Blockchain 2.0 - From Bitcoin Transactions to Smart Contract applications

by Jerome Kehrli

Written on Tuesday Nov 22, 2016

Since Satoshi's White paper came online, other cryptocurrencies have proliferated on the market. But irrespective of the actual currency and the frequently debated deflation issues, the actual revolution here is the Blockchain protocol and the distributed computing architecture is supports.

Just as thirty years ago the open communications protocol created profitable business services by catapulting innovation, the blockchain protocol has the potential of being the same kind of breakthrough, by offering a just as disruptive foundation on which businesses start to emerge. Using the integrity lattice of the transactions, a whole suite of value trading innovations are beginning to enter the market.

The key innovation here are **Smart Contracts**. This relatively new concept involves the development of programs that can be entrusted with money.

Smart Contracts are autonomous **computer programs** that, once started, execute automatically and mandatorily the conditions defined beforehand, such as the facilitation, verification or enforcement of the negotiation or performance of a contract.
They are most of the time defined in a Programming Language, which in the case of the Ethereum Blockchain 2.0 technology form a Turing Complete Programming Language.
Smart Contracts are implemented as any other software program,
using *conditions*, *loops*, *function calls*, etc.

If blockchains give us *distributed trustworthy storage*, then smart contracts give us *distributed trustworthy computations*. To illustrate a possible use of smart contracts, let's take the example of travel insurance: finding that 60% of the passengers insured against the delay of their flight never claimed their money, a team created during a hackathon in London in 2015 an Automated Insurance system based on smart contracts.
With this service, passengers are automatically compensated when their flight is delayed, without having to fill out any form, and thus without the company having to

process the requests. The blockchain's contribution here consists in generating the confidence and security necessary to automate the declarative phases without resorting to a third party.

The main advantage of putting Smart Contracts in a blockchain is the guarantee provided by the blockchain that the contract terms cannot be modified. The blockchain makes it impossible to tamper or hack the contract terms.
By developing ready to use programs that function on predetermined conditions between the supplier and the client, smart programs ensure a secure escrow service in real time at near zero marginal cost.

Smart Contracts enable to reduce the costs of verification, execution, arbitration and fraud prevention. They enable to overcome the moral hazard problem. The american cryptograph Nick Szabo is deemed to be the inventor of the concept, whom he spoked about in 1995 already. He used to mention the example of a rented car, whose smart contract could return the control to the owner in case the renter forgives the paiements.
Interestingly, as a sidenote, Nick Szabo is also believed by some to be one of the person behind the Satoshi Nakamoto identity.

In a general way, Smart Contracts from the heart of the Ethereum blockchain. Even if Rootstock aims at enabling the implementation of Smart Contracts on the bitcoin blockchain, the development of Smart Contracts technology is really rather related to Ethereum. The next versions of Ethereum are increasingly targeted to offering end users an *App-Store*-like User Experience to Smart Contracts.

This article intents to be a pretty complete introduction to Blockchain 2.0 technology and Smart Contract applications, detailing both of them as well as list the state of the state of the art of possible use cases being currently studied or discussed.
A big part of this article focuses on the Ethereum blockchain.

(One might want first article in this serie : Blockchain explained that provides a pretty complete introduction to the initial bitcoin blockchain technology)

Also one might want to see part of this article as a slideshare presentation available here :http://www.slideshare.net/JrmeKehrli/blockchain-20-69472625.

Table of Contents

1. Blockchain 2.0 and Smart Contracts

Smart Contracts are most often the central component of the next-generation blockchain platforms.
Blockchain technology is much broader than just bitcoin. The sustained levels of robust security achieved by public cryptocurrencies have today really proven that this new wave of blockchain technologies can provide efficiencies and intangible technological benefits very similar to what the internet has done in the early 90s.
Blockchains are a very powerful technology, capable of going much further than only "*simple*" financial transaction; a technology capable of performing complex operations, capable of understanding much more than just how many bitcoins one currently has in his digital wallet.
This is where the idea of **Smart Contracts** comes in. They form the *cornerstone* for coming enterprise blockchain applications.

In this article, we will explore what a smart contract is, how it works, and how it is being used.

1.1 From Transactions to Smart Contracts

The Blockchain 2.0 is an evolution of the blockchain protocol enabling not only to exchange transaction but rather code and programs in the form of Smart Contracts
Now developers are allowed to build programs and API's on the Blockchain Protocol.
This relatively new concept involves the development of programs that can be entrusted with money.
Smart contracts are programs that encode certain conditions and outcomes.
For instance, When a transaction between two parties occurs, the program can verify

if the product/service has been sent by the supplier. And only after this verification is the sum transmitted to the suppliers account.

By developing ready to use programs that function on predetermined conditions between the supplier and the client, smart programs ensure a secure escrow service in real time at near zero marginal cost

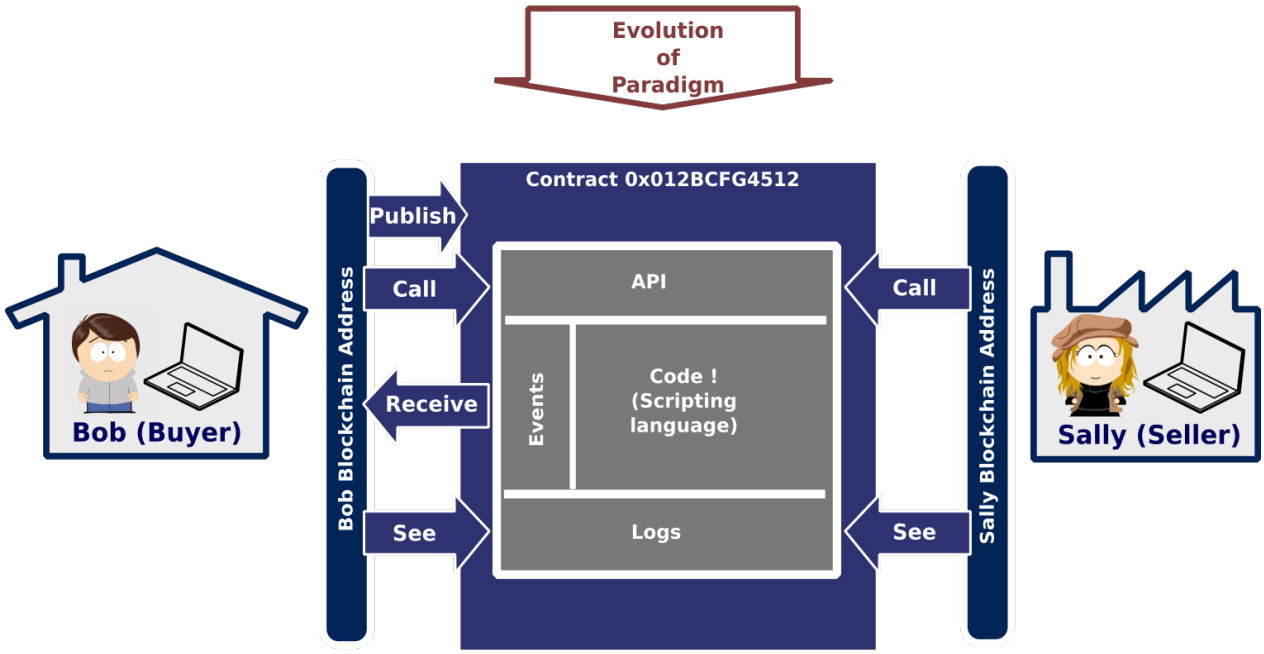
Apart from Financial transactions, smart contracts are now entering a whole lot of different industry.
One can refer to the section 5. Smart Contracts use cases to have a look at the use cases and the industries that can and will be disrupted by the blockchain technology.

Ethereum

Ethereum intends to bring together both a *crypto ledger* and a *Turing-complete* programming language. Ethereum's objectives is to turn a simple browser to a *Swiss-army knife* of blockchain applications by providing tools that allow non-technical users to truly leverage the technology.

Ethereum aims to implement a **globally decentralized, un-ownable, digital computer** for executing peer-to-peer contracts in the form of actual software programs.
Put more simply, Ethereum is a world computer you can't shut down.

Blockchain 1.0 - Bitcoin



Blockchain 2.0 - e.g. Ethereum, ...

Smart Contract Definition

Smart contract is a term used to describe computer program code that is capable of facilitating, executing, and enforcing the negotiation or performance of an agreement (i.e. contract) using blockchain technology.

