Blockchains & Server side Swift

Alex Tran Qui (alex@katalysis.io)

Swift Usergroup Netherlands, 30th January 2018

katalysis

https://katalysis.io

Monetisation and ownership management of digital content using blockchain technology

Tech stack

katalysis

Katalysis libraries

Zewo Server side Swift libraries

Ethereum Smart Contract Blockchain

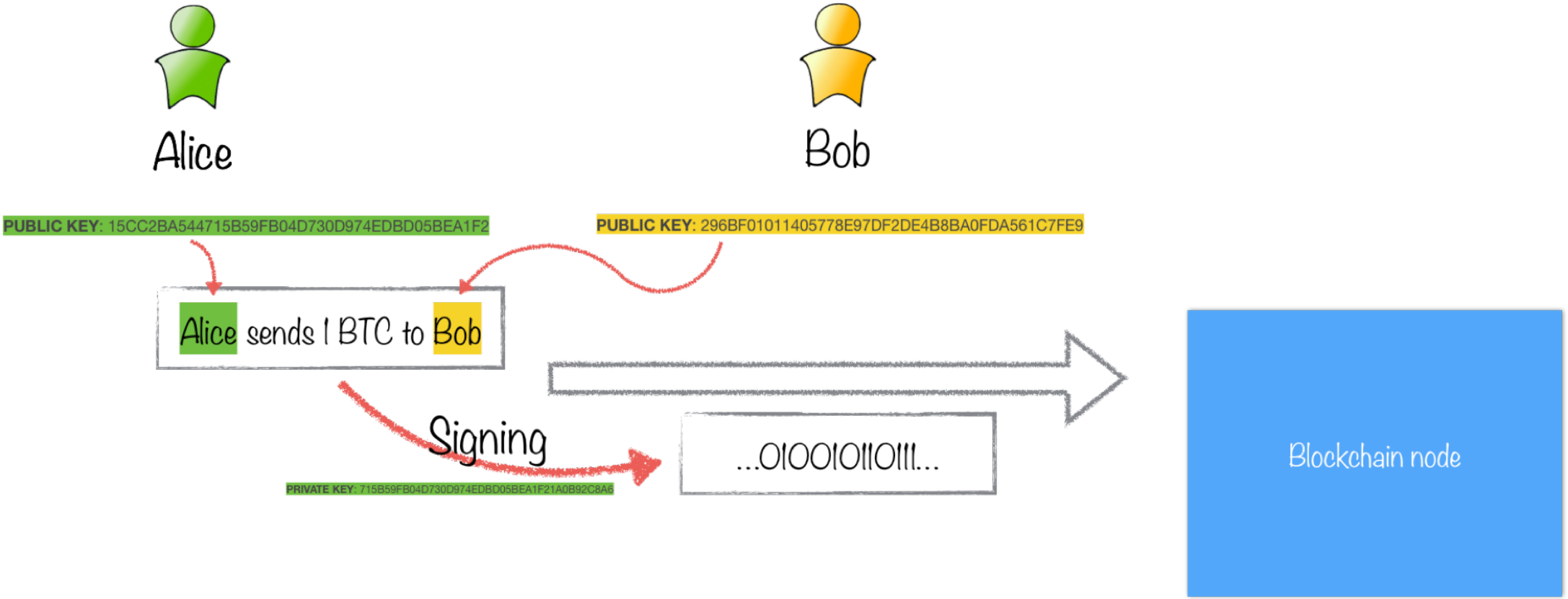
Blockchain

katalysis

- Peer to peer system secured by Crypto and Consensus Algorithms
- Enables Transactions between untrusted participants
- Underlying technology to Bitcoin, Ethereum

