MONASH University

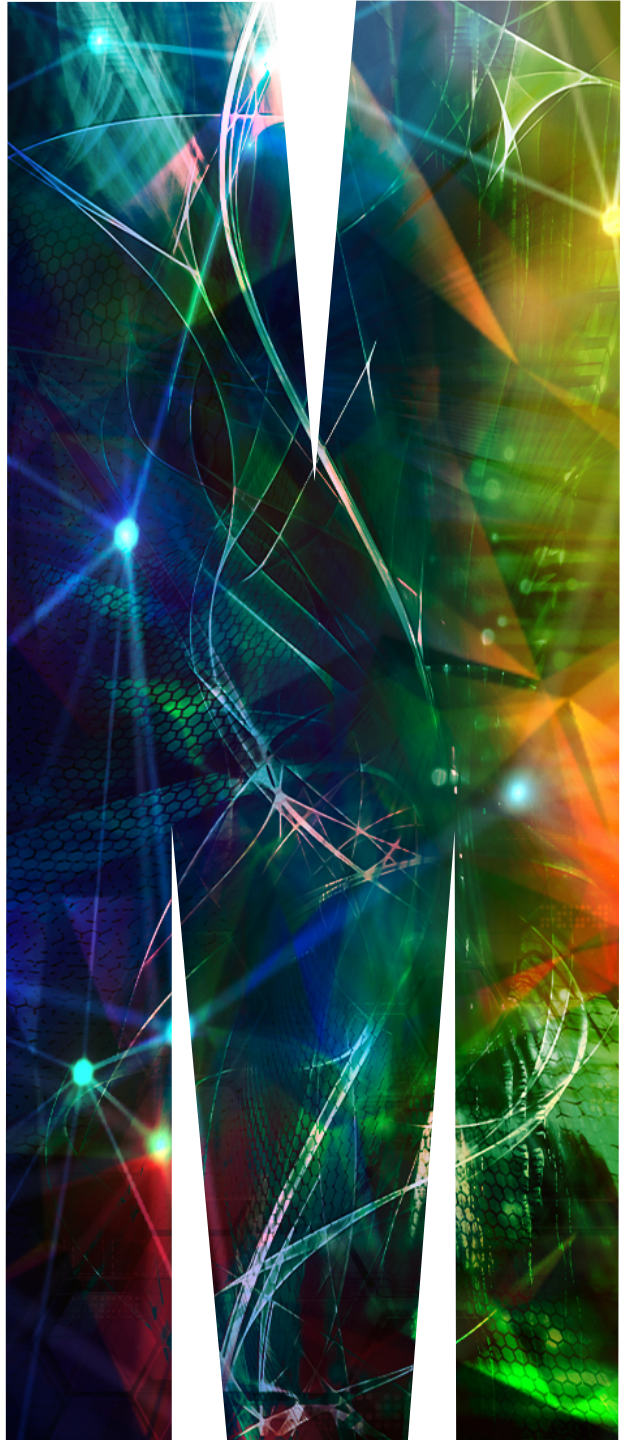
FIT5214: Blockchain

Lecture 3: Ethereum and Smart Contract

Lecturer: Rafael Dowsley

rafael.dowsley@monash.edu

https://dowsley.net



Unit Structure

- **Lecture 1: Introduction to Blockchain**
- **Lecture 2: Bitcoin**
- Lecture 3: Ethereum and Smart contracts
- Lecture 4: Proof-of-Work (PoW)
- Lecture 5: Attacks on Blockchains
- Lecture 6: Class Test/Alternatives to PoW
- Lecture 7: Proof-of-Stake (PoS)
- Lecture 8: Privacy
- Lecture 9: Byzantine Agreement
- Lecture 10: Blockchain Network
- Lecture 11: Payment Channels
- Lecture 12: Guest Lecture

Unit Structure

- **Lecture 1: Introduction to Blockchain**
- **Lecture 2: Bitcoin**
- **Lecture 3: Ethereum and Smart contracts**
  - ➡ Introduction to Ethereum and smart contract
  - ➡ Understand “DAO” and “DAO” attacks
  - ➡ Other key vulnerabilities
- Lecture 4: Proof-of-Work (PoW)
- Lecture 5: Attacks on Blockchains
- Lecture 6: Class Test/Alternatives to PoW
- Lecture 7: Proof-of-Stake (PoS)
- Lecture 8: Privacy
- Lecture 9: Byzantine Agreement
- Lecture 10: Blockchain Network
- Lecture 11: Payment Channels
- Lecture 12: Guest Lecture

Digital currency and smart contract - A brief history

Digital currency and smart contract - A brief history

The concept of e-Cash

- **1982** David Chaum

Digital currency and smart contract - A brief history

The concept of e-Cash

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)

Digital currency and smart contract - A brief history

The concept of e-Cash

The concept of PoW preventing email spam

The concept of smart contract

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)
  - **1994** First payment sent
- **1997** Hashcash (Hal Finney)
- **1997** Nick Szabo
- **1998** b-money (Wei Dai)

Digital currency and smart contract - A brief history

The concept of e-Cash

The concept of PoW preventing email spam

The concept of smart contract

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)
  - **1994** First payment sent
- **1997** Hashcash (Hal Finney)
- **1997** Nick Szabo
- **1998** b-money (Wei Dai)
- **1998** Bit gold (never implemented)
  by Nick Szabo

Digital currency and smart contract - A brief history

The concept of e-Cash

The concept of PoW preventing email spam

The concept of smart contract

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)
  - **1994** First payment sent
- **1997** Hashcash (Hal Finney)
- **1997** Nick Szabo
- **1998** b-money (Wei Dai)
- **1998** Bit gold (never implemented)
  by Nick Szabo
- **1998 PayPal**

Digital currency and smart contract - A brief history

The concept of e-Cash

The concept of PoW preventing email spam

The concept of smart contract

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)
  - **1994** First payment sent
- **1997** Hashcash (Hal Finney)
- **1997** Nick Szabo
- **1998** b-money (Wei Dai)
- **1998** Bit gold (never implemented)
  by Nick Szabo
- **1998 PayPal**
- **2008 Bitcoin (Satoshi Nakamoto)**

Digital currency and smart contract - A brief history

The concept of e-Cash

The concept of PoW preventing email spam

The concept of smart contract

- **1982** David Chaum
- **1990** DigiCash
  (Founded by Chaum)
  - **1994** First payment sent
- **1997** Hashcash (Hal Finney)
- **1997** Nick Szabo
- **1998** b-money (Wei Dai)
- **1998** Bit gold (never implemented)
  by Nick Szabo
- **1998 PayPal**
- **2008 Bitcoin (Satoshi Nakamoto)**
- **2009** Bitcoin software released
- **2014 Ethereum (Vitalik Buterin)**

MONASH University

Smart Contract

The concept of smart contract was first proposed by Nick Szabo in **1997.**

Szabo defines **smart contracts** as agreements that are derived from legal principles, but enforced by cryptographic protocols.

Smart Contract

The concept of smart contract was first proposed by Nick Szabo in **1997.**

Szabo defines **smart contracts** as agreements that are derived from legal principles, but enforced by cryptographic protocols.

A **smart contract** is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract.

— Wikipedia

Smart Contract

So, a contract is *"**smart**"* if

- it is run by a machine rather than a person;
- the assets in the contract are moved automatically.

Smart Contract

User-defined programs running on top of a blockchain

Smart Contract

User-defined programs running on top of a blockchain

Cryptocurrency

Transfer 5 coins from Address X to Address Y

Distributed ledger, or “Blockchain”

Smart Contract

User-defined programs running on top of a blockchain

Smart Contract

User-defined programs running on top of a blockchain

Smart Contract

If the 200th block contains more transactions than the 201th block, then transfer 5 coins from Address X to Address Y

Distributed ledger, or “Blockchain”

Group Discussion:

Does Bitcoin support smart contracts?

Answer:

Answer:

Bitcoin supports smart contracts via a Turing-incomplete Script language, examples include *multi-signature accounts*, *payment channels*, *atomic cross-chain trading*.

Smart Contract Languages

Smart contracts on a blockchain can be written in different programming languages, such as

- Solidity (Ethereum)
- Golang/Java/JavaScript (Hyperledger Fabric)
- C++ (EOS)
- …

Decentralized Application (DApp)

- A DApp is a computer application that runs on a distributed system.
- With Blockchain, a **smart contract is a DApp — it runs on the distributed ledger.**

Ethereum

Ethereum is an open source programmable blockchain platform.

- **Nov. 2013.** Ethereum was proposed by Vitalik Buterin.
- **July - August 2014.** Ethereum crowdsourcing campaign >$18 million through initial coin offering (by selling Ether – the Ethereum coins/tokens).
- **30th July 2015.** The first release.
- **Mar. 2016.** Official release.

Potential applications

Potential applications

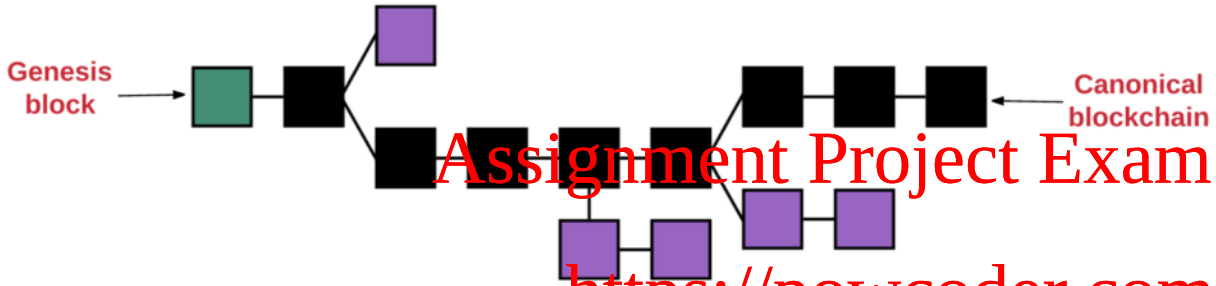
- Crowdfunding/initial coin offering (ICO)
- Gambling
- Supply-chain management
- Power trading and management
- Internet of Things
- Asset Issuance
- Marketplaces
- …

Ethereum Blockchain

Different from Bitcoin:

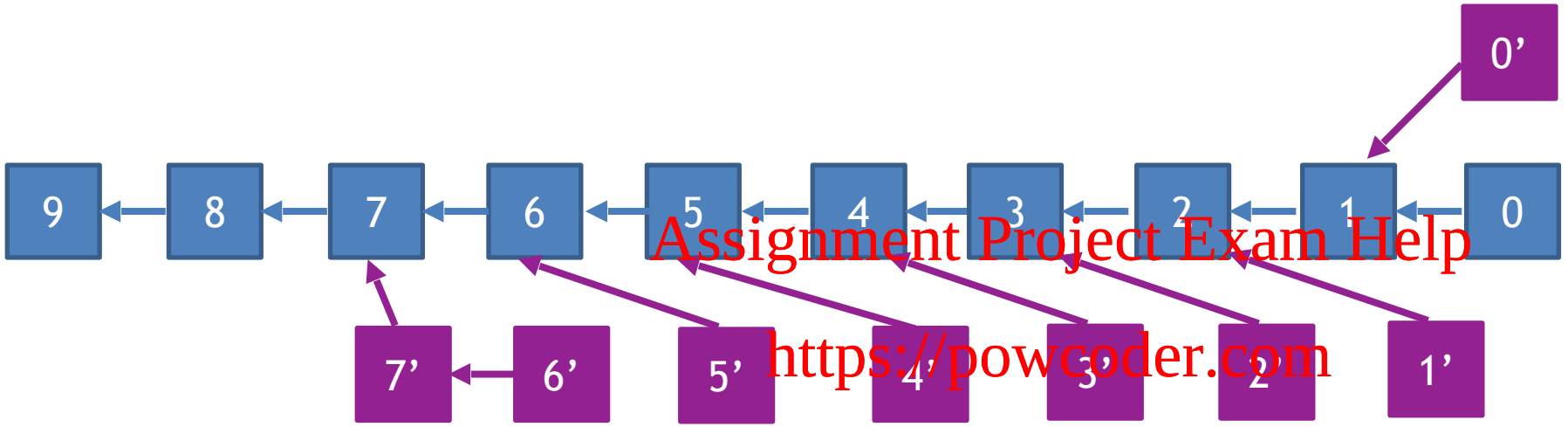
- Block generation: on average one block every 12 seconds through PoW
- Mining algorithm: Ethash
  - Memory bound
  - Helps mitigate ASIC and GPU advantages
- Difficulty is adjusted every block
- Hash algorithm:
  - SHA3_256 (a.k.a. Keccak-256) and
  - SHA3_512 (a.k.a. Keccak-512)

Simplified GHOST (Greedy Heaviest Observed Subtree)



Choose the path that has the most computation done

Uncle blocks



Uncles are orphan blocks, i.e., with parents that are ancestors (max 6 blocks back) of a block in the canonical blockchain.

Ethereum mining reward

A successful PoW miner receives:

- A static block reward of X Ether
  - Before Block 4369999 (16-Oct-2017)
    X=5
  - Block 4,370,000 — Block 7,279,999 (16-Oct-2017 — 28-Feb-2019)
    X=3
  - Since Block 7,280,000 (28-Feb-2019 — present)
    X=2
- Transaction fees (all the gas consumed, explained later)
- An extra of 1/32 of 2 Ether for including Uncles as part of the block. (Can include up to two Uncles)
  - Uncles receive 7/8 of the static block reward, an uncle can only be included by one valid block.

EVM and Solidity

EVM	Solidity

EVM and Solidity

EVM	Solidity
❖ **EVM** is a virtual machine for Ethereum, it can run smart contracts.	❖ **Solidity** is a language for writing smart contracts supported by EVM.

EVM and Solidity

EVM	Solidity
❖ **EVM** is a virtual machine for Ethereum, it can run smart contracts.	❖ **Solidity** is a language for writing smart contracts supported by EVM.
❖ EVM "understands" EVM bytecode	❖ Solidity (High-level language) is deterministically compiled into EVM bytecode.

EVM and Solidity

EVM	Solidity
❖ **EVM** is a virtual machine for Ethereum, it can run smart contracts.	❖ **Solidity** is a language for writing smart contracts supported by EVM.
❖ EVM "understands" EVM bytecode	❖ Solidity (High-level language) is deterministically compiled into EVM bytecode.
❖ All Ethereum nodes need to have EVM to run all smart contracts to verify the transactions, like in BTC. Thus, they all share the same EVM states.	❖ Solidity programs are capable of expressing all tasks accomplishable by computers, making them theoretically Turing complete. (C, C++, C#, Java, Python.. are all Turing complete.)

Smart contract in Ethereum

- Smart contract is written in Solidity;
- Solidity is deterministically compiled into EVM bytecode
- Each smart contract (bytecode) is contained in a transaction, just like BTC, and will be recorded in the blockchain;
- When one user uploads a smart contract through their Ethereum node, it is included in one block and propagated around the network, where it is stored on every other node in the network.
- Since smart contract needs to be run and verified on every full node, it needs to be guaranteed to execute, and the result is deterministic (for verification).

Ether and Gas

Ether and Gas

- **Ether is the currency**
  Similar to Bitcoin, in Ethereum miners are paid “ether” for mining.

Ether and Gas

- **Ether is the currency**
  Similar to Bitcoin, in Ethereum miners are paid “ether” for mining.
- **Gas** is a unit of work measuring the computational cost of an operation in EVM. Users can buy gas by using Ether
  - ➡ **Ethereum Gas is the “fuel” for running smart contract**
    Just like the fuel for cars :-)
  - ➡ Gas guarantees that all “works” to process a smart contract or a transaction is properly priced.
  - ➡ Complicated contracts require more gas to operate — this prevents DoS attacks.

Ether and Gas

- **Ether is the currency**
  Similar to Bitcoin, in Ethereum miners are paid “ether” for mining.

- **Gas** is a unit of work measuring the computational cost of an operation in EVM. Users can buy gas by using Ether
  - ➡ **Ethereum Gas is the “fuel” for running smart contract**
    Just like the fuel for cars :-)
  - ➡ Gas guarantees that all “works” to process a smart contract or a transaction is properly priced.
  - ➡ Complicated contracts require more gas to operate — this prevents DoS attacks.

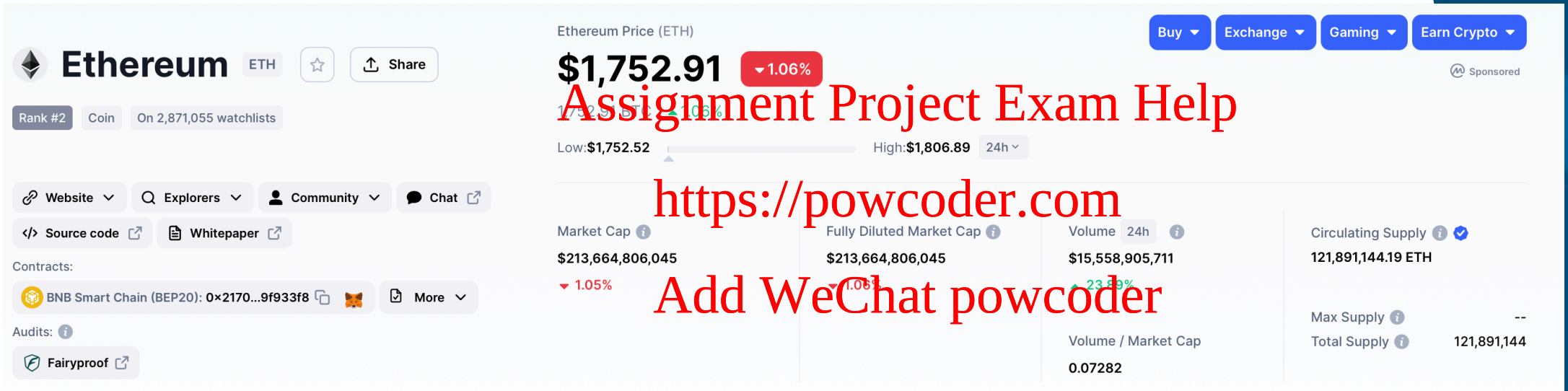
- **Miners are paid “gas” for running smart contracts on the network**
  Use Ether to buy Gas, and use Gas as “transaction fees” to store and run the smart contract

Denominations of ether

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

http://ethdocs.org/en/latest/ether.html

Ether



https://coinmarketcap.com/currencies/ethereum/

MONASH University

Ether

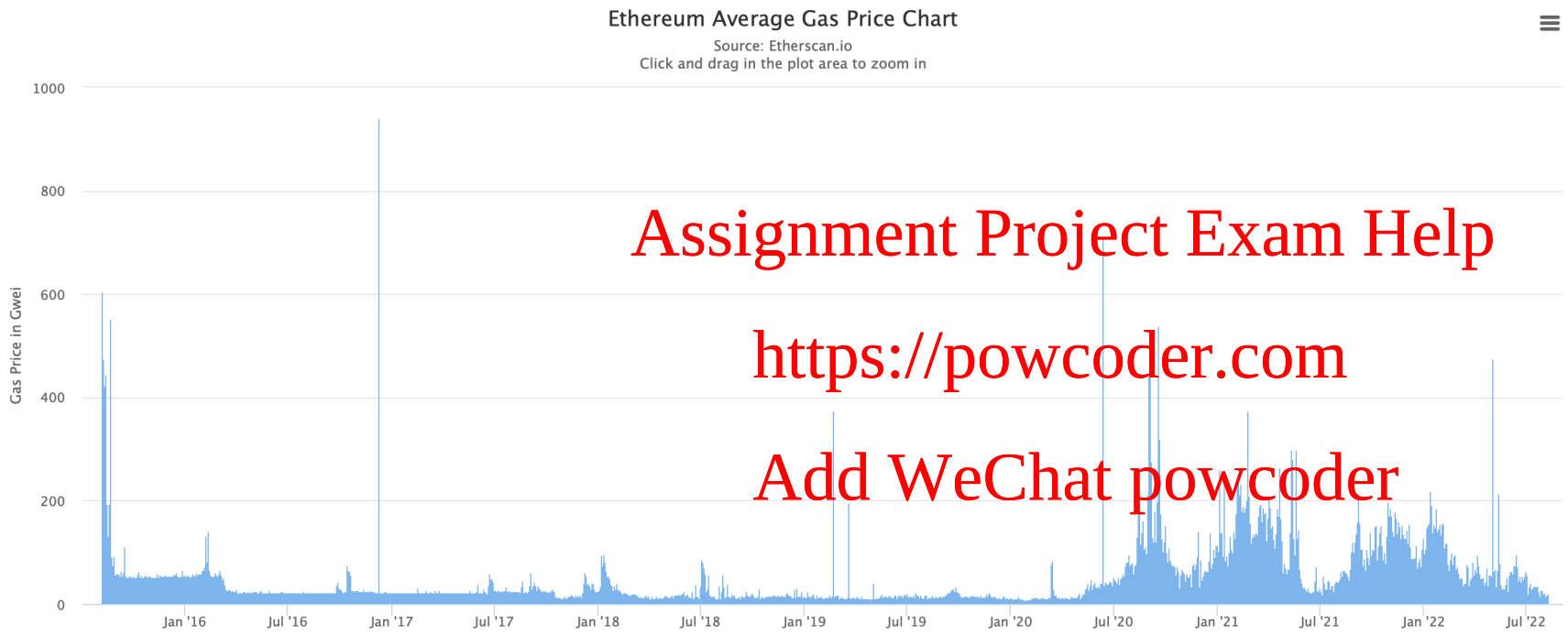
https://coinmarketcap.com/currencies/ethereum/

Gas cost

Each operation (Opcode) in the EVM was assigned a number of how much gas it consumes.

