



Bring dApps to the Real World

Icetea Whitepaper

Version 0.1

POSITION PAPER

ICETEA PLATFORM

BRING DAPPS TO REAL WORLD

version 0.1 (draft, subject to change)

26 June 2019

thi@icetea.io

Trada Tech

Table of Contents

1. Abstract	4
2. Introduction	4
3. The Problems of Existing dApp Platforms	7
3.1 Unacceptable UX	8
3.2 Lack of Essential Services for Developers	9
3.3 Poor Scalability	9
4. Characteristics of a Good dApp platform	11
4.1 User Experience	11
4.2 Technical Characteristics	12
5. Icetea Platform Architecture	12
6. Icetea Platform's Solutions	14
6.1 UX-enabling Features	14
REGULAR ACCOUNT	14
EXTERNAL PAYER	15
6.2 Essential Services	16
ACCOUNT PERMISSIONS	16
ASSET INHERITANCE	17
NAMING SYSTEM	20
DECENTRALIZED GATE	21
PRIVACY COMPUTATION	24
DECENTRALIZED CHATBOT	25
6.3 Scalability Solutions	27
ROBUST CONSENSUS ENGINE	27
PARALLEL TRANSACTION EXECUTION	28
SIDECHAIN AND AUTONOMOUS AREA	29
6.4 Developer-friendly Features	30
PROGRAMMING LANGUAGES & TOOLS	30
REUSE OF EXISTING LIBRARIES	32
ENHANCED DEBUGGING	32
7. General Concepts	33
7.1 Asset System	33
7.2 Platform Economics	34
7.3 Smart Contracts	35

FORMAT AND EXECUTION ENGINE	35
CONTRACT TYPES	36
DEPLOYMENT	36
EXECUTION	37
CONTRACT SAFETY	39
VERSIONING	39
7.4 Light Clients	40
7.5 Cross-chain Communication	40
7.6 Governance	40
7.7 Migration from Ethereum	41
ASSET MIGRATION	41
CONTRACT MIGRATION	42
Disclaimer	43

1. Abstract

Blockchain is a useful and revolutionary technology which is getting a lot of attention recently. However, despite much anticipation, no significant blockchain adoption has been observed.

Icetea is a platform designed to enable the *first useful and widely-adopted* decentralized application (dApp). In the long run, it aims to be a viable decentralized alternative to cloud computing platforms.

The blockchain industry is yearning for a first 'killer app' to validate this technology. Although plenty of blockchains exist with billions of USD invested, such an application remains elusive. If Icetea can do that, it is a game-changer.

In this paper, we will discuss the market potential as well as current limitations of dApps, architecture and operation of Icetea Platform, and how it will resolve those challenges.

2. Introduction

In essence, blockchain is the technology which offers a new way to freely and safely store and transfer assets and data, bypassing the middlemen. People can use verifiable tools instead of depending on institutions. This has the potential of huge impact on the structure of our society because, currently, organizations (governments, banks, big companies, etc.) often utilize their role as middlemen to control access, impose rules, and gain insights about people against their will.

There are 2 main categories of public blockchain use cases: Payment and Application Platform.

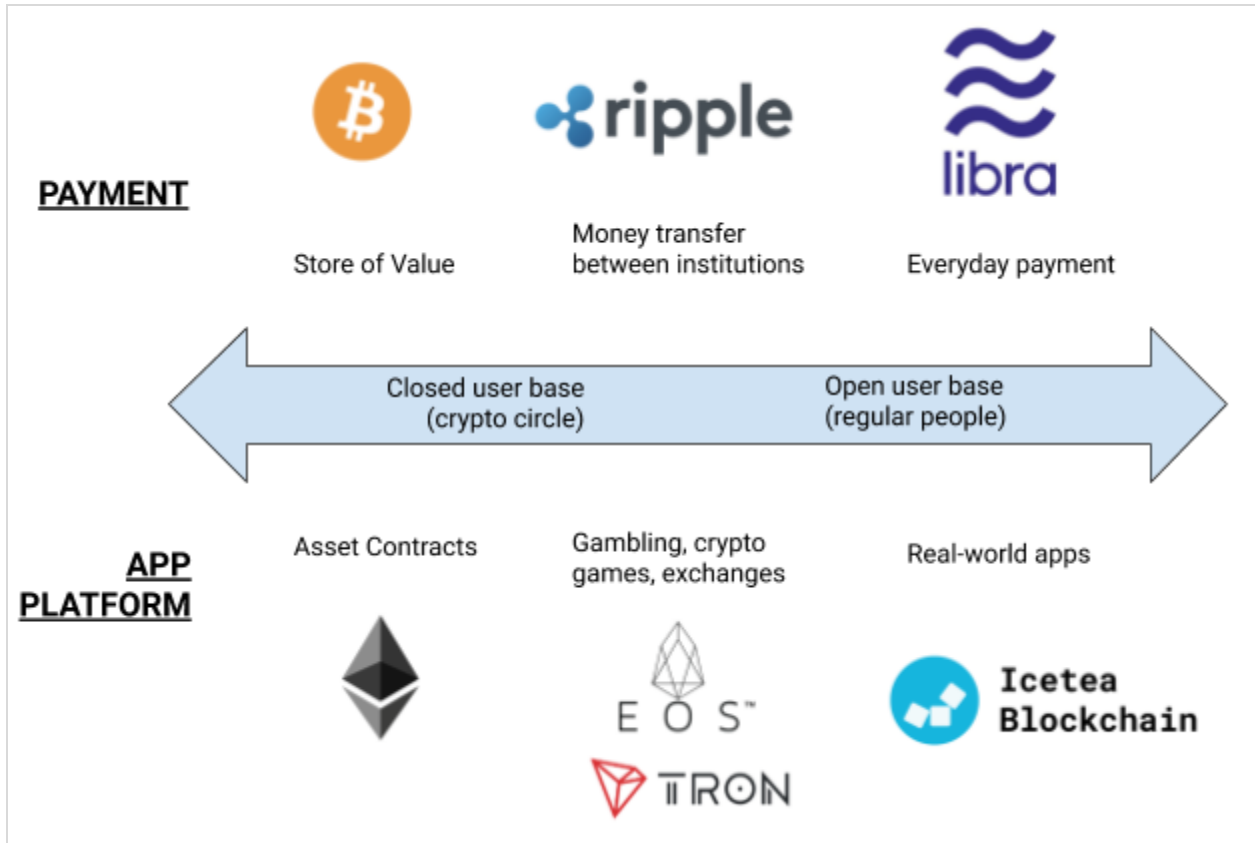


Figure 1: Use cases of public blockchain and its players (Libra is not a public blockchain but it plans to be one in the future)

Each of these categories faces several challenges. While the biggest challenge for cryptocurrency payment is legal issues, challenges for application platform are the technical constraints of the blockchain itself.

According to DAppRadar as of June 26, 2019, current leading dApps across all well-known blockchains have only some thousands of daily users. Most of them are gambles, trivial games, exchanges, and are used only by the small and closed crypto circle.





















#	Name	Category	Protocol	Users 24h
1	 TRONbet	Gambling	 TRON	4.3k -0.86%
2	 My Crypto Heroes	Games	 ETH	2.7k +0.07%
3	 EOS Knights	Games	 EOS	2.7k -1.06%
4	 HyperSnakes	Games	 ETH	2.5k -28.25%
5	 HyperSnakes	Games	 TRON	2.4k -6.18%
6	 Endless Game IOST	Gambling	 IOST	2.3k +24.41%
7	 MakerDAO	Other	 ETH	2.1k +14.27%
8	 EOS Dynasty	Games	 EOS	1.9k -11.08%
9	 RocketGame	Gambling	 TRON	1.8k +85.61%
10	 TronTrade	Exchanges	 TRON	1.5k +10.14%

Figure 2: Top 10 dApps, according to dAppRadar.com on 26 June, 2019

IceTea Platform's sole goal is to enable the first useful and widely-adopted blockchain application - one that is used by 'regular people'. Such an application would validate blockchain technology, infuse trust, and open the door to a new era of dApps.

The final, long-term goal is to offer a viable alternative platform comparable to cloud computing application platforms, then developers could choose either centralized or decentralized technology, or combination of both, depending solely on the value of

application data and trustworthiness of relevant parties, not the mature and usability of the underlying technology.

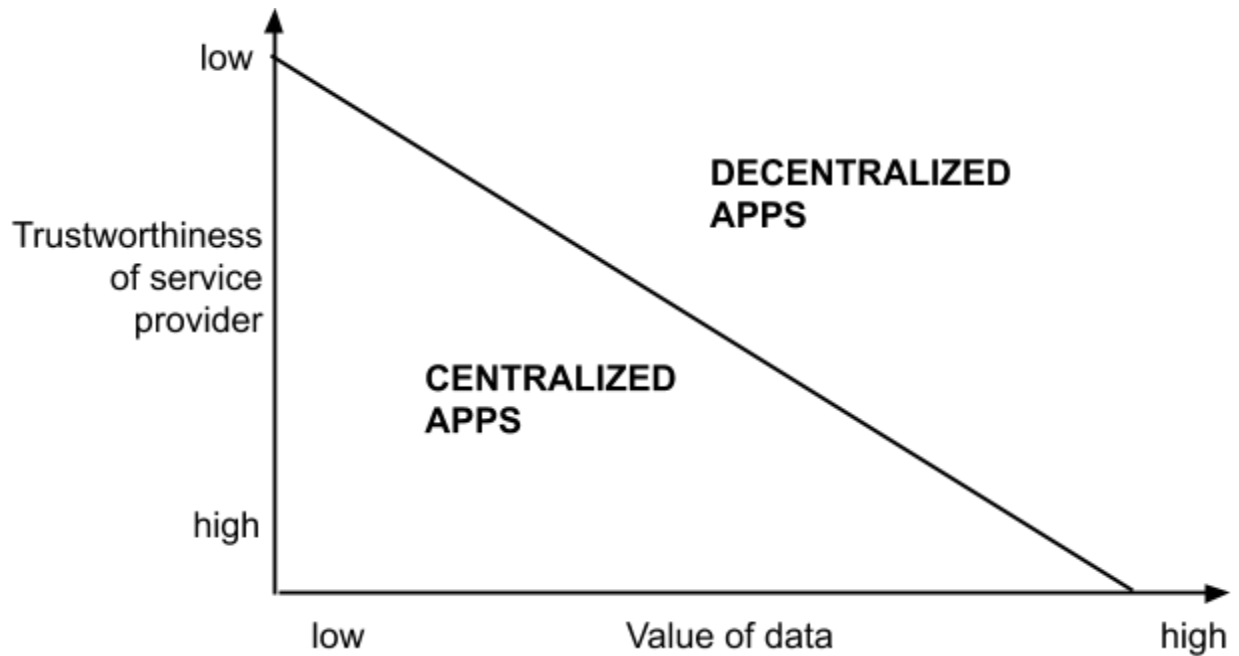


Figure 3: Areas of application of centralized vs decentralized platform

Nowadays, users are becoming more and more sensitive about privacy as well as true ownership of their data and assets. People in many countries are reluctant to use and start to boycott products which abuse their data. This creates a huge opportunity for decentralized, trust-free apps to emerge and replace current data-monster, trust-abusing centralized apps.