The entire process is automated can act as a complement, or substitute, for legal contracts, where the terms of the smart contract are recorded in a computer language as a set of instructions

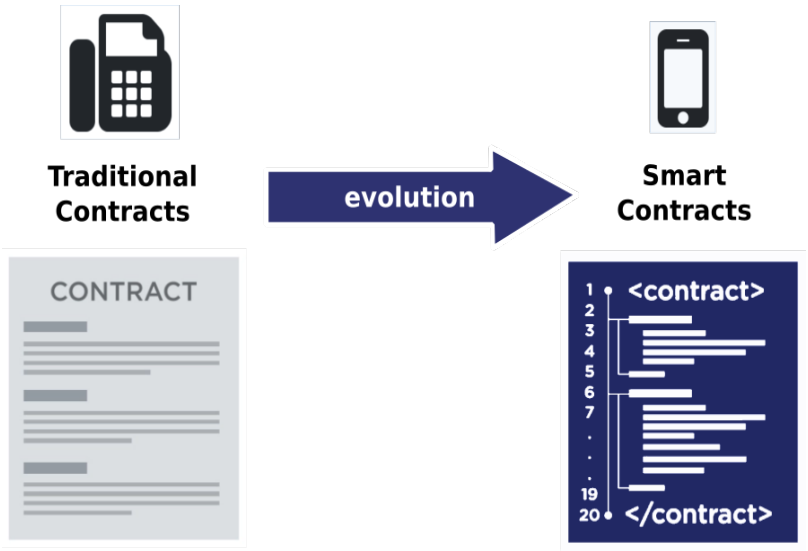
1.2 Smart Contracts Overview

A smart contract is a digitally signed, computable agreement between two or more parties. A virtual third party - a software agent - can execute and enforce (at least some of) the terms of such agreements.
In the context of the blockchain, where it truly takes it sense, a smart-contract is an

event-driven program, with state, that runs on a replicated, shared ledger and which can take custody over assets on that ledger.

Smart contracts provide a viable method of issuing tracking ownership of unique digital representations of value, which we call money.

Smart contracts are simply computer programs that act as agreements where the terms of the agreement can be preprogrammed with the ability to self-execute and self-enforce itself.
The main goal of a smart contract is to enable two anonymous parties to trade and do business with each other, usually over the internet, without the need for a trusted middleman.
The origin and history of smart contracts is much older than bitcoin and dates back to the 90's. The term "**Smart Contract**' was first used in 1993 by one of bitcoin's alleged creators, Nick Szabo, and referred to **self-automated computer programs** that can carry out the terms of any contract.



- **Traditional physical contracts**, such as those created by legal professionals today, contain legal language on a vast amounts of printed documents and heavily rely on third parties for enforcement. This type of enforcement is not only very time consuming, but also very ambiguous. If things go astray, contract parties must often rely on the public judicial system to remedy the situation, which can be very costly and time consuming, down to sometimes useless.
- **Smart contracts** on the blockchain, created by computer programmers, are entirely digital and written using programming code languages. This code defines the rules and consequences in the same way that a traditional legal document would, stating the obligations, benefits and penalties which may be

due to either party in various different circumstances. The big difference is that this code is automatically executed by a distributed ledger system, in a non-repudiable and unbreakable way.

Smart Contract code have some unique characteristics.

- **Deterministic** : Since a Smart Contract code is executed on multiple distributed nodes simultaneously, it needs to be deterministic i.e. given an input, all nodes should produce the same output. That implies the Smart Contract code should not have any randomness; it should be independent of time (within a small time window because the code might get executed a slightly different time in each of the nodes); and it should be possible to execute the code multiple times (idempotence).
- **Immutable** : Smart Contract code is immutable. This means that once deployed, it cannot be changed. This of course is beneficial from the trust perspective but it also raises some challenges (e.g. how to fix a code bug) and implies that Smart Contract code requires additional due diligence/governance.
- **Verifiable**: Once deployed, Smart Contract code gets a unique address. Before using the Smart Contract, interested parties can and should view or verify the code.

Smart Contracts benefits

For a wide range of potential applications, blockchain-based smart contracts could offer a number of benefits:

- **Speed and real-time updates** : because smart contracts use software code to automate tasks that are otherwise typically accomplished through manual means, they can increase the speed of a wide variety of business processes.
- **Accuracy :** automated transactions are not only faster but less prone to manual error.
- **Lower execution risk.** The decentralized process of execution virtually eliminates the risk of manipulation, nonperformance, or errors, since execution is managed automatically by the network rather than an individual party.
- **Fewer intermediaries :** smart contracts can reduce or eliminate reliance on third-party intermediaries that provide "trust" services such as escrow between counterparties.
- **Lower cost :** new processes enabled by smart contracts require less human intervention and fewer intermediaries and will therefore reduce costs.

- **New business or operational models** : because smart contracts provide a low-cost way of ensuring that the transactions are reliably performed as agreed upon, they will enable new kinds of businesses, from peer-to-peer renewable energy trading to automated access to vehicles and storage units.

1.3 A little glimpse of Smart Contracts from finance perspective

Just as a Bank account with embedded instructions

There are some elements of bank accounts that behave like smart contracts.
For instance, a bank account has a balance. Every month, for instance, there can be automated payments deducting amounts to pay various bills or fees, for instance for pension plans, If there isn't enough money on the bank account, the payment fails, the owner can get fined, and another workflow is triggered.
There are instructions that need to be set up and associated with the account.

This is similar to what a smart contract can do, except that a smart contract running on a blockchain is run by many parties rather than being controlled by a single one.

How is this different to automated banking payments?

- **Control** : The bank is the ultimate guardian of any bank account. It has complete control, and can arbitrarily add money to an account (well, that never happens !) or subtract some (this does happen, and one needs to argue to get it back). In a blockchain ecosystem, there are no single source of control and participant agrees on decisions by distributed consensus, meaning that multiple parties are constantly checking and re-checking updates to the ledgers, and anything that doesn't conform to pre-agreed rules is rejected by all participants.
- **Code** : With a bank account, there is some logic creating transactions on a monthly basis. That code sits on one computer and is executed by one party (the bank). There are internal controls and reconciliations, but there is no external validation.
  With smart contracts running on a blockchain, the logic is run in parallel on all the participating computers, and the results are compared by all participants. Participants only change their own version of the ledger if they agree the results. No one can cheat a blockchain, in theory of course.
- **Transparency** : For all participants in a blockchain ecosystem to run the same code, each verifying the other, the logic of the smart contract must be visible to all. This means anyone can look into a smart contract, and if use it if one wants.
  There will be smart contracts for general usage, and also very specific smart

contracts. The transparency is both a pro and a con. It's useful to all stakeholders of the contract to agree on what happens; on the other hand it's not just the stakeholders that can see what happens - it's everyone on the network. Privacy in blockchains is a contentious issue. There are solutions to the privacy-vs-validation tension being discussed, some using zero-knowledge proofs; which will be the subject of another post.

- **Flexibility** : The single logic that applies to a bank account is pretty much limited to automating payments. It would be difficult, for instance, to automate a payment from a salary account to a savings account every day it's sunny, then have it all sent back when there is a storm (the 'saving up for a rainy day' smart contract).
  A so-called "Turing complete" smart contract can do anything that a normal computer can do, though the blockchain version will run much more slowly and be more expensive to run than on a regular computer (depending on the set-up of the blockchain), because ultimately all computers on the network need to the code in parallel and of course they have to be paid for it.

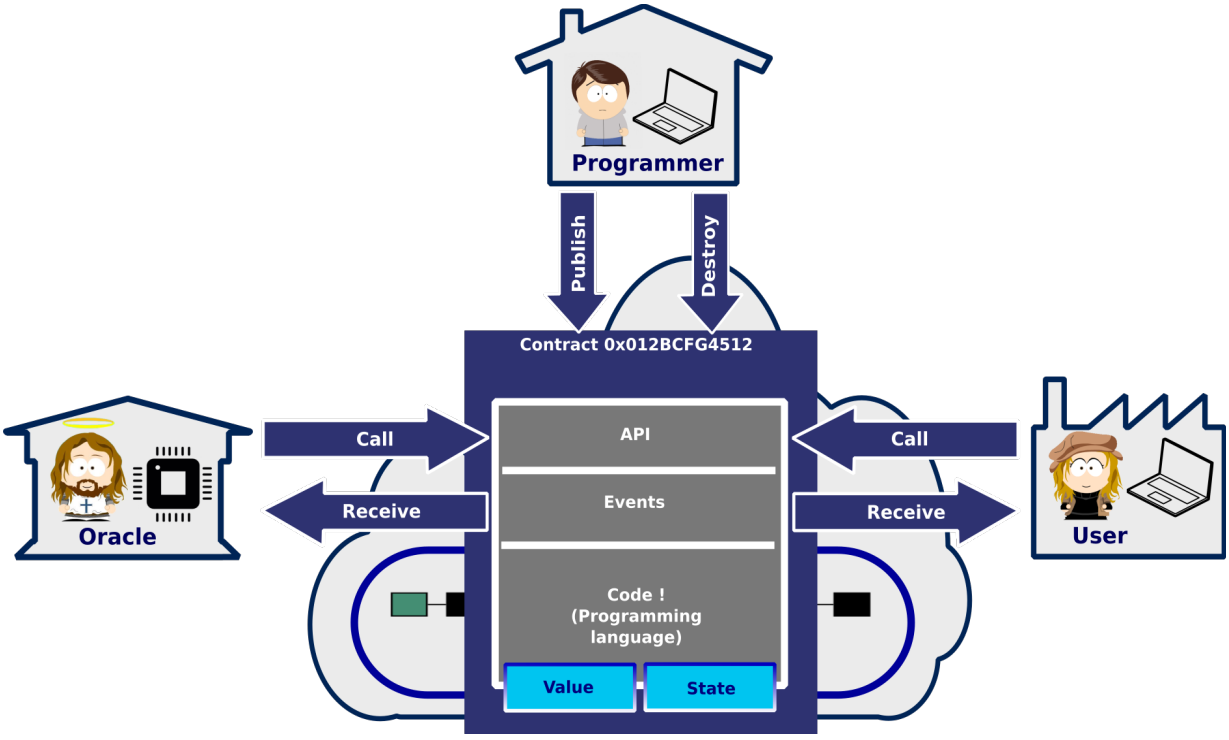
2. Smart Contracts Operation

In order to understand how smart contracts work, it is important to first make the distinction between the smart contract code and how/what that code is being applied to.

