smart contracts

principles and beyond

Blockchain and the Law. March 2022, seminar UCP

Giovanni Sileno, University of Amsterdam. g.sileno@uva.nl

smart contract

- term introduced by Nick szabo in the 1996

Smart Contracts: Building Blocks for Digital Markets

Glossary

(This is a partial rewrite of the article which appeared in Extropy #16)

Introduction

The contract, a set of promises agreed to in a "meeting of the minds", is the traditional way to formalize a relationship. While contracts are primarily used in business relationships (the focus of this article), they can also involve personal relationships such as marraiges. Contracts are also important in politics, not only because of "social contract" theories but also because contract enforcement has traditionally been considered a basic function of capitalist governments.

Whether enforced by a government, or otherwise, the contract is the basic building block of a free market economy. Over many centuries of cultural evolution has emerged both the concept of contract and principles related to it, encoded into common law. Algorithmic information theory suggests that such evolved structures are often prohibitively costly to recompute. If we started from scratch, using reason and experience, it could take many centuries to redevelop sophisticated ideas like property rights that make the modern free market work [Hayek].

smart contract

- term introduced by Nick szabo in the 1996

- ...well before the introduction of blockchain (2008)

Smart Contracts: Building Blocks for Digital Markets

Glossary

(This is a partial rewrite of the article which appeared in Extropy #16)

Introduction

The contract, a set of promises agreed to in a "meeting of the minds", is the traditional way to formalize a relationship. While contracts are primarily used in business relationships (the focus of this article), they can also involve personal relationships such as marraiges. Contracts are also important in politics, not only because of "social contract" theories but also because contract enforcement has traditionally been considered a basic function of capitalist governments.

Whether enforced by a government, or otherwise, the contract is the basic building block of a free market economy. Over many centuries of cultural evolution has emerged both the concept of contract and principles related to it, encoded into common law. Algorithmic information theory suggests that such evolved structures are often prohibitively costly to recompute. If we started from scratch, using reason and experience, it could take many centuries to redevelop sophisticated ideas like property rights that make the modern free market work [Hayek].

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of

Smart Contracts Glossary, Nick Szabo (1995)

“smart contract”

A set of promises,

> *including protocols within which the parties perform on the other promises.*

The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics,

> *thus these contracts are "smarter" than their paper-based ancestors.*

No use of artificial intelligence is implied.

"smart contract"

a bilateral contract is a formal agreement in which both parties exchange promises to perform

***A set of promises**,*

including protocols within which the parties perform on the other promises.

The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics,

thus these contracts are "smarter" than their paper-based ancestors.

No use of artificial intelligence is implied.

"smart contract"

A set of promises,

rules, procedures constraining the interaction

including ***protocols within which the parties perform*** *on the other promises.*

The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics,

thus these contracts are "smarter" than their paper-based ancestors.

No use of artificial intelligence is implied.

Smart Contracts Glossary, Nick Szabo (1995)

"smart contract"

A set of promises,

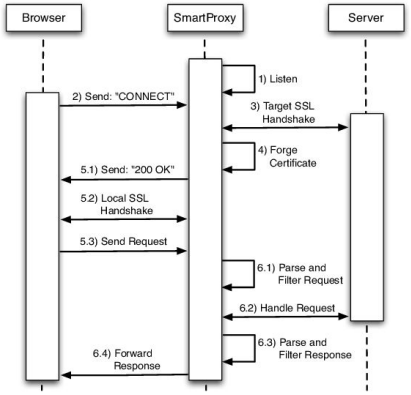
rules, procedures constraining the interaction

including protocols within which the parties perform on the other promis

*The **protocols** are usually **implemented with programs** on a computer network, or in other forms of digital electronics,*

thus these contracts are "smarter" than their paper-based ancestors.

No use of artificial intelligence is implied.



"smart contract"

A set of promises,

> *including protocols within which the parties perform on the other promises.*

The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics,

> *thus these contracts are **"smarter"** than their paper-based ancestors.* ???

No use of artificial intelligence is implied.

"smart contract"

A set of promises,

> *including protocols within which the parties perform on the other promises.*

The protocols are usually implemented with programs on a computer network, or in other forms of digital electronics,

> *thus these contracts are "smarter" than their paper-based ancestors.*

No use of artificial intelligence is implied.

then, smart in what sense?

- According to the vulgata:
  - as the contract will perform exactly as designed
  - it eliminates the need for “trust” amongst the parties
  - ...and traditional third-parties (lawyers, notaries, courts) whose activity is meant to increase the confidence in the contract

then, smart in what sense?

- According to the vulgata:
  - as the contract will perform exactly as designed
  - it eliminates the need for “trust” amongst the parties
  - ...and traditional third-parties (lawyers, notaries, courts) whose activity is meant to increase the confidence in the contract

- smart perhaps in the sense of *economically profitable*?

$$$

then, smart in what sense?

crucial critical point to all this enterprise!!!

- According to the vulgata:
  - as **the contract will perform exactly as designed**
  - it eliminates the need for “trust” amongst the parties
  - ...and traditional third-parties (lawyers, notaries, courts) whose activity is meant to increase the confidence in the contract

- smart perhaps in the sense of *economically profitable*?

then, smart in what sense?

crucial critical point to all this enterprise!!!

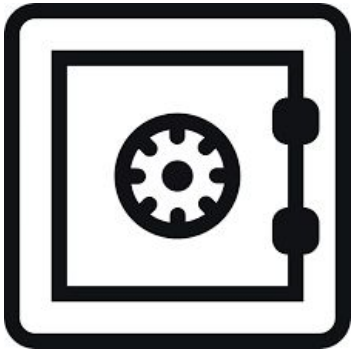
- According to the vulgata:
  - as **the contract will perform exactly as designed**
  - it eliminates the need for “trust” amongst the parties
  - ...and traditional third-parties (lawyers, notaries, courts) whose activity is meant to increase the confidence in the contract

- smart perhaps in the sense of *economically profitable*?

how? making breaches prohibitively expensive

$$$

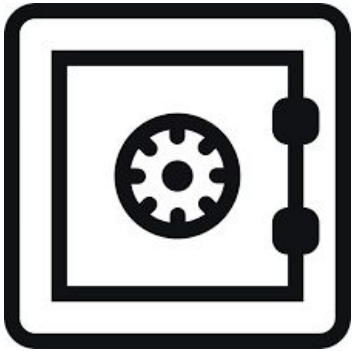
essential idea: technology as a vault



the *possibility* of breaching is
prohibitively expensive

***ex-ante* enforcement**

essential idea: technology as a vault



the *possibility* of breaching is prohibitively expensive

***ex-ante* enforcement**

tokens!