We are aware of Ethereum and their quest for a purely decentralized world computer. While their works are respectful and inspiring, we believe that Icetea Platform's agile and pragmatic approach, which attracts and gets 'regular users' feedback early and often, would also greatly benefit the blockchain industry. Therefore, Icetea Platform and Ethereum are not competitors but complement one another.

3. The Problems of Existing dApp Platforms

Making a widely adopted centralized application is very hard, yet making one on a decentralized platform is much harder. While the benefit of blockchain is still abstract

and vague to users, the hassles it introduces are *very* concrete and immediate. We believe that eliminating the hassles while still keeping the benefit is the only way to mainstream adoption.

Existing blockchains have explored a variety of experiments, but there remains 3 substantial obstacles to overcome.

3.1 Unacceptable UX

Let's first review the details of these hassles.

- Users have to learn new and foreign concepts, like private key, mnemonic, wallet, blocks, transactions, cryptocurrency, gas price, digital signature. All those concepts are not directly related to the application from the user perspective
- Users have to buy cryptocurrency to pay for the application from the start. There's no free, trial, or in-app-purchase mode. Furthermore, they have to go to some exchange service to buy coins to pay. Most of the regular users perceive crypto exchanges as risky and cumbersome.
- Users have to manage the keys themselves. There is no 'forget password' feature. This puts much burden on users. In addition, they are continuously asked to confirm to approve or 'sign' something that's hard to understand.

While it is fine to introduce new concepts and new ways of doing things to users, it is expected to improve the user experience instead of dramatically degrade it. The UX of current dApps shuts the door upon users from the first step. Without users, all other blockchain solutions might be wrong or at least ineffective because they are trying to solve *imaginary problems*.

The unusable UX of blockchain apps is *not the matter of dApp UI design*. It lies deeper at the core blockchain level as part of the blockchain trilemma which we will discuss more at a later section.

In reality, even a free and user-friendly application still has a hard time to attract users, let alone an dApp full of friction.

All those hassles overshadow the benefit of blockchain.

3.2 Lack of Essential Services for Developers

Decentralized code runs on every node. This leads to several requirements.

- It has to yield the same result when running on every node (i.e. deterministic)
- It has to finish within a limited amount of resources and time
- Because the code runs on the same environment which manages all the blockchain state, it must run in an overly strict security sandbox

This makes decentralized code very limited in capacity. It cannot utilize advanced programming languages and container's features. It cannot reuse most of the abundance of existing packages designed for centralized apps. Debugging capacity is very limited. As a result, it is hard and time-consuming to make any non-trivial dApp.

In addition to that, essential services which most of non-trivial applications require are missing or immature, and scattering. Those services are:

- Authorization (permission configuration)
- Off-chain data access
- Privacy of input/output data and business-secret
- High performance computation
- Large data storage and query

This forces the developers to 'swim in the ocean'.

An adopted platform should free users from infrastructure and framework worries so that they can focus on the business logic of their own application.

3.3 Poor Scalability

Let's start with the blockchain trilemma made famous by Vitalik Buterin, founder of Ethereum.

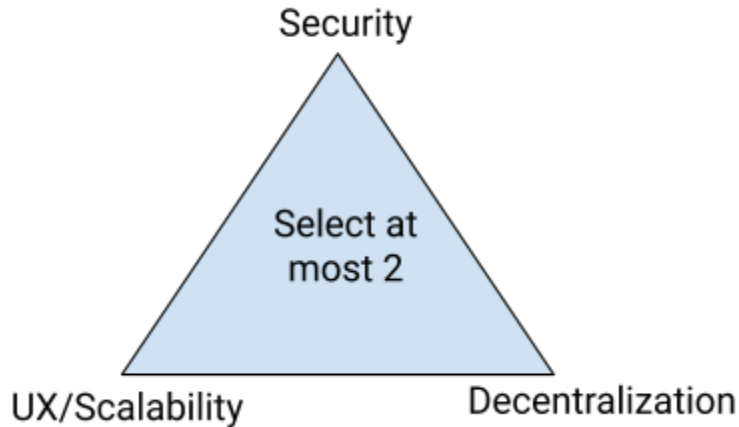


Figure 4: Blockchain Trilemma

We add UX to the Vitalik Buterin's original trilemma. As you can see in the previous section, increase security and decentralization will hamper UX. Scalability can be considered an aspect of UX.

This trilemma explains why current dApps are not scalable. Current version of Ethereum is very slow (~30 seconds for a transaction to get confirmed), has low throughput (15~20 transactions per second (TPS)), offers weak finality (transaction might still get 'reorganized' after confirmation), is expensive, and not scalable enough for simple games like CryptoKitties. Some recent blockchains utilizing PoS consensus mechanism have succeeded to scale to 1000~2000 TPS. Despite that huge improvement, this number is still very far from that of payment networks like VISA's, let alone popular real-world huge-traffic apps like Twitter.

The acceptable level of security, decentralization, and scalability varies depending on the domain of the blockchain (for instance, a general-purpose blockchain vs a domain-specific blockchain). However, for a specific blockchain, this trilemma is true as long as:

1. The blockchain is already fully optimized, to the point that any significant improvement leads to a trade-off
2. It runs as a fixed and closed system

The former is rarely the case, since blockchain technology is immature at the moment and contains plenty of room to optimize. In later section, we will introduce some

improvements employed by Icetea blockchain which dramatically improve UX and scalability without sacrificing either security or decentralization.

The latter could be solved by scaling vertically (adding more resources like RAM, CPU, etc.) and/or horizontally (clustering, sharding, sidechain, outsourced computation and storage, etc.). The question is how to implement those strategies effectively and efficiently.

4. Characteristics of a Good dApp platform

As we have pointed out the problems of existing dApp platforms, in this section, we will describe our vision about the characteristics of a dApp platform when done right, both from user's and developer's perspective.

4.1 User Experience

In general, the value that blockchain adds to the application must be sustainable and greater than the friction it produces. Thus, the platform should provide features to enable dApp to create frictionless UX.

The onboarding experience should be simple. Complex setup, if ever needed, is delayed so that users can try the app first.

The account and password/key management should be simplified. There should be convenient equivalents to "password forgetting" and biometric access features of centralized apps. If the app manages high-value assets, extra security is present only when needed, similar to 2nd-factor authentication mechanism.

Until cryptocurrency payment become prevalent, users should have options to seamlessly pay by traditional means like credit cards inside the app.

Applications should be performant and responsive. While it is OK to wait for 1 minute for money transfer, it is unacceptable to wait for several seconds for a chat message to get delivered. Regular apps must be fast.

4.2 Technical Characteristics

The platform should support different account types with different workflows for finance apps and regular apps.

The platform should enable large applications to divide its architecture into several areas, each requires different level of decentralization of data and computation.

For example, a decentralized chatbot could implement its input and output processing on main chain, while outsourcing the computation to a trusted execution service. A game might store user coins and virtual goods on main chain while implementing all of its game logic off-chain. A social network could be implemented with an autonomous area bridging to main chain.

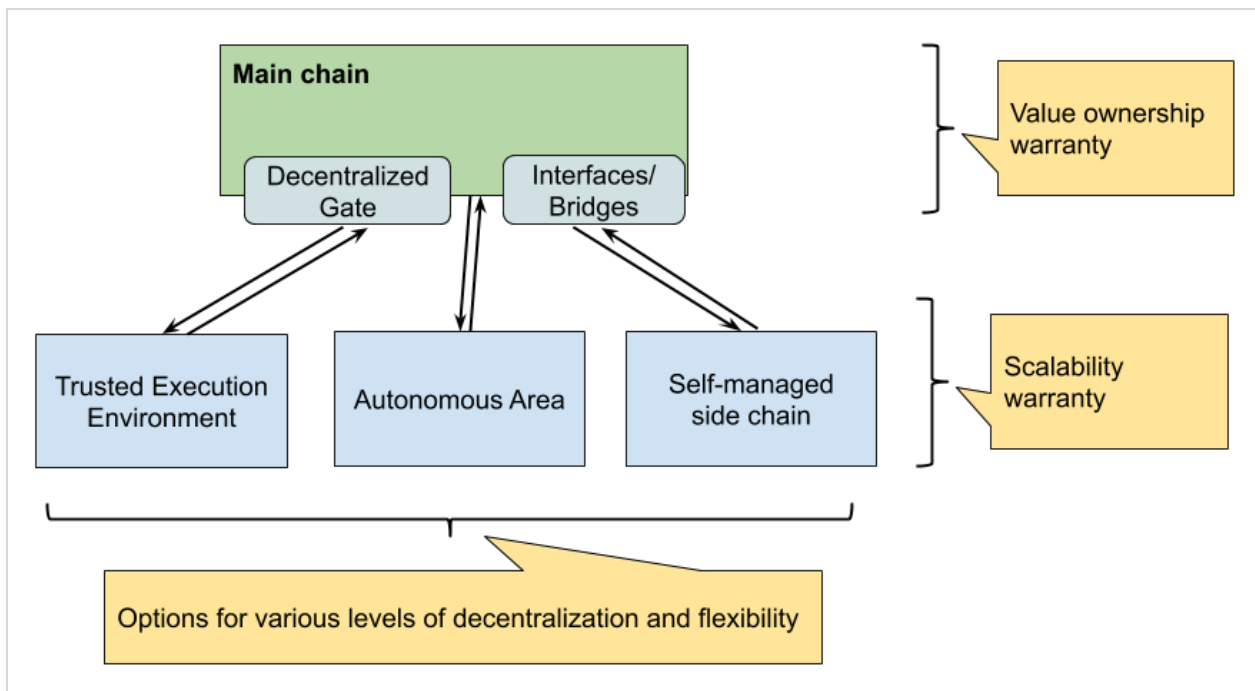


Figure 5: Application layers

5. Icetea Platform Architecture

To understand how Icetea Platform resolves those challenges, first we need to take a look at its system architecture.

