Smart Contracts

History

- Smart contracts were first theorized by *Nick Szabo* in the late 1990s,
  - 20 years before the true potential and benefits of them were truly appreciated
- Idea of smart contracts was implemented in a limited fashion in bitcoin in 2009,
  - Where bitcoin transactions can be used to transfer the value between users, over a peer-to-peer network where users do not necessarily trust each other and there is no need for a trusted intermediary.

Definition

- Smart contract is a **secure** and **unstoppable computer program** representing an **agreement** that is **automatically executable** and **enforceable**.
  - Computer program that is written in a language that a computer or target machine can understand.
  - Encompasses agreements between parties in the form of business logic.
  - Smart contracts are automatically executed when certain conditions are met.
  - Enforceable: All contractual terms are executed **as defined and expected**, even in the presence of adversaries.

Definition

- Smart Contracts are secure and unstoppable,
    - Computer programs are required to be designed in such a fashion that they are fault tolerant and executable in reasonable amount of time.
    - Program should be able to execute and maintain a healthy internal state, even if external factors are unfavorable
- it will provide greater benefits in the long run if security and unstoppable properties are included in the smart contract.
- Certain inputs that need to be provided by people can and should also be automated via the use of Oracles.

Definition

- Smart contracts usually operate by managing their internal state using a state machine model.
- Smart contract might be more acceptable in legal situations
  - If smart contract code readable not only by machines but also by people
  - If humans and machines can both understand the code written in a smart contract
- Smart contracts are inherently required to be deterministic in nature.
  - This property will allow a smart contract to be run by any node on a network and achieve the same result

Properties

- Smart contract has the following four properties:
  - Automatically executable
  - Enforceable
  - Semantically sound
  - Secure and unstoppable.
- The first two properties are required as a minimum
- Other two properties may not be required or implementable in certain scenarios and can be relaxed.

Ricardian contracts

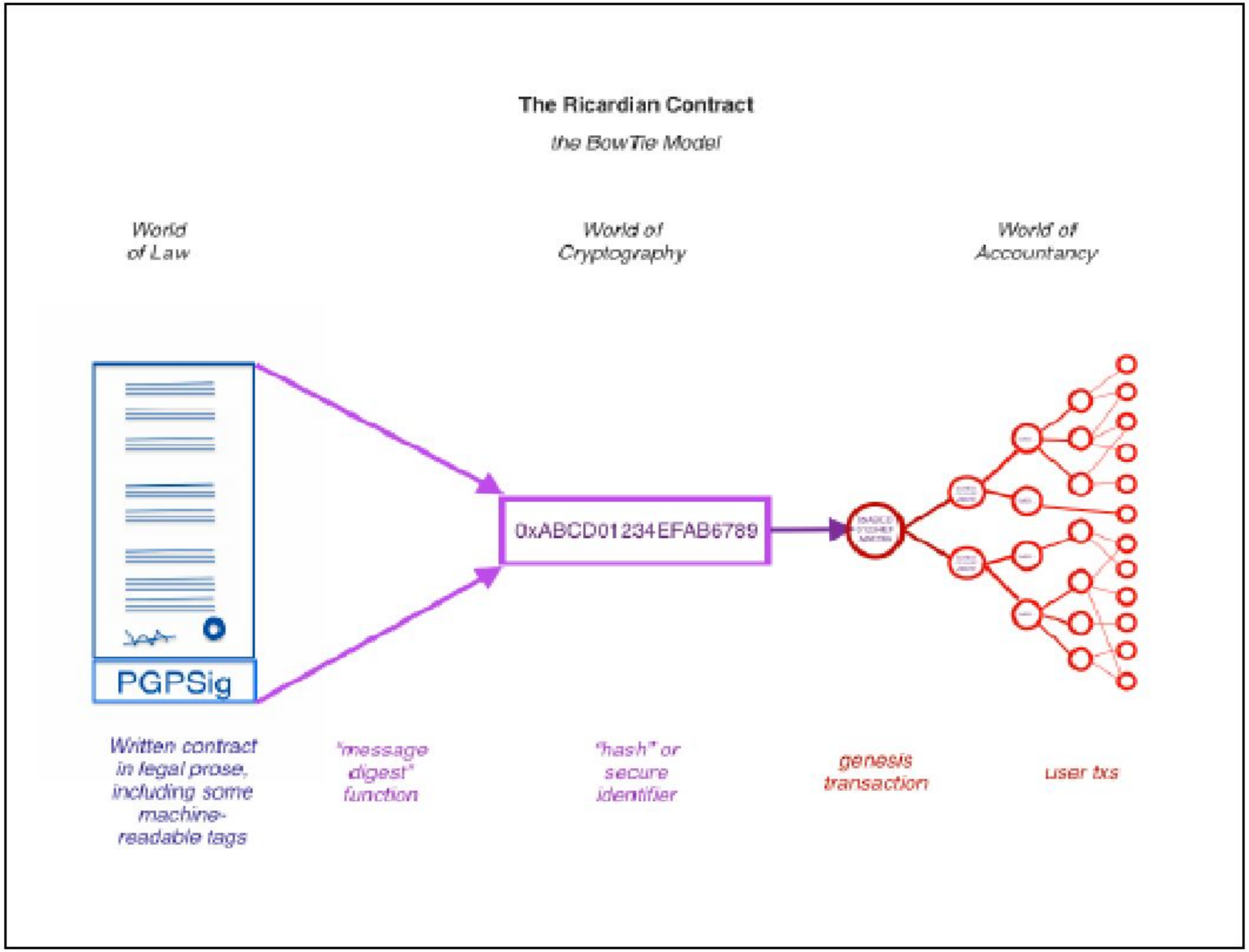
- Ricardian contracts were originally proposed in the *Financial Cryptography in 7 Layers* paper by *Ian Grigg* in late 1990s.
  - Contracts were used initially in a **bond trading** and **payment system** called **Ricardo**.
- key idea is to **write a document** which is **understandable** and **acceptable** by both a court of law and computer software.
- Ricardian contracts address the challenge of issuance of value over the Internet.
  - It identifies the **issuer**
  - captures all the **terms** and **clauses** of the contract in a document in order to make it acceptable as a legally binding contract.