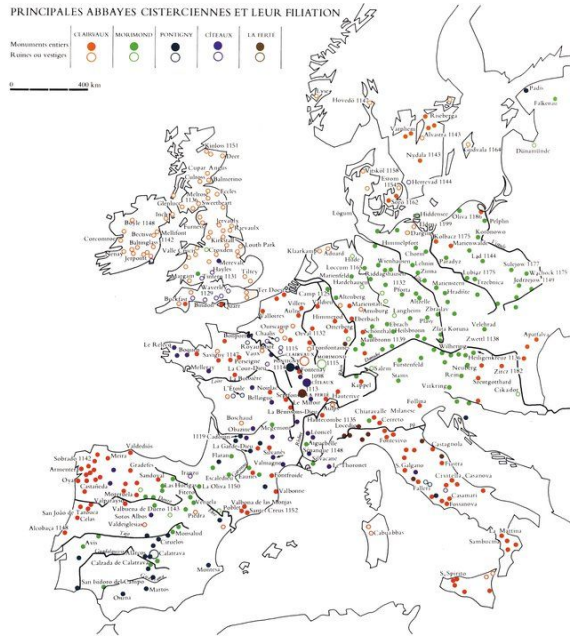
- What to protect?
  - (valuable?) digital assets
  - “contracts” (reified as assets)
  - automated performance

in terms of medieval technology

- How to protect (the content of) a book from invasions, wars, and plagues?

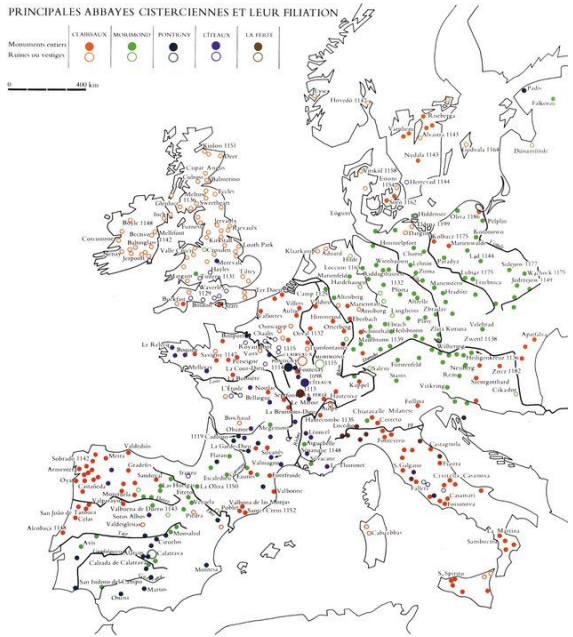
in terms of medieval technology

- How to protect (the content of) a book from invasions, wars, and plagues?
- Copy it, and distribute the copies through a network of monasteries!



in terms of medieval technology

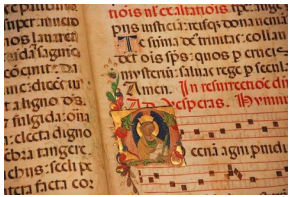
- How to protect (the content of) a book from invasions, wars, and plagues?
- Copy it, and distribute the copies through a network of monasteries!

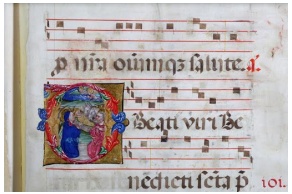


If a node is destroyed, a copy is still maintained in all others!!!

in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical



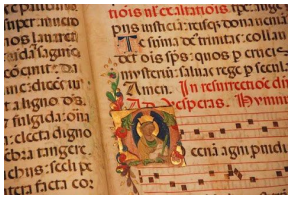


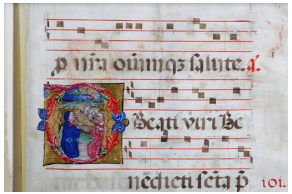
in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical

technological improvement:

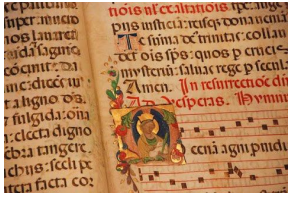
introduce **error checking** machinery, typically via additional content used to **check integrity**

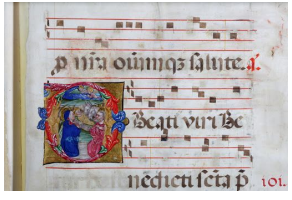




in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical

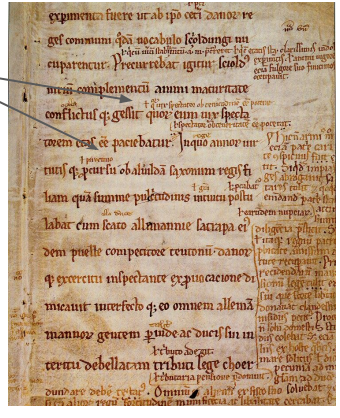




technological improvement:

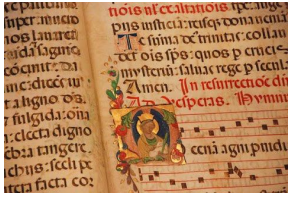
introduce **error checking** machinery, typically via additional content used to **check integrity**

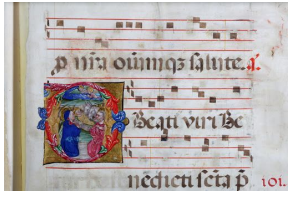
e.g. *glosses* rephrasing the meaning of some word



in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical

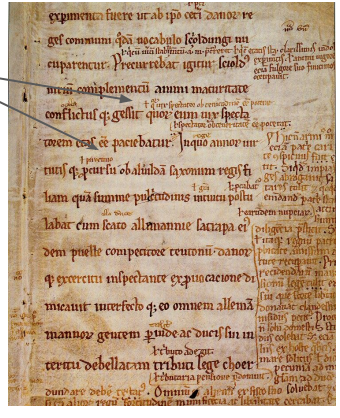




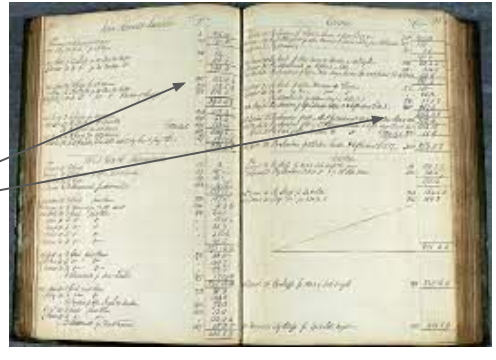
technological improvement:

introduce **error checking** machinery, typically via additional content used to **check integrity**