Operation Name	Gas Cost	Remark
step	1	default amount per execution cycle
stop	0	free
suicide	0	free
sha3	20	
sload	20	get from permanent storage
sstore	100	put into permanent storage
balance	20	
create	100	contract creation
call	20	initiating a read-only call
memory	1	every additional word when expanding memory
txdata	5	every byte of data or code for a transaction
transaction	500	base fee transaction
contract creation	53000	changed in homestead from 21000

Gas

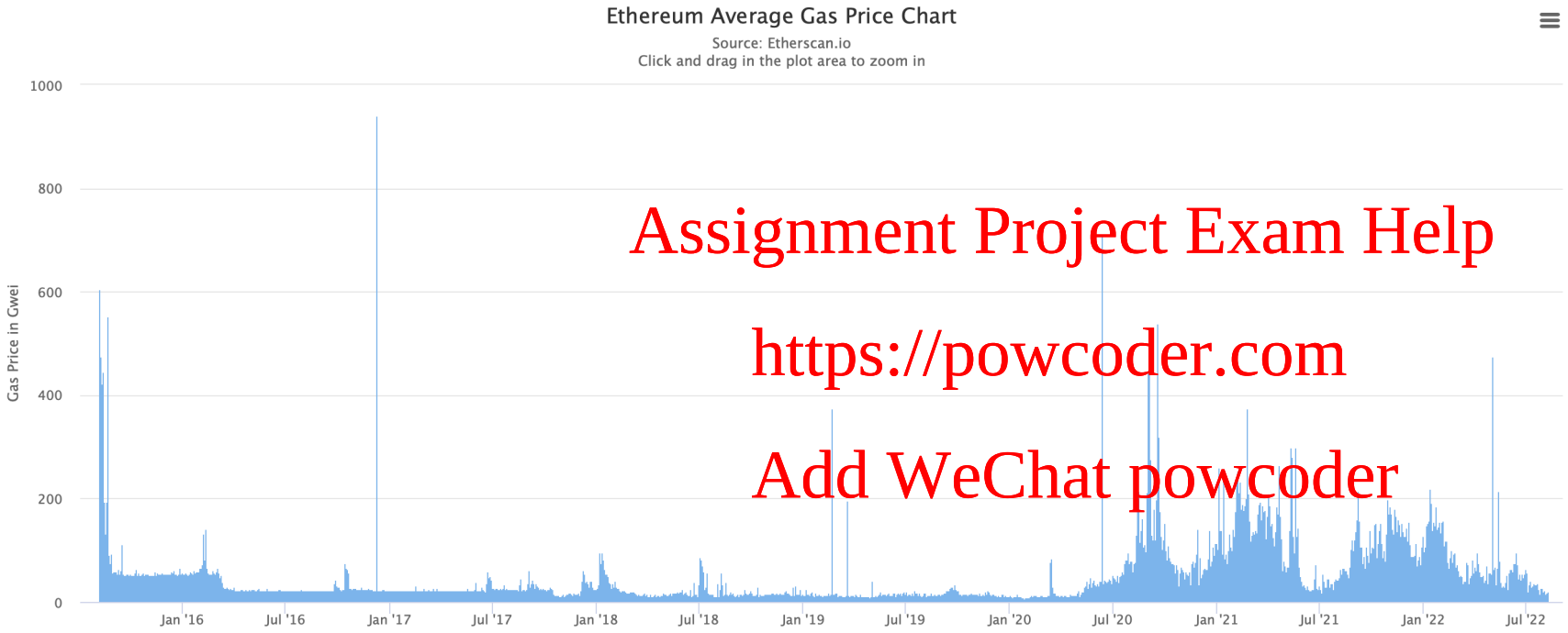


https://etherscan.io/chart/gasprice

\

MONASH University

Gas



https://etherscan.io/chart/gasprice

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

\

Gas Price

Gas Price: the price of a unit of gas the creator is willing to pay.

Like transaction fees, the higher the Gas price is, the faster the transaction will be chosen by a miner.

Category	Value
Cheapest Gas Price (gwei)	18.50654929
Highest Gas Price (gwei)	815
Median Gas Price (gwei)	35
Cheapest Transfer Fee	$1.4162
Highest Transfer Fee	$50.18
Total Transactions	8939
% Empty Blocks	0
% Full Blocks	28

https://ethgasstation.info

MONASH University

DeFi

ETH25 LEADERBOARD — **Last 30 Days**

RANK	NAME	ETH SPENT	AVE. GWEI	USD VALUE
1	Uniswap	1.50K	112	$2.00M
2	Tether USD	1.24K	133	$1.65M
3	1inch Exchange	292	125	$390K
4	USDC	195	133	$260K
5	0x0000000000007...e9f56	179	176	$239K
6	SushiSwap	148	155	$197K
7	Aave	114	135	$152K
8	Metamask Swap	100	119	$134K
9	0x	94.4	122	$126K
10	0x00c7a37b03690...75446	92.3	145	$123K
11	0xa57bd00134b28...dd6cf	92.2	308	$123K
12	0xf570deefff684...f7419	84.3	155	$112K
13	ChainLink	78.6	140	$105K
14	0x860bd2dba9cd4...78f66	78.4	158	$105K
15	0x78a55b9b3bbef...c55e8	71.9	207	$95.8K
16	0x00000063f648c...ff184	71.8	167	$95.8K
17	Bitstamp	67.2	172	$89.5K
18	0x03f34be1bf910...59659	63.2	131	$84.3K

MONASH University

Why using gas rather than Ether?

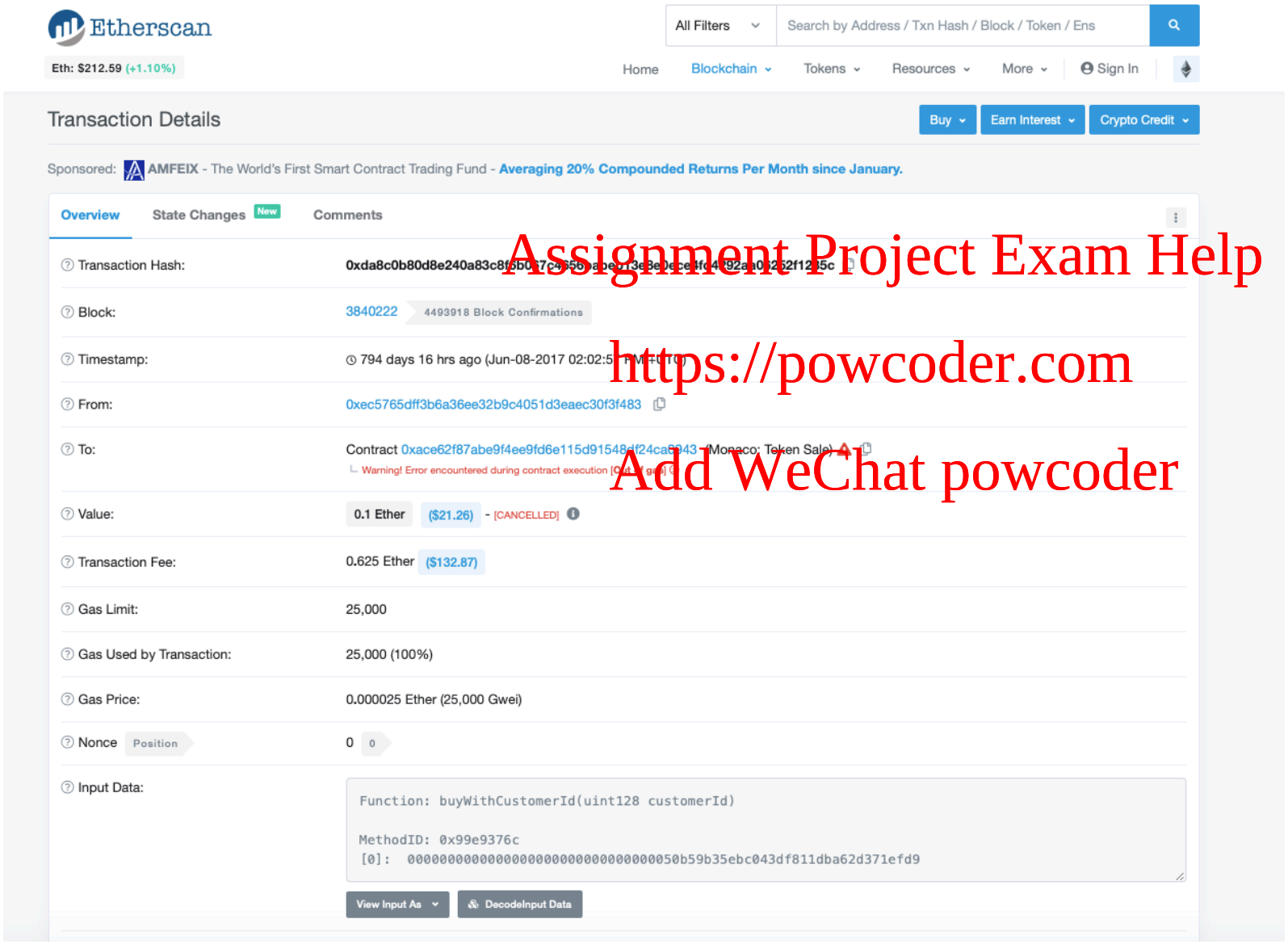
- Ether is a cryptocurrency, and the price of one ether is ever changing.
- If using ether, the price of running the same operation/contract can be very different.
- So, Ethereum uses Gas, where users can define the gas price!