Ricardian contracts

- Ricardian contract is a document that has several of the following properties:
  - Contract offered by an issuer to holders
  - Valuable right held by holders, and managed by the issuer
  - Easily readable by people (like a contract on paper)
  - Readable by programs (parseable, like a database)
  - Digitally signed
  - Carries the keys and server information
  - Allied with a unique and secure identifier

Ricardian contracts

- Contracts are implemented by producing a single document that contains the
  - Terms of the contract in **legal prose**
  - Required machine-readable tags.
- Document is **digitally signed by the issuer** using their private key.
- Document is then **hashed using a message digest function** to produce a hash by which the document can be identified.
- Hash is then further used and signed by parties during the performance of the contract in order to link each transaction, with the identifier hash thus serving as evidence of intent.
- This is called as ***bowtie* model.**


The Ricardian Contract
the BowTie Model
World of Law
World of Cryptography
World of Accountancy
PGPSig
0xABCD01234EFAB6789
Written contract in legal prose, including some machine-readable tags
"message digest" function
"hash" or secure identifier
genesis transaction
user txs

[illegible]

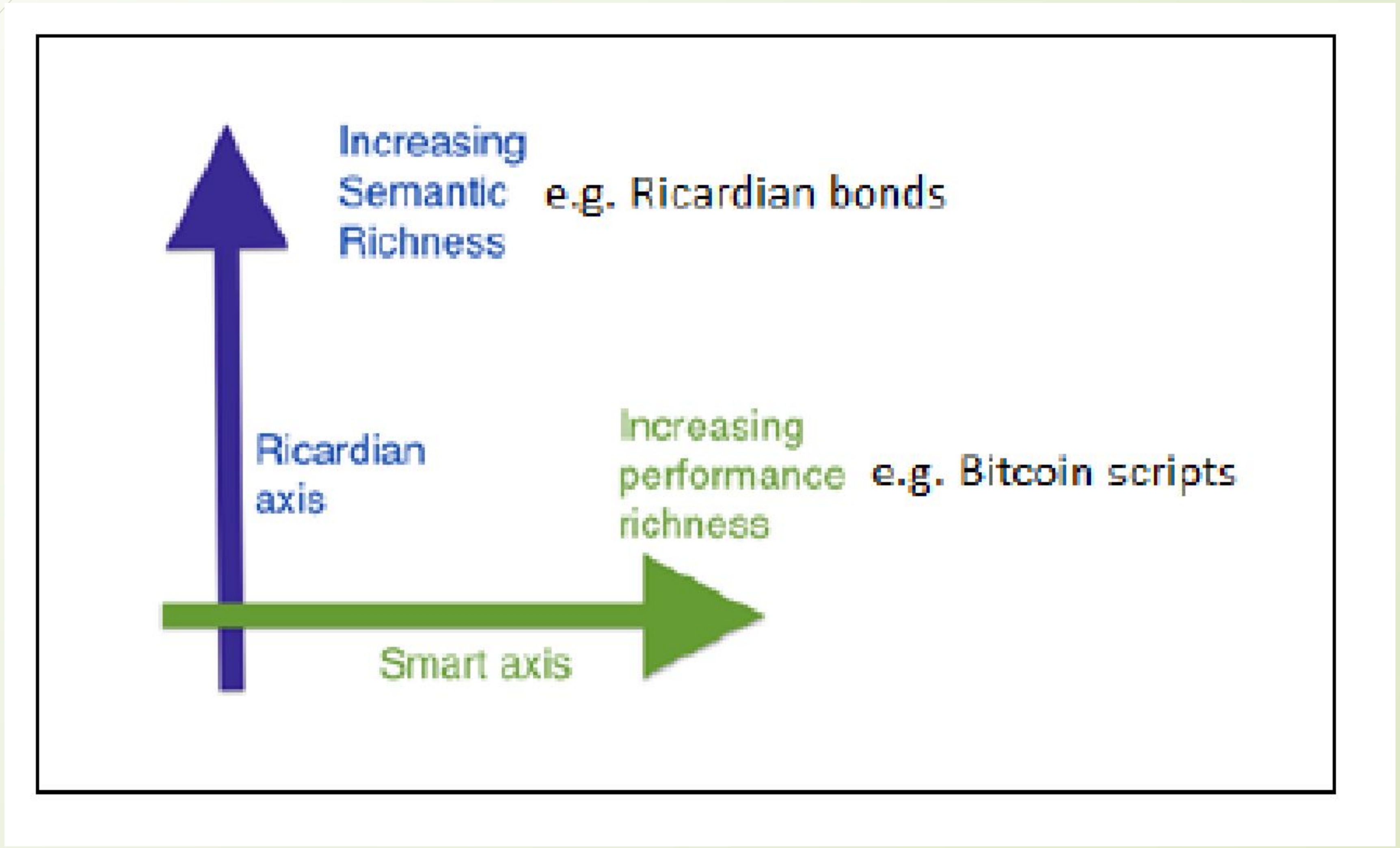
Ricardian contracts

- Ricardian contract is different from a smart contract
  - **Smart contract does not include any contractual document** and is focused purely on the execution of the contract.