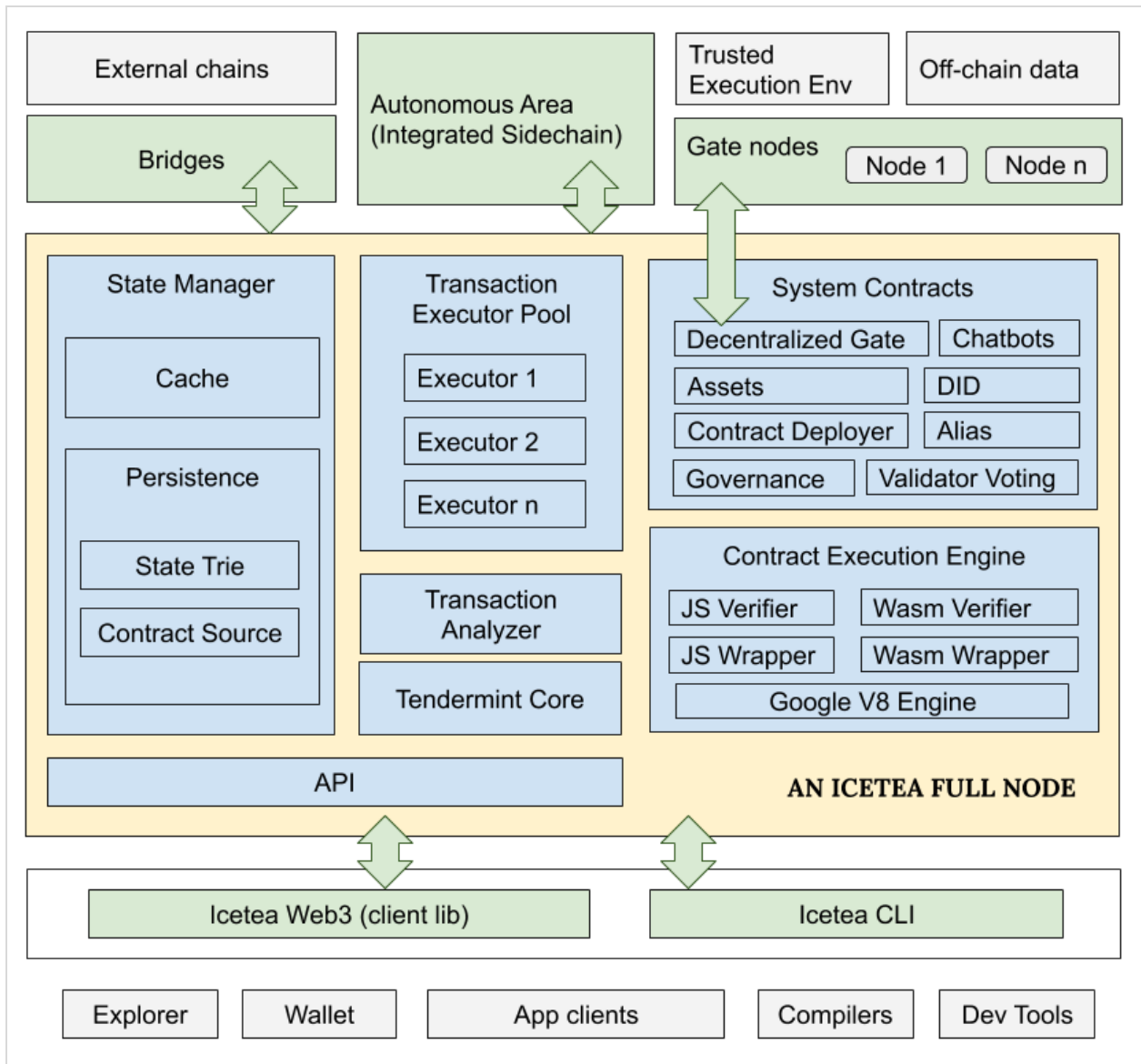


Figure 6: Icetea platform architecture

Icetea blockchain is an account-based (as opposed to UTXO) blockchain. Every transaction is a contract call. For example, to issue or transfer an asset, one calls the 'issue' or 'transfer' function, respectively, of the 'Assets' system contract. To deploy a smart contract one calls the 'deploy' function of 'ContractDeployer' system contract, supplying the contract's bytes. This way, the blockchain core can handle all transactions the same way, avoiding the maintenance overhead of the approach of using transaction types.

Under the hood, a full node includes the following primary components.

1. **Tendermint Core** for peer-to-peer, consensus and transaction raw data management. Tendermint also includes an RPC interface for clients to submit transaction and query state
2. A **Transaction Analyzer** to analyzer transactions, put them into groups, and dispatch them to the Transaction Executor Pool
3. A **Transaction Executor Pool** which is a process pool to isolate contract execution environment from the main process for safety reasons. Transactions can run in parallel by different processes if the Transaction Analyzer determines so
4. A **Contract Execution Engine** which executes the smart contract. It is a wrapper around Google V8 Engine, supporting both JavaScript and WebAssembly
5. A **State Manager** to manage global state. It persists state to a LevelDB and maintain a state cache similar to Redis. State can be accessed from other processes like ones from the Transansaction Execution Pool.
6. A bunch of **System Contracts** to enable Icetea Blockchain features like Asset Management, Alias (Naming) Management, Digital Identity, Permissions, Decentralized Gate, etc. Decentralized Gate is where a contract can communicate with external, off-mainchain world

The Icetea Platform specifies how the blockchain shall communicate with off-chain data sources, trusted execution environments, and other blockchains. It also defines a protocol for working with a type of integrated sidechain called 'autonomous area'.

6. Icetea Platform's Solutions

This section describes Icetea Platform's features that resolve current dApp Platform problems and make widely-adopted dApps possible.

6.1 UX-enabling Features

REGULAR ACCOUNT

The first feature of Icetea Platform is simple: distinguishing between 'bank account' and 'regular account'. Regular accounts are accounts which cannot store, receive, or transfer assets.

To understand why this is useful, let's review that of centralized apps. Bank accounts differentiate themselves from regular apps' account (like Twitter account). Imagine how inconvenient it would be if you merge your bank account and Twitter account? At that time, you'll have to re-login very often due to session timeout, and need to input OTP code and pay a small fee every time you make a simple tweet!

The distinction of account types provide extra information and guarantee to applications and wallets so that they can apply different security policies accordingly. For example, application can lengthen the session timeout or offer an 'remember me' option if it works with regular account. Wallets could provide a 'don't ask again' option when confirming signature permission. Furthermore, it becomes acceptable in terms of security for wallets to integrate some form of password management or biometrics access check for regular accounts.

Type of an account is baked into its address. Thus, one account cannot switch from 'regular' to 'bank' or vice versa. This ensures a stronger guarantee compared to implementation based on account settings.

Because regular accounts cannot store and transfer assets, it is only useful when combining with other features like 'external payer' and 'decentralized gate' which we will discuss in later sections.

EXTERNAL PAYER

Current blockchain requires users to either pay fees for transactions they make, or stake some amount of cryptocurrency first. This creates too much friction for user to get onboard. Icetea Platform solves this by allowing a dApp to pay transaction fees for its users. The dApp then could make money by other means, for example.

- It could be a free app, a donation-based app, or earning by some way unrelated to blockchain (like in-app ads, etc.)
- The dApp could grant usage rights to users based on their on-chain or off-chain activities. Example activities are proof of sharing on social media, proof of on-chain payment, proof of off-chain payment (prepaid cards, credit cards, bank transfer). Off-chain activities are available to the dApp via Icetea Platform's decentralized gate.

Technically, Icetea Platform transactions contain an extra optional field 'payer' pointing to another account or smart contract. If it is an account (external-owned account), the current account must have payment permission on behalf of the payer account (see more at Account Permissions section later). If it is a contract, the runtime will call a predefined function on that contract to see if it agrees to pay for this transaction.

6.2 Essential Services

ACCOUNT PERMISSIONS

Any non-trivial app needs some form of authorization. Icetea Platform has built-in support for a flexible permission scheme. It not only covers frequently-used use cases like multi-signature and 2-factor authentication out of the box but also is flexible enough for contract-based custom permission.

Icetea Platform implements this feature with a digital identity system contract. The contract is a registry mapping between address and account settings, which include permission settings.

The system supports the following types of permissions:

- Make transaction: make transaction on behalf of other account
- Payment: make transactions that cost money on behalf of other account
- Admin: can change account settings
- Custom: custom permission, pointing to a custom contract

Each type of these permissions can be restricted further by defined conditions based on transaction data. Permissions are attached to groups. Below is an example of a group.

Group name: middle managers

Description: managers who can pay from company account if the amount is less than 100 units

Permission type: payment

Permission condition: `asset.type = 'asset.tea' AND asset.value < 100`

Permission threshold: 1

Each group has a permission threshold. Each account added to the group is assigned a weight. Permission is granted if the total weight of accounts signing a transaction is

greater or equal to group threshold. For example, if you add 03 accounts to the group above.

Account address	Weight
account1	1
account2	1
account3	1

If the group's permission threshold is 1, either of account1, account2, or account3 alone could perform the action allowed by the permission. This is useful for cases when you want to add multiple keys into the same account in case you might forget one.

If the group's permission threshold is set to 2, at least 2 of the 3 accounts above must sign the transaction in order for it to pass permission check. This is useful for cases like 2-factor authentication, or 3 co-founders of a company jointly manage company fund.

Icetea Platform uses a single registry contract to manage accounts instead of the 'each account is a contract' approach. Only custom permission requires custom contract.

Although the permission system is very flexible, applications and wallets should try to make easy-to-understand permission-settings UI which focuses on common user needs, hiding complexity away.

ASSET INHERITANCE

Asset inheritance is a pain point of blockchain. If someone dies, no one else can access his/her account, and thus, all the assets of that account is permanently lost.

The permission system cannot solve this problem because it allows other accounts to access one account when the account owner is still alive. We need a mechanism which allows valid inheritors to access to an account after its owner was determined dead.

Asset inheritance workflow is also managed by the *digital identity* contract, the one that manage permissions.

The inheritance workflow consists of 03 steps.

Step 1: configure inheritors

In this step, the account owner will add a list of inheritors. He/she can also optionally configure the address of a certifier, who will certify the death.

Step 2: the inheritor claims inheritance permission

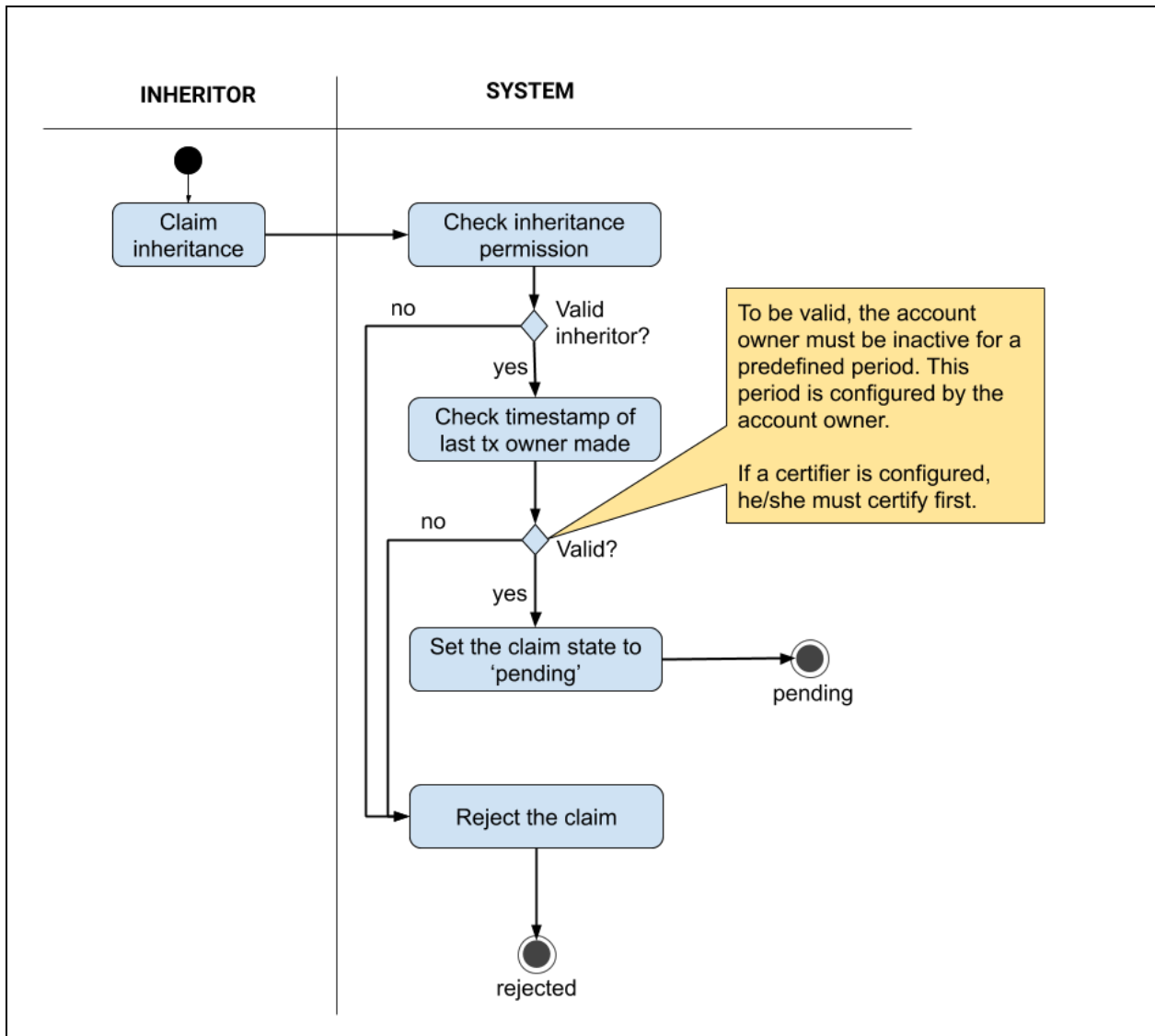


Figure 7: Claim inheritance

Step 3: Enter challenge period

When an inheritance claim passes validation, its state change to 'pending'. The inheritor has to wait for a predefined duration (called 'challenge period'). During this duration, if the account owner happens to be alive, he can reject the claim.

