KU LEUVEN

DistriNet

Trust through replication: Research challenges in decentralized applications

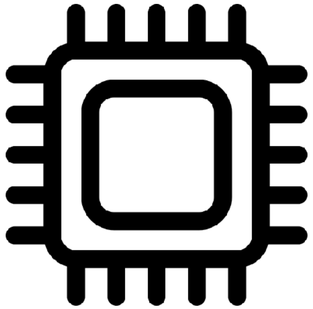
Tom Van Cutsem
DistriNet, KU Leuven
June 15, 2023

tvcutsem.github.io
be.linkedin.com/in/tomvc
github.com/tvcutsem
twitter.com/tvcutsem
@tvcutsem@techhub.social

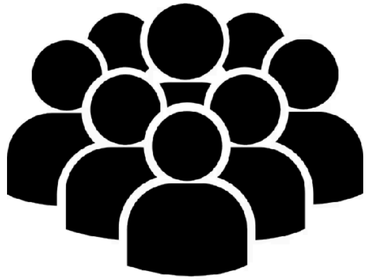
Trust through replication?

Trust rooted in
math

Trust rooted in
hardware



Trust rooted in
consensus



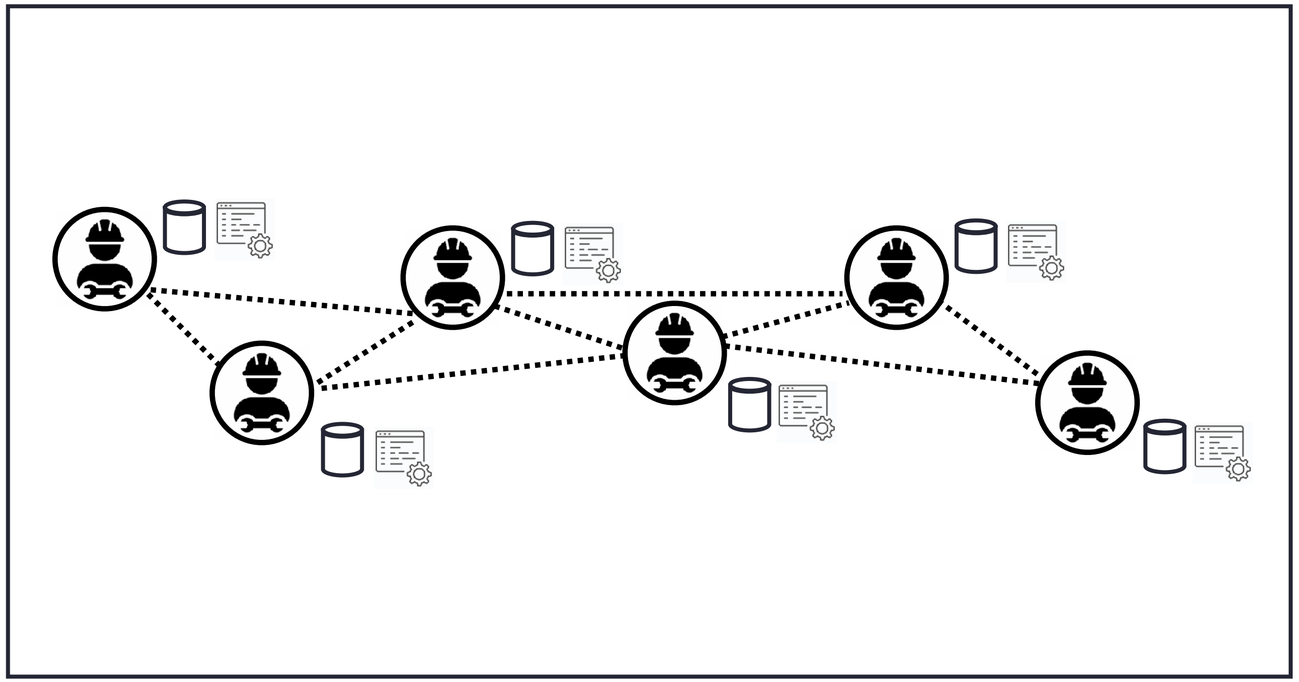
Web3: decentralized applications and services

Decentralized…

Payments	Bitcoin
Names	ENS
Identity	Civic
Networks	Helium
Storage	Filecoin
Computers	Ethereum

…

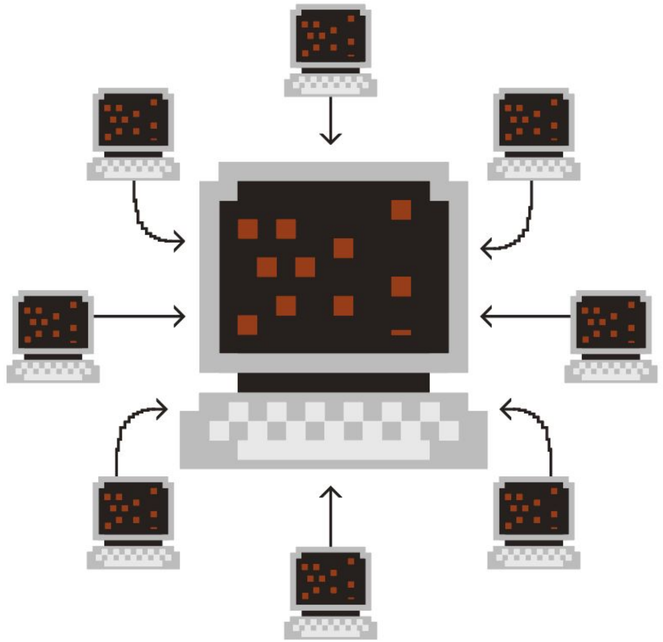
Decentralized services …



… operated by multiple independent parties

Web3 counterbalances the trend toward internet consolidation

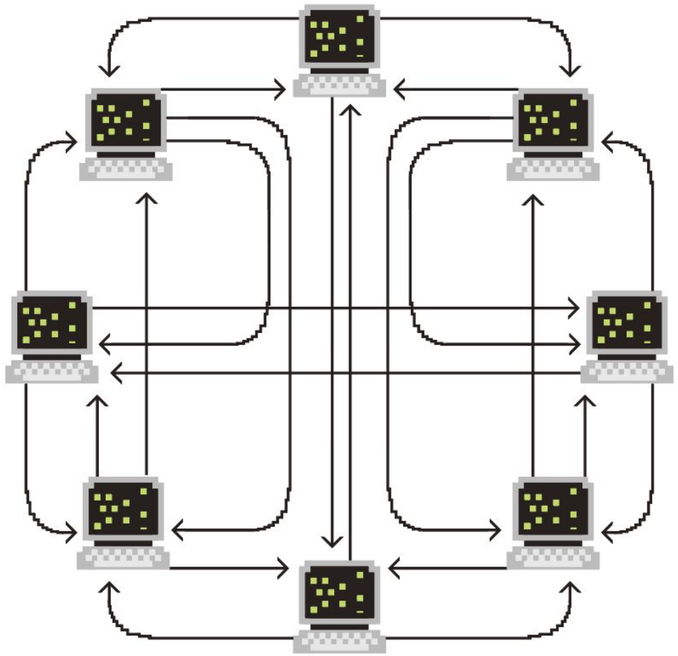
Big Tech



3 companies now generate a third of all global web traffic.

