Module -3

Bitcoin Clients and APIs

Types of Bitcoin Core clients

1. **Bitcoind :** This is the core client software that can be run as a daemon
   -it provides the JSON RPC interface.
2. **Bitcoin-cli:** This is the command line feature-rich tool to interact with the daemon.
   Bitcoin-cli calls only JSON-RPC functions and does not perform any actions on its own on the blockchain.
3. **Bitcoin-qt :** This is the Bitcoin Core client GUI. When the wallet software starts up first, it verifies the blocks on the disk and then starts up. The verification process is not specific to the Bitcoin-qt client; it is performed by the Bitcoind client as well.

Alternative Coins (altcoin)

- an altcoin is generated in the case of a hard fork.
- Altcoins must be able to attract new users, trades, and miners otherwise the currency will have no value.
- Currency gains its value, especially in the virtual currency space, due to the network effect and its acceptability by the community.
- If a coin fails to attract enough users then soon it will be forgotten.
- Users can be attracted by providing an initial amount of coins and can be achieved by using various methods
  Methods of providing an initial number of altcoins are discussed as follows:
  Create a new blockchain:
- Altcoins can create a new blockchain and allocate coins to initial miners
- pump and dump schemes where initial miners made a profit with the launch of a new currency and then disappeared.
  Proof of Burn (PoB): An approach to allocating initial funds to a new altcoin is PoB called a one-way peg or price ceiling.
  users permanently destroy a certain quantity of bitcoins in bitcoins were destroyed then altcoins can have a value no greater than some bitcoins destroyed. This means that bitcoins are being converted into altcoins by burning them.
  Proof of Ownership: Instead of permanently destroying bitcoins, an alternative method is to prove that users own a certain number of bitcoins. This proof of ownership can be used to claim altcoins by tethering altcoin blocks to Bitcoin blocks. For example, this can be achieved by merged mining in which effectively bitcoin miners can mine altcoin blocks while mining for bitcoin without any extra work.
  Pegged sidechains: Sidechains are blockchains separate from the bitcoin network but bitcoins can be transferred to them. Altcoins can also be transferred back to the bitcoin network. This concept is called a two-way peg.

Smart Contracts

Smart contracts were first theorized by Nick Szabo in the late 1990s in an article named Formalizing and Securing Relationships on Public Networks.

Definition

A smart contract is a secure and unstoppable computer program representing an agreement that is automatically executable and enforceable.

smart contracts are automatically executed when certain conditions are met. They are enforceable, which means that all contractual terms are executed as defined and expected, even in the presence of adversaries.

They are secure and unstoppable, which means that these computer programs are required to be designed in such a fashion that they are fault tolerant and executable in a reasonable amount of time. These programs should be able to execute and maintain a healthy internal state, even if external factors are unfavorable. For example, imagine a typical computer program that is encoded with some logic and executes according to the instruction coded within it. However, if the environment it is running in or external factors it relies on deviate from the normal or expected state, the program may react arbitrarily or simply abort. It is essential that smart contracts be immune to this type of issue.

If humans and machines can both understand the code written in a smart contract it might be more acceptable in legal situations. This is achieved using a markup language that describe natural language contracts by combining both smart contract code and natural language contract through linking contract terms with machine understandable elements. An example of this type of markup language is called **Legal Knowledge Interchange Format (LKIF),** which is an XML schema for representing theories and proofs.

Smart contracts are not really smart, they are simply doing what they are programmed to do. This deterministic nature of smart contracts is highly desirable in blockchain platforms due to consistent consensus requirements. A deterministic feature ensures that smart contracts always produce the same output for a specific input. In other words, programs, when executed, produce a reliable and accurate business logic that is entirely in line with the requirements programmed in the high-level code.

In summary, a smart contract has the following four properties:

- Automatically executable
- Enforceable
- Semantically sound
- Secure and unstoppable

Ricardian contracts

Ricardian contracts were proposed by Ian Grigg in late 1990s. These contracts were used initially in a bond trading and payment system called **Ricardo**.