The value of challenge period is configured by the account owner.

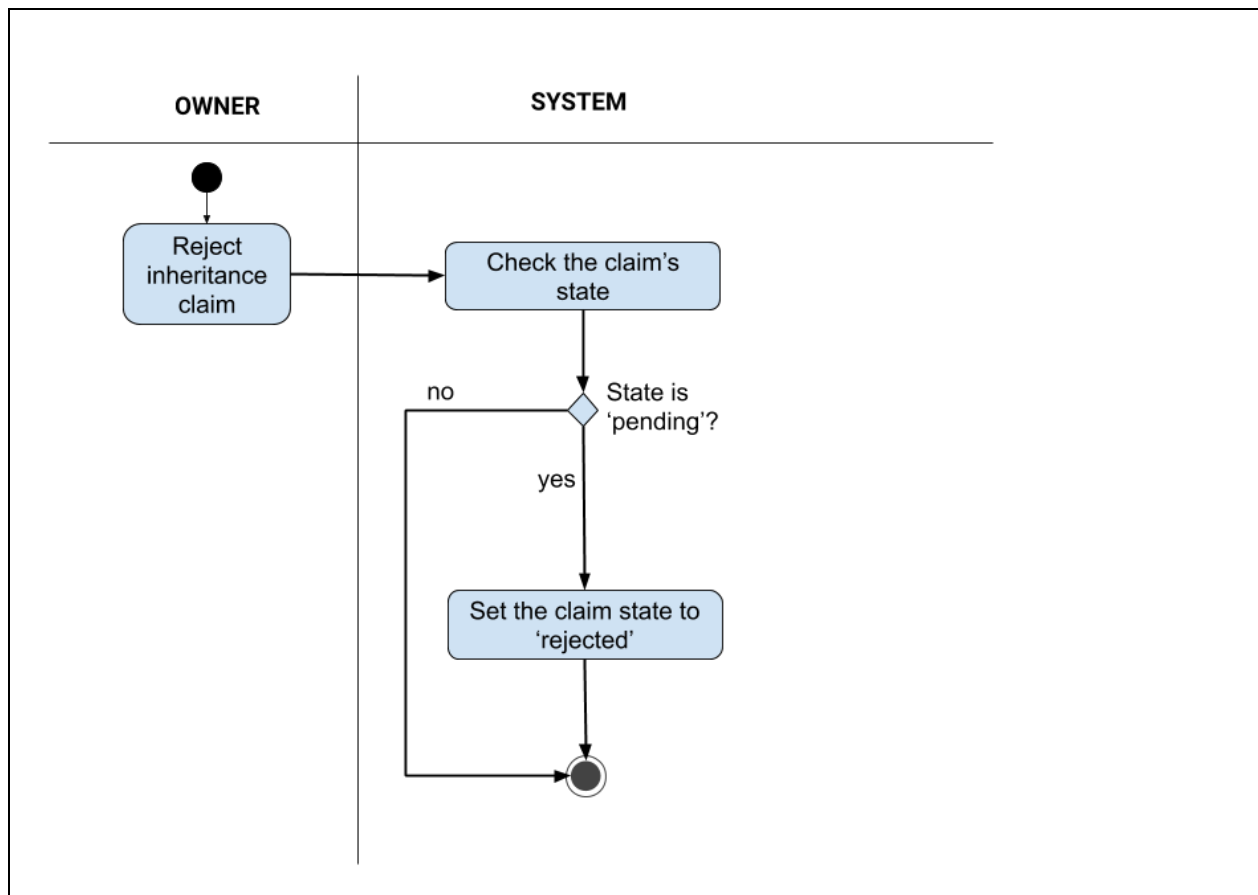


Figure 8: Reject inheritance claim

If a claim is rejected, the respective inheritor is locked for a predefined *lock period* before he can claim again.

Step 4: withdraw the money

When the challenge period passed and the inheritance claim is not rejected, the claimer now can execute his/her inheritance right. For example, he may spend on behalf of the owner, or withdraw all the assets to his/her own account.

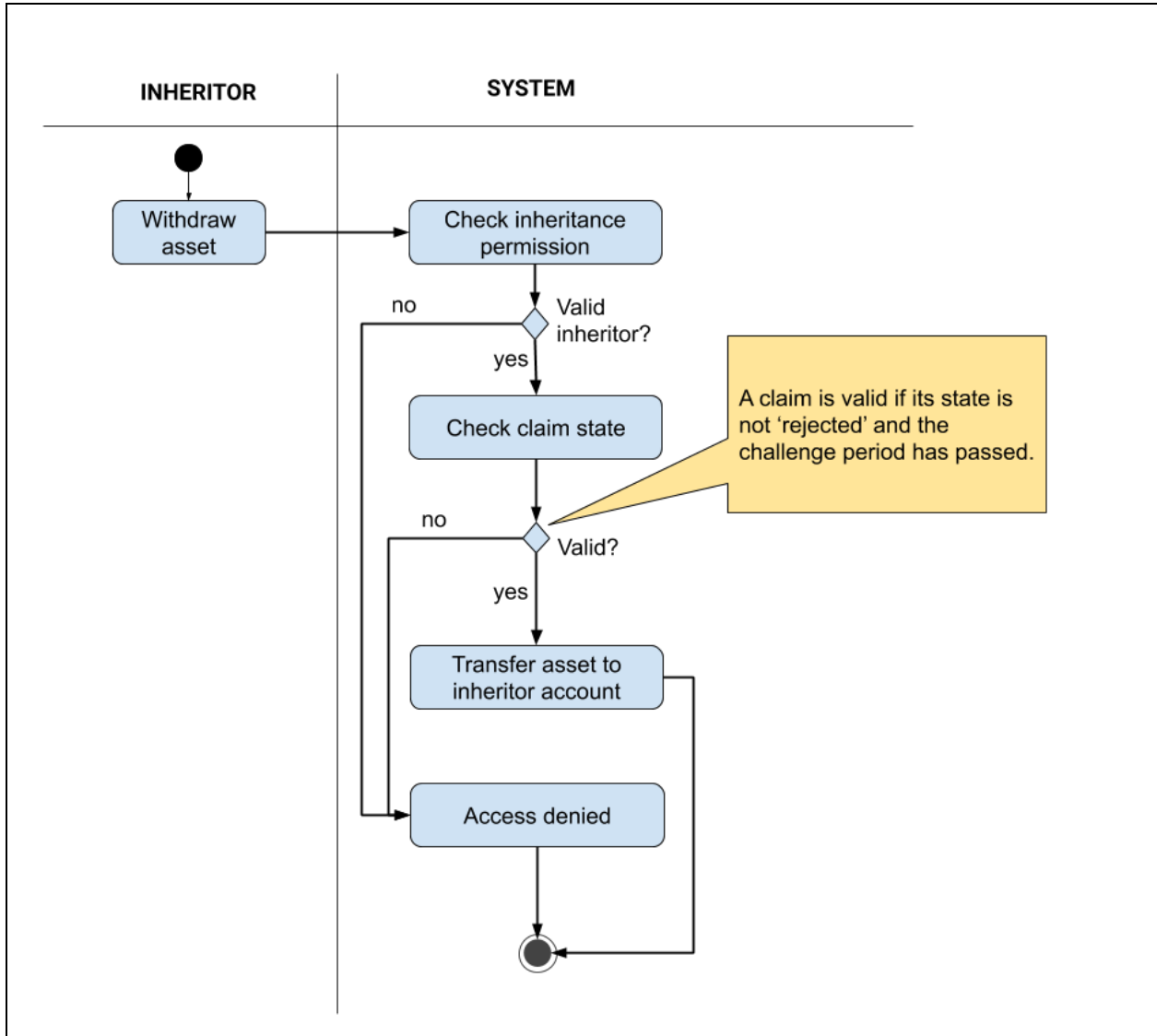


Figure 9: Execute inheritance right

NAMING SYSTEM

Icetea Platform includes a system contract for alias management. It support a namespace-based naming system similar to Domain Name System for IP address. One

can register aliases for accounts, contracts, assets. This makes it much easier for users to access blockchain features since they can use easy-to-remember names instead of long and complex addresses.

DECENTRALIZED GATE

Blockchain apps need to communicate with the outside (off-chain) world. Some examples.

- Apps that accept off-chain payment (bank transfer, credit cards, etc.) need to listen to payment events. This is a very important use case, and it works perfectly well with other Icetea Platform features like Regular Account and External Payer.
- Event prediction apps need the results of real life events to decide who predict correctly
- Apps that want to outsource part of its computation and/or storage requirements because of performance, privacy or business-secret reasons

Because of the deterministic nature of blockchain, smart contracts are not allowed to access the off-chain world. If it does, the result of such access could be different on each node, leading to inconsistent blockchain state. Technically, that leads to continuous blockchain forks, hence the blockchain is no longer usable.

Therefore, the blockchain must always be in passive mode. Any change made to blockchain state must be initiated from outside in the form of a transaction.

Many current dApps is designed so that some admins can input off-chain data into the contract. That creates a single point of trust for contract users.

Icetea Platform comes with a system contract called *Decentralized Gate* for contracts to communicate with the external world. In addition, because of Icetea Platform's powerful support for smart contract, third-parties could implement their own oracle solution or integrate with existing protocols like Chainlink or Oraclize if desired.

The following diagram describes the overview of the decentralized gate.