5 companies represent 50% of the Nasdaq 100's total market cap, up from 25% a decade ago.

web3

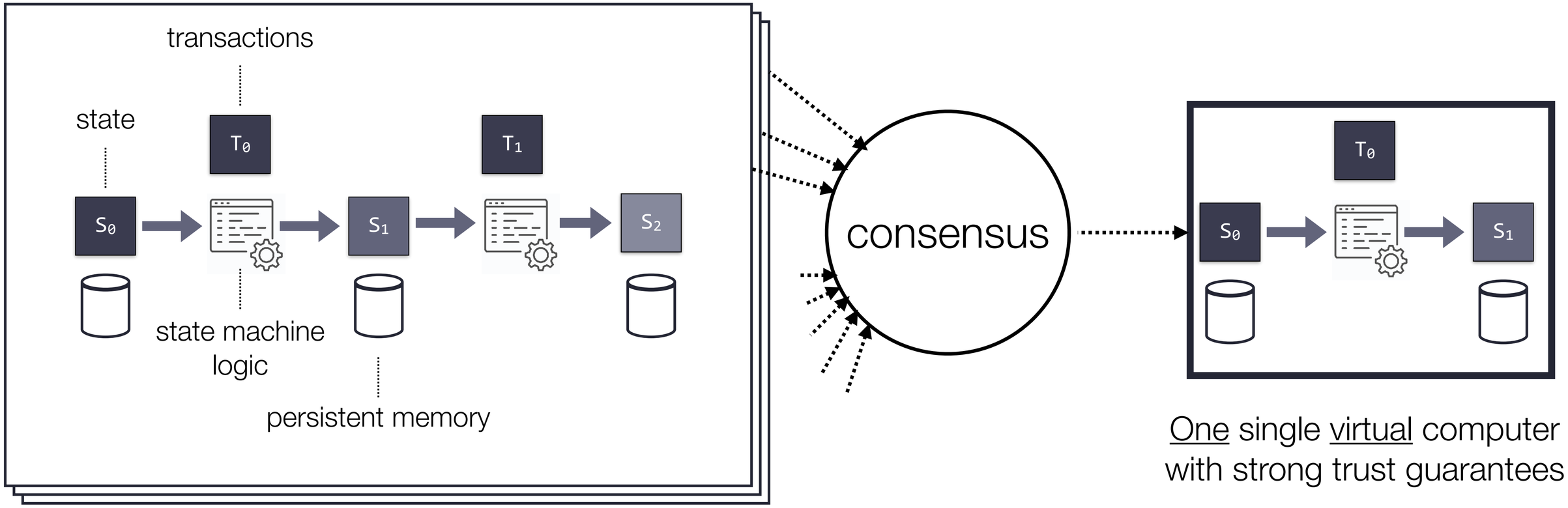


Blockchains transfer control from *centralized* entities to *decentralized* communities.

Source: CapIQ, SimilarWeb.

“Blockchains are computers that can make *credible commitments*”

Blockchains are computers that can make “credible commitments”



Many (1000s) untrustworthy physical computers

One single virtual computer with strong trust guarantees

Programs that run on blockchain computers are called “smart contracts”

What is a smart contract?

"A software program that automatically moves digital assets according to arbitrary pre-specified rules"

(Vitalik Buterin, Ethereum White Paper, 2014)

What is a smart contract?

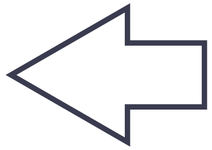
A program that can receive, store & send digital assets

A program with its own “bank account”

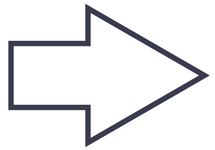
Essentially, a program that can “own things”

Smart contracts: basic principle

- A vending machine is an **automaton** that can trade **physical** assets



1. insert coins



2. dispense drink + change

Smart contracts: basic principle

- A smart contract is an **automaton** that can trade **digital** assets

code

1. insert digital coins (tokens)

2. dispense other digital assets or electronic rights

But who should we trust to faithfully execute the automaton's code?

- A smart contract is an **automaton** that can trade **digital** assets

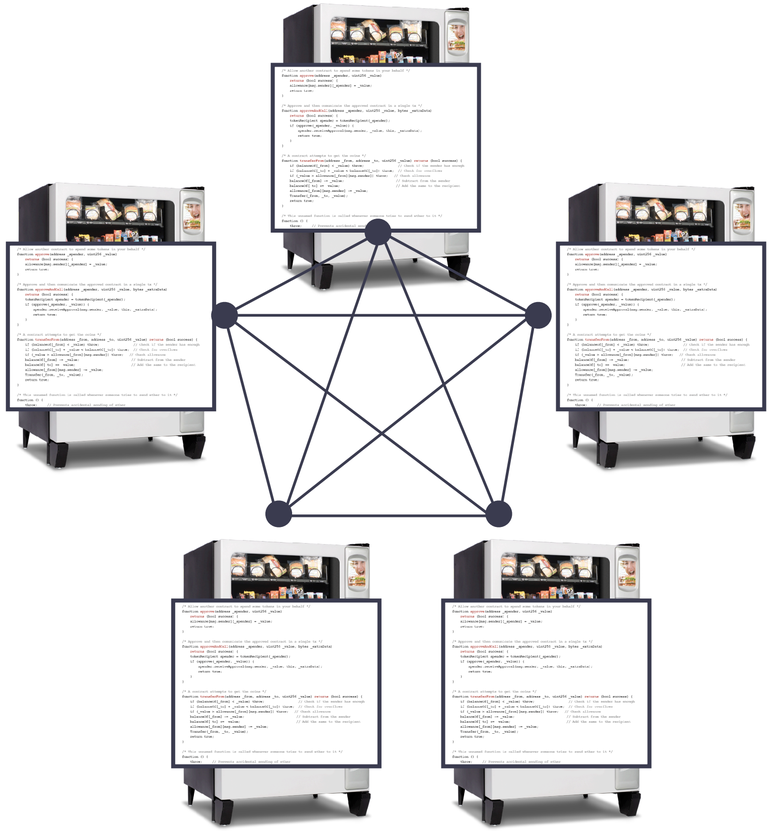
code

1. insert digital coins (tokens)

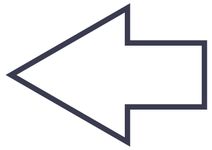
2. dispense other digital assets or electronic rights

Delegate trust to a decentralised network

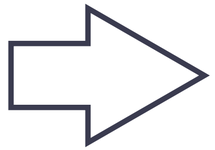
- A smart contract is a **replicated automaton** that can trade **digital** assets



replicated code



1. insert digital coins (tokens)



2. dispense other digital assets or electronic rights

Research challenges

Securely interacting with blockchain computers directly as a user is hard.

Can we build better “terminals” to connect to blockchain computers?

Programming blockchain computers is hard and unforgiving.

Can we find better, safer ways to program blockchain computers?

Research challenges

Securely interacting with blockchain computers directly as a user is hard.

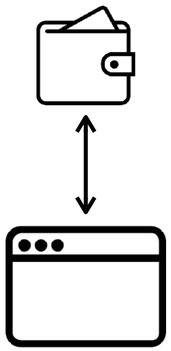
Can we build better “terminals” to connect to blockchain computers?

Programming blockchain computers is hard and unforgiving.

Can we find better, safer ways to program blockchain computers?

Web3 has a centralization problem

Wallet



Browser
or mobile
(Dapp UI)

REST

Web2 middlemen!

INFURA

alchemy

coinbase

BINANCE

OpenSea

Rarible

Node providers

Crypto exchanges

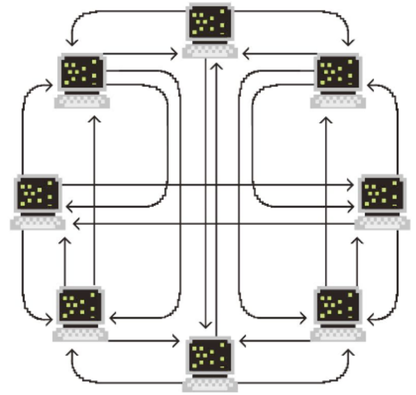
NFT marketplaces

...