  - **Ricardian contract is more concerned with the semantic richness and production of a document** that contains contractual **legal prose.**
- Semantics of a contract can be divided into two types:
  - Operational semantics
    - Defines the actual execution, correctness and safety of the contract,
  - Denotational semantics.
    - Concerned with the real-world meaning of the full contract.

Ricardian contracts

- Some researchers have differentiated between smart contract code and smart legal contracts
  - Smart contract is only concerned with the execution of the contract
  - Smart legal contracts encompasses both the denotational and operational semantics of a legal agreement
- Bitcoin, a very simple implementation of a smart contract can be observed which is fully oriented towards the execution of the contract,
  - whereas a Ricardian contract is more geared towards producing a document that is understandable by humans, with some parts that a computer program can understand.

legal semantics vs operational performance



Ricardian contracts

- smart contract is made up to have both performance and semantics embedded together
- Ricardian contract can be represented as a tuple of three objects, namely
  - ***Prose***, ***parameters*** and ***code***.
- **Prose** represents the legal contract in regular language;
- **Code** represents the program that is a computer-understandable representation of legal prose;
- **Parameters** join the appropriate parts of the legal contract to the equivalent code.
- Ricardian contracts have been implemented in many systems, such as CommonAccord, OpenBazaar, OpenAssets, and Askemos.