A Ricardian contract is a digital contract that functions as a legally binding agreement between two parties based on agreed-upon terms and conditions. The contract is cryptographically signed and verified using the blockchain, but is readable by both people and machines.

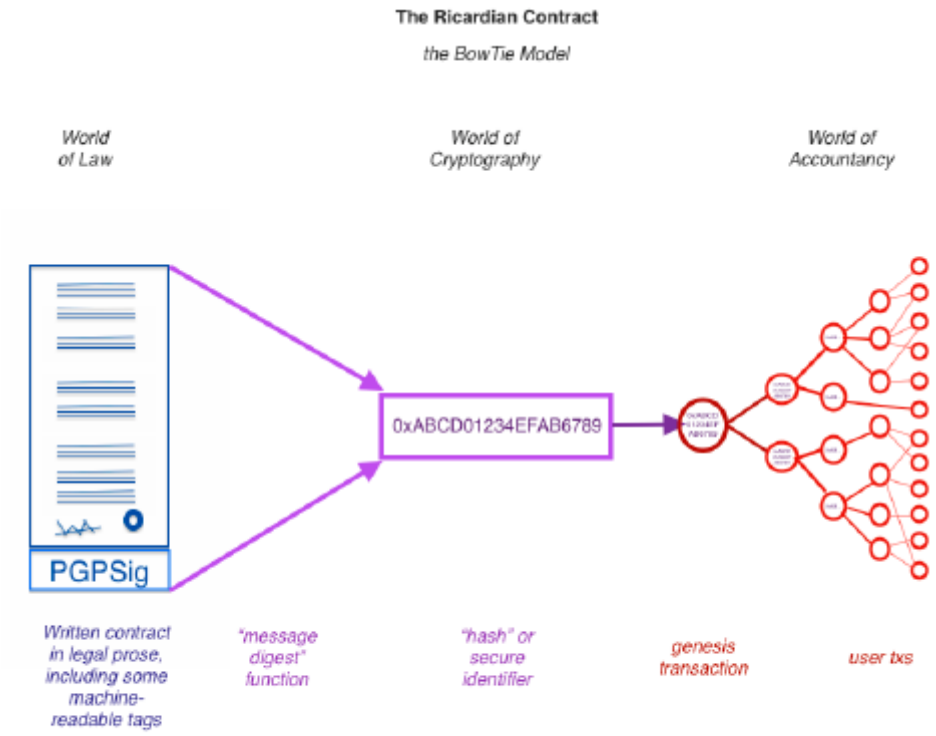
The fundamental idea is to write a document that is understandable and acceptable by both a court of law and computer software.

A Ricardian contract is a document that has several of the following properties:

- A contract offered by an issuer to holders
- A valuable right held by holders and managed by the issuer
- Easily readable by people (like a contract on paper)
- Readable by programs (parsable, like a database)
- Digitally signed
- Carries the keys and server information
- Allied with a unique and secure identifier

The contracts are implemented by producing a single document that contains the terms of the contract in legal prose and the required machinereadable tags. This document is digitally signed by the issuer using their private key. This document is then hashed using a message digest function to produce a hash by which the document can be identified. This hash is then further used and signed by parties

during the performance of the contract to link each transaction, with the identifier hash thus serving as an evidence of intent. This is depicted in bowtie model.



Ricardian contracts, bowtie diagram

The diagram shows number of elements:

- The World of Law on the left-hand side from where the document originates. This document is a written contract in legal prose with some machine-readable tags.
- This document is then hashed.
- The resultant message digest is used as an identifier throughout the World of Accountancy, shown on the right-hand side of the diagram

The World of Accountancy element represents any accounting, trading, and information systems that are being used in the business to perform various business operations. The idea behind this flow is that the message digest generated by hashing the document is first used in a so-called genesis transaction, or first transaction, and then used in every transaction as an identifier throughout the operational execution of the contract.