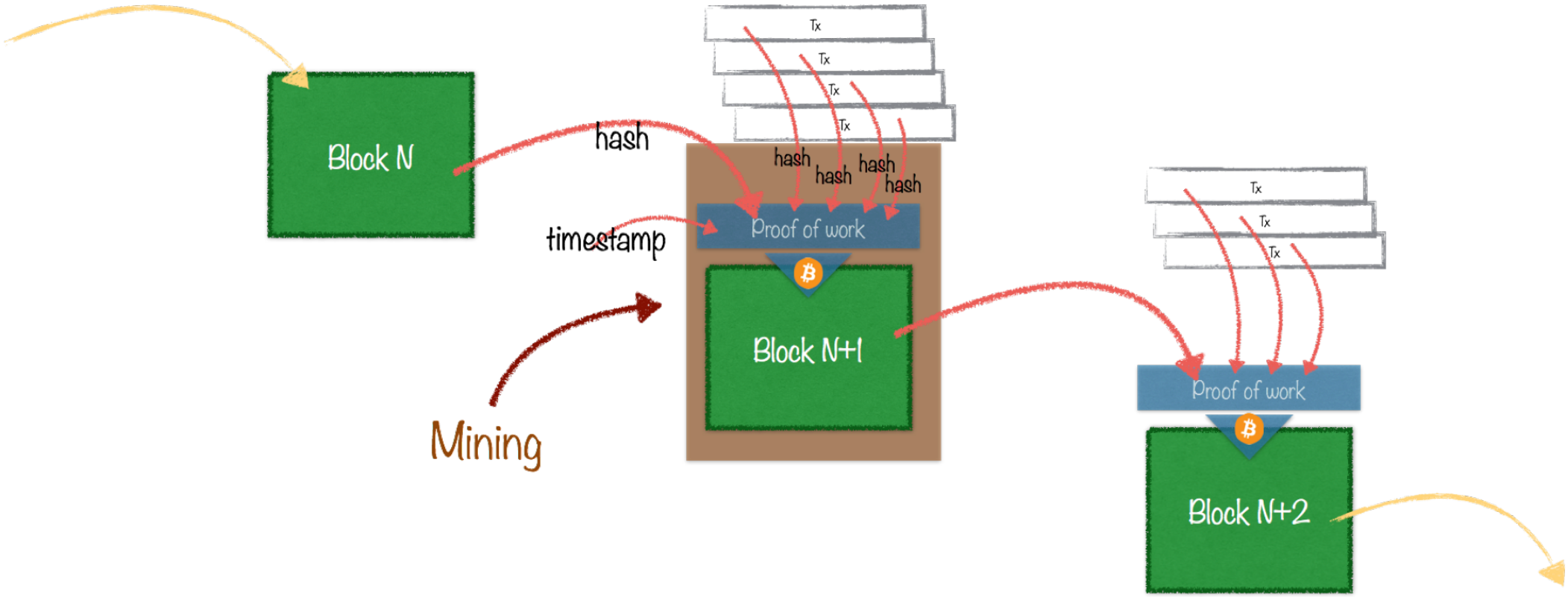
Signing

katalysis



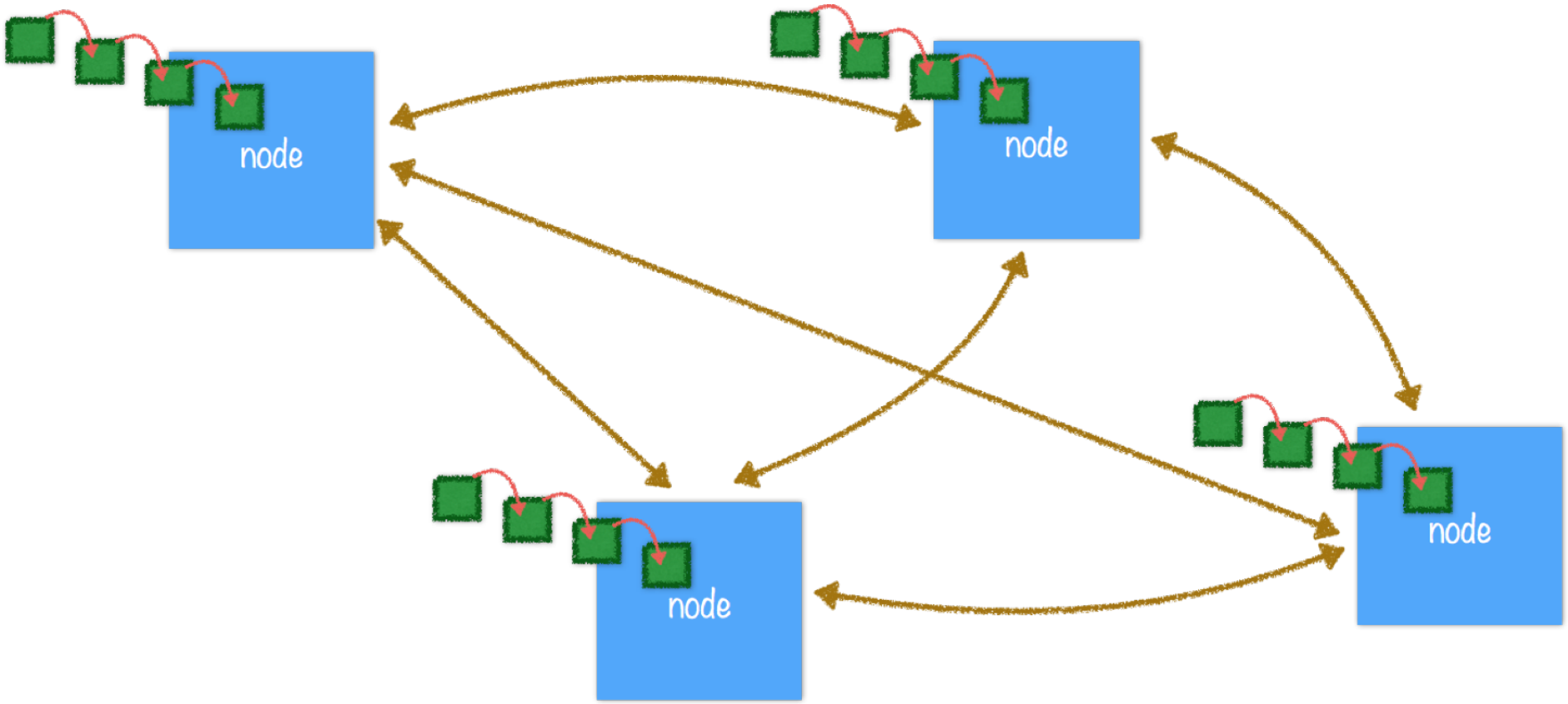
State Engine

katalysis



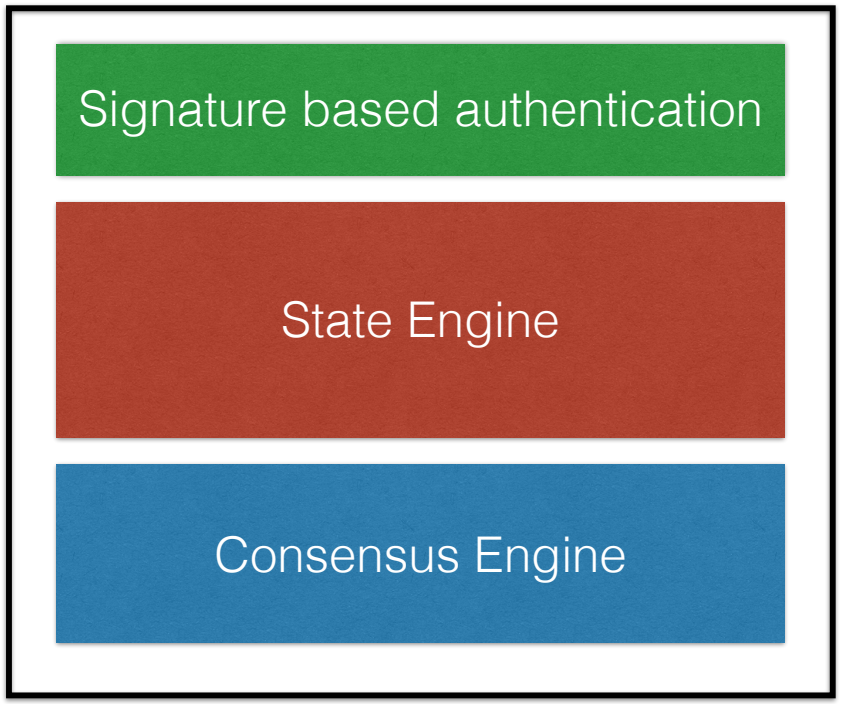
Consensus over P2P

katalysis



Blockchain node

katalysis



- Secured by crypto algorithms (hashing and signing)
- State engine stores transactions and possibly code and an execution environment
- Proof of Work (PoW), Proof of Stake (PoS), …