e.g. *glosses* rephrasing the meaning of some word

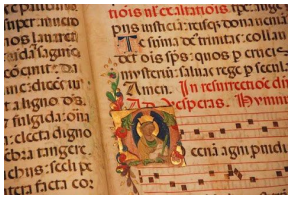


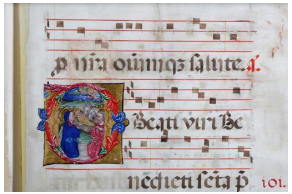
double entry bookkeeping (Florence, Venice ~1300)



in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical

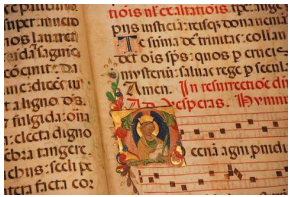


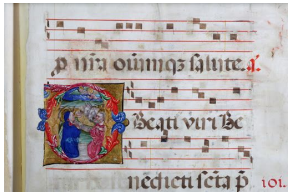



- Introduce **conflict resolution strategies**
  - philology provides several methods to “reconstruct” the original source, e.g. majority of sources, comparative analysis, provenance history

in terms of medieval technology

- Typically copying bring errors, exemplars of the book are never identical



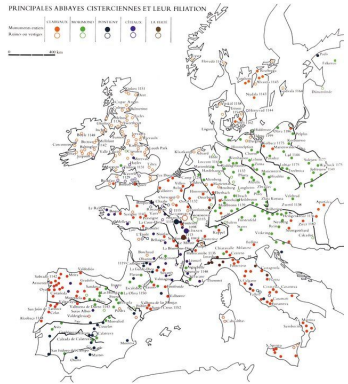



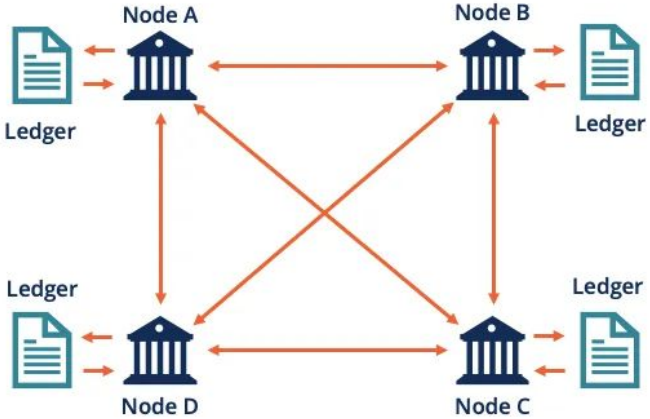
“consensus” protocol

- Introduce **conflict resolution strategies**
  - philology provides several methods to “reconstruct” the original source, e.g. *majority of sources*, comparative analysis, provenance history

“blockchain” =

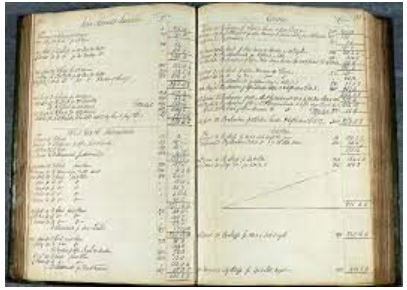
- **distributed** ledger + consensus protocol



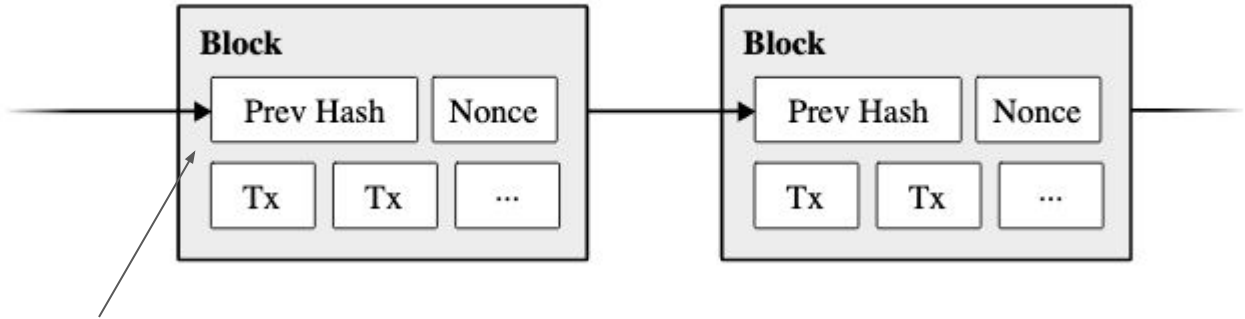


the same ledger is copied across the network

"blockchain" =



- distributed ledger + consensus protocol

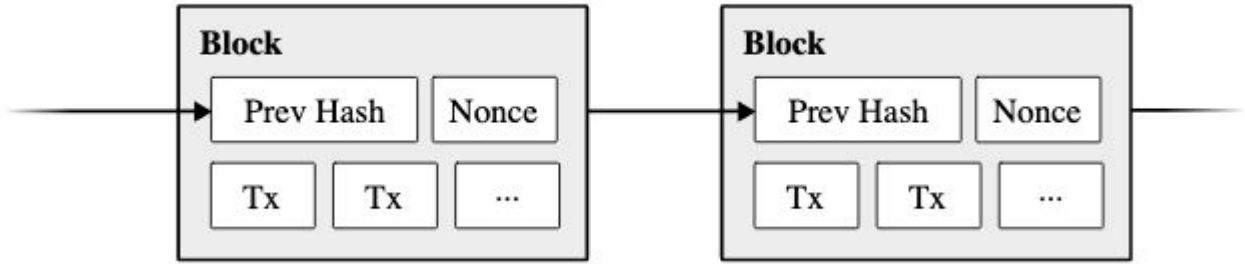


the ledger embeds machinery to test its integrity

compressed information (a sort of *fingerprint*) of the previous block, if the previous block is modified its hash is not the same

"blockchain" =

- distributed ledger + consensus protocol



checks for integrity are performed across the network, no central authority

techniques: **Proof of Work (PoW), Proof of Stake (PoS), etc.**

“smart contract”

= code running on blockchain

“smart contract”

= code running on blockchain

in most cases Ethereum

which code?

