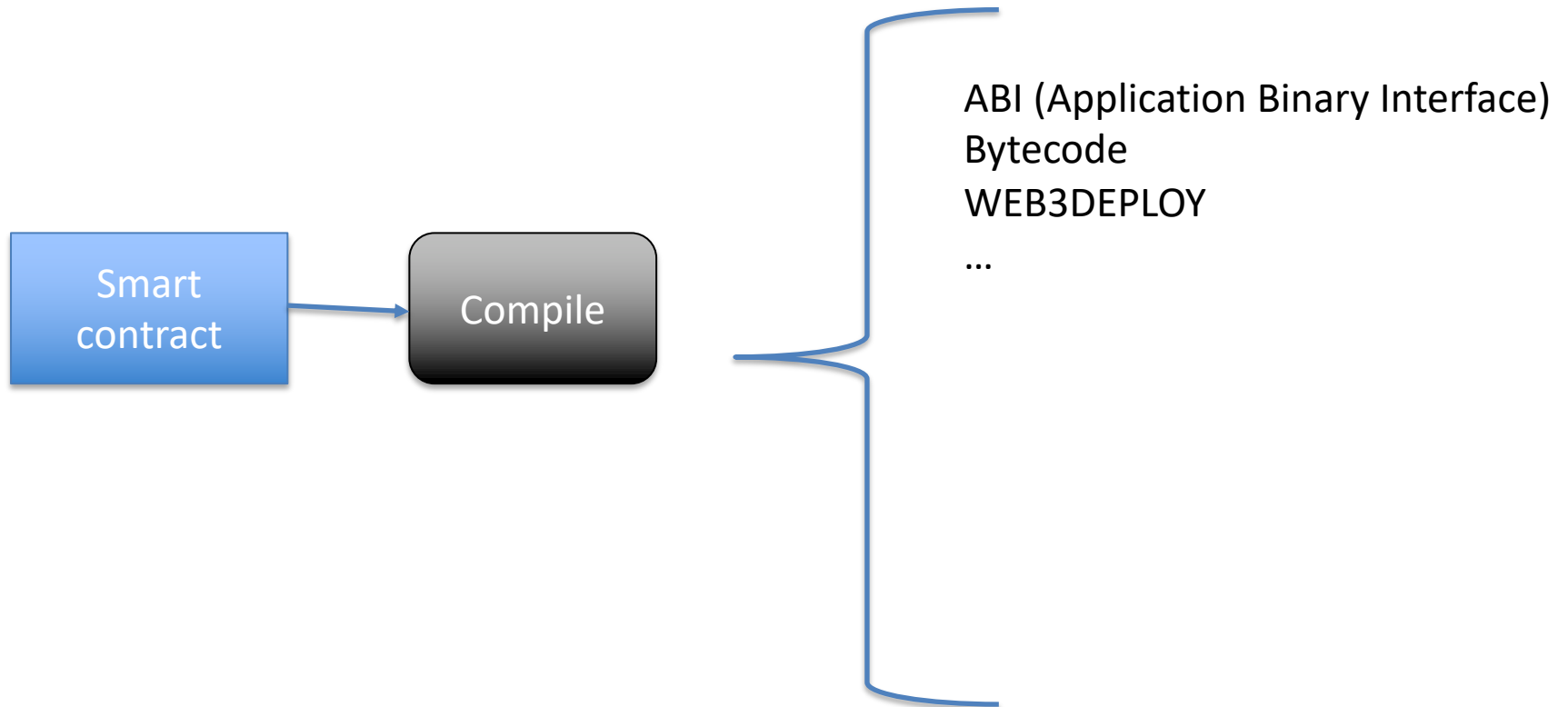
- There are currently 3 different testnets for the Ethereum public chain:
 - Ropsten
 - Rinkeby
 - Kovan

check out the [link](#).

Deploy and use contracts

Putting It All Together

Compile Artifacts



Let's see a demo on Remix IDE

Address of contract creator

The screenshot displays the Remix IDE interface. On the left, a file explorer shows a project named 'browser' containing 'FirstContract.sol', 'Greeter.sol' (selected), 'ballot.sol', and 'ballot_test.sol'. Below it is a 'config' section. The main editor shows the Solidity code for 'Greeter.sol' (lines 1-27). The code defines a 'Mortal' contract with an 'owner' variable and a 'kill()' function. It then defines 'Greeter' as inheriting from 'Mortal', with a constructor that sets the 'greeting' and a 'greet()' function that returns the 'greeting'.

On the right, the 'Run' tab is active, showing deployment settings. The 'Environment' is set to 'JavaScript VM'. The 'Account' is '0xca3...a733c (99.99999999999987830)'. The 'Gas limit' is '3000000'. The 'Value' is '0' 'wei'. The 'Contract' dropdown is set to 'Greeter'. The 'Deploy' button is highlighted in red, and the 'HelloMate' string is entered in the adjacent field. Below this, the 'At Address' section has a 'Load contract from Address' button. The 'Transactions recorded' section shows 1 transaction. The 'Deployed Contracts' section shows the 'Greeter' contract at address '0x8c1...401f5 (memory)', with 'kill' and 'greet' buttons below it.

Two blue arrows are present: one pointing from the top right towards the 'Account' field, and another pointing from the bottom right towards the 'Greeter at 0x8c1...401f5 (memory)' entry in the 'Deployed Contracts' section.

Address of the contract (will be created after it is deployed)

Deploy a smart contract

- ▶ 1. Write the code offline
- ▶ 2. Paste it into [remix.Ethereum.org](https://remix.ethereum.org)
- ▶ 3. Get Ropsten ether for metamask
- ▶ 4. Deploy it with metamask

There are more than one way!

Main references

Ethereum:

<https://www.ethereum.org/>

Solidity documentation:

https://solidity.readthedocs.io/en/latest
/

Solidity with examples:

<https://learnxinyminutes.com/docs/solidity>