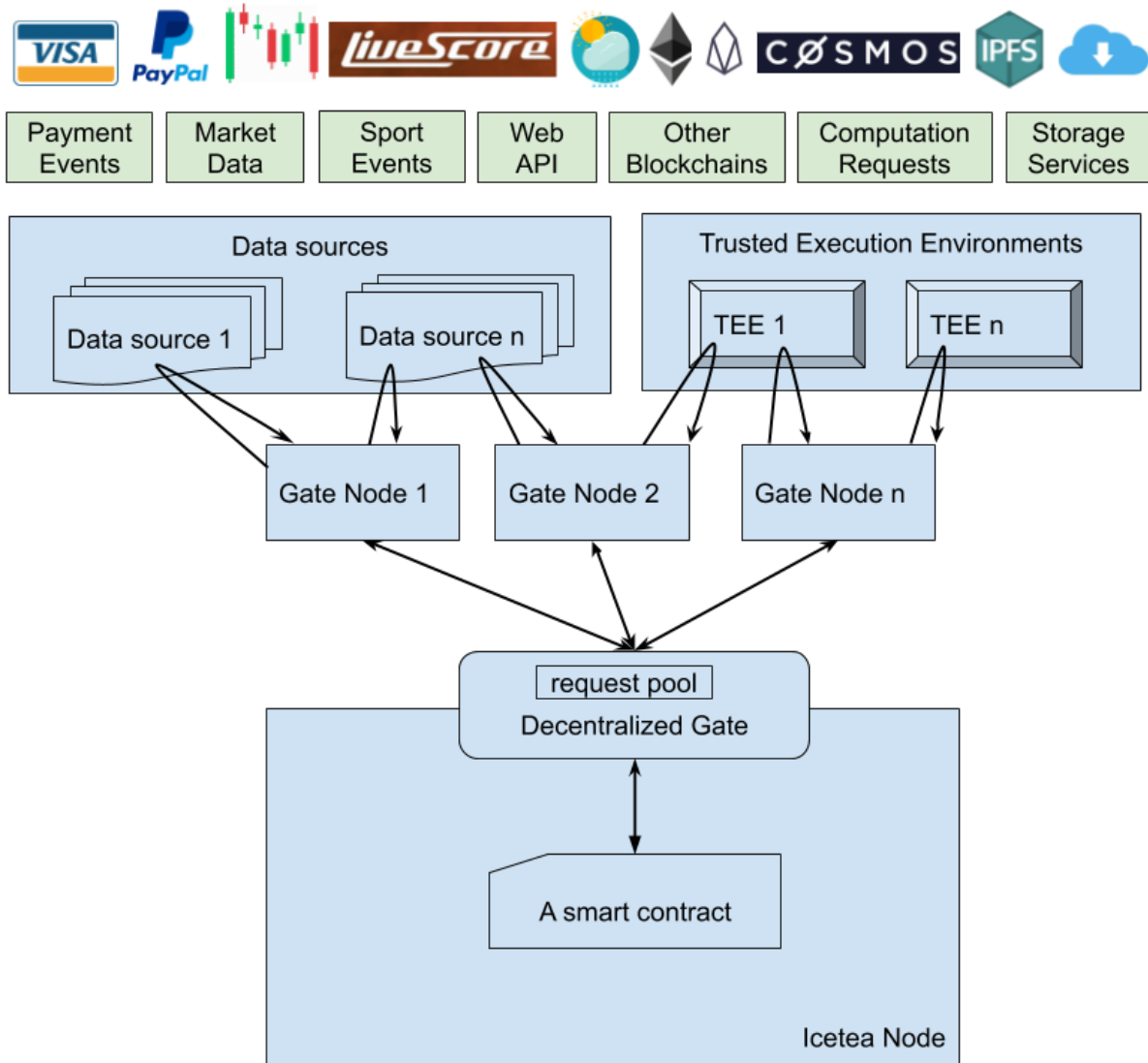


Figure 10: Decentralized gate overview

The decentralized gate is a 2-way gate. Contract not only can query external data, but also request execution like making random numbers, trusted execution (with privacy support), storage request (to IPFS, cloud services).

The Decentralized Gate achieves its decentralization by maintaining a list of *Gate Nodes*. Each gate node must deposit a predefined amount of assets when registering with the Decentralized Gate. When a smart contract sends a request through the gate, it attaches an amount of reward. The gate nodes will bid to gain the right to serve that request. Depending on the content and conditions of the request and the history and reputation of the nodes, the Decentralized Gate will select which gate nodes have the

right to serve the request. Upon gathering the results from all assigned gate nodes, the Decentralized Gate will validate, sanitize, and aggregate the result before setting final results to the requested contract. Reward is then distributed to the relevant gate nodes.

For example, a contract's request may look like this.

Type: data query
Data type: weather
Data specs

Fields

- Temperature
- Humidity

Where

- **Date:** today
- **Location:** Tokyo

Execution conditions

- **No of Provider:** 3 (at least 3 providers)
- **Reputation:** 4 (providers must have reputation ≥ 4)
- **Timeout:** 10 (10 blocks)

Aggregation options

- **Conflict:** average/reject/favor_high_reputation (what to do if gate nodes return conflict data)
- **Tolerance:** 2% (more than 2% will be considered an outlier and excluded)

If a gate node does not respond in a timely manner or if it provides invalid data, portion of its deposit is slashed.

The contract owner (i.e. the one who deployed the contract) could rate the quality of the returned data. That rating affects the node's reputation. Only the owner whose contract actually used the service can rate, one vote per request.

Gate node network is independent from Icetea node network. However, in phase 1, each Icetea validator is requested to run a gate node as well.

Decentralized gate is the main mechanism for integration with the outside world in phase 1 of Icetea Blockchain. In later phases, we will introduce the *Autonomous Area*, a specialized version of sidechain, which may be better suited for some of the off-mainchain computation use cases.

PRIVACY COMPUTATION

Icetea blockchain does not support privacy features on its mainchain. Privacy computation must be delegated to either a trusted execution environment (TEE) or a privacy-focused sidechain. Thus, they are layer-2 solutions.

However, Icetea Platform does provide a reference implementation of TEE. The development of a privacy-focused sidechain is left for third parties and community.

Using TEE, one can:

- Users can hide the input and output of a contract call, which may contain their sensitive data
- Developer can hide the algorithm of the computation. Possible reason might be that it is security-sensitive or business-secret

The following cannot be achieved with TEE.

- Users cannot hide transaction data that is not the input/output of contract calls (for example, information about who pays the transaction)
- TEE works with contract calls only. Other types of transactions like asset transfers and contract deployment are fully processed on mainchain and no part could be delegated to a TEE.

For those privacy requirements which are not covered by TEE, a possible solution is first transferring assets to a privacy-focused sidechain, performing the confidential transactions there, and optionally withdrawing the assets back to main chain. Those sidechains could utilize techniques such as zero knowledge proofs, ring signatures, etc. to protect user privacy.

The TEE works behind the decentralized gate. It supplies its public key when registering itself with the gate. The client and the TEE will use ECDH to derive a shared secret for secure data exchange.

The following diagram denotes the flow of exchange data between a client and a TEE.

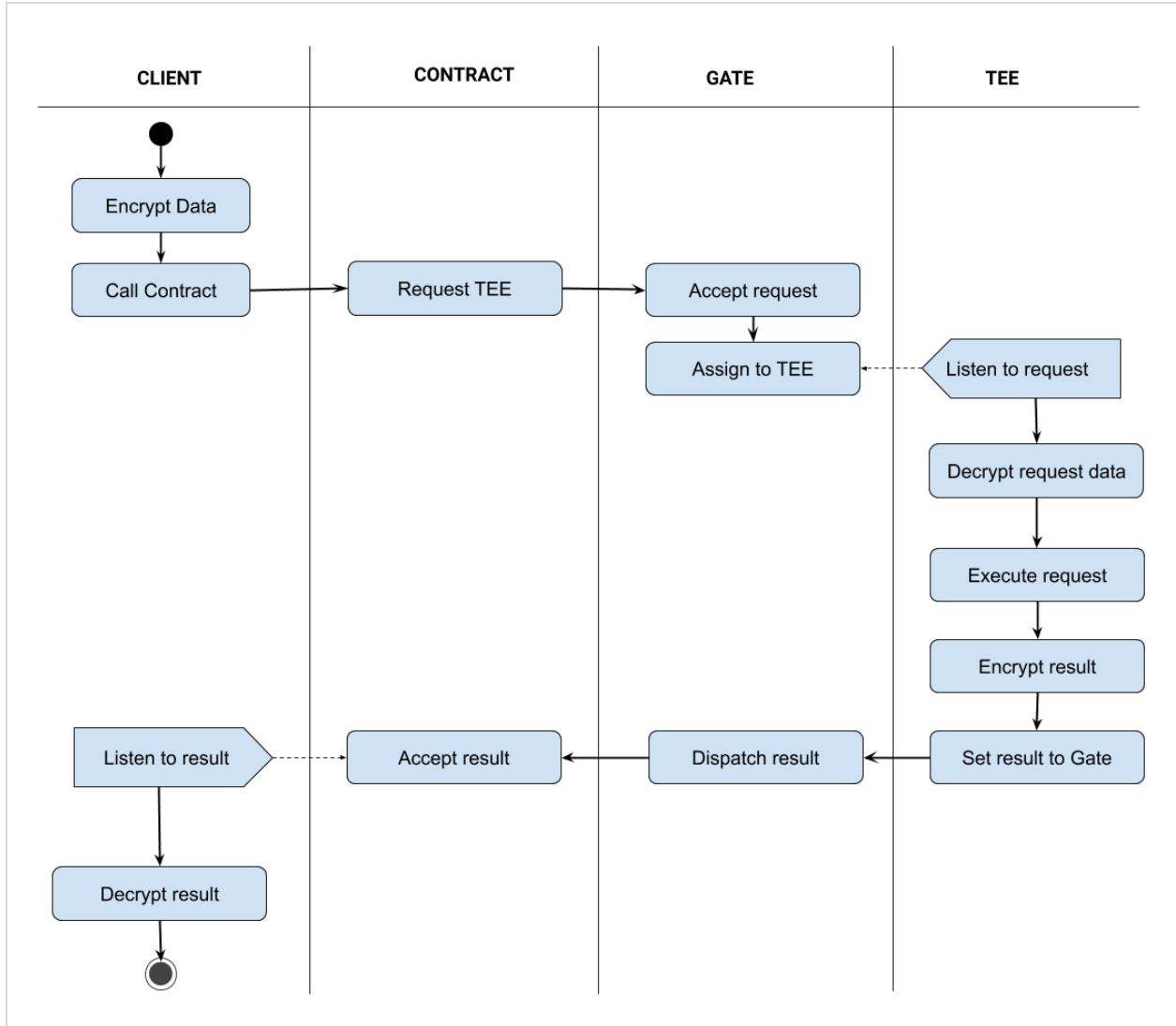


Figure 11: Data exchange between client and a TEE

It is not a requirement that a TEE node utilizes hardware trusted execution technology. However, contracts might specify such hardware requirement as one of their requests' execution conditions, thus leaving those TEE nodes out of eligible candidates for selection.

DECENTRALIZED CHATBOT

A current typical blockchain wallet does exactly what its name suggests: managing, sending, and receiving cryptocurrency. Some wallet features a 'dApp Browser' which is an embedded webview to run whatever webapps, even those which have nothing to do with blockchain or crypto.

In our vision, the wallet is the place to run all dApps - just like the browser is the place to run all webapps. In the future, there will be 'native dApps' run inside the wallet, not just web-based apps. Decentralized chatbot, a concept invented by Icetea, is a step toward that vision.

A decentralized chatbot is a dApp utilizing conversational UI and users can chat directly to it right from within their wallets. A dApp runs natively inside the wallet.

To understand how it works, image you chat to a smart contract directly from within your wallet, without an external app or UI. Icetea blockchain features a chatbot registry system contract (i.e. a *bot store*) where you could find your favorite bots and chat to them.