“smart contract”
= code running on blockchain

in most cases Ethereum

which code?

source code

```
pragma solidity >=0.4.22 <0.6.0;

contract Ballot {
    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    struct Proposal {
        bytes32 name;
        uint voteCount;
    }

    address public chairperson;

    [...]
```

“human-readable” instructions

[developer’s view]

“smart contract”
= code running on blockchain

which code?

in most cases Ethereum

source code

```
pragma solidity >=0.4.22 <0.6.0;

contract Ballot {
    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    struct Proposal {
        bytes32 name;
        uint voteCount;
    }

    address public chairperson;

    [...]
```

compilation

byte-code or machine code

```
1  6060604052600436106100576d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

“human-readable” instructions
[developer’s view]

low-level instructions
[user’s view]

“smart contract”
= code running on blockchain

which code?

in most cases Ethereum

libraries in byte-code

source code

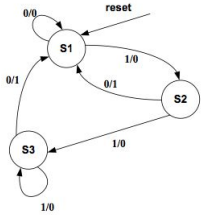
```
pragma solidity >=0.4.22 <0.6.0;

contract Ballot {
    struct Voter {
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    struct Proposal {
        bytes32 name;
        uint voteCount;
    }

    address public chairperson;

    [...]
```

compilation

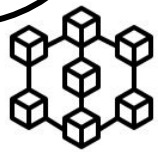
byte-code or machine code

```
1  6060604052600436106100d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

deployment



virtual machine on blockchain



“human-readable” instructions
[developer’s view]

low-level instructions
[user’s view]

“**smart contract**” most used meaning

= immutable low-level instructions
cloned on each node
running in a decentralized fashion

what are low-level instructions?

byte-code
or machine code

```
1  6060604052600436106100d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

- Individual primitive operations to be run on the virtual machine

eg. move a value from memory into a register, move a value from register to memory, perform operations between register and put value in register….

what are low-level instructions?

byte-code
or machine code

```
1  6060604052600436106100d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

- Individual primitive operations to be run on the virtual machine

eg. move a value from memory into a register, move a value from register to memory, perform operations between register and put value in register….

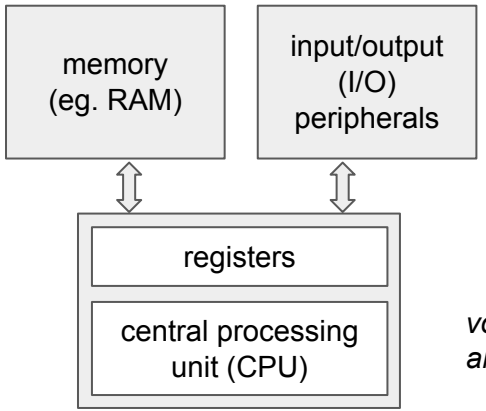
what are low-level instructions?

byte-code
or machine code

```
1  6060604052600436106100d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

- Individual primitive operations to be run on the virtual machine

eg. move a value from memory into a register, move a value from register to memory, perform operations between register and put value in register….



von Neumann architecture

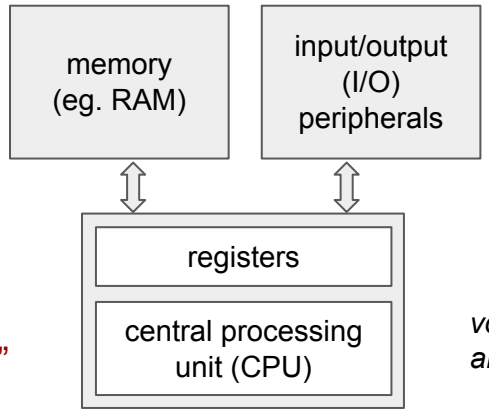
what are low-level instructions?

byte-code
or machine code

```
1  6060604052600436106100d576000357
2  000000: PUSH1 0x60
3  000002: PUSH1 0x40
4  000004: MSTORE
5  000005: PUSH1 0x04
6  000007: CALLDATASIZE
7  000008: LT
8  000009: PUSH2 0x006d
9  000012: JUMPI
10 000013: PUSH1 0x00
11 000015: CALLDATALOAD
12 000016: PUSH29 0x0100000000000000
13 000046: SWAP1
14 000047: DIV
15 000048: PUSH4 0xffffffff
16 000053: AND
17 000054: DUP1
18 000055: PUSH4 0x362186ed
19 000060: EQ
20 000061: PUSH2 0x008d
21 000064: JUMPI
22 000065: DUP1
```

- Individual primitive operations to be run on the virtual machine

eg. move a value from memory into a register, move a value from register to memory, perform operations between register and put value in register….



essentially,
computational “logistics”

von Neumann architecture

and “higher-level” instructions?

(source code)

and “higher-level” instructions?

(source code)

Solidity by Example

version 0.8.10

English Auction

English auction for NFT.

Auction

1. Seller of NFT deploys this contract.
2. Auction lasts for 7 days.
3. Participants can bid by depositing ETH greater than the current highest bidder.
4. All bidders can withdraw their bid if it is not the current highest bid.

After the auction

1. Highest bidder becomes the new owner of NFT.
2. The seller receives the highest bid of ETH.

https://solidity-by-example.org/app/english-auction/

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

interface IERC721 {
    function safeTransferFrom(
        address from,
        address to,
        uint tokenId
    ) external;

    function transferFrom(
        address,
        address,
        uint
    ) external;
}

contract EnglishAuction {
    event Start();
    event Bid(address indexed sender, uint amount);
    event Withdraw(address indexed bidder, uint amount);
    event End(address winner, uint amount);

    IERC721 public nft;
    uint public nftId;

    address payable public seller;
    uint public endAt;
    bool public started;
    bool public ended;

    address public highestBidder;
    uint public highestBid;
    mapping(address => uint) public bids;
```

and “higher-level” instructions?

(source code)

Solidity by Example

version 0.8.10

English Auction

English auction for NFT.

Auction

1. Seller of NFT deploys this contract.
2. Auction lasts for 7 days.
3. Participants can bid by depositing ETH greater than the current highest bidder.
4. All bidders can withdraw their bid if it is not the current highest bid.

After the auction

1. Highest bidder becomes the new owner of NFT.
2. The seller receives the highest bid of ETH.

https://solidity-by-example.org/app/english-auction/

```
constructor(
    address _nft,
    uint _nftId,
    uint _startingBid
) {
    nft = IERC721(_nft);
    nftId = _nftId;

    seller = payable(msg.sender);
    highestBid = _startingBid;
}

function start() external {
    require(!started, "started");
    require(msg.sender == seller, "not seller");

    nft.transferFrom(msg.sender, address(this), nftId);
    started = true;
    endAt = block.timestamp + 7 days;

    emit Start();
}

function bid() external payable {
    require(started, "not started");
    require(block.timestamp < endAt, "ended");
    require(msg.value > highestBid, "value < highest");

    if (highestBidder != address(0)) {
        bids[highestBidder] += highestBid;
    }

    highestBidder = msg.sender;
    highestBid = msg.value;

    emit Bid(msg.sender, msg.value);
}
```

and “higher-level” instructions?

(source code)

Solidity by Example

version 0.8.10

English Auction

English auction for NFT.