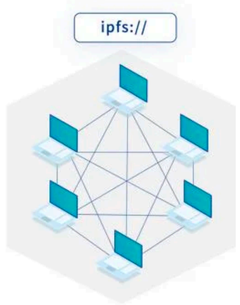
p2p

Web3

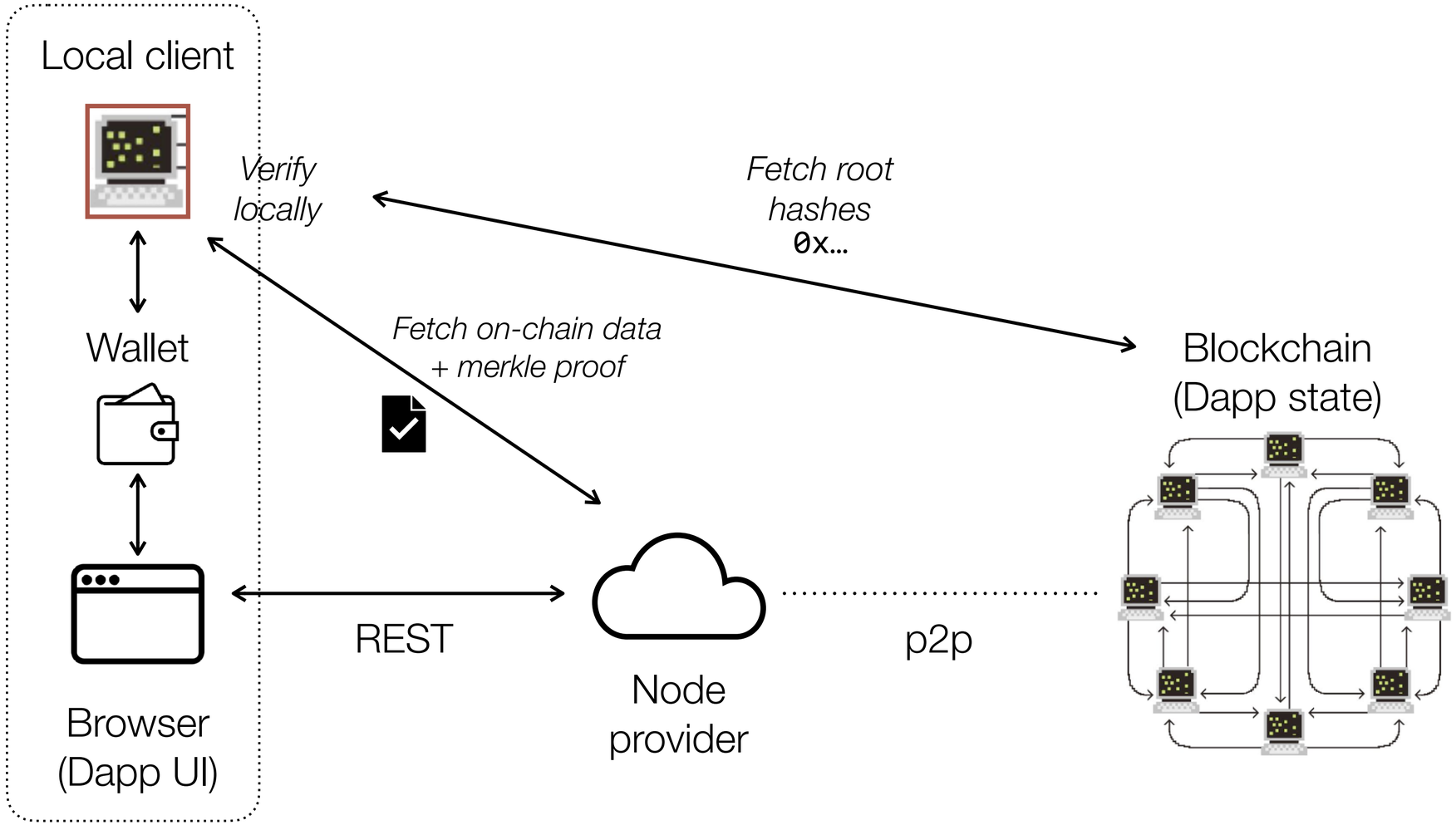
Blockchains



Decentralized data storage



Bridging Web3 and Web2: building better light clients



Bridging Web3 and Web2: building better light clients

TABLE I: Overview of light client schemes and implementations

Scheme	Consensus	Complexity	Compatibility	Crypto Primitives	Implementations
SPV [5]	Any	Linear	Any		
Ethereum 2.0 [21]	PoS	Linear	Fully compatible	Merkle proofs	Nimbus, Helios, Lodestar
PoPoW [7]	PoW	Sublinear	Modification	PoPoW	
NIPoPoW [8]	PoW	Sublinear	Modification	NIPoPoW	Ergo, WebDollar, Nimiq 1.0
FlyClient [9]	PoW	Sublinear	Modification	MMR proofs	ZCash
PoPoS [12]	PoS	Sublinear	Fully compatible	Merkle proofs	Kevlar
PoNW [11]	PoW	O(1)	Modification	SNARKs	
Mina [10]	PoS	O(1)	New System	SNARKs	Mina
DCert [14]	Any	O(1)	Any	Trusted Execution	DCert

(W. Wang and T. Van Cutsem, “Don’t Trust, Verify: Empowering Last-Mile Security and Privacy in Web3 ”. EuroS&P 2023 Poster)

Research challenges

Securely interacting with blockchain computers directly as a user is hard.

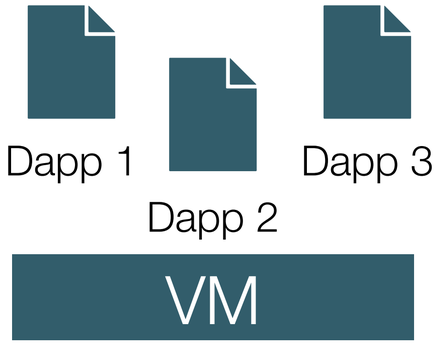
Can we build better “terminals” to connect to blockchain computers?

Programming blockchain computers is hard and unforgiving.

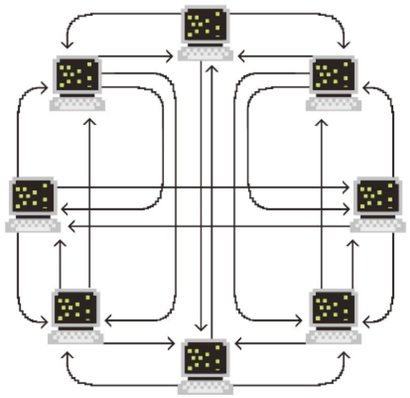
Can we find better, safer ways to program blockchain computers?