Smart contract templates

- Smart contracts can be implemented for any industry
  - Most current use cases are 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.
- *CLACK*, a common language for augmented contract knowledge has been proposed and research has begun to develop the language.
  - 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.

Smart contract templates

- Contracts in the finance industry is not a new concept
- Various domain-specific language DSLs are already in use in the financial industry to provide specific language for a specific domain.
- Domain-specific 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
  - not used to build general purpose large application programs

Smart contract templates

- **Solidity** is one such language that has been introduced with Ethereum blockchain to write smart contracts.
- **Serpent** is another language that has been introduced with Ethereum even though it's not used as much as Solidity.
- This idea of domain-specific languages for smart contract programming can be further extended to a graphical domain-specific language, a smart contract modelling platform
  - where a domain expert (not a programmer) can use a graphical user interface and a canvas to define and draw the semantics and performance of a financial contract.

Smart contract templates

- Once the flow has been drawn and completed
  - It can be emulated first to test
  - Then to deploy from the same system to the target platform, which can be a blockchain.
- Research should also be conducted in the area of developing high level DSLs that can be used to programme a smart contract in a user friendly graphical user interface
  - thus allowing a non-programmer to design a smart contract.

Oracles

- Oracles are an important component of the smart contract ecosystem.
- Limitation with smart contracts is that they cannot access external data
  - for example, the stock price of a security that is required by the contract to release the dividend payments.
- Oracles can be used to provide external data to smart contracts.
- Oracle is an interface that delivers data from an external source to smart contracts.
- 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

- Oracles are also capable of digitally signing the data proving that the source of the data is authentic.
- Smart contracts can subscribe to the Oracles,
- Smart contracts can either pull the data, or Oracles can 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.
- It is necessary that Oracles are unable to change the data.
  - This validation can be provided by using various notary schemes,.

standard or simple Oracles

- Smart contract designer can accept data for an oracle that is provided by a large reputable trusted third party
  - but the issue of centralization still remains.
- These types of Oracles can be called **standard or simple Oracles**.

Decentralized Oracles

- Oracles can be built based on distributed mechanism.
- Oracles can themselves source data from another blockchain which is driven by distributed consensus
  - thus ensuring the authenticity of data.
- **For example**
  - Institution running their own private blockchain can publish their data feed via an Oracle that can then be consumed by other blockchains.

Hardware Oracles

- 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.
- Requires a mechanism in which hardware devices cannot be tampered with.
  - tamper-proof devices.