2.1 Smart Contracts Design

A smart contract can be broken down into two separate components:

- **Smart Contract Code** - The code that is stored, verified and executed on a blockchain.
- **Smart Legal Contracts** - The use of the smart contract code that can be used as a complement, or substitute, for legal contracts.



A Smart Contract is a computer program that runs on a shared, replicated ledger, which can process and store information, as well as receive, store and send value.

How Smart Contracts work

1. **Coding** : what goes into a Smart Contract
   Because smart contracts work like computer programs, it is very important that they do exactly what the parties want them to do. This is achieved by inputting the proper logic when writing a smart contract (more on that later). The code behaves in predefined ways and doesn't have the linguistic nuances of human languages, thus, it has now automated the "**if** this happens **then** do that" part of traditional contracts.

2. **Distributed Ledgers** : how the smart contract is sent out
   The code is encrypted and sent out to other computers via a distributed network of nodes running a distributed ledger. If this is done via public permissionless blockchain such as bitcoin, the contract is sent out similar to the way that a network update of a bitcoin transaction would occur. This can also be done in a permissioned or hybrid distributed ledger platform such as the R3 Distributed Ledger.

3. **Execution** : how it is processed
   Once the computers in this network of distributed ledgers receive the code, they each come to the same agreement or consensus on the results of the code execution. The network would then update the distributed ledgers to

record the execution of the contract, and then monitor for compliance with the terms of the smart contract. In this type of system, single party manipulation is averted because control over the execution of the smart contract is no longer possible since that execution is not in the hands of a single party.

2.2 Smart Contract and Oracles

Some smart contracts systems, including the one built into Bitcoin, are strictly deterministic. In order to interact with the real world, these systems rely on informations (and cryptographic signatures) submitted by outside systems called "**oracles**".

Oracles are **trusted entities** which sign claims about the state of the world. Since the verification of signatures can be done deterministically, it allows deterministic smart contracts to react to the (non-deterministic) outside world.

Oracles are required to connect smart contracts to critical data feeds, any web API or various accepted payment methods.

As mentioned previously, Smart Contracts are executed by examining all the conditions of execution which have been defined in advance in the contract code. For example, if a contract includes a condition to run after January 1, 2017, it will be impossible to execute it before that date.
The problem then arises, naturally, from the validation of these conditions of execution. Two scenarios are then possible:

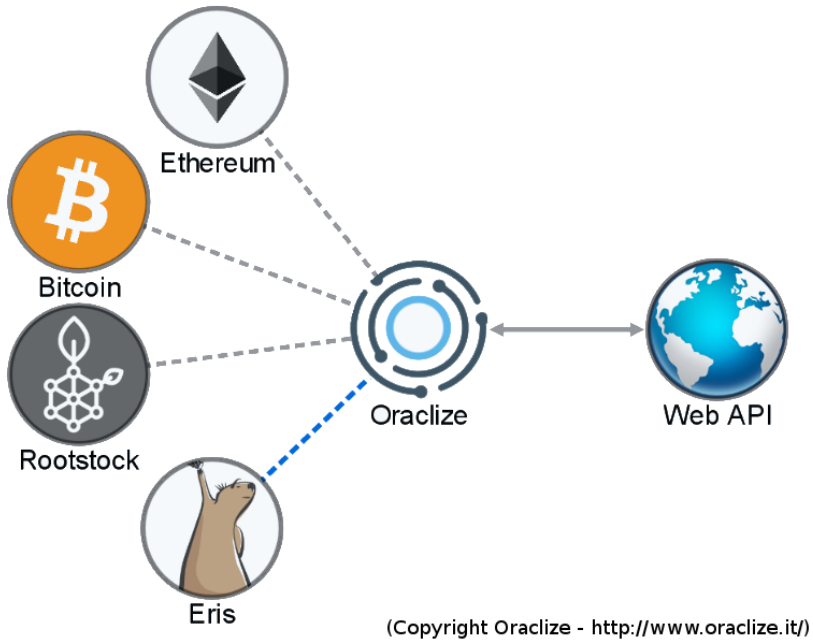
- The execution conditions of the contract are linked to other entries in the blockchain or are simple time markers. In this case, checking these execution conditions is very easy: the contract is programmed to verify that these entries exist or that the execution time is passed, and it executes when this is the case.
- The conditions of execution of the contract are outside the blockchain (realization of a service, occurrence of an event ...). In this case, the execution of the contract requires the use of a trusted third party, called in the Ethereum jargon an "oracle".

An oracle is instructed to enter the blockchain information reliably so that the contract can run properly and can be constituted in several ways:

- Prior designation of a trusted third party known to both parties;
- Reference to a database considered "trustworthy" (for example in the case of a sports betting, possibility of referring to the result recorded on the site of a sports newspaper);

- Using a decentralized Oracle service. It is an existing service on the blockchain involving many participants. Each participant votes for the result he / she considers to be accurate and it is the consensus among the participants that determines the final result sent to the contract. Decentralized oracle projects already exist, notably the Oraclize project

Long story short, the Oracle stands in between of the External world data or API and the Smart Contract.
As seen on the Oraclize web site :



The challenge with Oracles

An oracle, in the blockchain sense, is a third party which sends to your on-chain smart contract some specific data your smart contract code cannot fetch by itself. As the oracle is a centralized party, one shouldn't take its response for granted.
While hacking the blockchain is proven to be very difficult and unlikely to happen, Oracle form a *Single Point of Failure* and are vulnerable to attacks.

For example, if one asks an oracle to give last trading price of ETH/USD (forex), then the oracle needs to fetch this data from some exchanges on the Internet and than send this data back.
An attacker could compromise the link between the Oracle and the Forex Exchange company and make the Oracle send back to the blockchain a compromised value.
This is the reason why, by itself, the oracle cannot be trusted.

In the context of Smart Contracts, it is key to be able to trust the data fetched from the outside world and the data provider (in our example this is the exchange trading

the ETH/USD pair) and this is difficult.
It the example above, the risk can be mitigated by using different data-sources and using them to determine consensus.

Oraclizes provides its own solution to this trust issue and other initiatives work on other kind of solutions, for instance Codius.

Discussing this issue and its solutions further exceeds the scope of this article.

2.3 DAO

Smart Contracts and blockchain 2.0 technologies have made possible the emergence of special organizations called DAOs.

A **Decentralized Autonomous Organization (DAO)** is an organization that is run through rules encoded in computer programs called smart contracts. These rules provide a community with unbreakable, secured, universal, public and untamperable governance rules.
A DAO's financial transaction records and program rules are maintained on a blockchain.

A DAO is organized to run without any form of human managerial interactivity, provided the smart contracts are supported by a Turing complete platform. Ethereum is such a platform and thus enables such DAOs.

A DAO is a form of incorruptible organization owned by the people who created it and finance it and whose rules are public.
It brings three new elements in comparison with a traditional organization:

- A DAO cannot be stopped or broken
- No single person or organization can control the DAO. None can tamper with it, hack it or cheat with it.
- Everything in the DAO is public and transparent

It really is a global organization, aimed at being open to everyone. It suffers of no jurisdiction and works with software code and where nobody can fraud.

Corporations and businesses as DAOs

Corporations are, if you strip everything away down to the bare bones, a complex set of contracts and agreements. Most simplistically, employment contracts set the terms for workers pay, duties and responsibilities. Contracts with vendors and customers ensure supply chains are established and maintained. Lease agreements cover office space, vehicles, large machinery and rights to intellectual property.
And so on ... other parts or functions are covered by other contractual elements.

Smart contracts exist without the need for those institutional layers. An organization can be built where all of these agreements are replaced by such smart contracts, and in essence the corporation will exist entirely as an entity on a blockchain. As such it will be a decentralized organization, existing across all the nodes of the network.