Two approaches to program a blockchain computer

“World Computer” vision

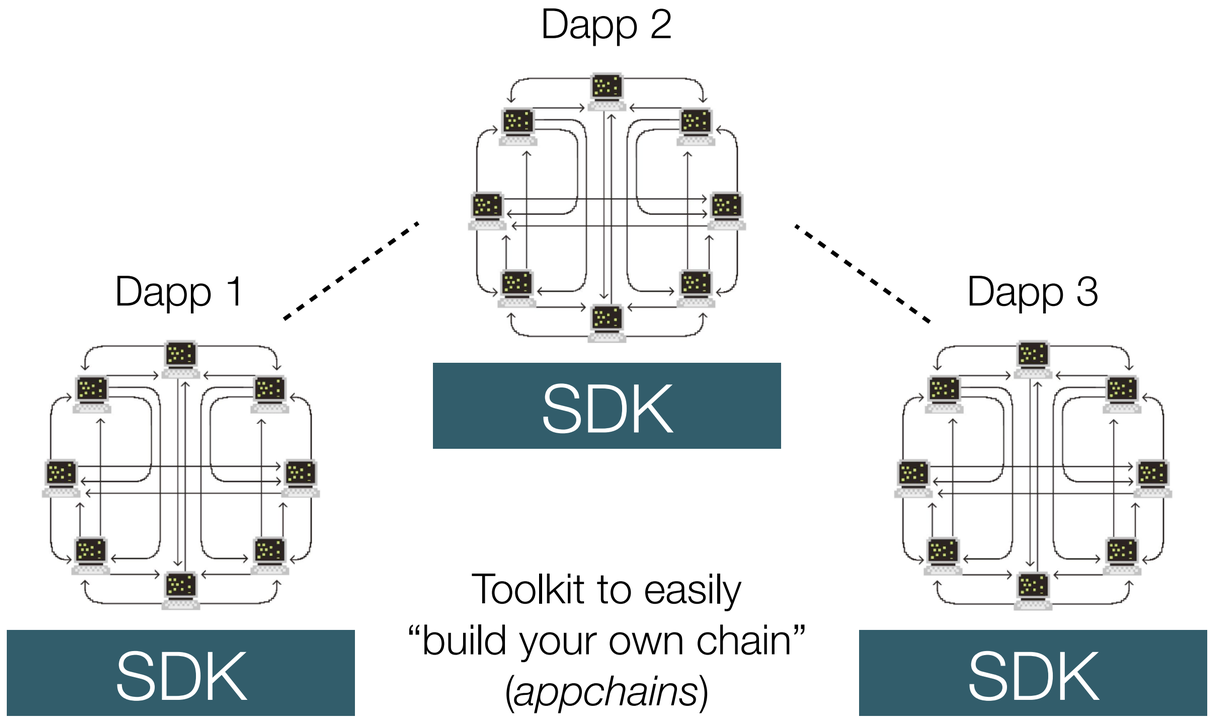


Programmable “Layer 1” chain



ethereum Algorand

“Internet of Blockchains” vision



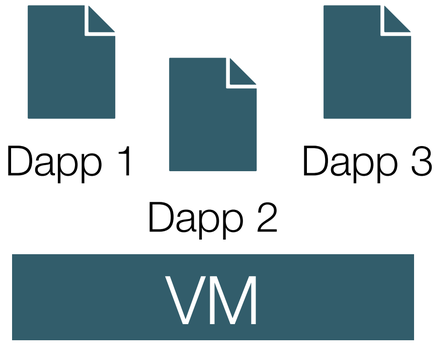
Toolkit to easily
“build your own chain”
(*appchains*)