Datastore with following properties:

katalysis

- **Immutability** = “very difficult” to change history
- **Transparency** amongst participants
- **Decentralised**
- **Pseudonymity**

Ethereum Smart Contracts

katalysis

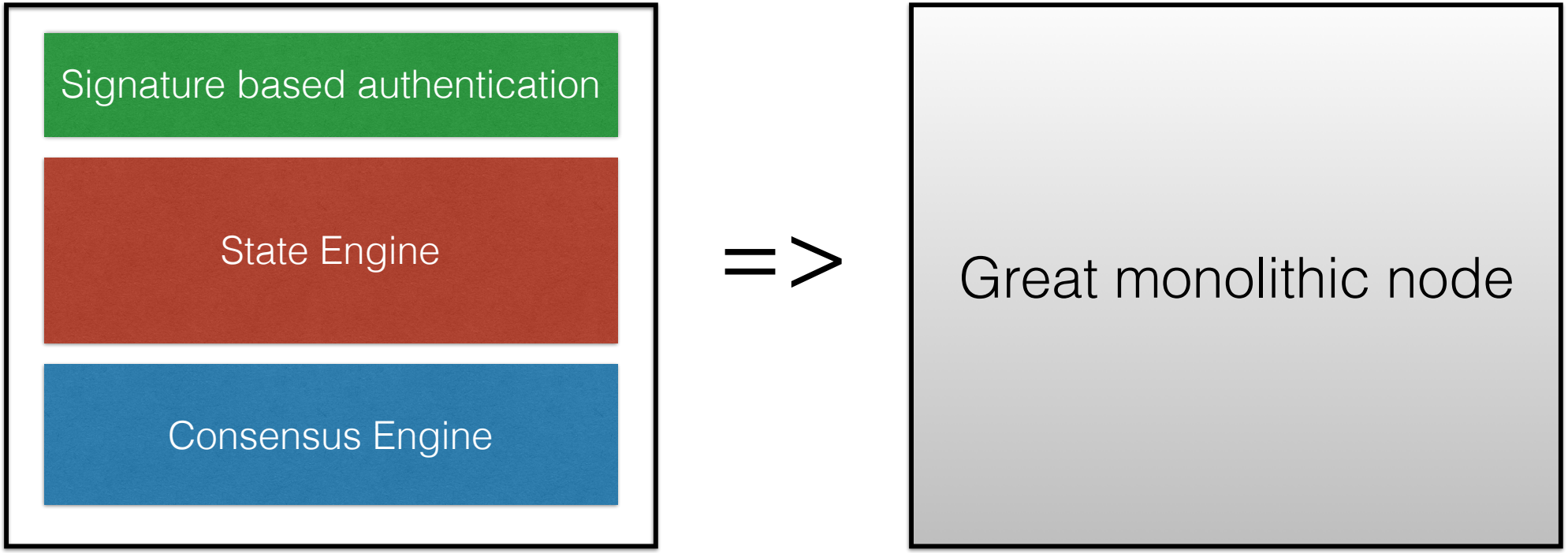
- https://ethereum.org
- Ethereum Virtual Machine (EVM)
- Storage of data AND code in the State Engine
- EVM language is a JS like language called **Solidity**