Characteristics	Smart Contracts	Ricardian Contracts
Adaptability	Limited in its process to adapt properties of Ricardian Contracts	Easily adapts Smart Contracts chracteristics
Scalability	Not flexible to add/alter new terms in Contract	Scalable and flexible to incorporate new Contract terms
Readability	Machine-readable	Machine and human readable
Purpose	Automatically executes instruction based on event triggers	Physically recorded legal agreement

Benefits

Smart Contracts		Ricardian Contracts	
Secured	Cost effective	Legally binding	Extremely secure
Trust	Autonomous	Can act as smart contract	
Accurate, always	Backup	Save costs and time	
Transparent		Support automation of operation	

eliNext

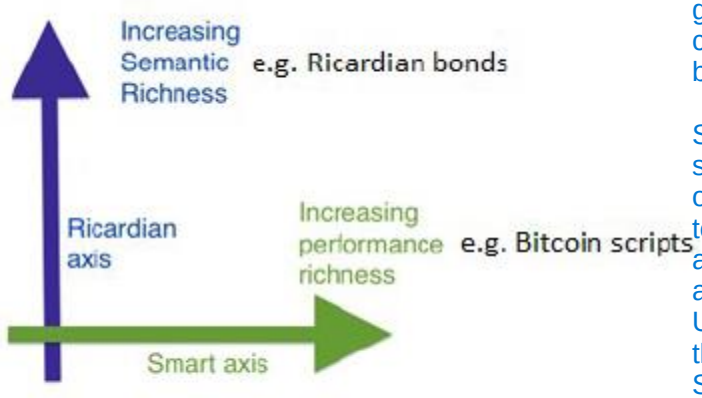


Diagram explaining performance versus semantics are orthogonal issues as described by Ian Grigg, slightly modified to show examples of different types of contracts on both axis

Richardian Contracts refer to traditional contracts that are based on the legal system and are enforceable by the courts. They are written agreements between two parties that specify the terms and conditions of a transaction. Richardian Contracts are governed by the law and enforceable by the courts, which can enforce penalties for breaches of contract.

Smart Contracts, on the other hand, are self-executing computer programs that run on a blockchain network. They contain the terms of the agreement between parties, and the execution of the contract is automatic when certain conditions are met. Unlike Richardian Contracts, which rely on the legal system to enforce the agreement, Smart Contracts are enforceable through code. They are transparent, secure, and immutable, which means that once a contract is executed, it cannot be altered or deleted.

Ricardian contracts have been implemented in many systems, such as CommonAccord, OpenBazaar, OpenAssets, and Askemos.

Smart contract templates are pre-written code snippets that provide a foundation for creating a smart contract. They are designed to be used as a starting point for developers and are meant to simplify the process of creating smart contracts by providing a set of commonly used functionalities.

Smart contract templates

Smart contracts can be implemented for any industry where required, but mostly related to the financial industry. Recent work in smart contract space specific to the financial industry has proposed the idea of smart contract templates. The idea is to build standard templates that provide a framework to support legal agreements for financial instruments.

A language named **CLACK, a common domain-specific language** for augmented contract knowledge has been proposed . This language is intended to be very rich and provide a large variety of functions ranging from supporting legal prose to the ability to be executed on multiple platforms and cryptographic functions.

It is also important to understand the concept of domain-specific languages as this type of languages can be developed to program smart contracts. These languages are developed with limited expressiveness for a particular application or area of interest. Domain-specific languages (DSLs) are different from general-purpose programming languages (GPLs). DSLs have a small set of features that are sufficient and optimized for the domain they are intended to be used in and, unlike GPLs, are usually not used to build general purpose large application programs.

Solidity and Vyper are languages that has been introduced for Ethereum smart contact development.

It is proposed that research should also be conducted in the area of developing high-level DSLs that can be used to program a smart contract in a user-friendly graphical user interface, thus allowing a non-programmer domain expert (for example, a lawyer) to design smart contracts.