COSMOS INTERNET OF BLOCKCHAINS Polkadot

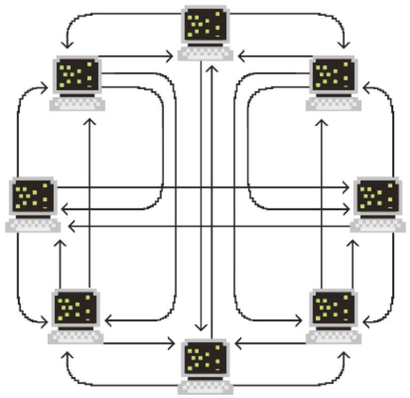
KU LEUVEN DistriNet

Two approaches to program a blockchain

“World Computer” vision

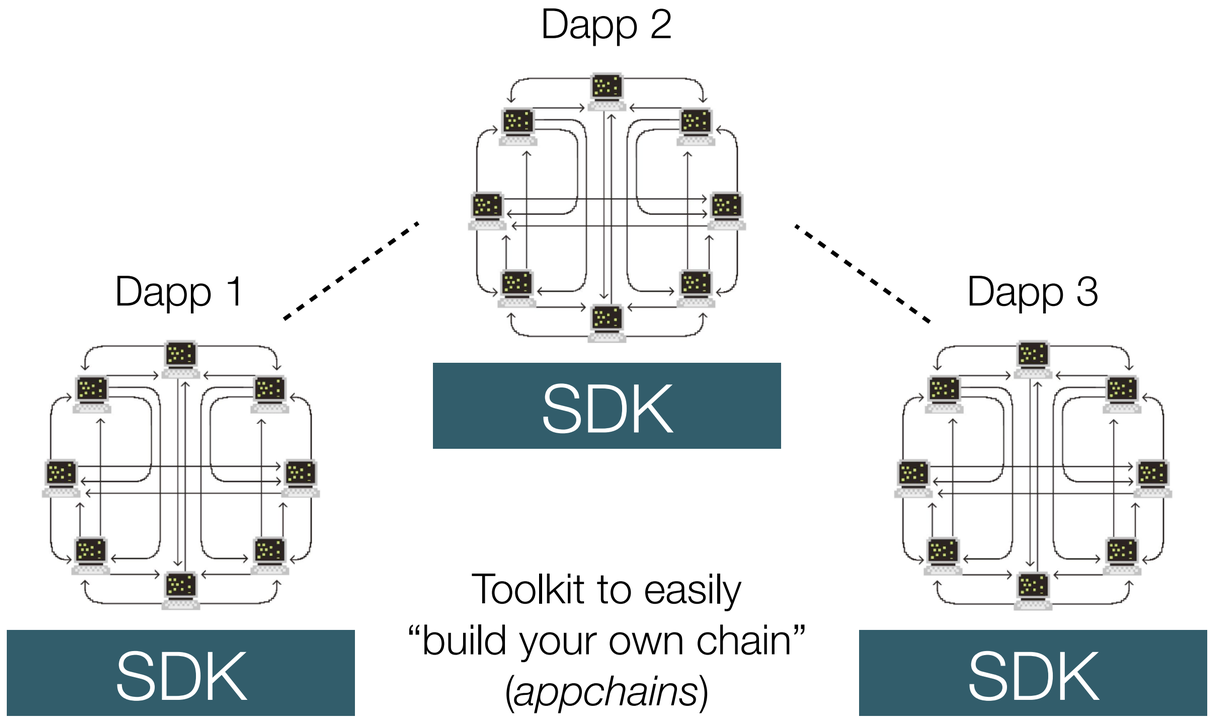


Programmable “Layer 1” chain



ethereum Algorand

“Internet of Blockchains” vision



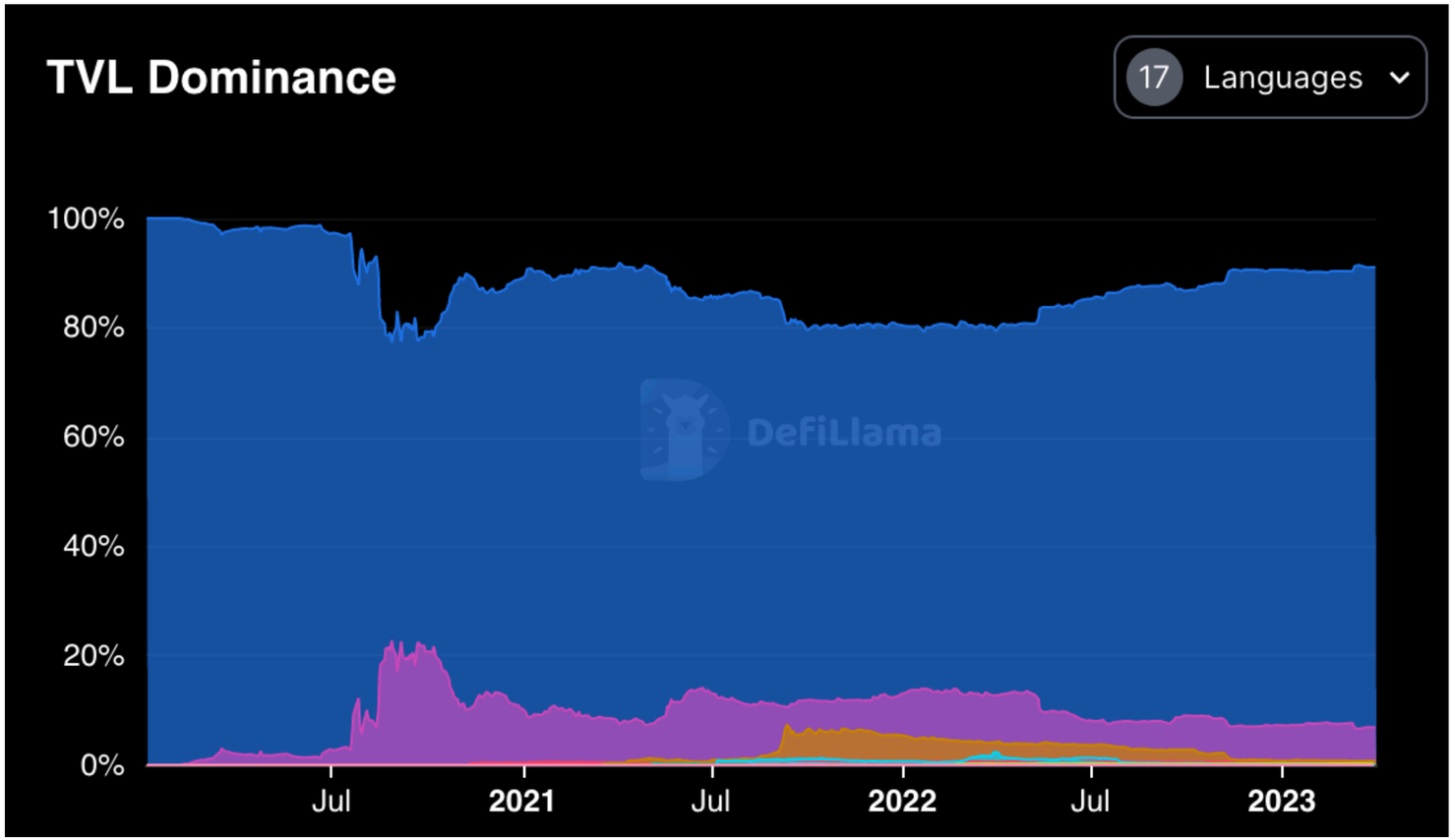
Toolkit to easily
“build your own chain”
(*appchains*)

COSMOS INTERNET OF BLOCKCHAINS Polkadot

KU LEUVEN DistriNet

Solidity is by far the most important smart contract language today

TVL = Total value locked in smart contract programs



31 mrt 2023

- Solidity 91.05%
- Vyper 6.91%
- Rust 0.7%
- Cairo 0.49%
- Bitcoin Script 0.24%
- Haskell 0.19%
- Ride 0.11%
- C++ 0.11%
- C# 0.1%
- Java 0.04%
- Others 0.07%

(Source: Defillama, april 2023)

Solidity has many safety issues that lead to vulnerabilities

Vulnerabilities

Not So Smart Contract	Description
Bad randomness	Contract attempts to get on-chain randomness, which can be manipulated by users
Denial of Service	Attacker stalls contract execution by failing in strategic way
Forced Ether Reception	Contracts can be forced to receive Ether
Incorrect Interface	Implementation uses different function signatures than interface
Integer Overflow	Arithmetic in Solidity (or EVM) is not safe by default
Race Condition	Transactions can be frontrun on the blockchain
Reentrancy	Calling external contracts gives them control over execution
Unchecked External Call	Some Solidity operations silently fail
Unprotected Function	Failure to use function modifier allows attacker to manipulate contract
Variable Shadowing	Local variable name is identical to one in outer scope
Wrong Constructor Name	Anyone can become owner of contract due to missing constructor

Crytic, (2018). Not so smart contracts. https://github.com/crytic/not-so-smart-contracts

SWC Registry

Smart Contract Weakness Classification and Test Cases

The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples. The links in the ID and Test Cases columns link to the respective SWC definition. Links in the Relationships column link to the CWE Base or Class type.

ID	Title	Relationships	Test cases
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	• odd_even.sol • odd_even_fixed.sol
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	• deposit_box.sol • deposit_box_fixed.sol • wallet.sol • wallet_fixed.sol
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	• hardcoded_gas_limits.sol
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	• access_control.sol • access_control_fixed_1.sol • access_control_fixed_2.sol

Smart Contract Weakness Classification https://swcregistry.io/

Can we design safer smart contract languages?

CHAPTER 3

The Next 700 Smart Contract Languages

Ilya Sergey, *Yale-NUS College and National University of Singapore, Singapore*

3.1 INTRODUCTION

Smart contracts are a mechanism for expressing replicated computations powered by a decentralized consensus protocol [Szabo 1994]. They are most commonly used to define custom logic for transactions operating over a blockchain—that is, a decentralized Byzantine-fault-tolerant distributed ledger [Bano et al. 2019, Pirlea and Sergey 2018]. In addition to a typical state of computations, a blockchain stores a mapping from *accounts* (public keys or addresses) to quantities of *tokens* owned by said accounts. Execution of an arbitrary program (aka a smart contract) is done by *miners*, who run the computations and maintain the distributed ledger in exchange for a combination of *gas* (transaction fees based on the execution length, denominated in the intrinsic tokens and paid by the account calling the smart contract) and *block rewards* (inflationary issuance of fresh tokens by the underlying protocol). One distinguishing property of smart contracts, not found in standard computational settings, is the management of token transfers between accounts. While simple forms of smart con-

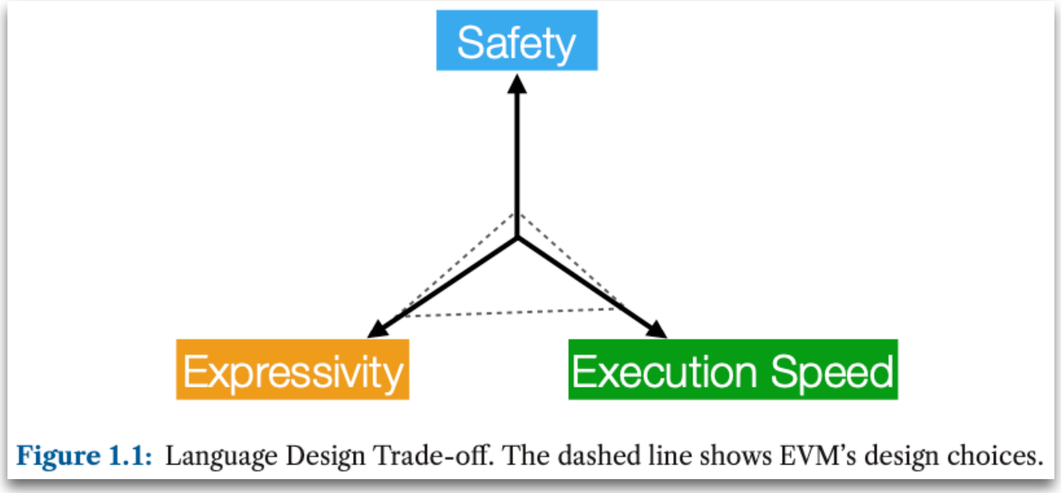


Figure 1.1: Language Design Trade-off. The dashed line shows EVM's design choices.