“Smart Contracts”

katalysis

Blockchain node

katalysis



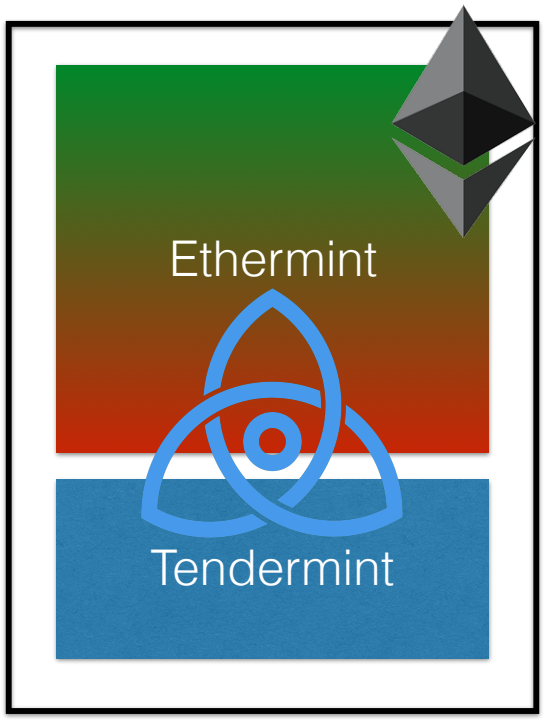
Tendermint

katalysis

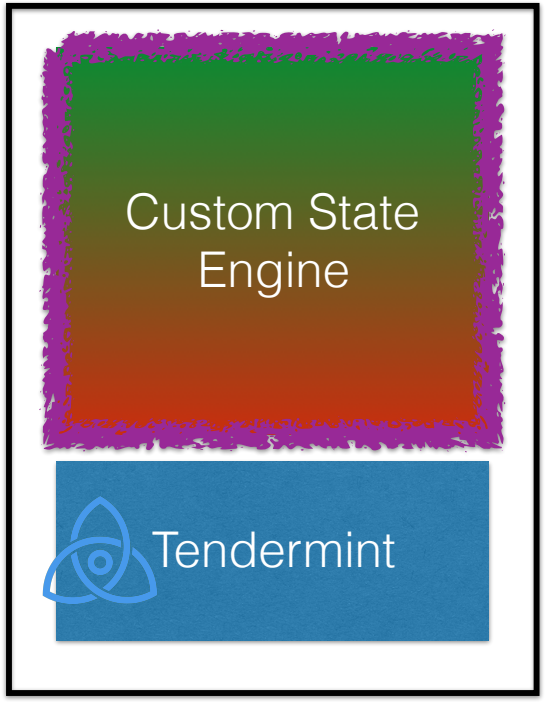
- **Toolkit** for building blockchains
- tendermint/ethermint: fully Ethereum compatible node, either public or private
- https://github.com/tendermint/ethermint/
- https://tendermint.com

Tendermint

katalysis



or



Tendermint

katalysis

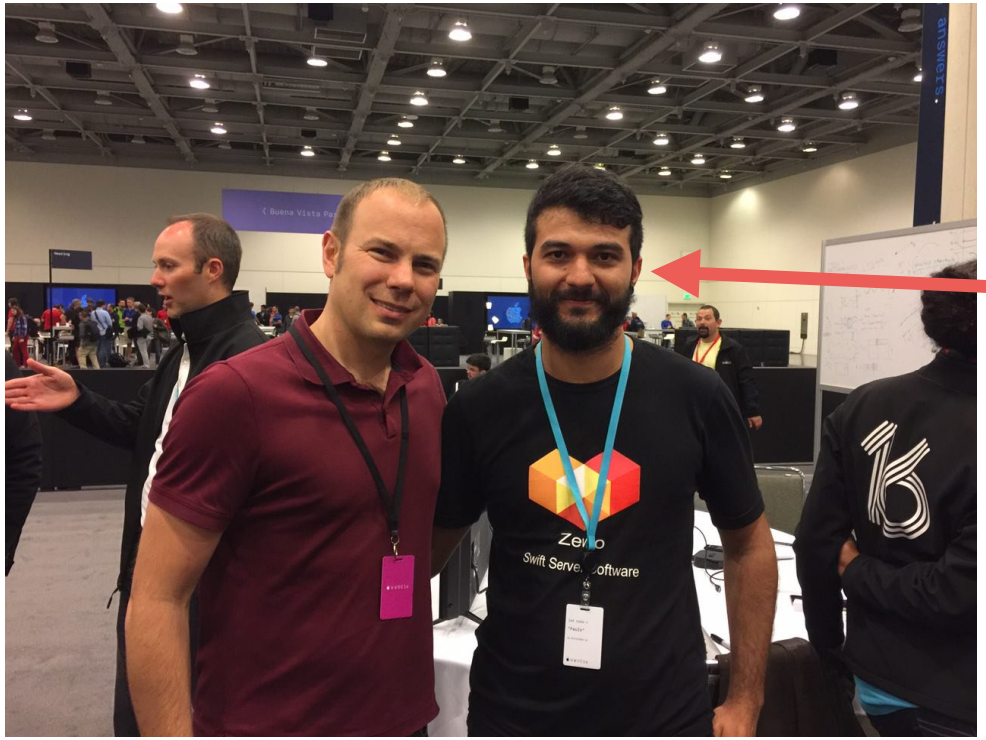
Written in Go.