A DAO would be in the business of generating economic profits if it were structured as a corporation (Decentralized Autonomous Corporation), and it could raise capital through crowdsales of tokens directly to the blockchain, akin to shares in a public company. Tokenholders would be entitled to their share of profits in the form of dividends, and could vote on the direction of the company. Those tokens could also trade on a secondary market (also on the blockchain) for people to buy and sell them at will.

3. Blockchain 2.0 projects

After the initial blockchain of the Bitcoin, many other projects started to flourish pretty soon. Interstingly, the blockchainb appear to be able to give life to some concepts designed or discussed many years before.
Nick Szabo described Smart Contracts 20 years ago. Interestingly, he has been involved pretty early in the bitcoin project as well.

3.1 New Blockchain technologies

In the world of Blockchain 2.0, the main difference between the different initiatives and technologies is really related to the form of support for Smart Contracts:

	No Smart Contracts	Smart Contracts possible	Turing-Complete Smart Contracts
What?	Distributed transaction or accounting storage	Distributed computing of logic available in pre-designed templates	Distributed computing of any logic
Example technologies	Bitcoin (public) Litecoin (public) Multichain (private)	NXT (public) R3 (private)	Ethereum (public) Eris (private) Clearmatics (private)

Presentation of some of them

Bitcoin's platform is great for processing bitcoin transactions, but otherwise has very limited compute ability. Within the scripts of bitcoin transactions there is only very limited ability to implement rich logic.
An example of what is possible in bitcoin is logic requiring multiple signatories to sign a transaction before a payment is made, like needing two signatories in a

cheque. However major changes would need to be made to both the mining functions and the mining incentivisation schemes to enable smart contracts proper on Bitcoin's blockchain.

Sidechains, i.e. blockchains connected to Bitcoin's main blockchain could enable smart contract functionality: by having different blockchains running in parallel to Bitcoin, with an ability to jump value between Bitcoin's main chain and the side chains, side chains could be used to execute logic.

NXT is an public blockchain platform which includes a selection of smart contracts that are currently live. However it is not *Turing-complete*, meaning that it's not possible to code up anything one wants, one has to use the existing templates.

R3/Corda is the private blockchain technology of the R3 consortium. The R3 consortium is constituted by more than 70 of the world biggest financial institutions in research and development of blockchain usage in the financial system. The consortium's joint efforts have created an open-source blockchain platform called *Corda* especially geared towards the financial world as it handles more complex transactions and restricts access to transaction data. The aim of Corda is to provide a platform with common services to ensure that any services built on top are compatible between the network participants, whilst still fostering innovation and faster time to market as the underlying infrastructure would be accepted and understood by at least the founding firms.

Ethereum is a public blockchain platform which is currently the most advanced smart contract enabled blockchain. With a "Turing complete" coding system, theoretically one can put any possible logic into an Ethereum smart contract, and it will be run by the whole network. There are mechanisms in place to prevent abuse, and of course one needs to pay for compute power, by passing in "ETH" tokens, which act as payment for the miners who run the code.

3.2 A focus on R3/Corda and Smart Contract *Templates*

Corda is a distributed ledger platform designed from the ground up to record, manage and synchronize financial agreements between regulated financial institutions. It is heavily inspired by and captures the benefits of blockchain systems, without the design choices that make blockchains inappropriate for many banking scenarios.

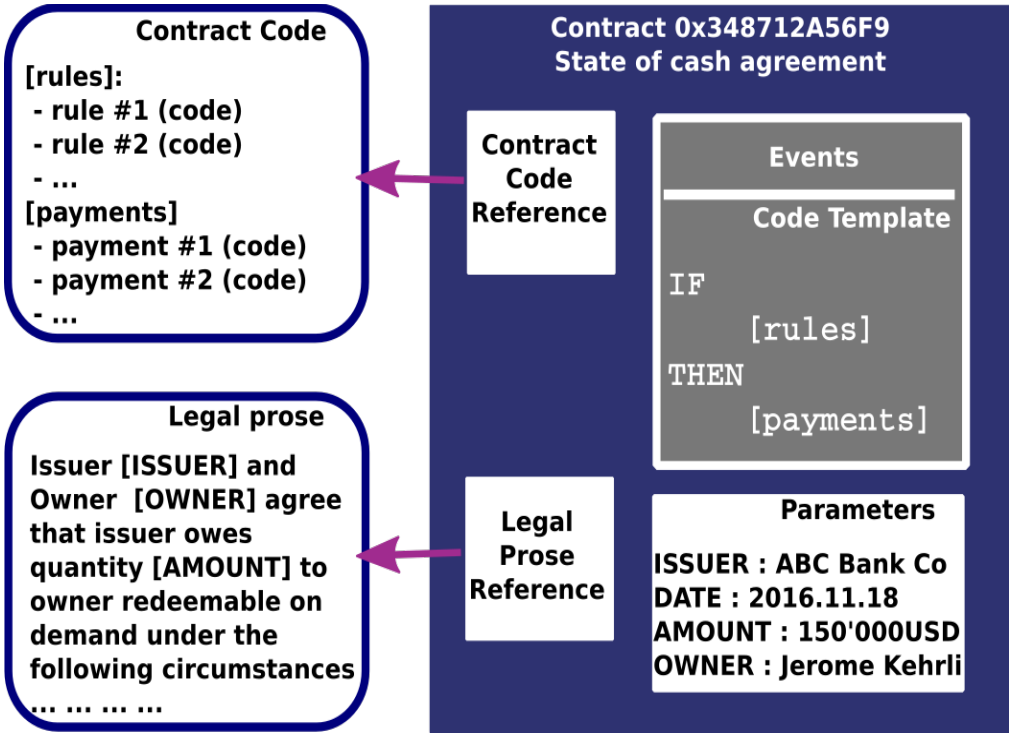
Key features

R3/Corda:

- has no unnecessary global sharing of data: only those parties with a legitimate need to know can see the data within an agreement

- choreographs workflow between firms without a central controller
- achieves consensus between firms at the level of individual deals, not the level of the system
- benefits from a design directly enabling regulatory and supervisory observer nodes
- provides transactions validated by parties to the transaction rather than a broader pool of unrelated validators
- supports a variety of consensus mechanisms
- records an explicit link between human-language legal prose documents and smart contract code
- has no native cryptocurrency

R3/Corda uses **Smart Contract Templates** which captures a Smart Contract as **Ricardian Contract** triple of "*prose, parameters and code*".



This is a very specific model that won't be described further in this article. We will now mostly focus on *Turing-Complete* Smart Contracts and specifically the Ethereum project.

3.3 A first focus on Ethereum and *Turing Complete* Smart Contracts

Ethereum is software running on a network of computers that ensures that data and small computer programs called smart contracts are replicated and processed on all the computers on the network, without a central coordinator. The vision is to create an unstoppable censorship-resistant self-sustaining decentralised world computer. The official website is https://www.ethereum.org

It extends the blockchain concepts from Bitcoin which validates, stores, and replicates transaction data on many computers around the world (hence the term 'distributed ledger'). Ethereum takes this one step further, and also runs computer code equivalently on many computers around the world.

What Bitcoin does for distributed data storage, Ethereum does for distributed data storage plus computations. The small computer programs being run are called smart contracts, and the contracts are run by participants on their machines using a sort of operating system called a "Ethereum Virtual Machine".

4. Ethereum in details

The vision of Ethereum is an **unstoppable censorship-resistant sefl-sustaining decentralized world computer**.

Concept	Description
ETH	Ethereum's inbuilt native cryptocurrency, used for paying for smart contracts to run
Ethereum VM, Swarm and Whisperer	Decentralized Computer, file storage and communication protocols
Solidity, Serpent and LLL	Smart Contract Programming Language
geth, eth, pyethapp	The main Ethereum software, written in different language
Frontier, *Homestead*,*Metropolis*, *Serenity*	Friendly names for different releases

We will now see the concepts of Ethereum in a more detailed way.

4.1 Ethereum concepts

Computers need to be able to calculate, store data, and communicate. For Ethereum to realize its vision as an unstoppable censorship-resistant self-sustaining decentralized 'world' computer, it needs to be able to do those three things fairly efficiently and in a robust way.

Ethers

The digital tokens or coins in Ethereum are called ether. Ether are used as crypto fuel or costs of transaction. Developers have to use ether to submit smart contract rules/code to the blockchain and users have to spend or burn ether to invoke transactions for an application. Transactions will roll back if they run out of gas (the amount of ether specified). Ether can be traded against bitcoins and other fiat currencies via cryptocurrency exchanges.

As a matter of fact, there are several different currency names for different scales:

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

We'll indistrincly use Ether or simply *ETH* below to design the Ethereum cryptocurrency. **Swarm and Whisper**

The **Ethereum Virtual Machine** is the "*virtual world computer*' compotent that can runs and executes smart contract's logic.
This is computation without relying on a central server.

Swarm is Peer-to-Peer file sharing, similar to BitTorrent, but incentivized with micropayments of Ethers. Files are split into chunks, distributed and stored with participating volunteers. These nodes that store and serve the chunks are compensated with ETH from those storing and retrieving the data.
This is file storage without relying on a central server.