Oracles

The limitation with smart contracts is that they cannot access external data, which might be required to control the execution of the business logic; for example, the stock price of a security product that is required by the contract to release the dividend payments. Oracles can be used to provide external data to smart contracts. **An Oracle is an interface that delivers data from an external source to smart contracts.**

Oracle is a crucial component of smart contract technology, as it allows smart contracts to interact with external data and off-chain resources. Smart contracts are self-executing contracts with the terms of the agreement directly written into code. However, as they run on blockchain networks, they are unable to access data outside the blockchain. This is where Oracles come into play.

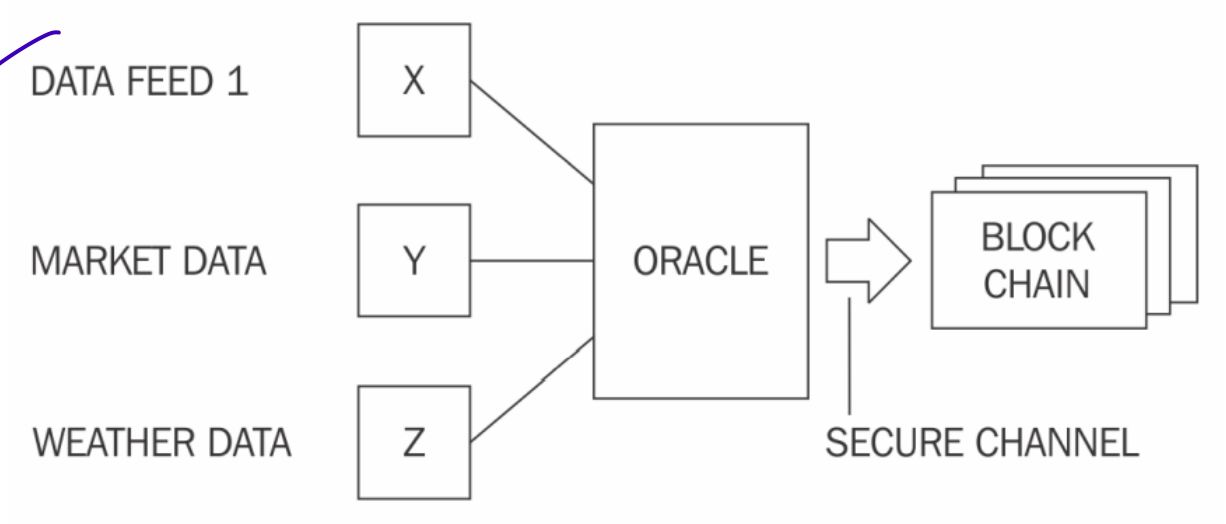
An Oracle acts as a bridge between the smart contract and the external world, providing data to the smart contract that can trigger its execution. For example, an Oracle can provide real-time stock prices to a smart contract that executes trades based on market conditions. Without an Oracle, smart contracts would be limited to data that is already stored on the blockchain, making them much less useful.

Oracles are data feeds that bring data from off the blockchain (off-chain) data sources and puts it on the blockchain (on-chain) for smart contracts to use. This is necessary because smart contracts running on Ethereum cannot access information stored outside the blockchain network.
Depending on the industry and requirements, Oracles can deliver different types of data ranging from weather reports, real-world news, and corporate actions to data coming from Internet of Things (IoT) devices. Oracles are trusted entities that use a secure channel to transfer data to a smart contract.

Oracles are also capable of digitally signing the data proving that the source of the data is authentic.
Smart contracts can then subscribe to the Oracles, and the smart contracts can either pull or push the data to the smart contracts. Oracles should not be able to manipulate the data they provide and must be able to provide authentic data.
Even though Oracles are trusted, it may still be possible in some cases that the data is incorrect due to manipulation. Therefore, it is necessary that Oracles are unable to change the data.

Oracle is provided by a large, reputable, trusted third party, but the issue of centralization remains. These types of Oracles can be called standard or simple Oracles. For example, the source of the data can be from a reputable weather reporting agency or airport information system relaying the flight delays.