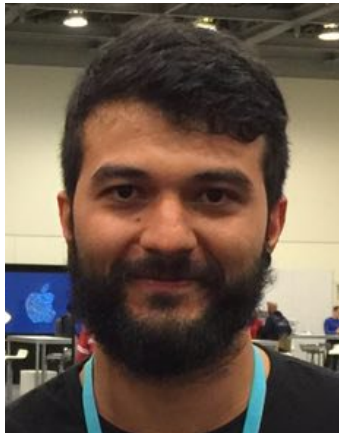
Zewo

katalysis

Zewo

katalysis





Paulo Faria

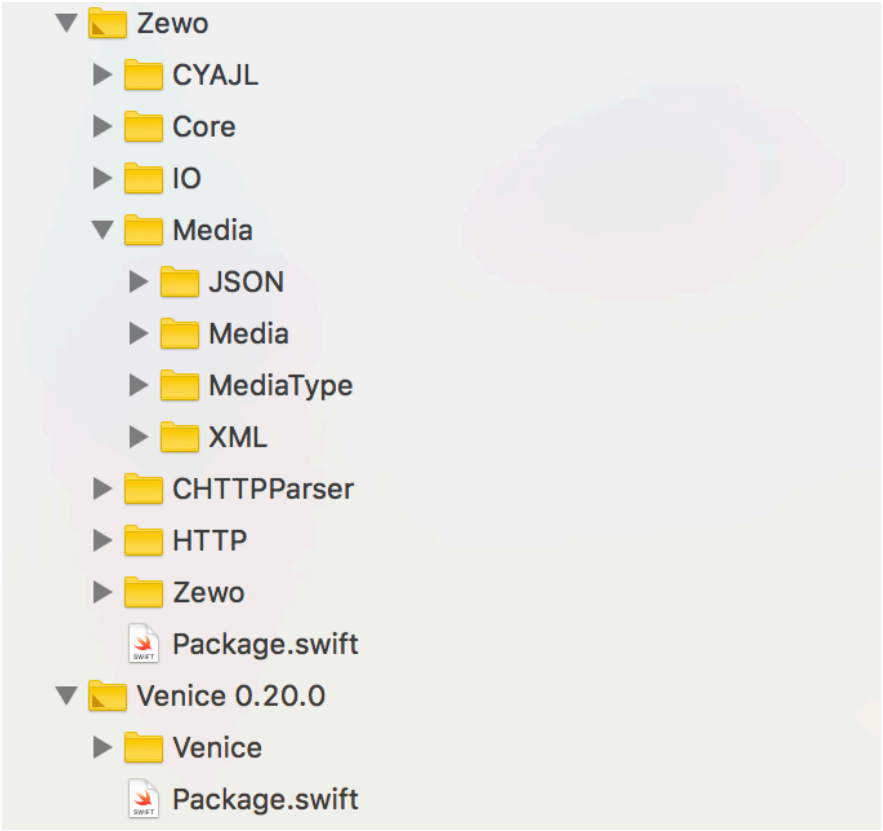
Zewo

katalysis

Lightweight libraries for web application in Swift

Zewo

katalysis



Zewo

katalysis

“Don’t communicate by sharing memory. Share memory by communicating.”