Whisper is an encrypted messaging protocol that allows nodes to send messages directly to each other in a secure way and that also hides the sender and receiver from third party snoopers.
This is communications without relying on a central server.

Smart Contract languages: Solidity / Serpent, LLL

There are three common languages smart contracts are written in, which can be compiled into smart contracts and run on Ethereum Virtual Machines. They are:

- **Solidity** : similar to the language Javascript. This is currently the most popular and functional smart contract scripting language.
- **Serpent** : similar to the language Python, and was popular in the early history of Ethereum.
- **LLL (Lisp Like Language)** : similar to Lisp and was only really used in the very early days. It is probably the hardest to write in.

Ethereum software: geth, eth, pyethapp

The official Ethereum clients are all *open source*. The most popular clients are:

- **geth** : written in a language called GO - https://github.com/ethereum/go-ethereum
- **eth** : written in C++ - https://github.com/ethereum/cpp-ethereum
- **pyethapp** : written in Python - https://github.com/ethereum/pyethapp

These are all command-line based programs (think green text on black backgrounds) and so additional software can be used for a nicer graphical interface.
Currently the official and most popular graphical one is Mist
- https://github.com/ethereum/mist, which runs on top of geth or eth.

Accounts

In Bitcoin, there is a concept called *address* where bitcoins are stored - like a bank account number, but for bitcoins. In Ethereum these are commonly called accounts and there are two types:

- **Accounts that only store ETH** : these are similar to Bitcoin addresses and are sometimes known as Externally Owned Accounts (EOAs). One makes payments from these accounts by signing transactions with the appropriate private key.
- **Accounts that store ETH and have code (smart contracts) that can be run** : these smart contracts are activated by a transaction sending ETH into it. Once the smart contract has been uploaded, it sits there waiting to be activated.

Apart from the fact whether an account stores code or not, the EVM treats the two types equally, though. Every account has a persistent key-value store mapping 256-bit words to 256-bit words called storage. Furthermore, every account has a balance

in Ether (in "Wei" to be exact) which can be modified by sending transactions that include Ether.

The fact that Smart Contracts on Ethereum have their own account is important. As a matter of fact, Smart Contracts instances can own ETH.

Gas and Gas Price

Gas is the internal pricing for running a transaction or contract in Ethereum. It's purpose is to decouple the unit of Ether (ETH) and its market value from the unit to measure computational use (gas).
A miner will use an amount of gas directly dependent on the amount of operations it needs to execute to run a Smart Contract or support an API all. If need be, the price of gas, i.e. the price of one unit of gas, can be increased or decreased independently in order to avoid a situation in which an increase in the price of ETH would cause the need to change all computing cost prices expressed in gas.

The usage of this more concrete notion of gas and using a gas price in ETH makes the computation system independent from actual ETH market value.

The gas system is not very different from the use of Kw for measuring electricity home use. One difference from actual energy market is that the originator of the transaction sets the price of gas, to which the miner can or not accept.

The **gas price** is a value set by the creator of the transaction, who has to pay [gas_price * gas] up front from the sending account. If some gas is left after the execution, it is refunded in the same way.
This way, If ETH market value goes up, miners can agree on decreasing the gas price to keep computing cost constant, and the other way around.

In addition, the **gas price** per transaction or contract is set up to deal with the Turing Complete nature of Ethereum and its EVM (Ethereum Virtual Machine Code) - the idea being to limit infinite loops. So for example 10 Szabo, or 0.00001 Ether or 1 Gas can execute a line of code or some command. If there is not enough Ether in the account to perform the transaction or message then it is considered invalid. The idea is to stop denial of service attacks from infinite loops, encourage efficiency in the code - and to make an attacker pay for the resources they use, from bandwidth to CPU calculations through storage.

The more complex the program one wants to execute, the more gas (and thus Ether) one has to pay. For example if A wants to send B one Ether unit - there would be a total cost of 1.00001 Ether to be paid by A. However if A wanted to form a contract with B depending on the future price of Ether, there would be more lines of code executable and more energy consumption placed on the distributed Ether network - and therefore A would have to pay more than the 1 Gas done in the transaction.

4.2 Ethereum and bitcoin differences

	Bitcoin	Ethereum
Applications	Digital Cash Merchant Payments Currency Trading	Smart Contracts Application Platform
Ownership	No one owns Bitcoin. Developer community and mining pool consensus drives roadmap	Ethereum Foundation develops the platform and roadmap for Ethereum
Blockchain	Blockchain is public and decentralized. New blocks mined every 10 minutes. SHA-256 Proof of Work and consensus verification.	Blockchain is public and decentralized. New blocks mined every 12 seconds. Ethash proof of work and consensus verification. Uncle blocks accepted
Coins	15.3 million of a total possible of 21 millions bitcoins have been mined. Current block reward of 25 bitcoins halving every 210'000 blocks	72 million ethers premined 5 ethers generated as block reward, 4.3 for uncle blocks. No cap; linear rising annual ether limit.
Mining	Full and light nodes are allowed. Increasing barriers to CPU mining. Near centralized mining industry with ASIC and FGPA mining in large scale data centers.	Current Frontier release requires full nodes. Ethash proof of work is ASIC resistant and memory hard. GPU friendly by design.
Trading	Bitcoins have a market cap of about $6 billion and are trading around $400 in Q1 2016.	Ether market cap approaching $1 billion trading at $11 in 2016. Second largest cryptocoin by market cap.

Ethereum's block time is shorter

In Ethereum the time between blocks is around 14 seconds, compared with Bitcoin's ~10 minutes. This means that on average if one made a Bitcoin transaction and an Ethereum transaction, the Ethereum transaction would be recorded into Ethereum's blockchain faster than the Bitcoin transaction getting into Bitcoin's blockchain. One could say Bitcoin writes to its database roughly every 10 minutes, whereas Ethereum writes to its database roughly every 14 seconds.

If one followed carefully the first article int his serie (see introduction above), one knows that this has a consequence. A shorter time between blocks means much more *exctincts* blocks and branches being created. A solution needed to be found to make it still interesting to miner to try to mine blocks in Ethereum. See notion of *uncles* below.

Ethereum has smaller blocks

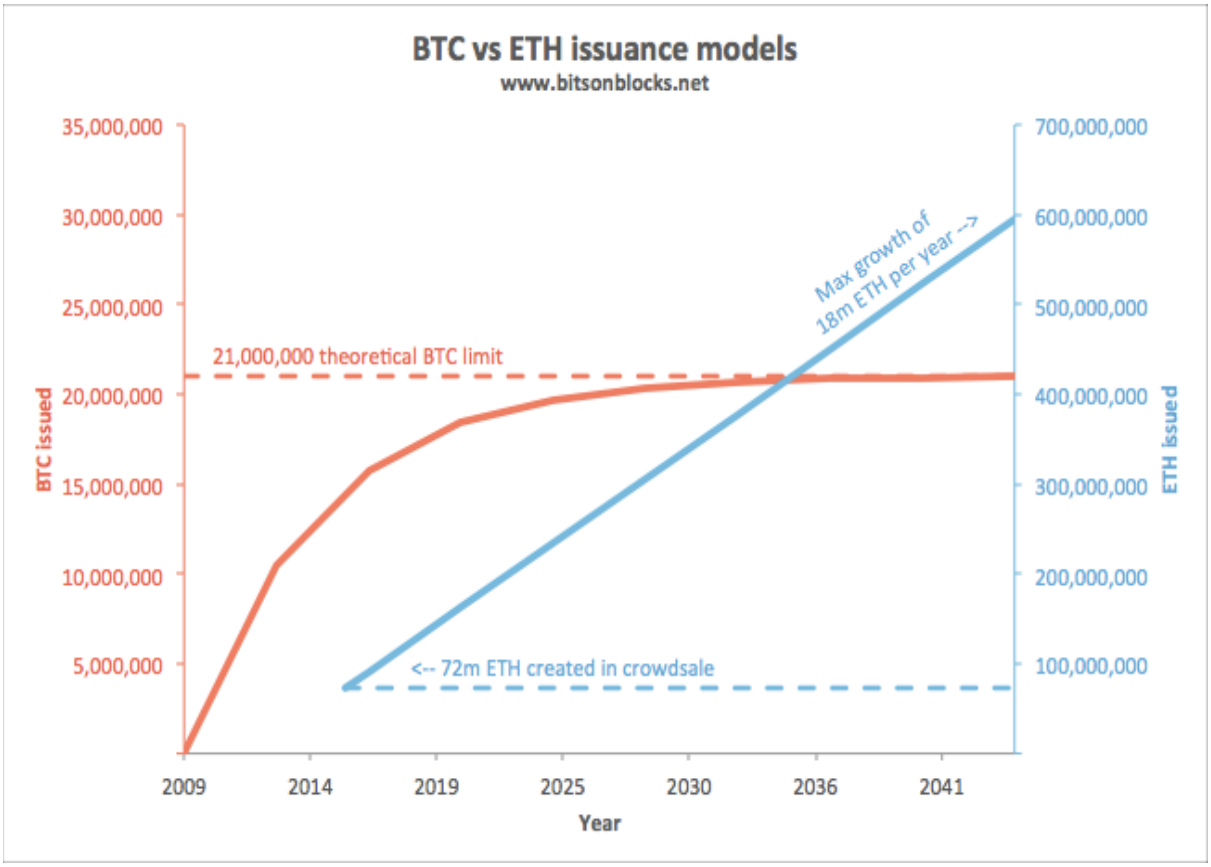
In Bitcoin, the maximum block size is specified in bytes (currently 1 MB) whereas Ethereum's block size is based on complexity of contracts being run - it's known as a Gas limit per block, and the maximum can vary slightly from block to block.
Currently the maximum block size in Ethereum is around 1,500,000 Gas. Basic transactions or payments of ETH from one account to another (ie not a smart contract) have a complexity of 21,000 Gas so one can fit around 70 transactions into a block (1,500,000 / 21,000). In Bitcoin one currently gets around 1,500-2,000 transactions in a block.
Data-wise currently most Ethereum blocks are under 2 KB in size.