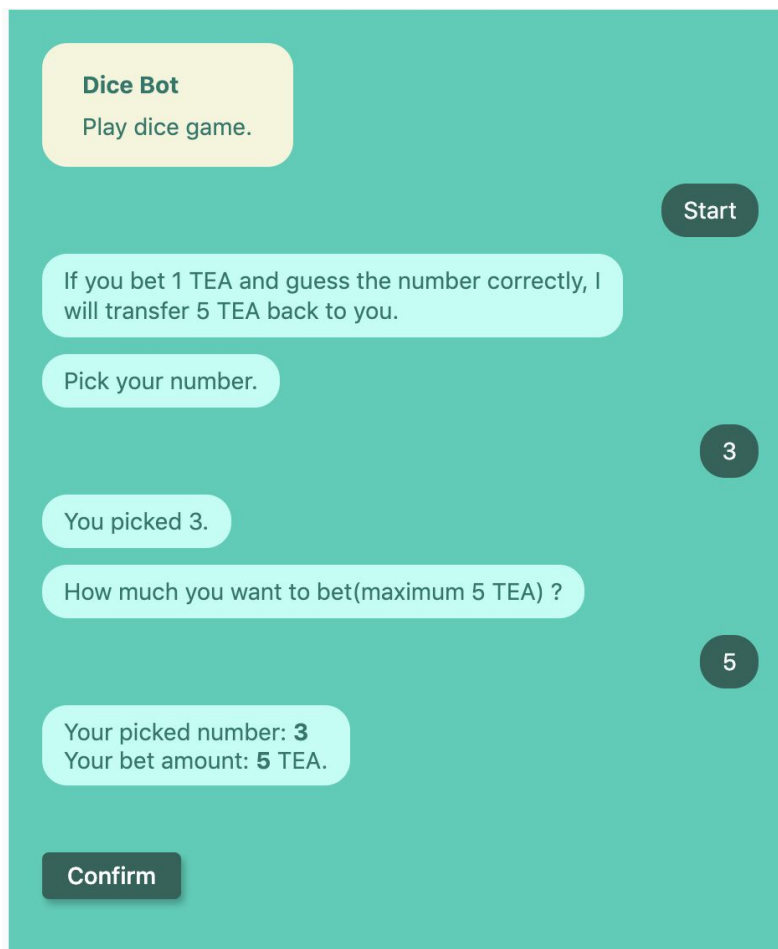


Figure 12: A sample chatbot

Under the hood, the chatbot is just a smart contract. The chatbot follow a predefined protocol so that users and dApp runners (e.g. wallets) can host them. Details about this protocol is described in a separate document.

6.3 Scalability Solutions

ROBUST CONSENSUS ENGINE

Icetea blockchain uses Tendermint Core as its peer-to-peer communication and consensus engine. It is the same engine which powers Cosmos Hub and Binance DEX and able to serve thousands of transactions per second.

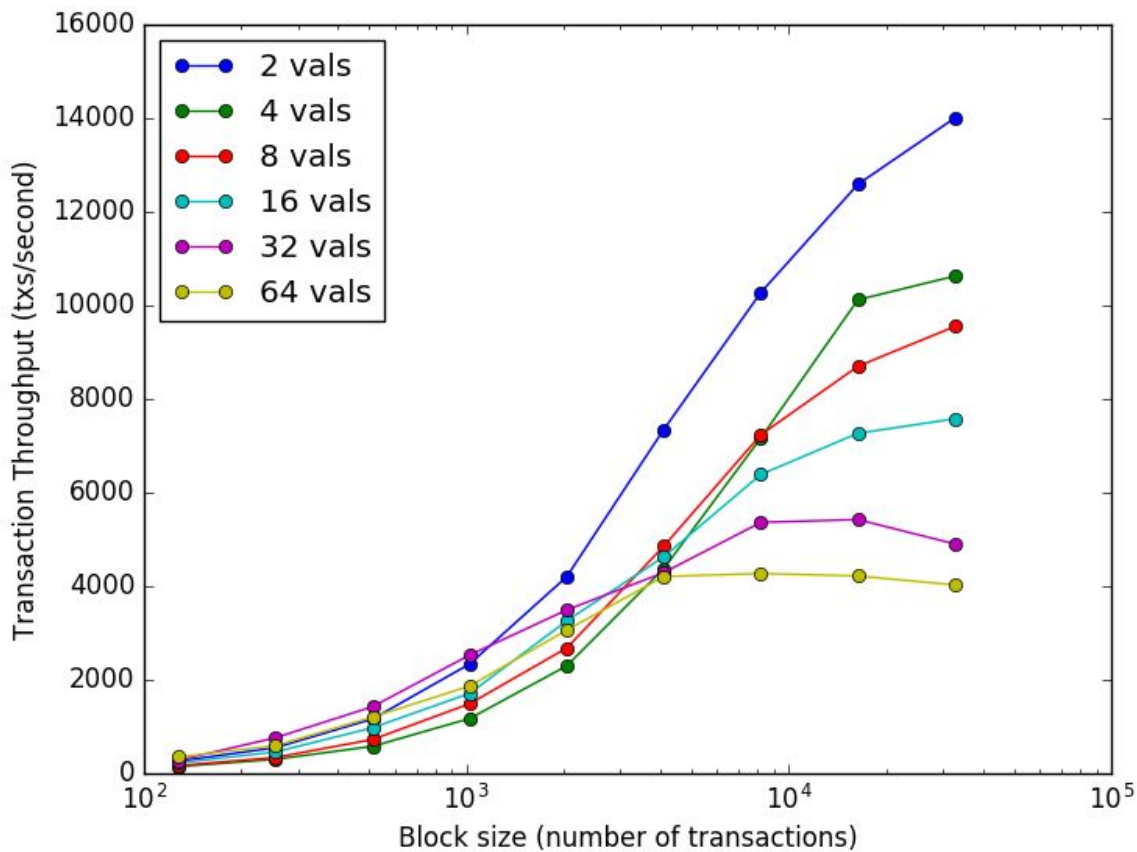


Figure 13: Tendermint engine can serve 4000 TPS on blockchain with 64 validators distributed across 7 data-centers on 5 continents, on commodity cloud instances.

Source: [Cosmos Whitepaper](#)

For more information on Tendermint, please checkout [its specification](#) and [paper](#).

Icetea blockchain features a Proof of Stake (PoS) system contract which allows stakeholders to vote for validators. The number of initial validators during phase 1 of Icetea blockchain is 50, tentatively. This number can be adjusted later on by voting following the Icetea blockchain governance rules.

PARALLEL TRANSACTION EXECUTION

Traditionally, a blockchain executes transactions sequentially. This is to avoid 'double spending' problem where two transactions spend the same amount of asset simultaneously. This makes the blockchain very slow if there are many transactions come at the same time.

Icetea Platform can execute part of the transactions of the same block in parallel. Furthermore, the decision whether or not to run in parallel depends on many factors, including the hardware specification of the node in question. Therefore, it is a requirement that even some nodes run transactions in parallel while other nodes run them sequentially, after the block commits all of those nodes must yield the same state.

In essence, if 2 transactions access separate state, or if both *read* a shared state, they could run in parallel. If one *update* a shared state while others access it, they must run sequentially.

In addition, the consistency of global state is critical to the correctness of the blockchain, so the runtime must ensure no parallel and conflicting accesses to a shared global state can occur. It cannot rely on the assumption that contracts are programmed correctly. Therefore, contracts are not allowed to use programming language level's locking constructs to proactively synchronize accesses to shared global state.

This leads to some execution strategies.

- **Optimistic Strategy:** assume that transactions won't access share state and thus always execute them in parallel. If shared state access detected during execution, the execution is canceled immediately and the runtime will re-execute the block in sequential mode.

- **Pessimistic-Dynamic Strategy:** at runtime, the execution engine keeps track of which state is being accessed by which transaction. If other transaction access the same state, the engine will make it wait until the current transaction finishes.
- **Pessimistic-Static Strategy:** each contract must declare which state it will access. By doing that, the system knows which transaction will access shared state and can organize them to groups in order to execute them in parallel or sequentially accordingly. If a contract access an undeclared state, the system will throw an error which reverts the transaction.

Icetea employs the *Pessimistic-Static Strategy*. All of the block's transactions go through the *Transaction Analyzer*, which will analyze and organize transactions into groups and dispatch them to the *Transaction Executor Pool* for processing.

Not every transaction that can be executed in parallel will be executed in parallel. For example, assets transfers might be executed sequentially and in-process, because the overhead of such parallel execution is often higher than the sequential alternative.

SIDCHAIN AND AUTONOMOUS AREA

Although Trusted Execution Environment (TEE) offers a powerful way to execute performance-sensitive and privacy-sensitive code, many large apps requires a different level of performance, decentralization, and flexibility.

There are 2 primary use cases of sidechains:

1. A domain-specific or feature-specific platform (e.g. a privacy-focused blockchain). We will call this the 'platform sidechain'
2. A single-application blockchain suitable for large and complex application. We will call this the 'app sidechain'

When designing a sidechain, the fundamental questions are:

1. Who is in charge of the sidechain's security and decentralization
2. How the mainchain can verify sidechain's transactions.

For current blockchains, security and decentralization of sidechain is taken care by the sidechain itself with no aid offer from the main chain. A smart contract on the main chain uses the sidechain's verifiable proofs (e.g. block headers) to verify its transactions. The question is that who submits the those proofs to the main chain?

Often times, it is the job of either a representative node from the sidechain itself, an off-chain relayer, or a network of incentivized relayers. Which type of relaying strategy selected depends on the trust level required by the problem in hand. In addition, one can reduce the dependence on the decentralization level of sidechain and the trustworthiness of relayers by designing a challenge mechanism like the ones used by Plasma and Lightning Network.

While you can implement all of the above-mentioned sidechain strategies using Icetea blockchain's smart contracts, the burden of ensuring decentralization on sidechain shy away many developers, especially developers of app-sidechains. They'd rather focus on their apps' logic than trying to recruit validators for their sidechains.

That is why Icetea Platform introduces the concept of *Autonomous Area*. An Autonomous Area is a sidechain whose security and decentralization are taken care by the main chain. In other words, the Autonomous Area 'recruits' its validators among the main chain's validators. An Autonomous Area candidate must implement a predefined interface (if it uses the Icetea engine, this interface is already included) so that the relaying job can be done automatically.

The Autonomous Area could and should come with one or some outside validators and only recruit some more from the validator network. That way, each Autonomous Area adds some validators to the whole validator network, each validator validate one or several Autonomous Areas.

Autonomous Area will be introduced in phase 2 of the Icetea Platform. We will discuss how it works in detail in a separate paper.

6.4 Developer-friendly Features

PROGRAMMING LANGUAGES & TOOLS

Icetea Platform support 2 smart contract formats.

1. JavaScript: the programming language that most developers are familiar with
2. WebAssembly: for performance-critical contracts

Because most of the current languages can be compiled/transpiled into either JavaScript, WebAssembly, or both, developers can use their favorite languages to author smart contract. This eases the learning curve for new developers.