Error: Out of Gas

If gas is not enough to complete all operations of a transaction, then:

1. the contract is terminated and transaction is cancelled.
2. the miner still gets the transaction fees.
3. the contract creator loses the transaction fees, but not the coin to be transferred.

Error: Out of Gas



MONASH University

How it works - basics

The high-level idea is the same as Bitcoin

How it works - basics

The high-level idea is the same as Bitcoin

Step 1. The very first block, called genesis block, is hardcoded to initialise the system.

How it works - basics

The high-level idea is the same as Bitcoin

Step 1. The very first block, called genesis block, is hardcoded to initialise the system.

Step 2. New blocks are created through the PoW-based mining process.

- The mining algorithm is Ethash rather than SHA-256 in Bitcoin
- A block is generated every 12 seconds
- *Ethereum plans to move to Proof-of-Stake.*

How it works - basics

The high-level idea is the same as Bitcoin

Step 1. The very first block, called genesis block, is hardcoded to initialise the system.

Step 2. New blocks are created through the PoW-based mining process.

- The mining algorithm is Ethash rather than SHA-256 in Bitcoin
- A block is generated every 12 seconds
- *Ethereum plans to move to Proof-of-Stake.*

Step 3. Miners get reward

- The mining reward is static
- Transaction fees go to the miner as well
- An extra of 1/32 of 2 Ether for including Uncles as part of the block. (Can include up to two Uncles)

How it works - basics

The high-level idea is the same as Bitcoin

Step 1. The very first block, called genesis block, is hardcoded to initialise the system.

Step 2. New blocks are created through the PoW-based mining process.

- The mining algorithm is Ethash rather than SHA-256 in Bitcoin
- A block is generated every 12 seconds
- *Ethereum plans to move to Proof-of-Stake.*

Step 3. Miners get reward

- The mining reward is static
- Transaction fees go to the miner as well
- An extra of 1/32 of 2 Ether for including Uncles as part of the block. (Can include up to two Uncles)

Step 4. Block creators gain the new coins as reward, and spend them through transactions.

How it works - basics

The high-level idea is the same as Bitcoin

Step 1. The very first block, called genesis block, is hardcoded to initialise the system.

Step 2. New blocks are created through the PoW-based mining process.

- The mining algorithm is Ethash rather than SHA-256 in Bitcoin
- A block is generated every 12 seconds
- *Ethereum plans to move to Proof-of-Stake.*

Step 3. Miners get reward

- The mining reward is static
- Transaction fees go to the miner as well
- An extra of 1/32 of 2 Ether for including Uncles as part of the block. (Can include up to two Uncles)

Step 4. Block creators gain the new coins as reward, and spend them through transactions.

Step 5. Transactions are included in blocks through Step 2.

How it works - basics

Smart contract:

1. Create a contract (Solidity)
2. Create a transaction containing the contract (EVM bytecode)
3. Submit the transaction to the blockchain

Account and wallet

Account and wallet

Two types of accounts:

- Externally-Owned Account
- Contract Account

Account and wallet

Two types of accounts:

- Externally-Owned Account
- Contract Account

Wallets:

- A set of one or more external accounts
- Used to store/transfer Ether

Externally-Owned Account (EOA)

Externally-Owned Account:

- the bank account for a user
- it keeps money (Ether) for the user
- it can send transactions
- controlled by the private key
- no code!

Contract Account

Contract Account:

- the account for a deployed contract
- it keeps money (Ether) to support the contract
- hold contract code in the memory
- can be triggered by other contracts
- no owner once released

Transaction

A transaction is a request to modify the state of the blockchain

- It can run code/contracts to change the global state of the blockchain

Transaction

A transaction is a request to modify the state of the blockchain

- It can run code/contracts to change the global state of the blockchain

Types of transactions:

- **Value Transaction:**
  Fund transfer between EOA
- **Creation Transaction:**
  Deploy a contract on Ethereum network
- **Invocation Transaction:**
  Invoke code (functions) on a Deployed Contract

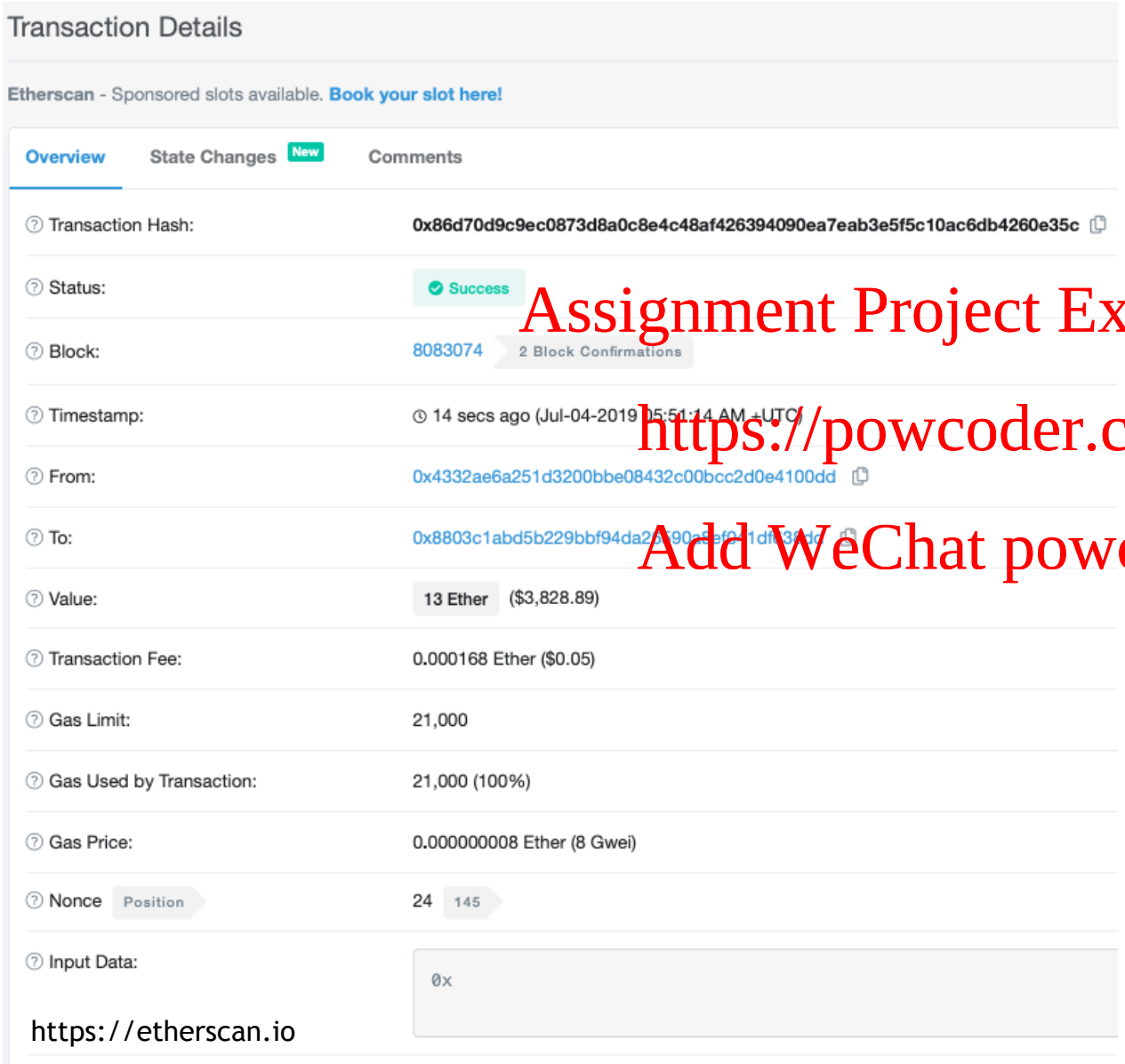
Transaction

Main fields:

- **From:** Fund sender, an EOA
- **To:** Fund receiver, an EOA
- **Value:** amount
  (wei is the unit)
- **Gas Limit:** the total amount of gas.
  (The default amount for a standard ETH transfer is 21,000 gas)
- **Gas Price:** the price of a unit of gas the creator is willing to pay
  (Gwei is the unit).

The total cost of a transaction (the “transaction fee”) is:
the Gas Limit * Gas Price