ETH (Ether) issuance

Bitcoin is a cryptocurrency and payments network. Bitcoins are designed to be in finite supply and deflationary. 21 million bitcoins will be generated through a halving mining block reward. As on April 2016, 15.3 million bitcoins have been mined and the block reward is 25 bitcoins. All bitcoins have been generated through mining.

Ether is the digital token used in Ethereum. Around 72 million ether were pre-mined and distributed through a crowdfunding sale in exchange for bitcoins to launch Ethereum platform development in 2014. After the Frontier platform went live, Ether are generated as mining block rewards. As on April 2016, there are 78.7 million ether in supply.

Five ether are generated in the form mining reward for new blocks, with five ether per block and uncle blocks are compensated at 7/8th of the block reward or 4.375 ether with a maximum 2 uncles per block which implies a new block confirmation can produce a maximum of 13.75 ether.



Let's just say that this is a lot more complicated than Bitcoin.

Block reward

Currently each block mined creates 5 fresh ETH. Doing the maths, if a block is mined every 14 seconds, and there are 31.5m seconds in a year (365x24x60x60), this means 2.25m blocks are mined per year.
2.25m blocks at 5 ETH per block = 11.3m ETH generated per year. This meets the commitment of less than 18m ETH generated per year.

Uncle reward

Some blocks are mined a little late and don't form part of the main blockchain. In Bitcoin these are called 'orphans' and are entirely discarded, but in Ethereum they are called 'uncles' and can be referenced by later blocks. If uncles are referenced as uncles by a later block, they create about 4.375 ETH for the miner of the uncle (7/8th of the full 5 ETH reward). This is called the uncle reward. Currently around 500 uncles are created per day, adding an additional 2,000 ETH into circulation per day (~0.7m ETH per year at this rate).

This achieves two important things:

- It incentivises miners to mine even though there is a high chance of creating a non-mainchain block (the high speed of block creation results in more orphans or uncles. Ethereum's 12 second block rate significantly increases the rate of orphan blocks and forks)
- It increases the security of the blockchain by acknowledging the energy spent creating the uncle blocks

Uncle referencing reward

And there's a bit more too: A miner who references an uncle also gets about 0.15 ETH per uncle (maximum 2 uncles).
This is called the *GHOST* protocol (Greedy Heaviest-Observed Sub-Tree).

4.3 Etherscript

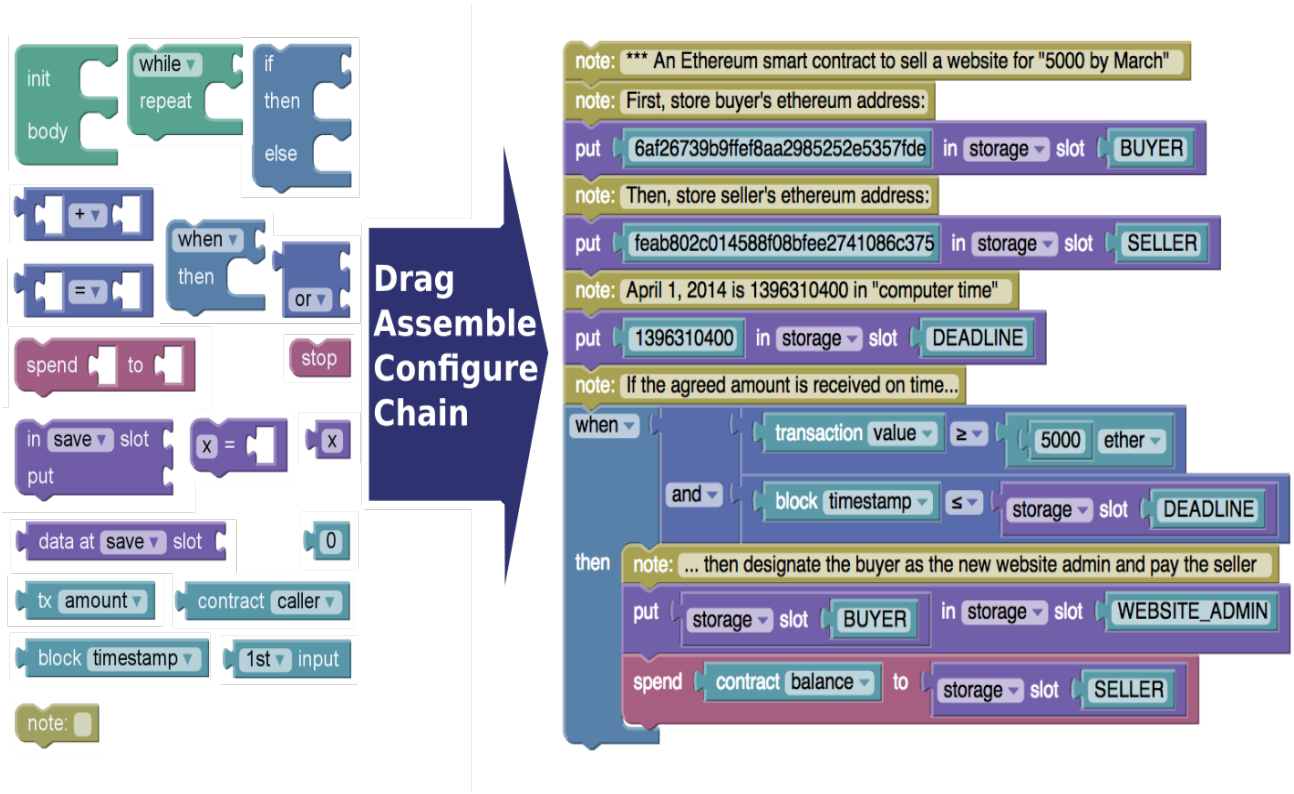
Ethereum's main difference is a turing-complete programming language, sometimes called **EtherScript**. Contracts live on the blockchain in an Ethereum-specific binary format (Ethereum Virtual Machine [=EVM] bytecode). However, contracts are typically written in some high level language such as solidity and then compiled into byte code to be uploaded on the blockchain.

We will later see the example of a SmartContract written in Solidity but before I would like to introduce **Etherscripter** available at http://etherscripter.com/.
Etherscripter is a **Visual Smart Contrat Builder for Ethereum**.
It works by providing the user with a Graphical User Interface (GUI) enabling him to drag-and-drop logic elements on a board representing the Smart Contract Code

Let's look at an example. Imagine the following scenario :

- One has built a website and someone wants to buy it for $5000 but they can only pay in March.
- In the traditional approach, one transfers control of the website and write down what's been agreed on a piece of paper.
- March arrives and it seems there has been some confusion.
- You assumed the contract meant this March but they insist they meant next March.
- Get ready to argue in court about the meaning of "March".

Using Ethereum, you might define the agreement in a form of EtherScript that's readable to both humans and the computer network.
Using Etherscripter, the resulting script would look as follows:



Reading agreements in this form could take some getting used to, but no more than the legalese produced by today's contract lawyers. Fill-in-the-blank scripts would likely become available for common uses. Specialists could craft very custom agreements, as done by lawyers today.
The big advantage here again is that Smart Contract eliminate confusion and remove all uncertainty over whether the other person will follow through. The script both defines and enforces the agreement.

4.4 Hello World in Ethereum

We have seen above a first example of a Smart Contract implemented using Etherscripter. This is interesting for the sake of introducing Smart Contract programming logic. In real life Smart Contract are rather written in one of the Ethereum scripting language, such as Solidity, Serpent or LLL.

We will see here a *Hello World* program developed using Solidity. Solidity is the prefered language by many Smart Contract developers and IMHO a *de-facto* standard.
I will present here the greeter program as presented in Ethereum Greeter.