Ilya Sergey, "The Next 700 Smart Contract Languages" in *Principles of Blockchain Systems*, 2021

How you represent digital assets in smart contract code matters

Solidity: assets are integers

```
contract Crowdfunding {
    address public owner;
    uint256 public deadline;
    uint256 public goal;
    mapping (address => uint256) public backers;

    function donate() public payable {
        require(block.timestamp < deadline);
        backers[msg.sender] += msg.value;
    }

    …
}
```

Move: assets are “resource types”

```
module crowdfunding {

  struct Deposit<phantom CoinType> has key {
    coin: Coin<CoinType>,
  }
  public entry fun donate<CoinType>(account: &signer, fund_addr: address,
                                    amount: u64) acquires Deposit, CrowdFunding {
      …
      let coin_to_deposit = coin::withdraw<CoinType>(account, amount);
      let cf = borrow_global_mut<CrowdFunding<CoinType>>(fund_addr);

      if (!exists<Deposit<CoinType>>(addr)) {
          let to_deposit = Deposit<CoinType> {coin: coin_to_deposit};
          move_to(account, to_deposit);
          let backers = &mut cf.backers;
          vector::push_back<address>(backers, addr);
      } else {
         let deposit = borrow_global_mut<Deposit<CoinType>>(addr);
         coin::merge<CoinType>(&mut deposit.coin, coin_to_deposit);
      }

      …
  }

  …
}
```

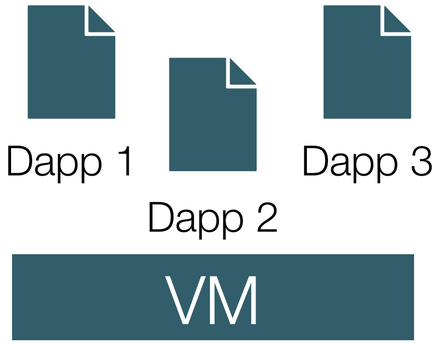
Safer smart contract languages. Example: Move

Vulnerability Class	# Vuln researched	Move	Solidity 0.8+
Overflow/ Underflow	12	12	12
Access control	15	8	2
Constructor naming	5	5	4
Control flow	7	7	0
Logic error	17	6	1
Wrong interface	8	8	0
Total	**64**	**46 (72%)**	**19 (30%)**

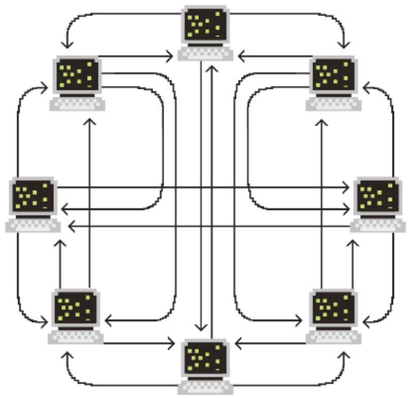
(S. Selleri, “Smart contract safety: A comparative study between Solidity and Move smart contract languages”. Masters’ thesis. 2023)