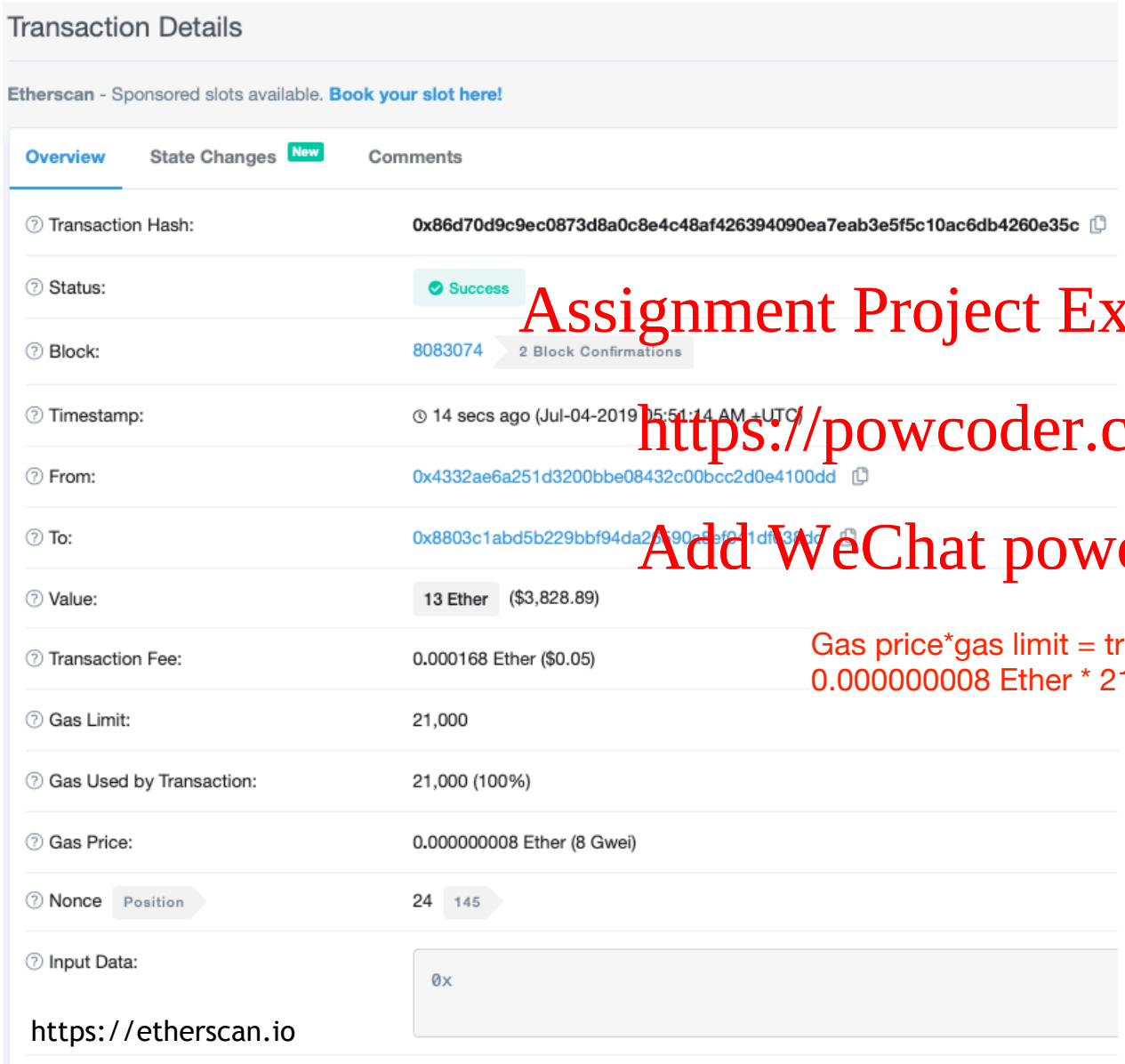
Transaction



https://etherscan.io

MONASH University

Transaction

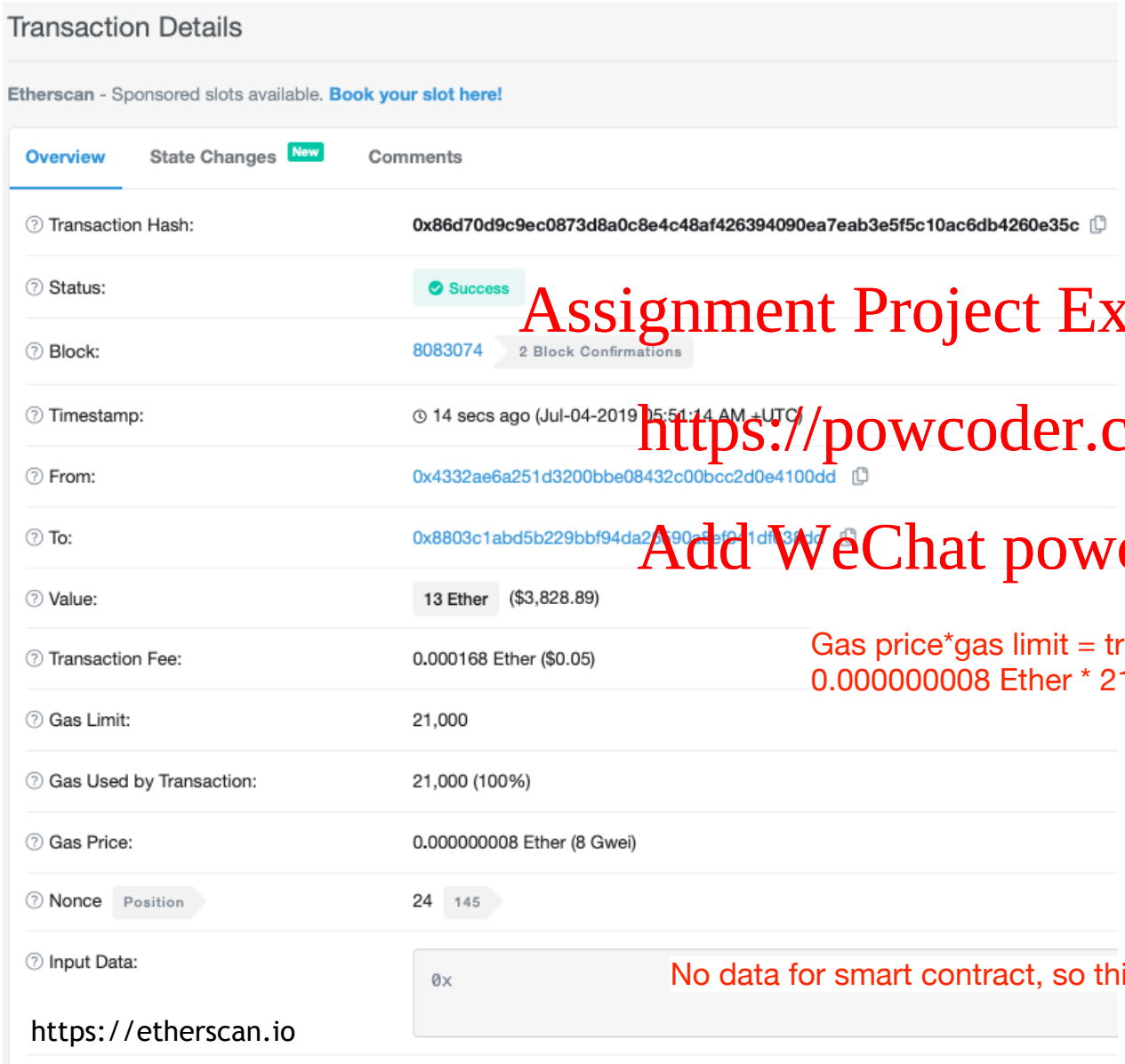


https://etherscan.io

Gas price*gas limit = transaction fee
0.000000008 Ether * 21000 = 0.000168 Ether

MONASH University

Transaction



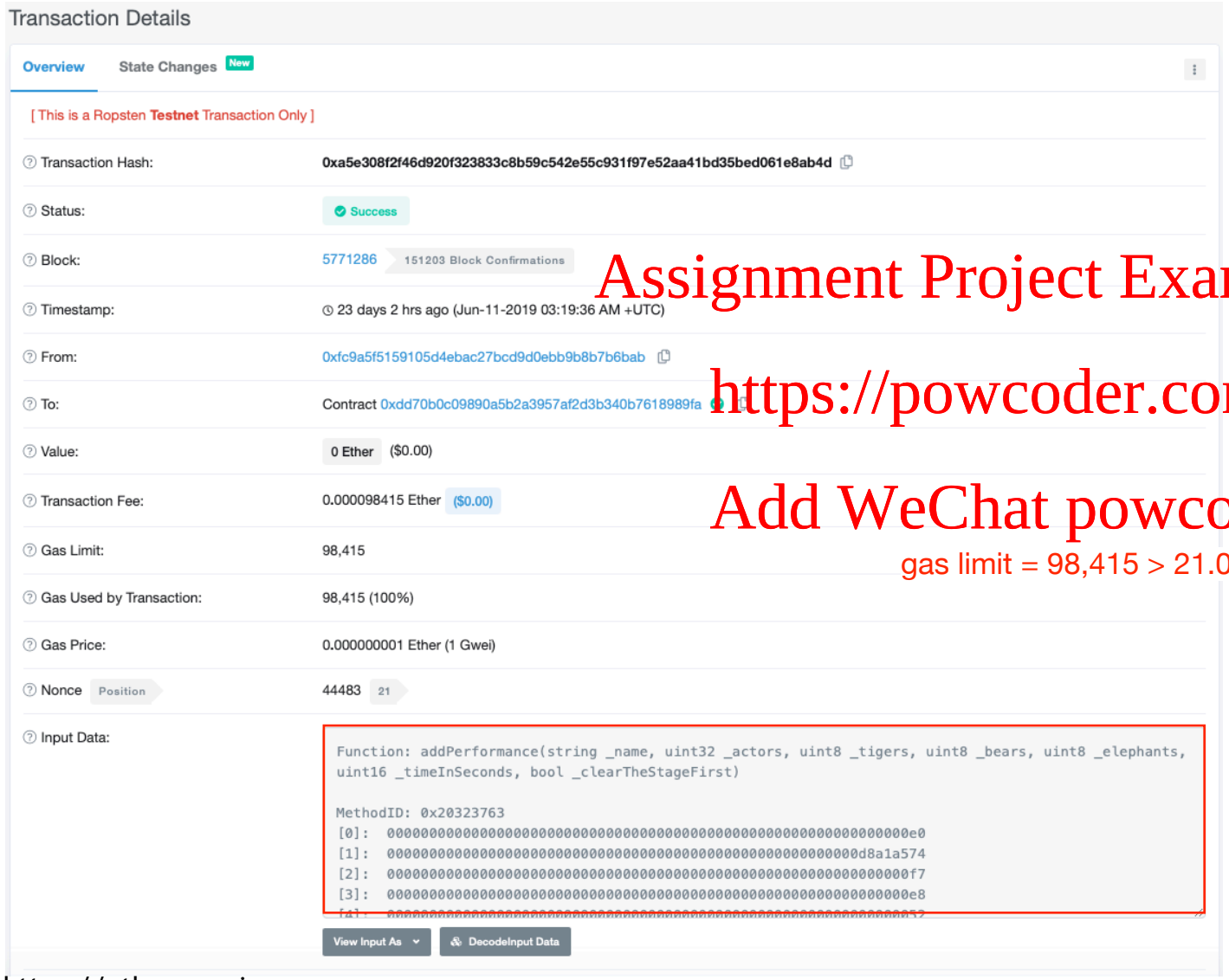
https://etherscan.io

Gas price*gas limit = transaction fee
0.000000008 Ether * 21000 = 0.000168 Ether

No data for smart contract, so this is only for transferring money (13 Ether)

MONASH University

Transaction



gas limit = 98,415 > 21.000

Bytecode of a contract

https://etherscan.io

Transaction

- Given a smart contract source code (Solidity), it is easy to verify the bytecode.