Another concept that can be used to ensure the credibility of data provided by third-party sources for Oracles is that data is sourced from multiple sources; even users or members of the public that have access and knowledge about some data can provide the required data.
Another type of Oracle, which essentially emerged due to the decentralization requirements, is called decentralized Oracles. These types of Oracles can be built based on some distributed mechanism.
Another concept of hardware Oracles is also introduced by researchers where real-world data from physical devices is required. For example, this can be used in telemetry and IoT.
The following diagram shows a generic model of an Oracle and smart contract ecosystem:



A generic model of an Oracle and smart contract ecosystem

For example, in Bitcoin blockchain, an Oracle can write data to a specific transaction and a smart contract can monitor that transaction in the blockchain and read the data.

Smart Oracles

Smart Oracles are entities just like Oracles, but with the added capability of contract code execution. Smart Oracles proposed by Codius run using Google Native Client, which is a sandboxed environment for running untrusted x86 native code.

Deploying smart contracts on a blockchain

Smart contracts may or may not be deployed on a blockchain, but it makes sense to deploy them on a blockchain due to the distributed and decentralized consensus mechanism provided by blockchain. Ethereum is an example of a blockchain platform that natively supports the development and deployment of smart contracts. Smart contracts on Ethereum blockchain are usually part of a broader application such as **Decentralized Autonomous organization (DAOs).**

For example, you can implement conditions such as "Pay party X, N amount of bitcoins after 3 months".

One example, of a basic smart contract, is to fund a Bitcoin address that can be spent by anyone who demonstrates a "hash collision attack".

Various other blockchain platforms support smart contracts such as Monax, Lisk, Counterparty, Stellar, Hyperledger fabric, corda, and Axoni core. Smart contracts can be developed in various languages. The critical requirement is determinism, wherever the smart contract code executes, it produces the same result every time and everywhere

Various languages have been developed to build smart contracts such as Solidity, which runs on Ethereum Virtual Machine (EVM).Hyperledger fabric which supports Golang, Java, and JavaScript for smart contract development.

The DAO

-is one of the highest crowdfunded projects,
-started in April 2016
- a set of smart contracts written to provide a platform for investment.
-Due to a bug in the code, this was hacked in June 2016
-This incident resulted in a hard fork on Ethereum to recover from the attack.
-It should be noted that the notion of code is the law or unstoppable smart contracts
-is evident from the recent events where the Ethereum foundation was able to stop and change the execution of The DAO by introducing a hard fork.
-it goes against the true spirit of decentralization, and the notion of code is law.
- resistance against this hard fork and some miners who decided to keep mining on the original chain resulted in the creation of Ethereum Classic.
-It also highlights the absolute need to develop a formal language for development and verification of smart contracts.
-The attack also highlighted the importance of thorough testing to avoid the issues that the DAO experienced.
-There have been various vulnerabilities discovered in Ethereum recently around the smart contract development language.