Two approaches to program a blockchain

“World Computer” vision

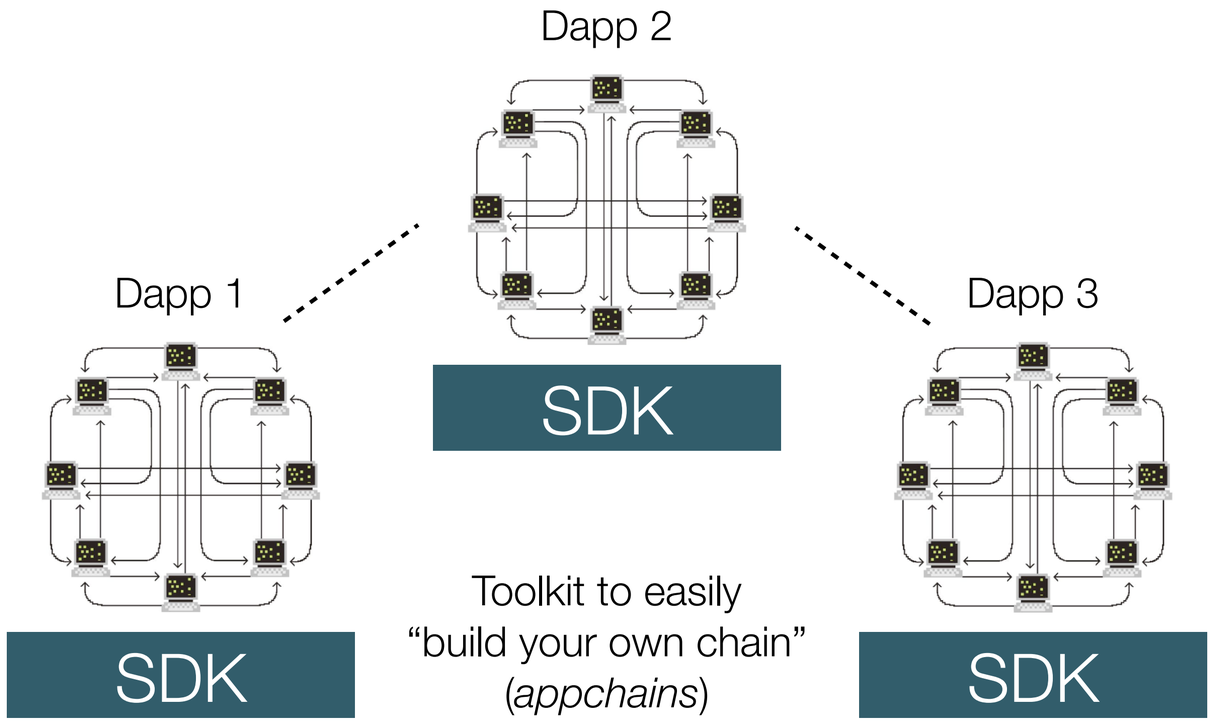


Programmable “Layer 1” chain



ethereum Algorand

“Internet of Blockchains” vision

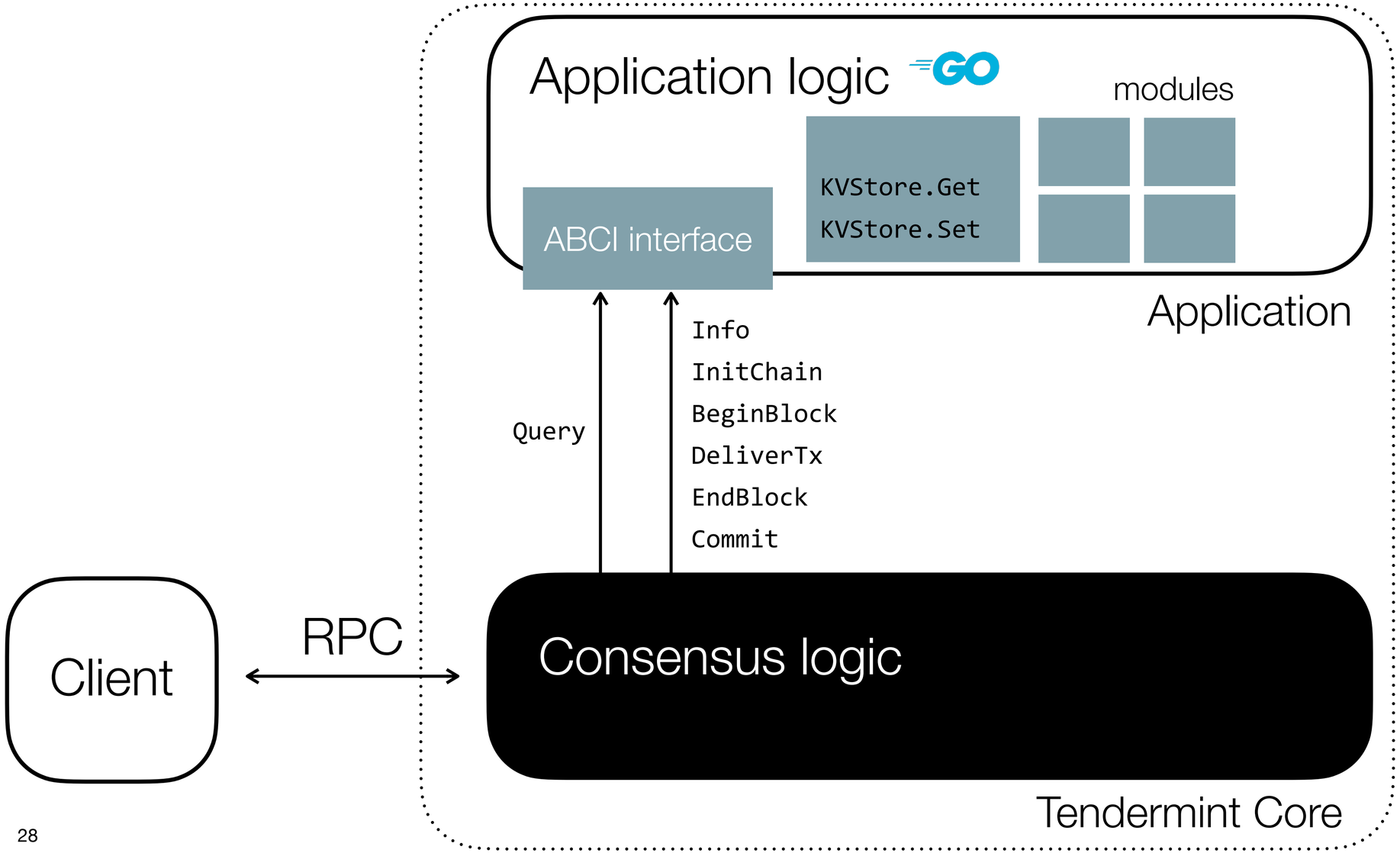


Toolkit to easily
“build your own chain”
(*appchains*)

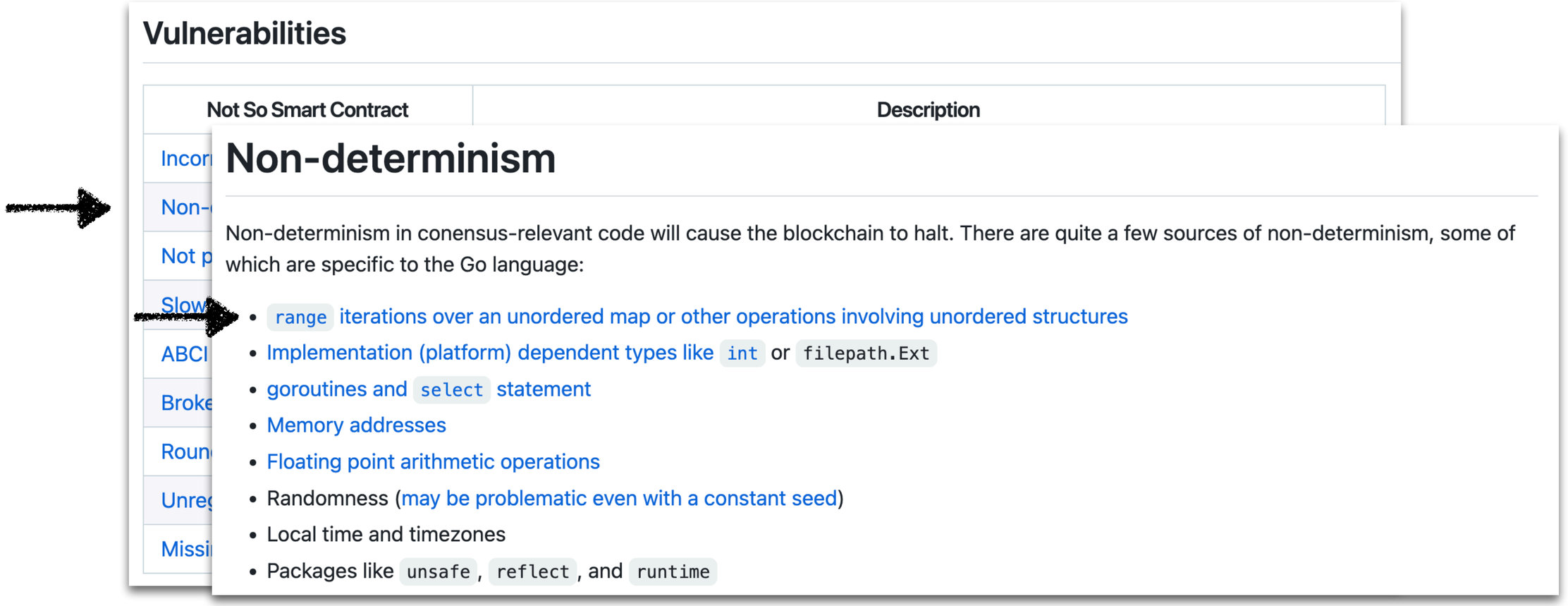
COSMOS INTERNET OF BLOCKCHAINS Polkadot

KU LEUVEN DistriNet

Cosmos SDK and ABCI



Vulnerabilities in Cosmos code



Vulnerabilities

Not So Smart Contract	Description
Incor	
Non-	
Not p	
Slow	
ABCI	
Broke	
Roun	
Unreg	
Missi	

Non-determinism

Non-determinism in conensus-relevant code will cause the blockchain to halt. There are quite a few sources of non-determinism, some of which are specific to the Go language:

- `range` iterations over an unordered map or other operations involving unordered structures
- Implementation (platform) dependent types like `int` or `filepath.Ext`
- goroutines and `select` statement
- Memory addresses
- Floating point arithmetic operations
- Randomness (may be problematic even with a constant seed)
- Local time and timezones
- Packages like `unsafe`, `reflect`, and `runtime`

(Source: Crytic)