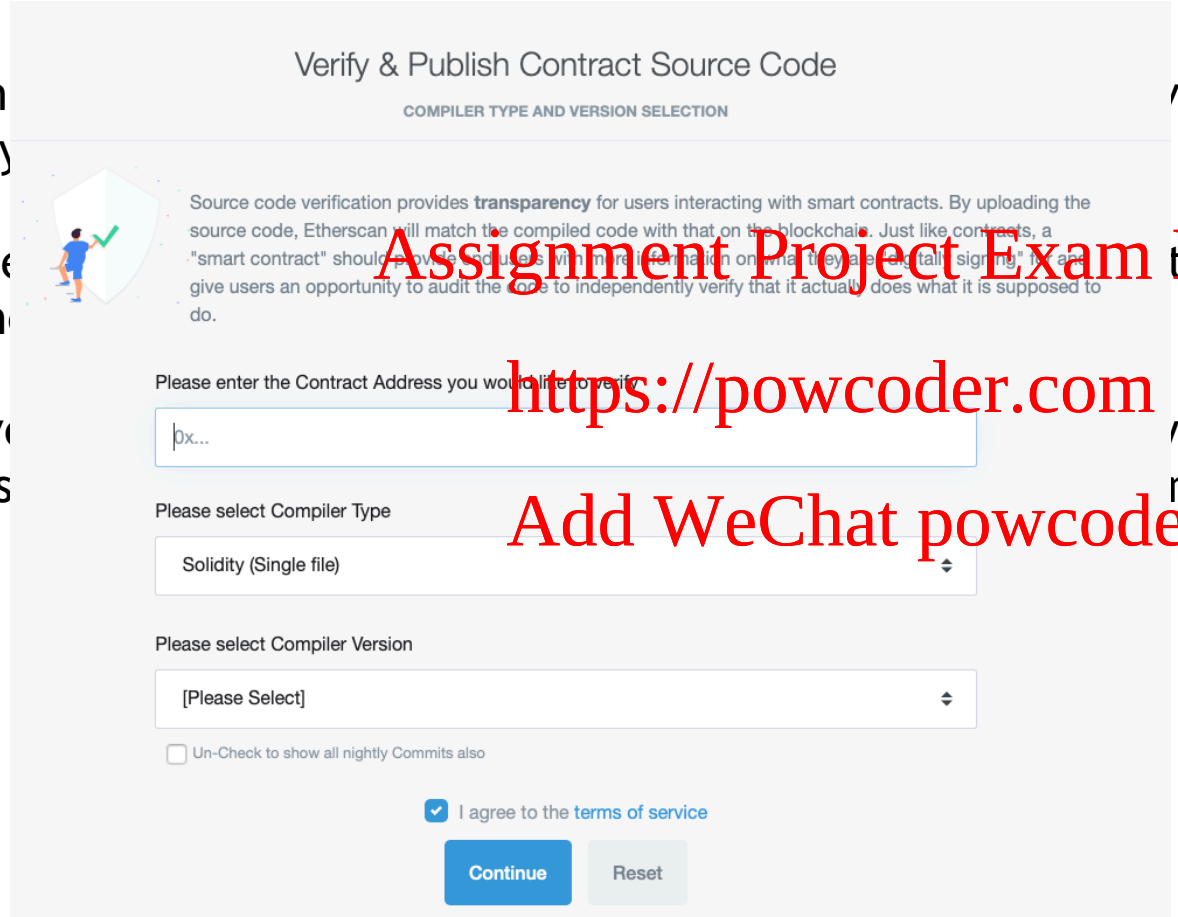
  You need to trust Etherscan is using the service. If you don't, you still need to trust EVM.

- It is very hard to decompile from bytecode to Solidity, but you can publish the verified smart contract source code at Etherscan.

https://etherscan.io

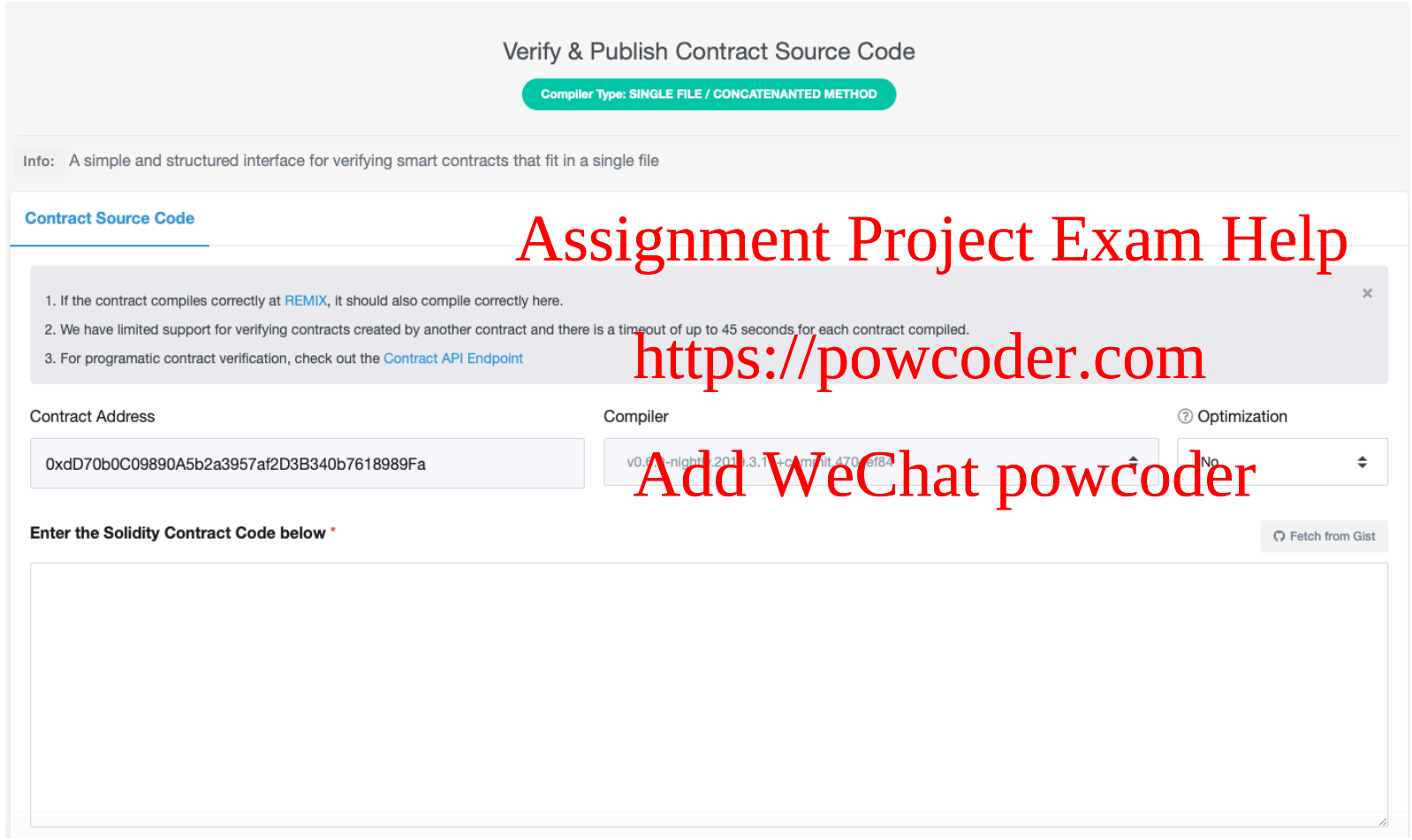
Transaction

- Given … verify the by…
  You ne… t, you still n…
- It is v… you can publis… n.