The Greeter is an intelligent digital entity that lives on the blockchain and is able to have conversations with anyone who interacts with it, based on its input.

Here is its code:

The greeter Smart Contract

```
/* A first abstraction representing a "mortal" entity */
contract mortal {

	/* Define variable owner of the type address*/
	address owner;

	/* this constructor is executed at initialization and sets the owner of the contract
	   It is run only when the contract is created. */
	function mortal() {
		owner = msg.sender; /* msg is an implicit parameter */
	}

	/* Function to recover the funds on the contract */
	function kill() {

		/* Any "mortal" can be destroyed only by its owner */
		if (msg.sender == owner) {
			selfdestruct(owner);
		}
	}
}

/* Our greeter "Smart Contract" */
contract greeter is mortal {

	/* define variable greeting of the type string */
	string greeting;

	/* this runs when the contract is executed */
	function greeter(string _greeting) public {
		greeting = _greeting;
	}

	/* main function */
	function greet() constant returns (string) {
```

```
        return greeting;
    }
}
```

One should notice that there are two different contracts in this code: "mortal" and "greeter".
This is because Solidity has inheritance, meaning that one contract can inherit characteristics of another.

The inherited characteristic "mortal" simply means that the greeter contract can be killed by its owner, to clean up the blockchain and recover funds locked into it when the contract is no longer needed. Contracts in ethereum are, by default, immortal and have no owner, meaning that once deployed the author has no special privileges anymore. One should consider this carefully when deploying SmartContracts on the blockchain.

The reader might want to consult the Ethereum Greeter page to discover about how to compile the greeter program using the SolC compiler.

Deploying on the Ethereum blockchain

The deployment and the initialization of a greeter Smart Contract (instance) is performed using the following commands in the geth console:

```
/* compile greeter source */
var greeterCompiled = web3.eth.compile.solidity(...)

/* The greeting we will use in our greeter "instance" */
var _greeting = "Hello World!"

/* Define greeter Smart Contract (no instance, just definition) */
var greeterContract = web3.eth.contract (greeterCompiled.greeter.info.abiDefinition);

/* Deploy and instantiate greeter smart contract */
var greeter = greeterContract.new (_greeting,
            {from : web3.eth.accounts[0],
             data : greeterCompiled.greeter.code,
             gas  : 300000
            },
            function(e, contract){  /* event listener callback */

    /* If no error occured ... */
```

```
    if (!e) {

      if (!contract.address) {
        console.log("Contract transaction send: TransactionHash: "
                    + contract.transactionHash + " waiting to be mined...");

      } else {
        console.log("Contract mined! Address: " + contract.address);
        console.log(contract);
      }

    } else {

      /* If an error occured, log the error */
      console.log(contract);
    }
});
```

After typing these commands on the geth console, one should wait up to thirty seconds before seeing a message like this:

```
Contract mined! address: 0xdaa24d02bad7e9d6a80106db164bad9399a0423e
```

This contract is estimated to need ~180 thousand gas to deploy, at the time of writing, gas on the test net is priced at 20 gwei (equal to 0.00000002 ether) per unit of gas.
Notice that the cost is not paid to the ethereum developers, instead it goes to the Miners, those peers whose computers are working to find new blocks and keep the network secure. Gas price is set by the market of the current supply and demand of computation. If the gas prices are too high, you can become a miner and lower your asking price.

Interacting with the greeter

In order to call the greeter bot, just type the following command in your terminal:

```
greeter.greet();
```

Since this call changes nothing on the blockchain, it returns instantly and without any gas cost. You should see it return your greeting:

```
'Hello World!'
```

In order for other people to interact with this greeter contract they need two things:

- the address where the contract is located and
- the ABI (Application Binary Interface) which is a sort of user manual, describing the name of its functions and how to call them to your JavaScript console.

In order to get each of them run these commands:

```
var ABI = greeterCompiled.greeter.info.abiDefinition;
var Address = greeter.address;
```

Recover these two variables ABI and Address and send them to whoever wants to interact with the greeter Smart Contract.
Then, on his remote computer, the other user can get access to the greeter using the following commands:

```
var greeter = eth.contract(ABI).at(Address);
```

That's it !

Clean-up

Whenever one has fun enough playing with the greeter, one should get rid of it by making it unusable and thus cleaning the blockchain from abandoned live contracts

A transaction will need to be sent to the network and a fee to be paid for the changes made to the blockchain after the code below is run. The self-destruct is subsidized by the network so it will cost much less than a usual transaction.

```
greeter.kill.sendTransaction({from:eth.accounts[0]})
```

This can only be triggered by a transaction sent from the contracts owner. You can verify that the deed is done simply seeing if this returns 0:

```
eth.getCode(greeter.address)
```

4.5 Further Ethereum Examples

The example above is actually not very relevant since it doesn't make any usage of what Ethereum is finally much more about : exchanging goods and money. Describing more advanced example exceeds the scope of this article and I will write another article when I come up myself with something relevant in the coming weeks. In the meantime hereunder are 2 examples I think are worth looking at and that make a great job in describing what Ethereum is able to do

Sub-currency example

The following contract will implement the simplest form of a cryptocurrency. It is possible to generate coins out of thin air, but only the person that created the contract will be able to do that (it is trivial to implement a different issuance scheme). Furthermore, anyone can send coins to each other without any need for registering with username and password - all you need is an Ethereum keypair.

See Sub-currency example on solidity

Democracy DAO

The example below creates a first decentralized autonomous organization, or DAO.

Think of the DAO as the constitution of a country, the executive branch of a government or maybe like a robotic manager for an organization. The DAO receives the money that your organization raises, keeps it safe and uses it to fund whatever its members want by voting about it. The robot is incorruptible, will never defraud the bank, never create secret plans, never use the money for anything other than what its constituents voted on. The DAO will never disappear, never run away and cannot be controlled by anyone other than its own citizens.

The token we distributed using the crowdsale is the only citizen document needed. Anyone who holds any token is able to create and vote on proposals. Similar to being a shareholder in a company, the token can be traded on the open market and the vote is proportional to amounts of tokens the voter holds.

See The democracy DAO example

4.6 Private chains in Ethereum

Ethereum software enables a user to set up a "private" or "testnet" Ethereum chain that is separate from the main Ethereum chain. This is useful for testing distributed apps built on Ethereum without having to expose your apps or trials to the real Ethereum network using real Ether. You either pre-generate or mine your own Ether on your private Ethereum chain, so it is a much more cost effective way of trying out Ethereum.

What are the components that tell geth that we want to use/create a private Ethereum chain? The things that dictate a private Ethereum chain are:

- Custom Genesis File
- Custom Data Directory
- Custom NetworkID
- (Recommended) Disable Node Discovery

This is actually pretty straightforward and one should simply follow this tutorial

4.7 *The DAO* and *The DAO Attack*

The DAO is Code

The DAO was a digital decentralized autonomous organization and a form of something in between an*investor-directed venture capital fund* and a *crowfounding platform*.
The DAO had an objective to provide a new decentralized business model for organizing both commercial and non-profit enterprises. It had no conventional management structure or board of directors. The code of the DAO is open-source.

The DAO was stateless, and not tied to any particular nation state. As a result, many questions of how government regulators would deal with a stateless fund were yet to be dealt with.
The DAO itself was crowdfunded via a token sale in May 2016. It set the record for the largest crowdfunding campaign in history.

The DAO was intended to operate as "*a hub that disperses funds (currently in Ether, the Ethereum value token) to projects.*" Investors received voting rights by means of a digital share token; they voted on proposals that are submitted by "contractors" and a group of volunteers called "curators" checked the identity of people submitting proposals and made sure the projects were legal before "whitelisting" them. The profits from the investments will then flow back to its stakeholders.
The DAO did not hold the money of investors; instead, the investors owned DAO tokens that give them rights to vote on potential projects. Anyone could pull out their funds until the time they first vote.
Its reliance on Ether has allowed people to send their money to it from anywhere in the world without providing any identifying information

Since 28 May 2016 the DAO tokens were tradable on various cryptocurrency exchanges.

The DAO Attack

A paper published in May 2016 noted a number of security vulnerabilities associated with The DAO, and recommended that investors in The DAO hold off from directing The DAO to invest in projects until the problems had been resolved.
An Ethereum developer on Github pointed out a flaw relating to "recursive calls" in early June that was picked up and blogged by Peter Vessenes, founder of the Blockchain Foundation on June 9.
By June 14, fixes had been proposed and were awaiting approval by members of The DAO.
On June 16 further attention was called to recursive call vulnerabilities by bloggers affiliated with the IC3.
It's important to note that **the vulnerabilities** discussed here **were not related to the blochchain technology itself**, but rather to the specific implementation of The DAO Smart Contract.

On June 17, 2016, The DAO was subjected to a hack that deployed a combination of vulnerabilities, including the one concerning recursive calls, and the hacker gained control of 3.6 million Ether, around a third of the 11.5 million Ether that had been committed to The DAO; the stolen Ether had a value of about $50M at the time of the hack.