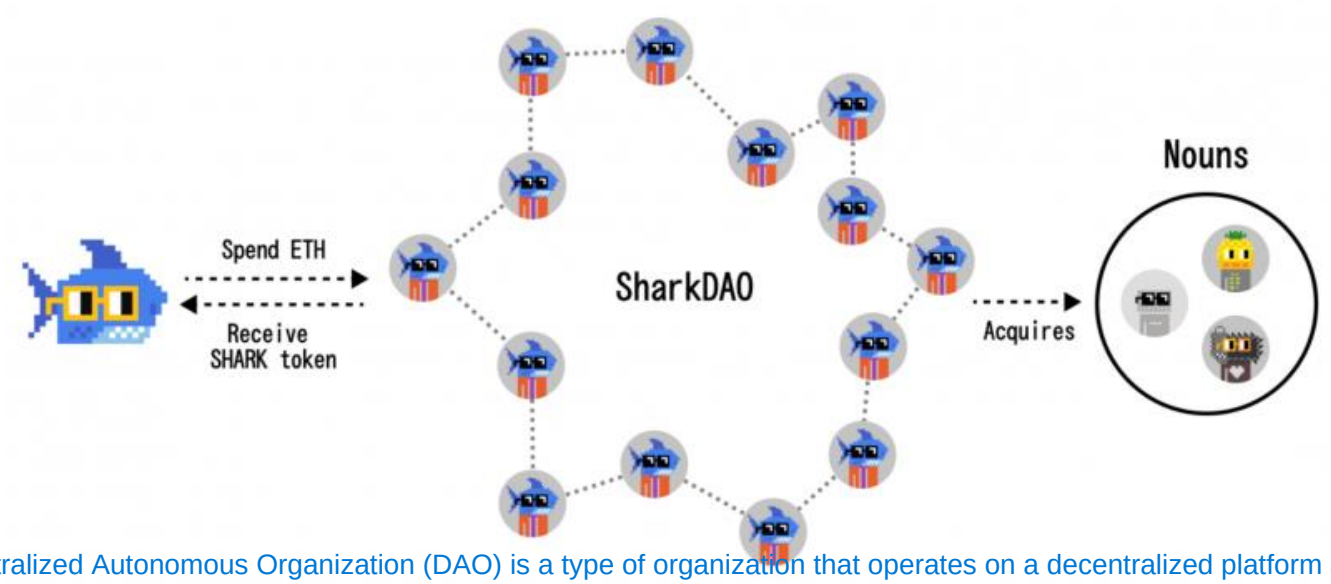
A DAO, or “Decentralized Autonomous Organization,” is a community-led entity with no central authority. It is fully autonomous and transparent: smart contracts lay the foundational rules, execute the agreed upon decisions, and at any point, proposals, voting, and even the very code itself can be publicly audited.

Ultimately, a DAO is governed entirely by its individual members who collectively make critical decisions about the future of the project, such as technical upgrades and treasury allocations.

Generally speaking, community members create proposals about the future operations of the protocol and then come together to vote on each proposal. Proposals that achieve some predefined level of consensus are then accepted and enforced by the rules instantiated within the smart contract.

How does a DAO work?

The rules of the DAO are established by a core team of community members through the use of smart contracts. These smart contracts lay out the foundational framework by which the DAO is to operate. They are highly visible, verifiable, and publicly auditable so any potential member can fully understand how the protocol is to function at every step.
Once these rules are formally written onto the blockchain, the next step is around funding: the DAO needs to figure out how to receive funding and how to bestow governance.
This is typically achieved through token issuance, by which the protocol sells tokens to raise funds and fill the DAO treasury.
In return for their fiat, token holders are given certain voting rights, usually proportional to their holdings.
Once funding is completed, the DAO is ready for deployment.
At this point, once the code is pushed into production, it can no longer be changed by any other means other than a consensus reached through member voting. That is, no special authority can modify the rules of the DAO; it is entirely up to the community of token holders to decide.



A Decentralized Autonomous Organization (DAO) is a type of organization that operates on a decentralized platform and is run through rules encoded as computer programs called smart contracts. The following are some of the key properties of DAOs:

Decentralization: DAOs are decentralized, meaning that they are not controlled by a single central authority or individual. Instead, they are run by a network of participants who have a stake in the organization.

Autonomy: DAOs are autonomous, meaning that they operate according to a set of predefined rules encoded in the form of smart contracts. These rules govern the behavior of the organization, and decisions are made based on consensus among its members.

Transparency: DAOs are transparent, meaning that their operations and decision-making processes are publicly visible and auditable. This helps to increase trust among participants and ensures that the organization is operating in a fair and transparent manner.

Security: DAOs are secure, meaning that their operations are protected from external interference and malicious actors. This is achieved through the use of cryptographic algorithms and distributed networks.

Inclusiveness: DAOs are inclusive, meaning that anyone with an internet connection can participate in the organization. This helps to increase the diversity of perspectives and ideas and allows for more democratic decision-making processes.

Programmability: DAOs are programmable, meaning that their rules and operations can be changed or updated over time based on the consensus of its members. This allows for the evolution and adaptation of the organization as circumstances change.

In summary, DAOs represent a new type of organization that leverages the power of decentralized technology to create more transparent, secure, and inclusive organizations.