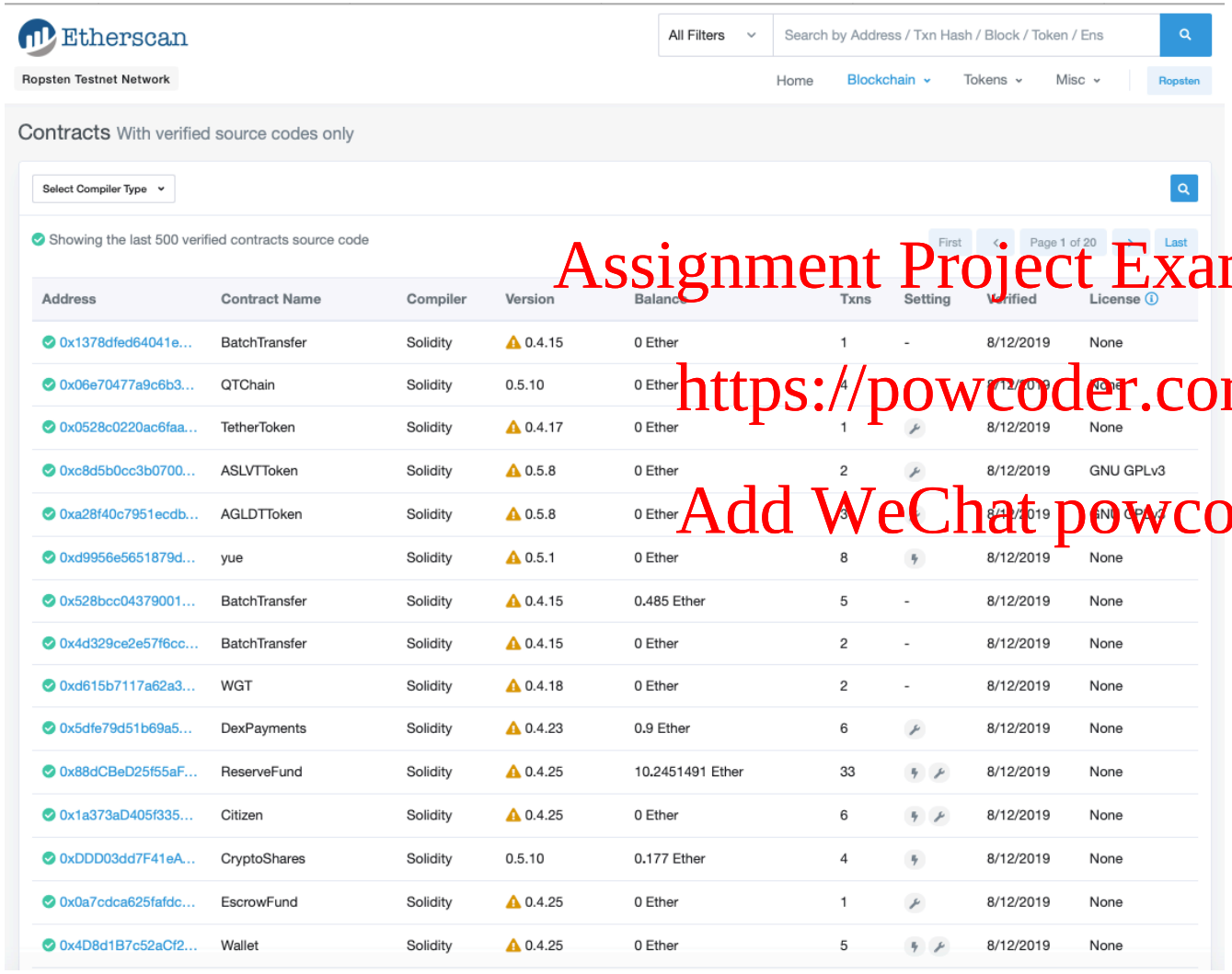
https://etherscan.io

Transaction



https://etherscan.io

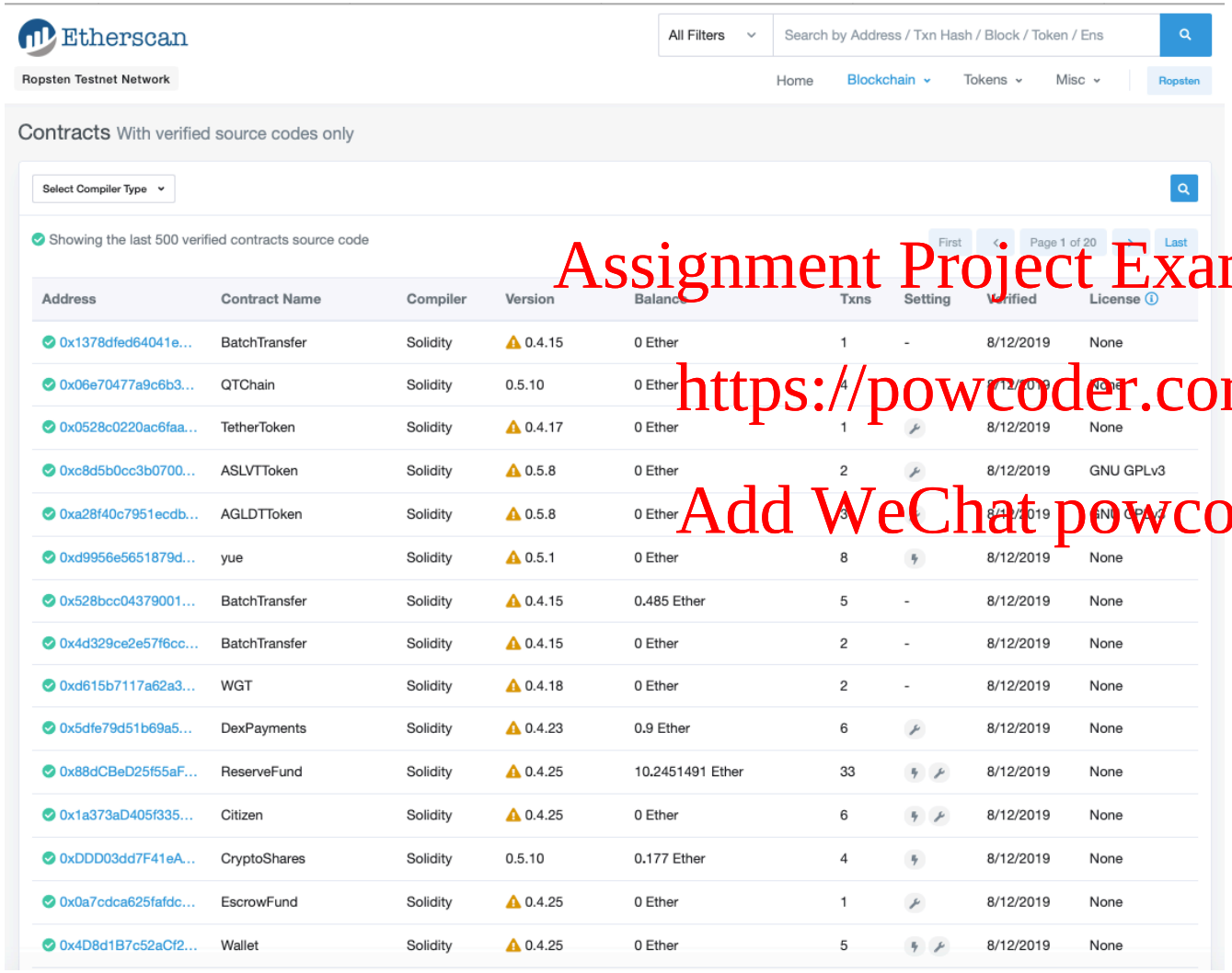
Contracts With verified source codes



Address	Contract Name	Compiler	Version	Balance	Txns	Setting	Verified	License
0x1378dfed64041e...	BatchTransfer	Solidity	0.4.15	0 Ether	1	-	8/12/2019	None
0x06e70477a9c6b3...	QTChain	Solidity	0.5.10	0 Ether	4		8/12/2019	None
0x0528c0220ac6faa...	TetherToken	Solidity	0.4.17	0 Ether	1		8/12/2019	None
0xc8d5b0cc3b0700...	ASLVTToken	Solidity	0.5.8	0 Ether	2		8/12/2019	GNU GPLv3
0xa28f40c7951ecdb...	AGLDTToken	Solidity	0.5.8	0 Ether	3		8/12/2019	GNU GPLv3
0xd9956e5651879d...	yue	Solidity	0.5.1	0 Ether	8		8/12/2019	None
0x528bcc04379001...	BatchTransfer	Solidity	0.4.15	0.485 Ether	5	-	8/12/2019	None
0x4d329ce2e57f6cc...	BatchTransfer	Solidity	0.4.15	0 Ether	2	-	8/12/2019	None
0xd615b7117a62a3...	WGT	Solidity	0.4.18	0 Ether	2	-	8/12/2019	None
0x5dfe79d51b69a5...	DexPayments	Solidity	0.4.23	0.9 Ether	6		8/12/2019	None
0x88dCBeD25f55aF...	ReserveFund	Solidity	0.4.25	10.2451491 Ether	33		8/12/2019	None
0x1a373aD405f335...	Citizen	Solidity	0.4.25	0 Ether	6		8/12/2019	None
0xDDD03dd7F41eA...	CryptoShares	Solidity	0.5.10	0.177 Ether	4		8/12/2019	None
0x0a7cdca625fafdc...	EscrowFund	Solidity	0.4.25	0 Ether	1		8/12/2019	None
0x4D8d1B7c52aCf2...	Wallet	Solidity	0.4.25	0 Ether	5		8/12/2019	None

https://ropsten.etherscan.io/contractsVerified

Contracts With verified source codes



Again, you need to trust Etherscan if using the service

https://ropsten.etherscan.io/contractsVerified

Running a transaction

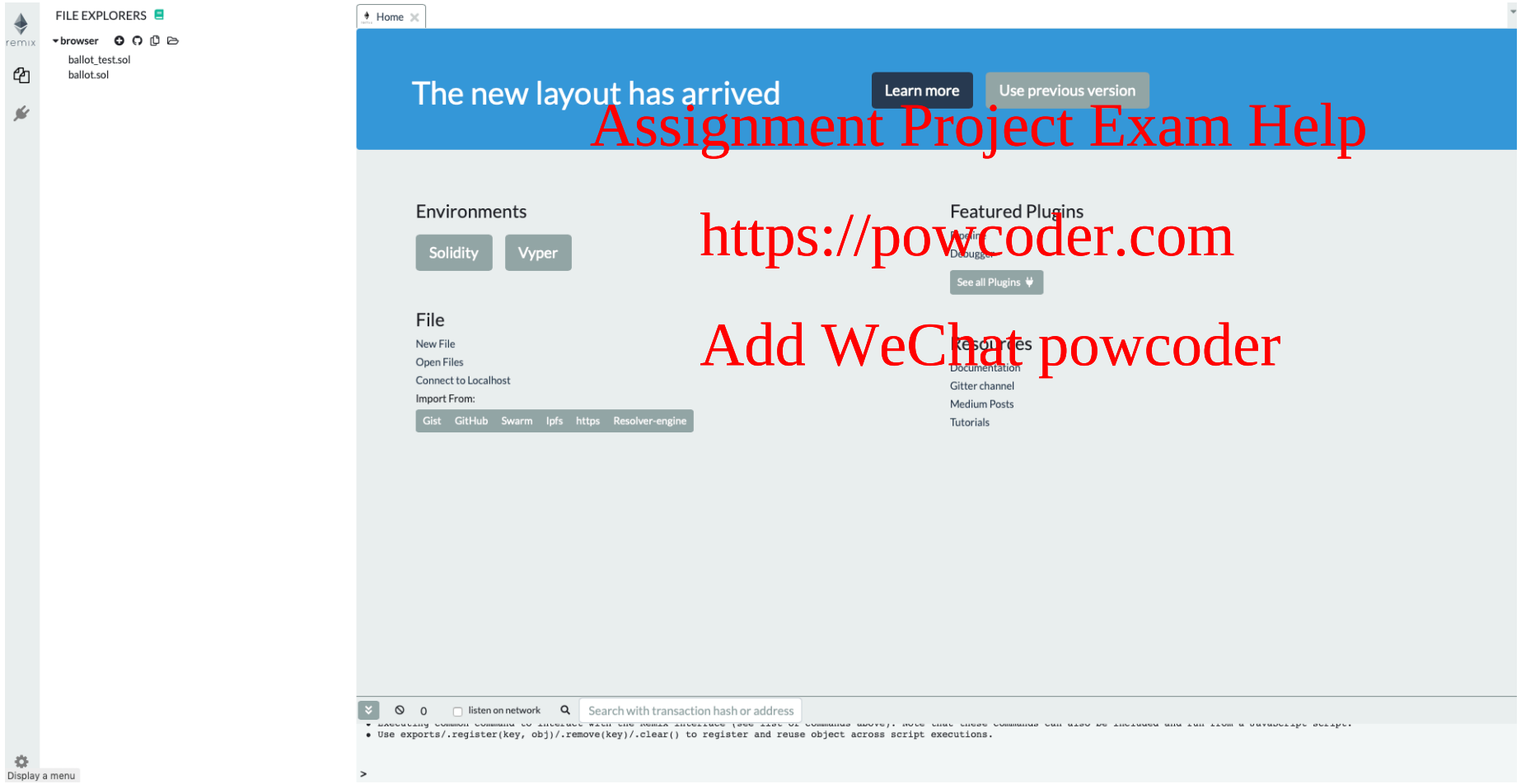
All full nodes do:

- **Verification:**
- The sender.balance should have at least **Value + Gas Limit * Gas price**

- **Update account balance:**
  - Recipient.balance += Value
  - Sender.balance -= Value + Gas Limit * Gas price
  - Sender.balance += The unused gas * gas price

Remix

An IDE to write, deploy, and run smart contracts in Solidity or Vyper.
You can access Remix from a web browser!



https://remix.ethereum.org

MONASH University

Decentralized Autonomous Organization (DAO)

Decentralized Autonomous Organization (DAO)

May 2016	DAO deployed a smart contract to raise 11.5 million Ether ($150 million USD at the time) in a token sale for funding

Decentralized Autonomous Organization (DAO)

May 2016	DAO deployed a smart contract to raise 11.5 million Ether ($150 million USD at the time) in a token sale for funding
June 2016	Vulnerability is found in DAO code, and 3.6 million Ether ($50 million USD at the time) was stolen.