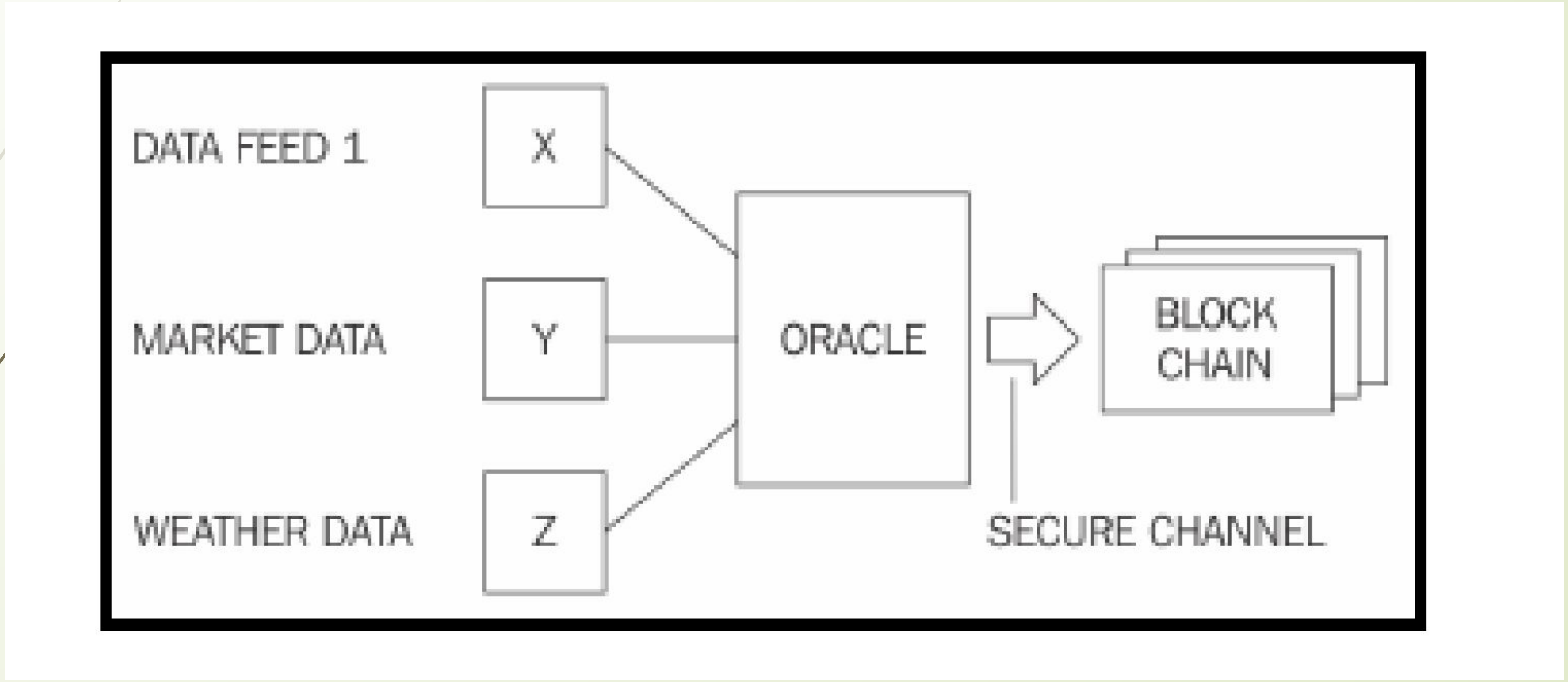
Oracles

- There are platforms available now to enable a smart contract to get external data using an Oracle.
- There are different methods used by an Oracle to write data into the blockchain depending on the type of blockchain used.
- For example in bitcoin blockchain, an oracle can write data to a specific transaction via an OP_RETURN Opcode,
  - smart contract can monitor that transaction and read the data.
- Various online services are available that provide oracle services such as
  - http://www.oraclize.it/
  - https://www.realitykeys.com

Oracles

- Another service at https://smartcontract.com/ is available
  - which provides external data and the ability to make payments using smart contracts.
- All these services enable the smart contract to get the data it needs to execute and make decisions.
- In order to prove the authenticity of the data retrieved by the Oracles from external sources,
  - mechanisms like TLSnotary can be used which produce proof of communication between the data source and the oracle.
  - This ensures that the data fed back to the smart contract is definitely retrieved from the source.

Smart contract Ecosystem



Smart Oracles

- Smart Oracles are basically 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