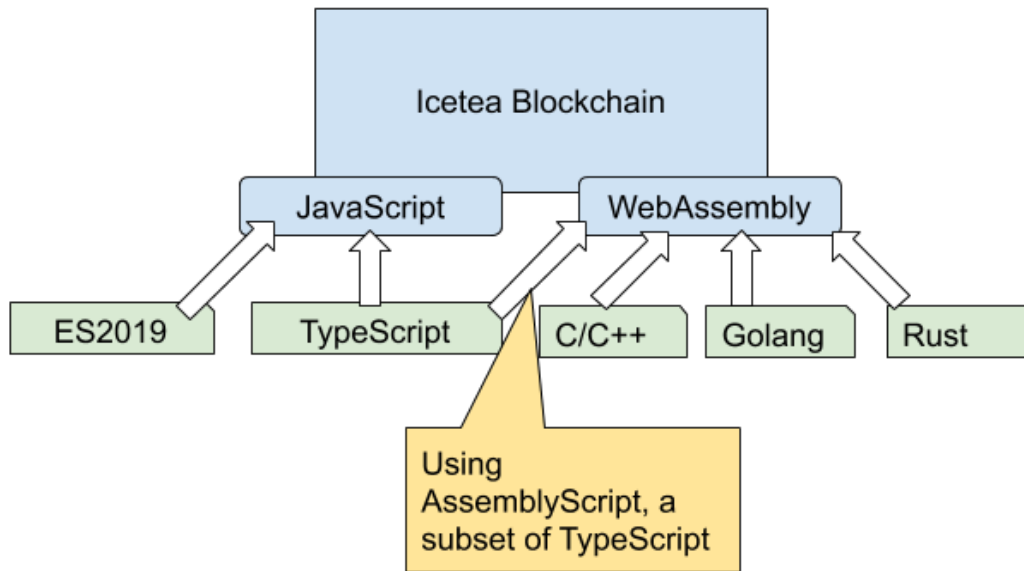


Figure 14: Programming languages for smart contracts

Although developers can write ‘raw’ code to communicate directly with the blockchain, we advise to use wrappers and tools to abstract away that interface. Icetea Platform comes with a tool named ‘sunseed’, which is a transpiler & bundler for JavaScript and Rust. One of its features is letting developers write contracts as ES6 classes similar to a Solidity contracts, using convention and annotations to denote metadata and interface.

Third-party can make similar tools, for example to support TypeScript, or to structure a dApp like a middleware-based [ExpressJS](#) app.

Because of the rich ecosystem around JavaScript and WebAssembly, it is possible for the community to create higher level tools like web editor, editor extensions, libraries and CLI for scaffolding, compiling, transforming, linting, deploying, testing, migrating from other blockchains, etc.

The Icetea documentation page maintains a list of available tools.

REUSE OF EXISTING LIBRARIES

Given the abundance of existing JavaScript and WebAssembly libraries especially NodeJS packages, it is very useful if developers could reuse them when making their dApps.

Prior to Icetea Platform, this kind of reuse is very limited because of 3 primary reasons.

- The blockchain works with a brand new programming language, hence, no existing libraries to reuse
- Existing libraries cannot be used because they were made for centralized apps and utilize many powerful features that the blockchain runtime does not support
- The library contains a large amount of source unused by the contract, thus cost unnecessary gas to deploy

With Icetea Platform, one can reuse most of the existing libraries in 1 of 2 ways.

1. **Whitelisted libraries:** these are a group of most frequently used and helpful packages like *lodash*, *hapi/poi*, etc. They are sanitized and verified by Icetea team and make available for every dApps. The Icetea documentation maintains a list of these packages.
2. **Other libraries:** developers can use *sunseed* tool to examine and polyfill these libraries. After that, there are 2 deployment strategies.
 - a. Deploy as a separate library: this is useful if the library is used by several contracts
 - b. Bundle the library into the contract source. *Sunseed* makes use of *tree-shaking* technique to remove all unused source and keep the contract source as small as possible

There are some libraries which cannot be reused though. For those libraries, *Sunseed* will produce an error message.

ENHANCED DEBUGGING

Debugging is a pain point for developers when working with Ethereum. On many occasions, developers are left desperate with an obscured 'transaction reverted' message with no further clues and must do the job of 'error guessing' in darkness.

With Icetea, developers can leverage NodeJS powerful debugging capacity or use Google [ndb](#) tool which improves debugging experience. The only thing developers have to do is turning on debug mode via either command line argument, config file, or environment variable. Then, Icetea client will output the verified, wrapped, and unminified contract source files to a specified folder. Developers can set breakpoints, step into, step over, watch variables within Chrome Developer Tools, VSCode, or your editor of choice with NodeJS debugging support.

In addition, when debug mode is on, developers can use 'console.log' for debugging purposes. It will output to Icetea client console, or redirect to a log file. This is convenient for quick debugging.

7. General Concepts

7.1 Asset System

Unlike Ethereum where each token has its own contract, Icetea Platform manages all assets via the Assets system contract. Think of it as a registry of all assets. It supports well-known asset types like fungible (similar to Ethereum's ERC-20) and non-fungible ones (similar to Ethereum's ERC-712) out of the box. Users can easily issue one by calling a function on the Assets contract, supplying asset parameters and sufficient issuance fees. No source code is needed.

In rare cases when you want to create a custom asset, you can deploy a smart contract extending the builtin asset with custom behaviors, or create a completely new type of asset.

There is no distinction between a *coin* and a *token*. All are assets and Icetea Platform treat them the same. At the time of mainnet launch, Icetea blockchain comes with a pre-issued asset named TEA. This asset is used for gas cost and fees. Future assets can also be used for gas cost and fees, by converting to TEA via the Exchanges system contract.

Compare to Ethereum token system, this asset system has some advantages.

- It allows the system to implement 'regular account' feature
- It allows the system to apply permissions to any assets' transfers

- Without the distinction between coins and tokens, apps can handle assets consistently and no need to create a wrapped token for native coin
- Contracts which do not expect to receive assets can explicitly refuse them. This prevents the problem of tokens get stuck in contracts as happened in Ethereum.
- Apps and Wallets can easily list all assets owned by a specific account. This helps users manage all of their assets conveniently
- Reduce the effort to code, audit, deploy hundreds to thousands of token contract similar to each other

Although it is possible for developers to create a token using a contract similar to that of Ethereum, doing so will exclude the token from built-in support for permission management, exchange, moving assets to sidechains, etc. Therefore, we strongly recommend using asset system to issue and manage assets.

7.2 Platform Economics

The Economics of Icetea Platform is designed with a single goal in mind: to sustain and develop the ecosystem in a stable and healthy manner.

There are the following platform stakeholders.

- Validators
- Providers of services, layer 2 solutions, tools
- dApp developers
- dApp users
- Holders (people who hold assets for future use or any reasons)

On-chain assets are used as a medium for storing and exchanging value. Initially, only the TEA asset can be used, but future assets may also be used if they are voted so by the governance body.

Each transaction execution costs some *gas* corresponding with the amount of resources required to execute it. DApp users can pay this gas directly or they can do something else that is valuable to the dApp (pay off-chain, promote, etc) and the dApp (smart contract) will pay it for them.

Transaction execution gas can be paid by:

- on-chain assets (e.g. TEA)

- other on-chain verifiable activities which are deemed good for the blockchain. For example, staking assets
- or a combination of both

Validators are the workhorses of the blockchain. Thus, most of the transaction fees will be awarded to them. The remaining is transferred to a reserved fund used to sustain the blockchain in the future. The ratio will be defined later and regulated by the governance rules.

To become a validator candidate, apart from satisfying hardware, software, and bandwidth requirements, a node has to deposit a required amount of on-chain assets. Portion of a validator's deposit will be slashed (i.e. forcibly move to the reserved fund) for wrongdoing or unfulfillment of validator responsibilities. Non-validators can delegate their assets to validators to share the award at the risk that if their delegated validator get slashed for misbehavior, they will also lose some amount of delegated assets accordingly.

On mainnet launch, part of the TEA supply is transferred to the reserved fund. This fund also receives part of every transaction's fees. It is used to cover asset staking benefit (i.e. free execution gas) and sponsor development bounties for improvement proposals voted by governance body.

The total supply of the TEA asset as well as its allocation will be decided and announced later.

At the beginning, all the economic parameters (e.g. ratios) might not be optimal, thus it will be monitored, analyzed, and then adjusted gradually by voting under governance rules.

7.3 Smart Contracts

FORMAT AND EXECUTION ENGINE

Smart contracts, or contracts for short, are at the heart of Icetea blockchain. Every transaction is a contract call.

Icetea blockchain supports 2 contract format: JavaScript and WebAssembly. Other languages can be compiled into either JavaScript or WebAssembly before deploying to the blockchain.

Under the hood, Icetea blockchain uses Google's V8 engine to execute contracts. V8 is currently the most powerful, advanced, and mature engine for both JavaScript and WebAssembly.

CONTRACT TYPES

There are 2 types of contracts.

SYSTEM CONTRACTS are contracts built into the blockchain. They handle system tasks like asset issuance and transfer, permission management, user-uploaded contract verification and deployment, etc.

Because system contracts are part of the system, they run in-process and in the same context with the system.

USER CONTRACTS are contracts that developers upload to the blockchain. Transactional calls to these contracts need to be verified and wrapped, grouped for possible parallel execution before dispatching to a process pool for execution.

Besides contracts, one can also deploy libraries, which can be called from other libraries or contracts but cannot be called directly from clients. Please refer to section *Reuse of Existing Libraries* for more details about libraries.

DEPLOYMENT

The following diagram denotes what happens when you deploy a smart contract.