Rob Pike (co-designer of Go)

Zewo

katalysis

Coroutines instead of Multi-threading

Zewo/Venice

katalysis

Coroutines in Swift

zewo.github.io/Venice/

Wraps libdill.org

libdill.org/structured-concurrency.html

libdill

katalysis

- Structured concurrency (coroutine nesting)
- Coroutines run on a single CPU core
- Coroutines must yield control to achieve concurrency
- Blocking functions automatically yield

Zewo/Venice

katalysis

```
func testWakeUpWithChannels() throws {
    let channel = try Channel<Int>()
    let group = Coroutine.Group()

    func send(_ value: Int, after delay: Duration) throws {
        try Coroutine.wakeUp(delay.fromNow())
        try channel.send(value, deadline: .never)
    }

    try group.addCoroutine(body: { try send(111, after: 30.milliseconds) })
    try group.addCoroutine(body: { try send(222, after: 40.milliseconds) })
    try group.addCoroutine(body: { try send(333, after: 10.milliseconds) })
    try group.addCoroutine(body: { try send(444, after: 20.milliseconds) })

    XCTAssert(try channel.receive(deadline: .never) == 333)
    XCTAssert(try channel.receive(deadline: .never) == 444)
    XCTAssert(try channel.receive(deadline: .never) == 111)
    XCTAssert(try channel.receive(deadline: .never) == 222)
    group.cancel()

}
```

Katalysis libraries

katalysis

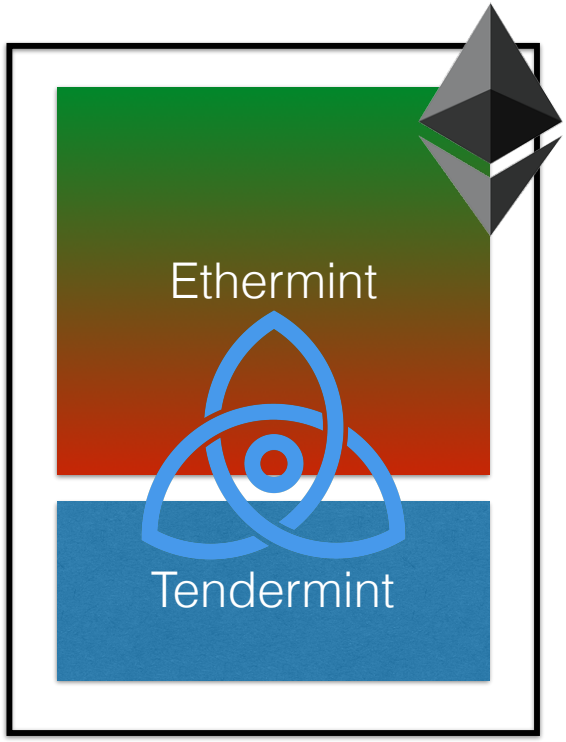
Problem: interact with a blockchain node from Swift

Solutions:

- EthereumBlockchain Swift library: Ethereum node compliant JSON-RPC calls, including transaction generation and signing.
- ABCISwift library: Tendermint ABCIServer allowing the use of the underlying Consensus Engine.

Ethereum Blockchain library

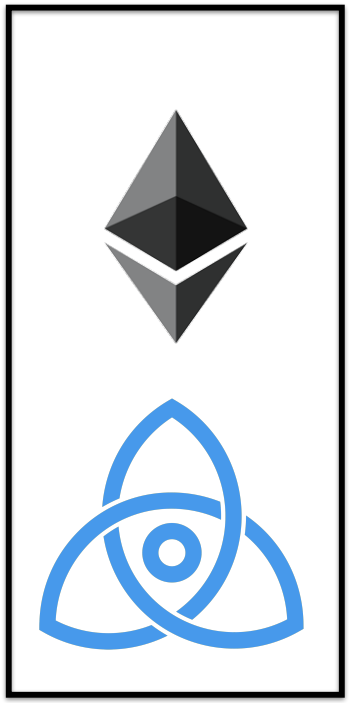
katalysis



Communicate to an Ethereum compatible node (web3.js)