Auction

1. Seller of NFT deploys this contract.
2. Auction lasts for 7 days.
3. Participants can bid by depositing ETH greater than the current highest bidder.
4. All bidders can withdraw their bid if it is not the current highest bid.

After the auction

1. Highest bidder becomes the new owner of NFT.
2. The seller receives the highest bid of ETH.

https://solidity-by-example.org/app/english-auction/

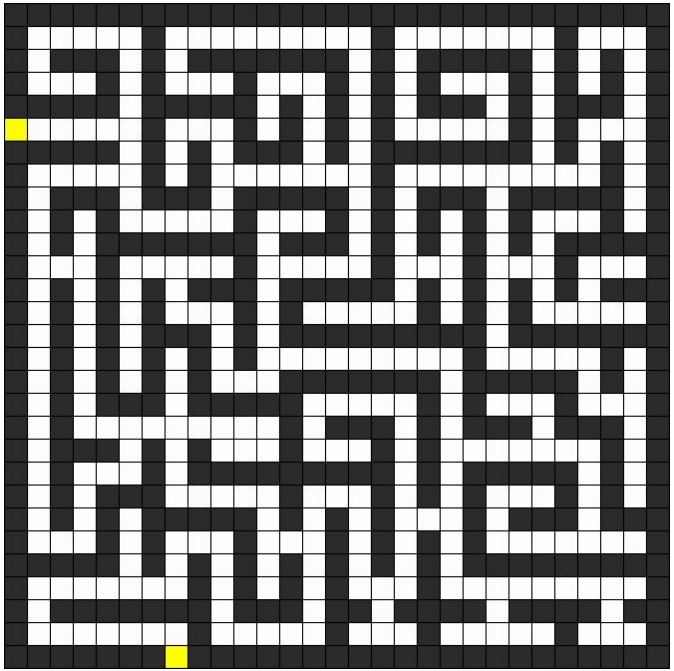
```
    function withdraw() external {
        uint bal = bids[msg.sender];
        bids[msg.sender] = 0;
        payable(msg.sender).transfer(bal);

        emit Withdraw(msg.sender, bal);
    }

    function end() external {
        require(started, "not started");
        require(block.timestamp >= endAt, "not ended");
        require(!ended, "ended");

        ended = true;
        if (highestBidder != address(0)) {
            nft.safeTransferFrom(address(this), highestBidder, nftId);
            seller.transfer(highestBid);
        } else {
            nft.safeTransferFrom(address(this), seller, nftId);
        }

        emit End(highestBidder, highestBid);
    }
}
```

not very different from computational “logistics”

difference between “imperative” and “declarative” programming languages

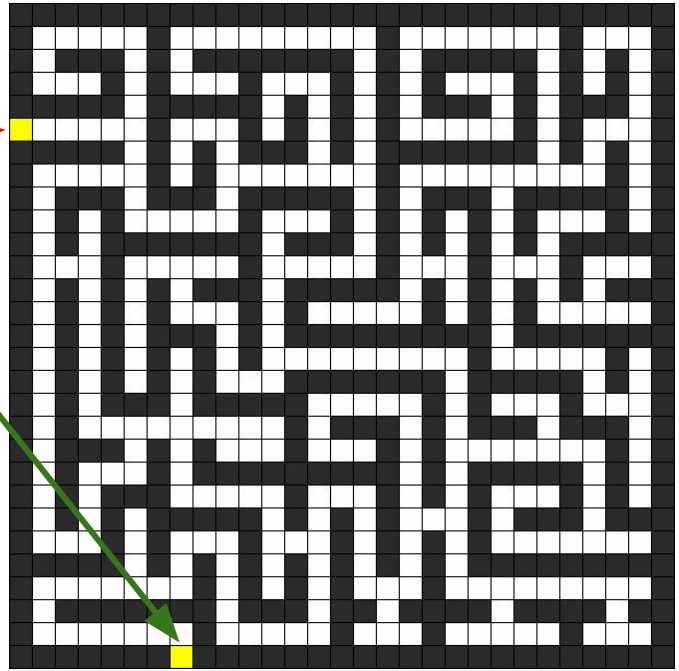
difference between “imperative” and “declarative” programming languages

We have a labyrinth.



difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**

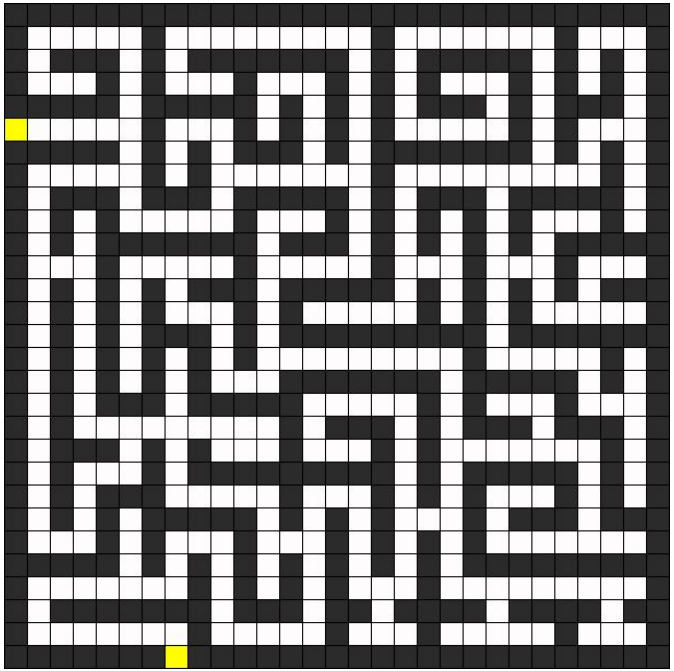


difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**

I can:

- write down the instructions to perform (*imperative programming*)

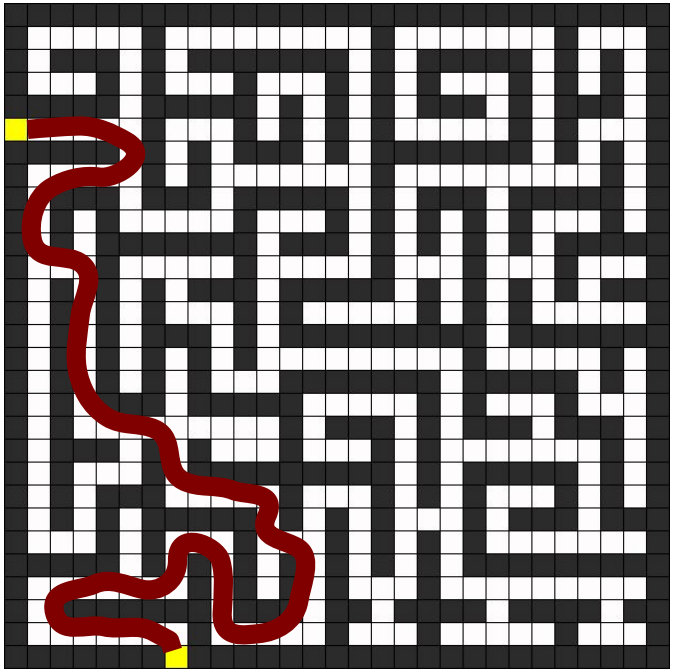


difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**

I can:

- write down the instructions to perform (*imperative programming*)

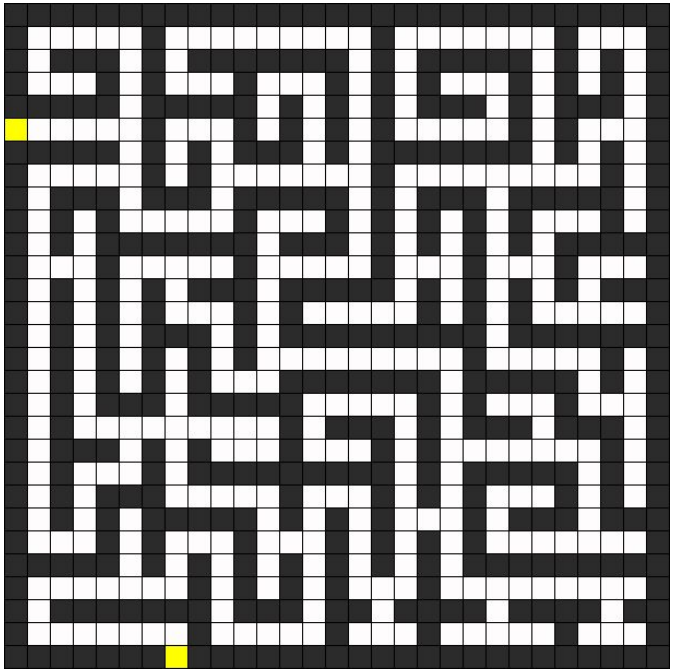


difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**.

I can:

- write down the instructions to perform (*imperative programming*)
- write down the initial point, the exit point, the labyrinth walls, and **let the computer to find the way** (*declarative programming*)



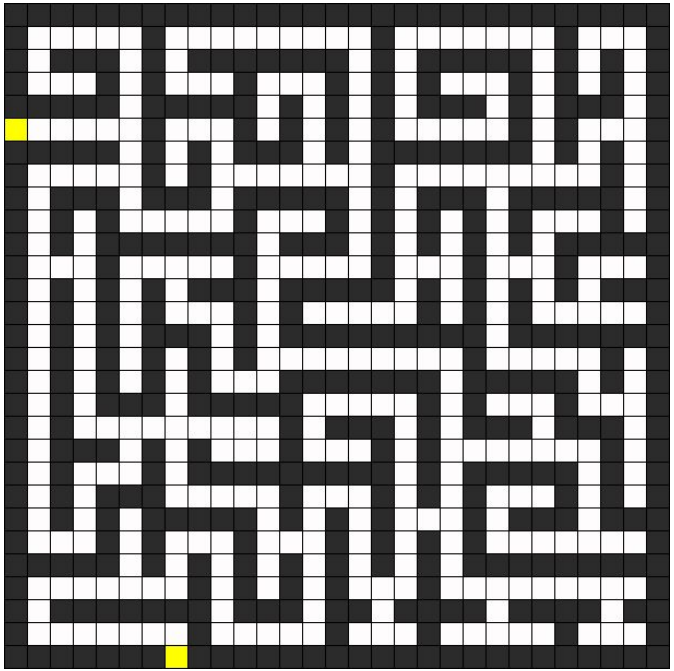
difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**.

I can:

- write down the instructions to perform (*imperative programming*)
- write down the initial point, the exit point, the labyrinth walls, and **let the computer to find the way** (*declarative programming*)

  via some problem-solving strategy, eg. trial and error with backtracking



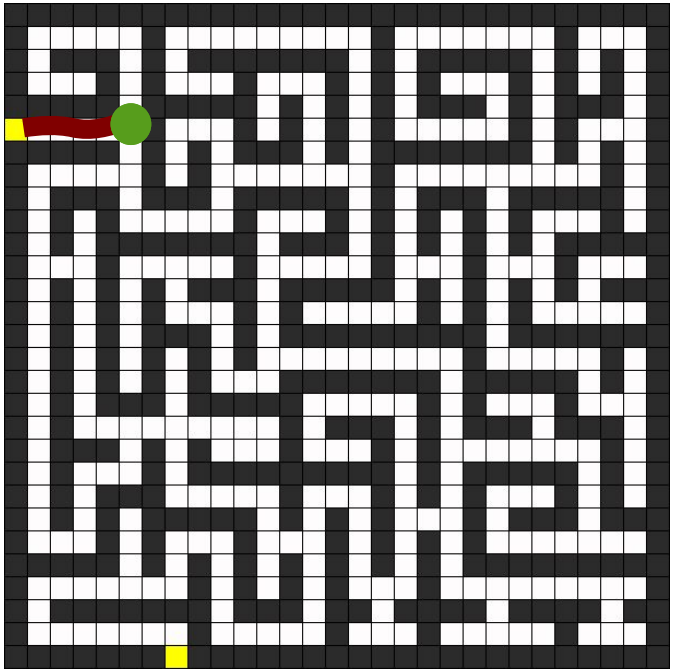
difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**.

I can:

- write down the instructions to perform (*imperative programming*)
- write down the initial point, the exit point, the labyrinth walls, and **let the computer to find the way** (*declarative programming*)

  via some problem-solving strategy, eg. trial and error with backtracking



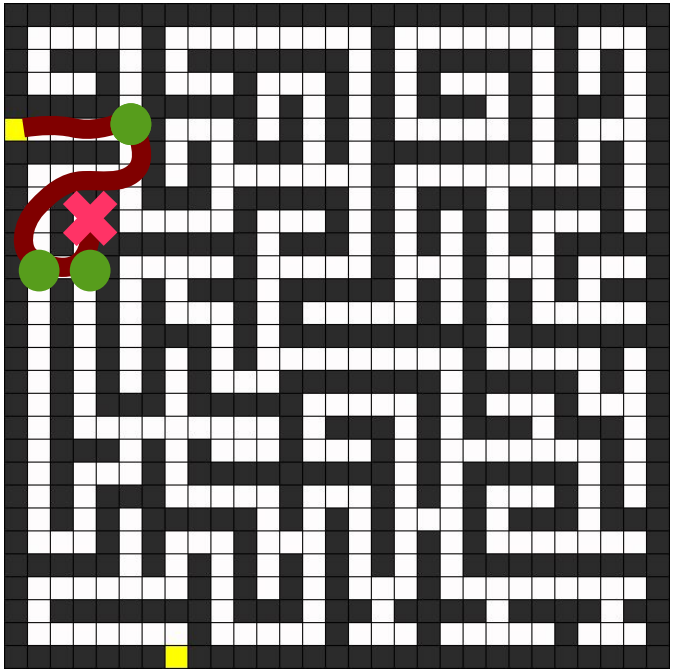
difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**.