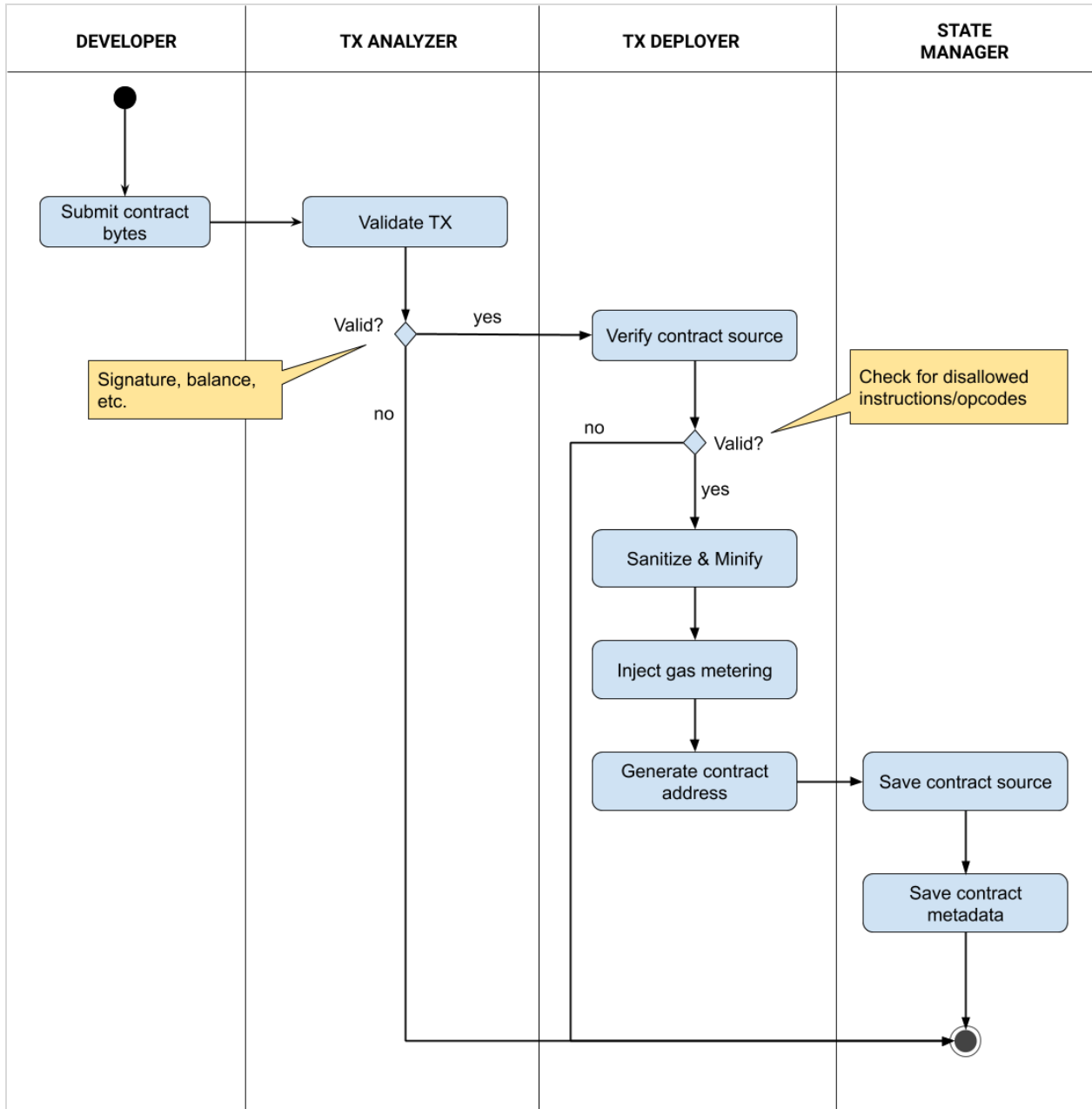


Figure 15: Smart contract deployment

EXECUTION

After deployment, you can send messages to the smart contract (that is, call the contract's functions). There are basically 2 types of contract functions: ones that do not change state and ones that does. Calls that change state must be sent as transactions which require consensus on the network and thus cost some gas. Calls that does not change state is gas-free and can return instantly.

The following diagram shows the flow of a contract call that changes state.

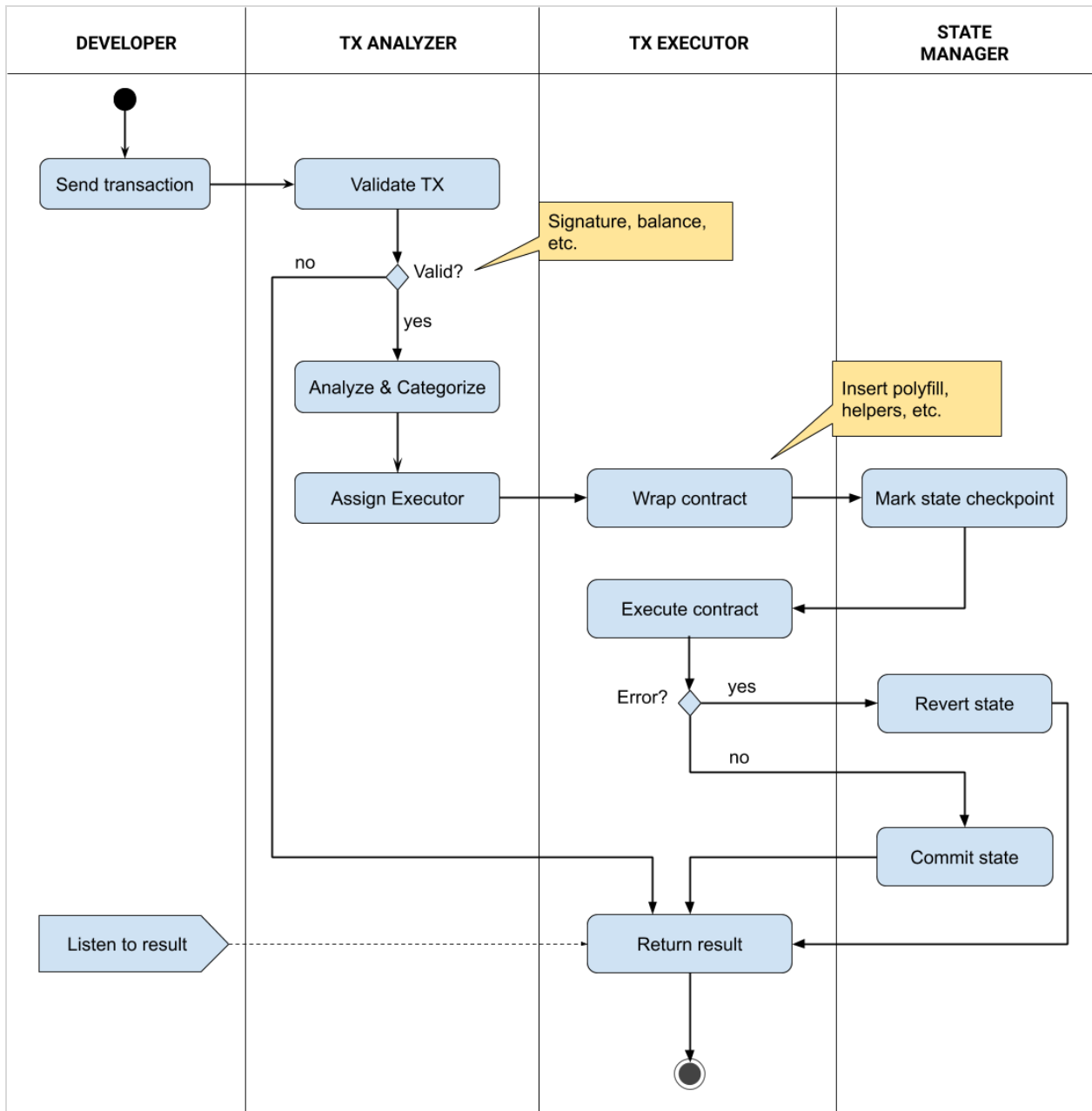


Figure 16: Smart contract execution

CONTRACT SAFETY

Contracts submitted by users can contain dangerous code, either unintentionally (bugs) or on purpose (exploits, hacks, vandalizing, etc.). Icetea blockchain performs the following measures.

- Gas-based strategy to limit the amount of resources a contract call can use. For more details about gas calculation, please refer to Transaction Fees section
- Scan for unsafe code: at deployment, the contract is scanned for unsafe code. This includes code that attempts to escape the execution sandbox and code that is deemed un-polyfillably indeterministic
- Wrap in sandbox: the contract is wrapped in an execution sandbox guarded against unsafe behaviors
- Running out-of process: the contract runs in a different process. If it crashes or leaks memory, the main process and other concurrently running contracts are not affected
- Validators can vote on malicious contracts to increase gas prices or ban them completely
- Dangerous patterns could be discovered over time and voted into unsafe code so that the unsafe code scanner (linter) can stop them in the future

It should be noted that the blockchain cannot detect contracts which attempt to scam users. Users should interact with verified and audited dApps only.

VERSIONING

Icetea blockchain supports versioning for both contracts and libraries. Version is an integer number, starting from zero. One can deploy an update contract or library to the same address. This will increase the version by one.

To avoid breaking changes, client can specify which version of the contract it wants to communicate with (default is latest version). A contract can also specify which version of the library it want to load.

7.4 Light Clients

Light client support is part of the Tendermint consensus and it works seamlessly with Icetea blockchain. The strong point of Tendermint light clients are that their proofs are *very succinct*. While a Bitcoin light client must sync chains of block headers and find the one with the most proof of work, a Tendermint light client just need to keep up with changes to the validator set. This makes it an ideal candidate for mobile and internet-of-things use cases.

7.5 Cross-chain Communication

Because Icetea blockchain uses Tendermint Core under the hood, it is trivial to connect Icetea to [Cosmos Hub](#). We would decide how and when to connect to it later.

Besides Cosmos, for direct cross-communication with other chains like Ethereum, EOS, Tronx, etc., corresponding 2-way bridges must be built. Icetea Blockchain's smart contract system is capable of building this kind of bridges. Because this is not specific to Icetea blockchain, this paper will not discuss in detail. Ones who are interested may refer to Kyber Network's [Waterloo Bridge](#) and other similar solutions for reference and inspiration. The advantage is that, due to the compactness of Tendermint light client proofs, you do not need to sync Icetea block headers to the other chain, just keep track of the changes of its validator set. This makes a 2-way bridge between Icetea and chains like EOS very cost effective.

One addition is that it is possible to utilize the Icetea's node gate network to relay information from one chain to others.

7.6 Governance

After releasing to mainnet and running stably, the governance of Icetea blockchain will be transferred to Icetea Foundation, a non-profit organization.

The governance rules of Icetea are designed with the following goals in mind.

1. Can handle emergency situations
2. Done on-chain and every stakeholder can vote

During childhood period (first 12 months from mainnet release), everyone can propose but only validators vote. Voting power of a validator is the sum of its own stake and the combined stake others delegate to it. Voting period is short. Part of the validator's deposit is slashed if it does not vote in time.

After childhood, the blockchain enters mature period. Voting period takes longer. Everyone can explicitly vote or delegate its stake to a validator. Changes to the blockchain are requested in the form of proposals. People must deposit sufficient amount to back the proposal before it becomes eligible for voting.

There are 3 types of proposals.

1. **Blockchain Parameter Changes.** This includes blocksize, blocktime, number of validator, ratios of transaction fees awarded to validator, whitelisted libraries, etc. Some parameters requires only validators to vote (e.g. blacklist a smart contract) while others open to all stakeholders (e.g. change compensation ratio). These parameters can be changed automatically after the proposal passes (i.e. this type of proposal does not require a software update)
2. **Emergency State Declaration:** put blockchain back to childhood state for a specified period of time. This is a special type of Blockchain Parameter Changes. The blockchain also supports a mechanism so that it can enter Emergency State automatically without the need of passing a proposal when certain vital indicators enter dangerous zone. However, the list of indicators are not yet determined at phase 1.
3. **Advanced Changes:** this includes both software update and update to the governance model itself. This type of proposal cannot be automated and must be presented in the form of an improvement proposal paper.

People can get their proposal-backing deposit back after the proposal passes, or it does not enter voting period.

7.7 Migration from Ethereum

ASSET MIGRATION

ETH or tokens on Ethereum could be moved to Icetea blockchain using a bridge as described in *Cross-chain Communication* section. The bridge could be 1-way (Ethereum to Icetea only) or 2 ways, depending on your specific use case.

CONTRACT MIGRATION

Because there are Abstract Syntax Tree (AST) parsers for both Solidity and JavaScript, it is possible to write a Solidity to Javascript transpiler. The *Sunseed* tool supports *Decorated JS*, a JavaScript class structure similar to Solidity contract, so it is easier to transpile Solidity to it. A sample tool for this could be found at <https://github.com/TradaTech/solidity2js>.

Disclaimer

This paper is for information only and subject to change. It is neither an advice nor an offer of investment. To the maximum extent permitted by all applicable laws, regulations and rules, the author, Trada Tech, or any of its affiliates shall not be liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including but not limited to loss of revenue, income or profits, and loss of use or data), arising out of or in connection with any acceptance of or reliance on this paper or any part thereof by you or any person to whom you transmit any part of the paper to (whether authorised or unauthorised by the author, Trada Tech, or any of its affiliates).