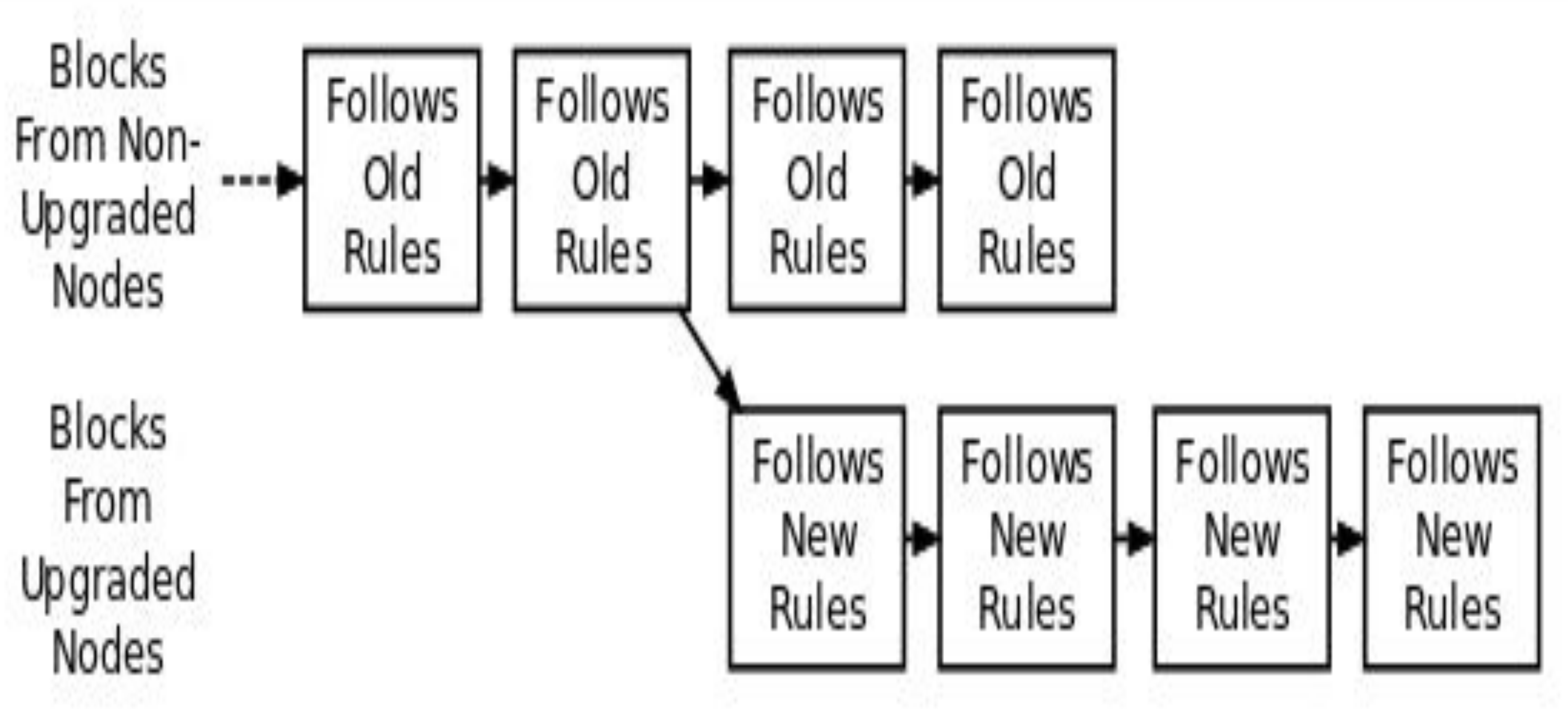
- Smart contracts may or may not be deployed on a blockchain
  - But it makes sense to deploy them on a blockchain due to the distributed consensus mechanism provided by blockchain.
- Ethereum is an example of a blockchain that natively supports the development and deployment of smart contracts.
- Smart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

Deploying smart contracts on a blockchain

- In bitcoin blockchain the lock_time field in the bitcoin transaction can be seen as an enabler of a basic version of a smart contract.
- lock_time field enables a transaction to be locked until a specified time or after a number of blocks,
  - thus enforcing a basic contract that a certain transaction can only be unlocked if certain conditions (elapsed time or number of blocks) is met
  - Can be viewed as basic smart contract.
- Bitcoin scripting language, though limited, can be used to construct basic smart contracts.

The DAO

- DAO is one of the highest crowdfunded projects, and started in April 2016.
  - set of smart contracts written in order to provide a platform for investment.
- Due to a bug in the code this was hacked in June 2016 and an equivalent of 50 million dollars was siphoned out of the DAO into another account.
  - Resulted in a hard fork on Ethereum in order to recover from the attack.
- It should be noted that the notion of *code is law*, or unstoppable smart contracts



A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain

The DAO

- Ethereum foundation was able to stop and change the execution of *The DAO* by introducing a hard fork.
  - hard fork goes against the true spirit of decentralization and the notion of *code is law*.
- Resistance against this hard fork
  - some miners who decided to keep mining on the original chain resulted in the creation of Ethereum Classic.
  - This is the original, non-forked Ethereum blockchain where *code is still law*.

References

- Imran Bashir. “Mastring BlockChain”, Packt
- Web Materials