Decentralized Autonomous Organization (DAO)

May 2016	DAO deployed a smart contract to raise 11.5 million Ether ($150 million USD at the time) in a token sale for funding
June 2016	Vulnerability is found in DAO code, and 3.6 million Ether ($50 million USD at the time) was stolen.
20 July 2016	Ethereum community at large decided to hard fork the chain to restore the funds to their original wallets and patch the vulnerability • **Ethereum Classic (ETC)**: The continued original chain containing the vulnerability in DAO code • **Ethereum (ETH)**: The hard forked chain with vulnerability of DAO code patched and stolen money returned to their owner. The majority of the community and the core developers continued working on this chain.

The DAO Attack (simplified)

A contract can have one unnamed function, called **fallback function**. This function does not take any arguments and it is triggered in three cases:

1. No functions of the call to a contract match any of the functions in the called contract.
2. No data was supplied.
3. **The contract receives Ether without extra data.**

Reentrancy Attack on DAO

A simplified example of DAO:

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Reentrancy Attack on DAO

A simplified example of DAO:

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

It maps an **address** to a **value**, and stores the value in a public variable **balances.**

Similar to dictionary[key]=value , it is a key-value pair or dictionary, where **balances** has a set of **unit** data type values mapped by **address** data type.

Reentrancy Attack on DAO

A simplified example of DAO:

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

It returned the balance of a sender's address, as defined before.

Reentrancy Attack on DAO

A simplified example of DAO:

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Transfers x from the balance of the sender in this contract to the sender's account address.

Reentrancy Attack on DAO

A simplified example of DAO:

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

The balance is recalculated only after sending the money out

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Honest User Contract

```
function MoneyWithdraw() {
    A.withdraw(5);
}
function() {
    EventMoneyReceived(msg.value);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Honest User Contract

```
function MoneyWithdraw() {
    A.withdraw(5);
}
function() {
    EventMoneyReceived(msg.value);
}
```


The unnamed function, fallback function

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Honest User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   EventMoneyReceived(msg.value);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Honest User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   EventMoneyReceived(msg.value);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Honest User Contract

```
function MoneyWithdraw() {
  A.withdraw(5);
}
function() {
  EventMoneyReceived(msg.value);
}
```


The user receives money if have enough balance in the DAO contact.

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

The user's balance in DAO is recalculated.

Honest User Contract

```
function MoneyWithdraw() {
  A.withdraw(5);
}
function() {
  EventMoneyReceived(msg.value);
}
```

The user receives money if have enough balance in the DAO contact.

Reentrancy Attack on DAO

Group discussion: what can go wrong?

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

The user's balance in DAO is recalculated.

Honest User Contract

```
function MoneyWithdraw() {
  A.withdraw(5);
}
function() {
  EventMoneyReceived(msg.value);
}
```

The user receives money if have enough balance in the DAO contact.

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```


The user receives money if have enough balance in the DAO contact, and asks to withdraw money again.

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```


The user receives money if have enough balance in the DAO contact, and asks to withdraw money again.

withdraw function is triggered again,
the user's balance in DAO is not recalculated!

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```


The user receives money if have enough balance in the DAO contact, and asks to withdraw money again.

withdraw function is triggered again,
the user's balance in DAO is not recalculated!

Looping until:

- exception
- Out of gas
- Stack limit is reached

Reentrancy Attack on DAO

DAO Contract

```
mapping (address => unit) public balances;

function withdraw(uint x) {
 if (balances[msg.sender] >= x) {
   msg.sender.call.value(x)();
   balances[msg.sender] -= x;
 }
}
```

Attacker User Contract

```
function MoneyWithdraw() {
   A.withdraw(5);
}
function() {
   A.withdraw(5);
}
```


The user receives money if have enough balance in the DAO contact, and asks to withdraw money again.

withdraw function is triggered again,
the user's balance in DAO is not recalculated!

Looping until:

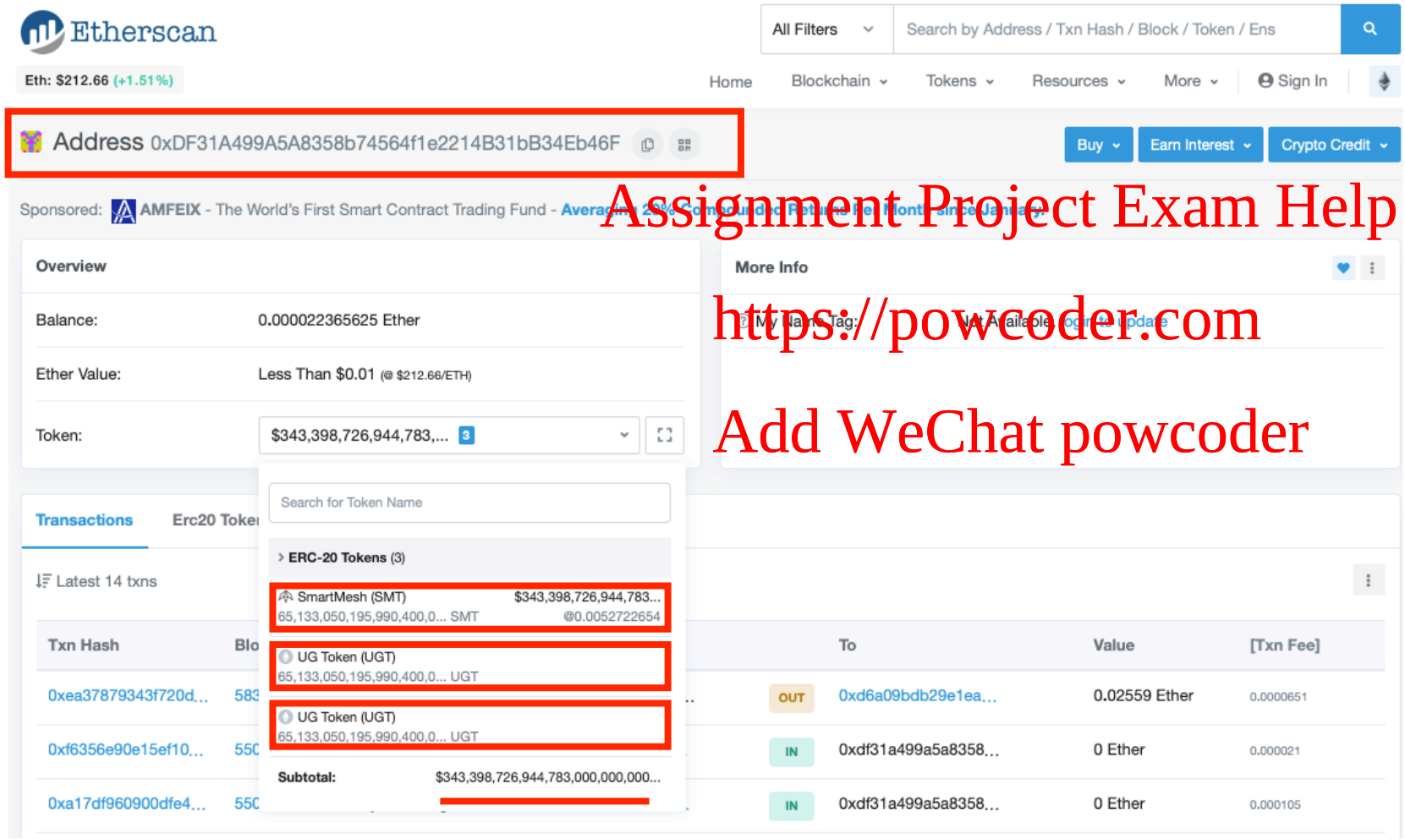
- exception
- Out of gas
- Stack limit is reached

The user's balance in DAO is now recalculated.

The overflow and underflow attack

The overflow and underflow attack

It happens …



MONASH University

Ethics

An ethical issue:

- Could an action (or product) be damaging to someone or some group?
- Identify the stakeholders; consult those who would be impacted
- Vulnerability disclosure
  - How to disclose vulnerabilities that impacts more than one blockchains is still an open challenge.
- Some key vulnerabilities still exist in the current systems
  - Not well regulated
  - DO NOT ATTACK!
    (you can try attacks on testnets and in our labs)

Ethics

An ethical issue:

- Could an action (or product) be damaging to someone or some group?
- Identify the stakeholders; consult those who would be impacted
- Vulnerability disclosure
  - How to disclose vulnerabilities that impacts more than one blockchains is still an open challenge.
- Some key vulnerabilities still exist in the current systems
  - Not well regulated
  - DO NOT ATTACK!
    (you can try attacks on testnets and in our labs)

Reading

A nice place to expand your knowledge about smart contract:
https://docs.soliditylang.org/en/v0.8.16/

The book "Introducing Ethereum and Solidity" also has information about Ethereum and Solidity (log in with Monash access):
https://link.springer.com/book/10.1007/978-1-4842-2535-6

Next lecture: Proof-of-Work