I can:

- write down the instructions to perform (*imperative programming*)
- write down the initial point, the exit point, the labyrinth walls, and **let the computer to find the way** (*declarative programming*)

  via some problem-solving strategy, eg. trial and error with backtracking



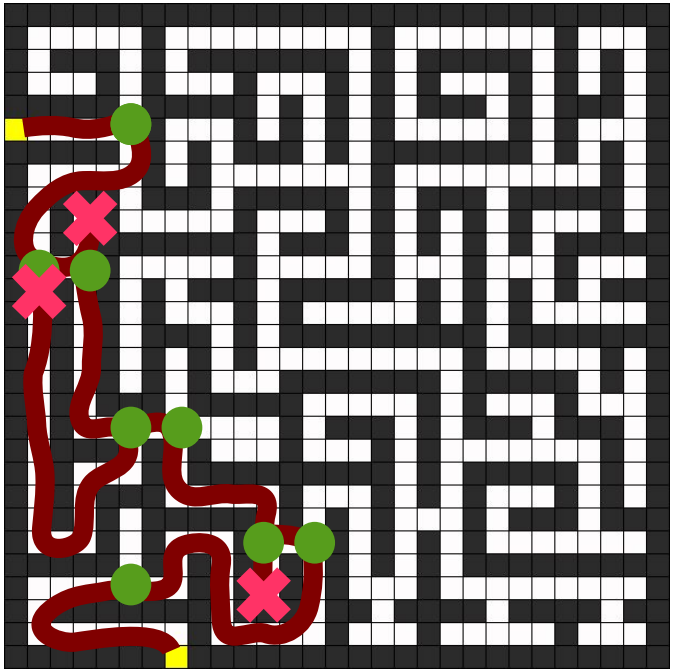
difference between “imperative” and “declarative” programming languages

We have a labyrinth. We know **entry** and **exit**.

I can:

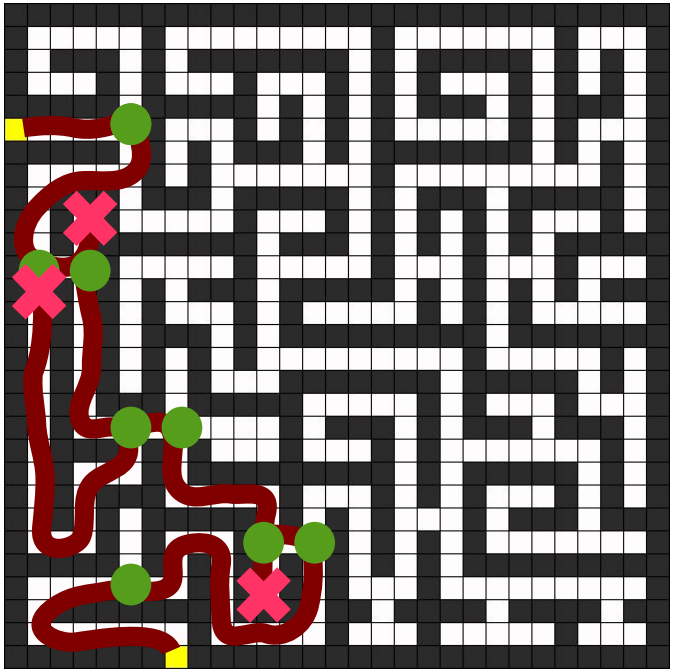
- write down the instructions to perform (*imperative programming*)
- write down the initial point, the exit point, the labyrinth walls, and **let the computer to find the way** (*declarative programming*)

  via some problem-solving strategy, eg. trial and error with backtracking



difference between “imperative” and “declarative” programming languages

If we accept the “labyrinth” may change, we need declarative forms of programming.

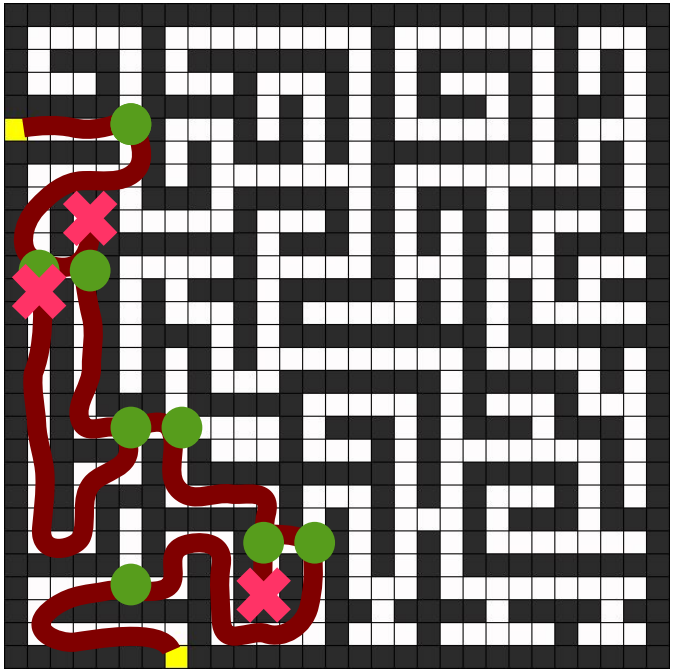


difference between “imperative” and “declarative” programming languages

If we accept the “labyrinth” may change, we need declarative forms of programming.

Why they haven’t been considered so far?