The hacked funds were put into an account likely subject to a 28 day holding period under the terms of the Ethereum contract so were probably not actually gone; members of The DAO and the Ethereum community debated what to do next, with some calling the hack a valid but unethical maneuver and others calling for the Ether to be re-appropriated and some calling for The DAO to be shut down

On the 20th July 2016, the Ethereum community decided to hard-fork the Ethereum blockchain to restore virtually all funds to the original contract. This was controversial, and led to a fork in Ethereum, where the original unforked blockchain was maintained as Ethereum Classic, thus breaking Ethereum into two separate active cryptocurrencies.

This fork of the blockchain is a tsunami. It really consisted in "rewriting history" which is **precisely what the blockchain should prevent and protect its users from**. This causes a lot of interesting questions :

- What is better ? A really unbreakable blockchain, making the above fork really impossible but making it also impossible to recover from thief or badly intentioned people exploiting vulnerabilities in a Smart Contract ?
- It raises the fact that one of the big challenges of the blockchain technology is **governance**. I'll get back to this in a next article on this topic

In the fallout of the incident, much was made about how The DAO was "hacked". Upon closer examination though, The DAO was not hacked at all. The attacker(s) used two features of The DAO's specialised code to siphon out ether in amounts small enough to not result in the destruction of their DAO tokens.

Moreover, it is perfectly legitimate to do whatever a smart contract's code permits, even if this is beyond the original intention of those who wrote the code. Like all technologies, "smart contracts" are dual use and might be used in ways that their creators did not intend. The complexity of the technology only compounds this issue.

5. Smart Contracts use cases

I have identified a wide range of applications - ranging from smart health records to pay-as-you-go insurance - that companies are either piloting right now or that are being considered by discussions around the blockchain:

Industry	Use Case	What Smart Contracts can do
Financial Services	**Trade Clearing and Settlement**	Manage Approval workflows between counterparties, calculate trade settlement amounts, and transfer funds automatically.
	Coupon Payments	Automatically calculate and pay periodic coupon payments and returns principal upon bond expiration.
	Commercial and Retail banking	support mortgage lending, loans and crowdfunding, etc.
	Insurance Claim Processing	Perform error checking, routing, and approval workflows, and calculate payout based on the type of claim and underlying policy
	Micro-insurance	Calculate and transfer micropayments based on usage data from an Internet-of-Things enabled device (example : pay-as-you-go automotive insurance)
	Cyber-insurance	Support insurance for the sharing economy, autonomous object insurances, peer-to-peer insurances, improve fraud prevention, etc.
	Corporate	Support decentralized venture capital fund

	Finance and Funding	that relies on a *wisdom-of-the-crowd* voting system to make investment decisions.
Life sciences and Health Care	**Electronic Medical Records**	Provide transfer and/or access to metrical health records upon multi-signature approvals between patients and providers.
	Population Health Data Access	Grant health researchers access to personal health information; micropayments are automatically transferred to the patient for participation.
	Personal health tracking	Track patients' health-related actions through IoT devices and automatically generates reward baseds on specific milestones.
Technology, Media and Telecom	**Royalty distribution**	Calculate and distribute royalty payments to artists and other associated parties according to contracts.
	Social networks	Decentralizing online communities with a clever rating system, censorship and moderation elimination
	IT Services and Web hosting	Eliminate risk of *Denial Of Service attacks* by having the site stored everywhere
Energy and resources	**Autonomous electric vehicles charging stations**	Process a deposit, enable the charging station and return remaining fund when complete .
Gaming	**Casinos, online gambling and lotteries**	Support *provably fair* casino style gambling, eliminating cheating and fees.
	Prediction platforms	combine prediction market and algorithms with power of decentralization to create forcasting tools
Public Sector	**Record-keeping / company registry**	Update private company share registries and capitalization table records, and distribute shareholder communication.
	Real-Estate / Land registry	Maintain land registry, track changes of ownership, update transactions register
	Person registry	Track civil status of persons, but also marriage contracts and wills, death settlements, etc.
Cross-industry	**Supply chain and trade finance**	Transfers payments upon (digital) multi-signature approval for letters of credit and issues port payments upon custody

	documentation	changes for bills of lading
	Product Provenance / Ownership and History	Facilitates chain-of-custody processes for products in the supply chain where the party in custody is able to log evidence about the custody. (This is a key use case in so many fields : luxury, gold and diamonds, medical goods, wines, applications, etc.)
	Peer to peer transacting	Match parties and transfer payments automatically for various peer-to-peer applications: lending, insurance, energy, credits, etc.
	Voting	Validate voter criteria, log vote in the blockchain, and initiate specific action as a result of the majority vote. Also applies to elections, in both public or private sectors.
	Person identification	Support trustworthy identification and proof of identity, proving authenticity of actions, reputation management, access management.
	Loyalty programs	support tracking of product or service usage (e.g Airline miles)

6. Issues and challenges

Smart contract technology is still in its early stages. One should track both technology and business developments surrounding smart contracts. On the technology side, certain advances will help broaden the applications and adoption of smart contracts.

Challenges with current blockchain are as follows :

Scalability

Smart contract platforms are still considered unproven in terms of scalability. Think of high frequency trading and the million of transaction on stock exchanges every second, or the hundreds of thousands of facebook updates. If Big Data technologies is at one side of the scale, then blockchain platforms at at the other side, for now.

The community is aware of this problem and is thinking of (more than working of) several approaches such as sharding of computations or off-chain computations.
It remains to be seen whether or not any of these solutions can preserve the key selling points of a public blockchain i.e. its trustful, permissionless and decentralized nature.

Access to real world information

As discussed above, because smart contracts can reference only information on the blockchain, trustworthy data services-known as "oracles" - that can push information to the blockchain will be needed.
Approaches for creating oracles are still emerging. While some initiatives are promising, this is really a new field and where nothing clear and straightforward has emerged now.
In addition, a lot of issues are not clearly addressed. For instance, what happens when things change: what happens if information sources go away, if previously independent sources merge, if new and better sources emerge?

Privacy

The code within smart contracts is visible to all parties within the network, which may not be acceptable for some applications.
For instance, some retailers may not want their deals with their suppliers to be public.

Latency and performance.

Blockchains suffer from high latency, given that time passes for each verified block of transactions to be added to the ledger. For Ethereum this occurs approximately every 17 seconds - a far cry from the milliseconds to which we are accustomed while using non-blockchain databases.
Smart Contracts make things even slower and Ethereum's target of a new block arriving every 12 seconds is quite ambitious and raises the risk of a node drowning under a backlog of work

Permissioning.

While excitement for smart contracts is growing in the realm of both permission-less and permissioned blockchains, the latter is likely to see faster adoption in industry, given that complexities around trust, privacy, and scalability are more easily resolved within a consortium of known parties.

Limits of application

There are often good reasons for optionality. In many contracts, clauses are written into things on purpose to create a channel for arbitration. For example in a flat rental agreement, wear-and-tear from tenants is acceptable, but major damage needs to be repaired. How does code define these things? *Force majeure* is present in many contracts to allow for wiggle-room for the parties involved. In a smart contract environment, how does one party call that without abusing it or referring to a human arbitrator ?

In addition some business may simply not be modeled in a way that would enable it to benefit from Smart Contracts or other blockchains.

Governance

If blockchains are to be sustainable in the long run, serious consideration of appropriate governance mechanisms is needed.

A skewed distribution of mining power and crypto-currency holdings is combined with pseudonymity of account holders and a strong incentive to game the system. This has all the makings for deceptive, unaccountable, fraudulent, and self interested decision making.
Until hard questions around governance of blockchains are asked, and solutions implemented, we should brace ourselves for more incidents like that which has befallen The DAO.

7. Conclusion

This (yet pretty long) article is really just an introduction to Smart Contracts and blockchain 2.0 technologies. There is so much more to say on both Blockchain 2.0 initiatives in general and Ethereum specifically. I guess I'll cover some more aspects following my discoveries when playing with Ethereum or discoveries I make on the technology at large.

What I would like to add for now is that the blockchain technology, despite all the hypes and all the investment in fancy startups around it, still need to mature and the issues I described above are real challenges that need yet to be overcome.

Even if Ethereum is widely discussed nowadays, its only a year and a half old and still in an initial stage. Smart Contracts application are being studied and almost every week a new startup is announcing working on a new use case. But still, proven businesses and wide adoption is still far ahead of us.

Having said that, this technology is a definitely a breakthrough and has an amazing potential. Many technology experts said that its the most disruptive technology they have seen since the apparition of the World Wide Web in the early 90's. And I believe them.

(Again, one might want first article in this serie : Blockchain explained that provides a pretty complete introduction to the initial bitcoin blockchain technology)

Also one might want to see part of this article as a slideshare presentation available here :http://www.slideshare.net/JrmeKehrli/blockchain-20-69472625.