Map iteration order in Go is non-deterministic

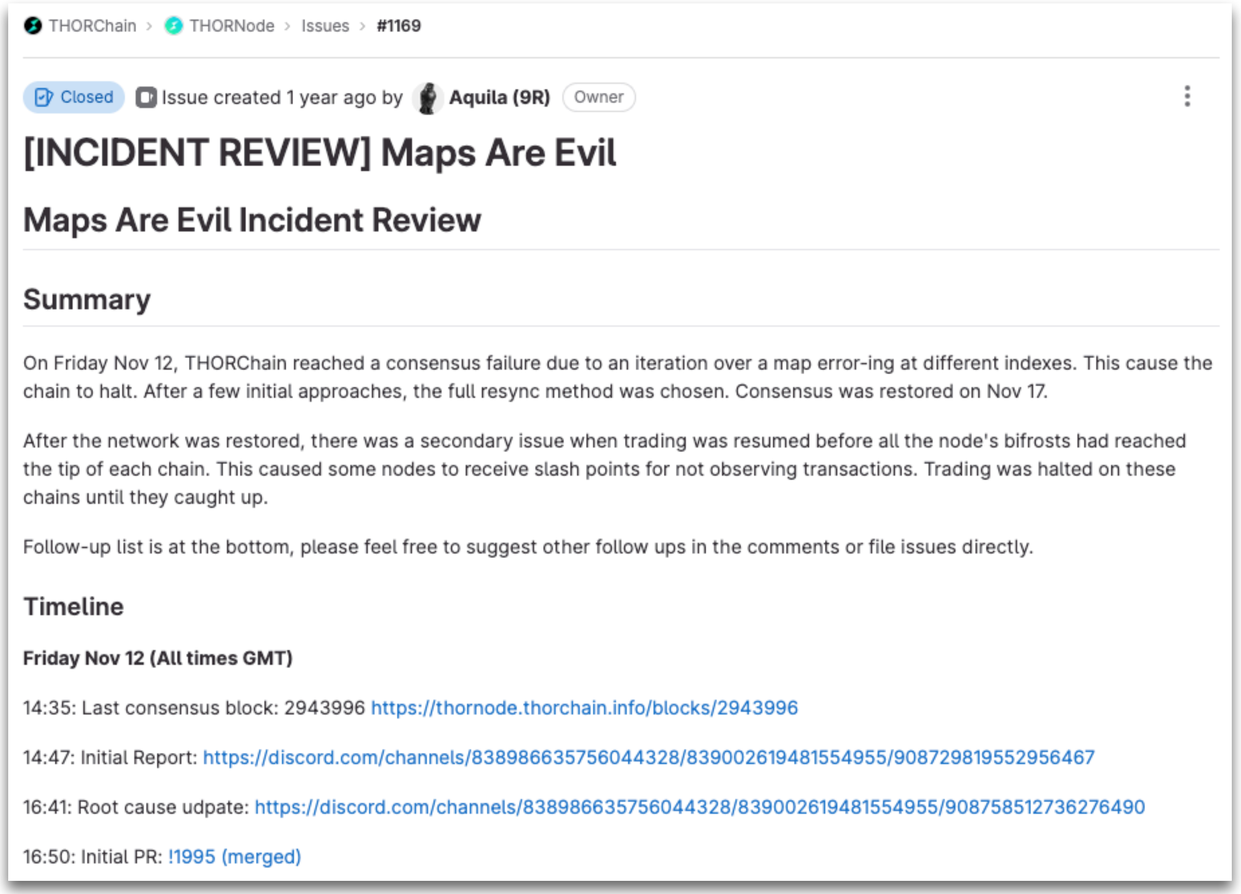
```go
func main() {
    m := make(map[string]int32)
    m["one"] = 1
    m["two"] = 2
    m["three"] = 3

    arr := []int{}

    for key, value := range m {
        append(arr, value)
    }

    // The order of values in arr is non-deterministic
}
```

Blockchain computers don’t like non-deterministic execution



THORChain › THORNode › Issues › #1169

Closed Issue created 1 year ago by Aquila (9R) Owner

[INCIDENT REVIEW] Maps Are Evil

Maps Are Evil Incident Review

Summary

On Friday Nov 12, THORChain reached a consensus failure due to an iteration over a map error-ing at different indexes. This cause the chain to halt. After a few initial approaches, the full resync method was chosen. Consensus was restored on Nov 17.

After the network was restored, there was a secondary issue when trading was resumed before all the node's bifrosts had reached the tip of each chain. This caused some nodes to receive slash points for not observing transactions. Trading was halted on these chains until they caught up.

Follow-up list is at the bottom, please feel free to suggest other follow ups in the comments or file issues directly.

Timeline

Friday Nov 12 (All times GMT)

14:35: Last consensus block: 2943996 https://thornode.thorchain.info/blocks/2943996

14:47: Initial Report: https://discord.com/channels/838986635756044328/839002619481554955/908729819552956467

16:41: Root cause udpate: https://discord.com/channels/838986635756044328/839002619481554955/908758512736276490

16:50: Initial PR: !1995 (merged)

Studying potential vulnerabilities in Cosmos code in the wild

Curated corpus of 11 representative Cosmos projects:

- Open Source
- Built with Cosmos SDK
- Mix of application use cases
- Mix of recent and mature projects
- Mix of large, medium, small market caps

Name	Use case	Date	Total Tx	# Func
Stride[1]	A liquid staking platform.	4 Sept. 2022	546,690	209
Osmosis[2]	The largest interchain decentralized exchange.	16 June 2021	607,470	1310
Cosmos Hub[3] (Gaia)	The economic center of Cosmos providing IBC security and more.	13 Aug. 2019	289,710	6
Axelar[4]	Web3 integration across multiple chains	8 Mar. 2021	2,944,100	2965
Crypto.org[5]	Payment, DeFi and NFTs.	14 Oct. 2020	167,040	92
Fetch.ai[6]	Automation of Web3 systems using AI agents.	1 July 2020	264,000	2
Regen[7]	Originate and invest in ecological regeneration projects.	5 June 2019	0	444
Jackal[8]	Cloud storage solution.	22 Oct. 2022	0	257
Medibloc[9]	Patient-centered health data ecosystem.	26 Aug 2019	14,101	513
Desmos[10]	Framework to build social media platforms.	10 Dec. 2019	3,202	758
Dig[11]	Tokenized real-estate.	13 Dec. 2021	1,370	2

(J. Surmont, “Static Application Security Testing in Application-specific Blockchains: a case study of Cosmos”. Masters’ thesis. 2023)

Perform CodeQL Analysis

```
1    ▶ Run github/codeql-action/analyze@v1
26   /opt/hostedtoolcache/CodeQL/0.0.0-20220214/x64/codeql/codeql version --format=terse
27   2.8.1
28   ▶ Extracting javascript
293  ▶ Finalizing javascript
297  ▶ Running queries for javascript
789  ▶ Interpreting results for javascript
1050 Analysis produced the following diagnostic data:
1051
1052 |          Diagnostic          |  Summary   |
1053 +------------------------------+------------+
1054 | Extraction errors            | 0 results  |
1055 | Successfully extracted files | 20 results |
1056 Analysis produced the following metric data:
1057
1058 |                                                                                  Metric                                                                                  | Value |
1059 +----------------------------------------------------------------------------------------------------------------------------------------------------------------------------+-------+
1060 | Total lines of JavaScript and TypeScript code in the database                                                                                                              |   607 |
1061 | Total lines of user written JavaScript and TypeScript code in the database                                                                                                 |   607 |
1062
1063 /opt/hostedtoolcache/CodeQL/0.0.0-20220214/x64/codeql/codeql database print-baseline
     /home/runner/work/_temp/codeql_databases/javascript
```

Identifying potential vulnerabilities using the CodeQL SAST tool

Statically detect 8 potential sources of non-determinism in “consensus-critical code” of Go applications that use Cosmos ABCI

Refactored / new queries	Positives	UTP	Noise Ratio	Precision
1. {Begin,End}Block panic	91	91	0%	100%
2. Map iteration	13	5	0%	38%
3. Hardcoded Bech32	0	0	N/A	N/A
4. Goroutines	0	0	N/A	N/A
5. Floating point	2	0	N/A	0%
6. Local time	0	0	N/A	N/A
7. Unsafe packages	5	4	0%	80%
8. Platform dependent types	44	35	0%	79.54%

(J. Surmont, “Static Application Security Testing in Application-specific Blockchains: a case study of Cosmos”. Masters’ thesis. 2023)

Summary

- Trust through replication: blockchains are trustworthy computers
- “World Computer” versus “Internet of Blockchains” execution model
- Can we build better “terminals” to connect to blockchain computers?
  - Build **light client-friendly blockchains**
- Can we find better, safer ways to program blockchain computers?
  - **Better language design** for blockchain-specific languages
  - **Better analysis tools** to “tame” general-purpose languages for blockchain