Ethereum Blockchain library

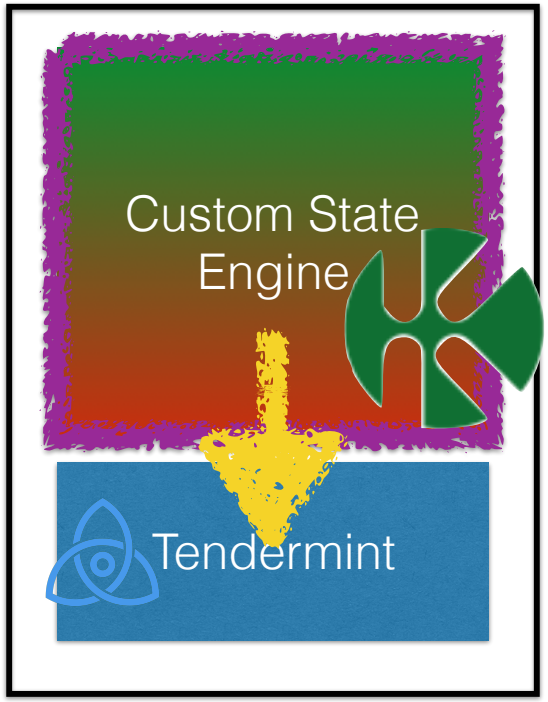
katalysis



```
import EthereumKeys
import EthereumBlockchain

func run() {
    do {
        let priv = [UInt8](repeating: 0x46, count: 32)
        let key = EthereumKey(seed:priv)!
        let ebc = EthereumBlockchain("http://localhost:8545/", key)
        let to = try? Address(bytes:[UInt8](repeating: 0x35, count: 20))

        let tx = TransactionRequest(to: to, value: 100)
        let txHash = try ebc.transact(tx)
        print("Transaction hash: \(txHash)")

    } catch let error {
        print ("\(error)")
    }
}
```

ABCISwift library

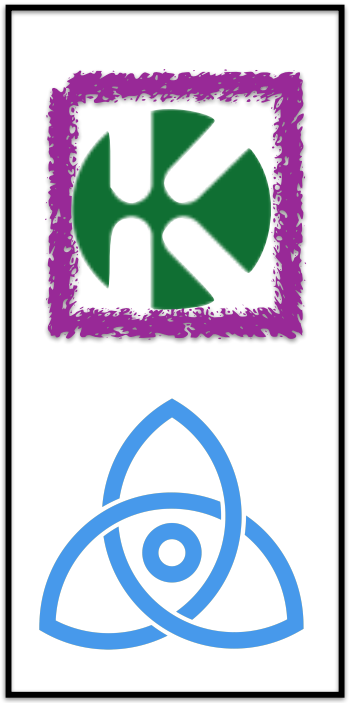
katalysis



Communicate directly to
a Consensus Engine
(ABCIClient)

ABCISwift library

katalysis



```swift
// ABCI apps should comply to the following protocol
public protocol ABCIApplication {

    func initChain(_ validators: [Validator])
    func info(_ version: String) -> ResponseInfo
    func echo(_ message: String) -> ResponseEcho
    func flush()
    func setOption(_ key: String, _ value: String) -> ResponseSetOption
    func deliverTx(_ tx: Data) -> Result
    func checkTx(_ tx: Data) -> Result
    func query(_ q: Query) -> ResponseQuery
    func beginBlock(_ hash: Data, _ header: Header)
    func endBlock(_ height: UInt64) -> ResponseEndBlock
    func commit() -> Result
}
```

Katalysis libraries

Current Status:

- Most building blocks (EthereumKey, RPL, Crypto algos, …) are already open sourced, work with Swift 4.0.x
- Earlier implementation of the EthereumBlockchain libraries (BurrowBlockchain) are open sourced
- Open Sourcing EthereumBlockchain and ABCISwift libraries is on the roadmap. Talk to us if you are interested.

Katalysis libraries

https://gitlab.com/katalysis

Recap

katalysis

https://gitlab.com/katalysis

https://github.com/Zewo


https://ethereum.org, https://tendermint.com

katalysis

We’re Hiring!

Alex Tran Qui (alex@katalysis.io)

Questions?


@katalysis_io


alex@katalysis.io


https://katalysis.io