…generally computationally more expensive



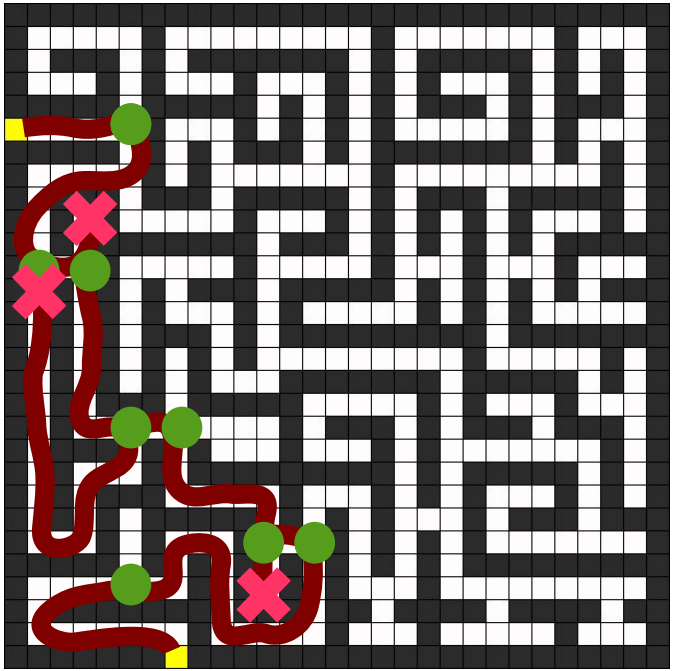
difference between “imperative” and “declarative” programming languages

If we accept the “labyrinth” may change, we need declarative forms of programming.

Why they haven’t been considered so far?

…generally computationally more expensive

Yet, higher-abstraction constructs are more intelligible to humans.



“smart contract” most used meaning

= immutable low-level instructions
cloned on each node
running in a decentralized fashion

- What are the **differences** with respect to other programs?

“smart contract” most used meaning

= immutable low-level instructions
cloned on each node
running in a decentralized fashion

- What are the **gaps** with usual contracts?

“smart contract” most used meaning

= immutable low-level instructions
cloned on each node
running in a decentralized fashion

- What are the **gaps** with usual contracts?

a bilateral contract is a formal agreement in which both parties exchange promises to perform

contracts vs “smart contracts”

- **Readability**: parties should understand what the agreement is about

contracts vs “smart contracts”

- **Readability**: parties should understand what the agreement is about
- **Control**: parties should maintain autonomy on performance. e.g. non-foreseeable conditions may cause justifiable release of duty, that can be assessed only *ex-post*

contracts vs “smart contracts”

- **Readability**: parties should understand what the agreement is about
- **Control**: parties should maintain autonomy on performance. e.g. non-foreseeable conditions may cause justifiable release of duty, that can be assessed only *ex-post*
- **Amendment**, **Delegation**, **Mandate,** etc. are all common constructs applicable on contracts (unless explicitly disabled), but not in smart contracts.

contracts vs “smart contracts”

- **Readability**: parties should understand what the agreement is about
- **Control**: parties should maintain autonomy on performance. e.g. non-foreseeable conditions may cause justifiable release of duty, that can be assessed only *ex-post*
- **Amendment**, **Delegation**, **Mandate,** etc. are all common constructs applicable on contracts (unless explicitly disabled), but not in smart contracts.
- **Regulation:** there is not an equivalent of contract law or private law regulations (a sort of meta-contracts) in smart contracts

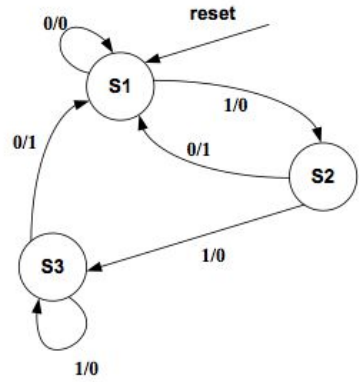
contracts vs "smart contracts"

- **Readability**: parties should understand what the agreement is about
- **Control**: parties should maintain autonomy on performance. e.g. non-foreseeable conditions may cause justifiable release of duty, that can be assessed only *ex-post*
- **Amendment**, **Delegation**, **Mandate,** etc. are all common constructs applicable on contracts (unless explicitly disabled), but not in smart contracts.
- **Regulation:** there is not an equivalent of contract law or private law regulations (a sort of meta-contracts) in smart contracts
- **Informational model (e.g. normative primitives):** contemporary smart contracts allow to specify *imperative instructions* driving performance mapping only to positive duties. What about prohibitions? What about permissions, legal competences?

cf. normative systems, computational theory of law

contracts vs “smart contracts”

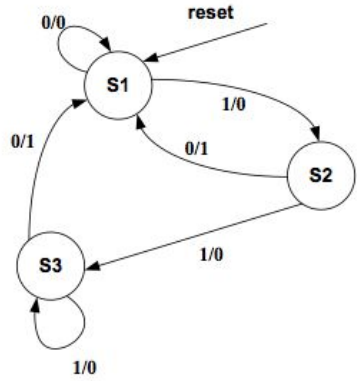
- Readability
- Control and *ex-post* enforcement
- Amendment, Delegation, Mandate
- Regulation
- Informational model



In most practical applications smart contracts depend on offline events, triggered by oracles. The environment “sets” the pace of execution.

contracts vs "smart contracts"

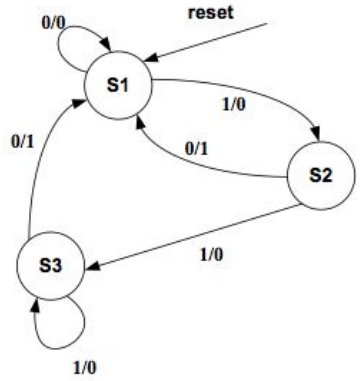
- Readability
- Control and *ex-post* enforcement
- Amendment, Delegation, Mandate
- Regulation
- Informational model



However, the **"dynamics" of contracts** depends not only on performance-related elements, but also **on contextual factors** (legal, social, physical) that ***modify the contract semantics*** itself.

contracts vs "smart contracts"

- Readability
- Control and *ex-post* enforcement
- Amendment, Delegation, Mandate
- Regulation
- Informational model

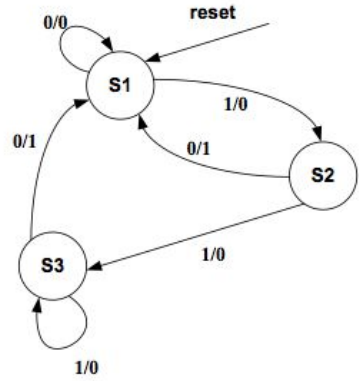


However, the **"dynamics" of contracts** depends not only on performance-related elements, but also **on contextual factors** (legal, social, physical) that ***modify the contract semantics*** itself.

The question is: **Can any socio-institutional systems (legal or not) be sustainable without these capacities?**

contracts vs "smart contracts" vs digital enforceable contracts?

- Readability
- Control and *ex-post* enforcement
- Amendment, Delegation, Mandate
- Regulation
- Informational model



However, the **"dynamics" of contracts** depends not only on performance-related elements, but also **on contextual factors** (legal, social, physical) that ***modify the contract semantics*** itself.

The question is: **Can any socio-institutional systems (legal or not) be sustainable without these capacities?**