Insolar Technical Paper

Version 1.0

January 30, 2019

Insolar Team

www.insolar.io ✉ tech@insolar.io @insolario

Abstract

The Insolar Business Network Platform is designed to enable the efficient formation, management, and reconfiguration of business networks. It focuses on business needs, enables shared business processes across enterprises, is aligned with current enterprise IT practices, considers the long-term total cost of ownership goals, and is both decentralized and scalable. It marks a significant maturity of blockchain technology in which blockchain takes on enterprise scalability, security, and governance characteristics while hiding its complexity.

This paper sets out to provide an initial technical description of the underlying Insolar platform: its goals, key features, and core technologies. A detailed description of the Business Services will be presented in another paper.

The technologies of the Insolar platform enable near-linear scalability starting at an average of 10,000 transactions per second on a network of 20 to 30 nodes, with the capability to scale to millions of transactions per second on networks consisting of 100,000 nodes. The platform is business-oriented. Business network onboarding and participation are simplified because business network participants are not required to own or run their own nodes. There is a dynamic consensus, which allows finding the right balance between transaction value and processing costs. On-chain documents provide business logic flexibility, and varied governance of networks—private, permissioned, public, or hybrid—allows interoperability between different Insolar-based networks.

Disclaimer

This paper is intended for a broad audience. As such, the focus is mainly on the unique and defining features of the Insolar platform. The technologies described herein are currently under development and are subject to change.

Updates and additional details will be published on an ongoing basis.

This paper may contain forward looking statements including statements regarding our intent, belief or current expectations with respect to the technical description of the Insolar platform. Readers are cautioned not to place undue reliance on these forward looking statements. Insolar does not undertake any obligation to publicly release the result of any revisions to these forward looking statements to reflect events or circumstances after the date hereof to disclose the occurrence of unanticipated events. While due care has been used in the preparation of forecast information, actual results may vary in a materially positive or negative manner. Forecasts and hypothetical examples are subject to uncertainty and contingencies outside of Insolar's control.

This paper is intended for informational purposes only. Insolar does not endorse, guarantee or approve, and assumes no responsibility for, the accuracy, reliability or completeness of the information presented.

Insolar expressly disclaims all representations and warranties (whether express or implied by statute or otherwise) whatsoever. Insolar shall have no liability for damages of any kind, whether or not the likelihood of such damages was known by Insolar, arising out of the use, reference to or reliance on the contents of this paper.

Introduction

Blockchain-based technology has shown great promise for business application across many industries. The promise of a trust framework that reduces the transactional friction between business network participants and reduces the need for expensive, time-consuming third parties is compelling. Unfortunately, blockchain technology is still in an early maturation phase in which its complexity and lack of standards have limited its impact. However, its potential is great. Gartner predicts that blockchain will add $US3.1 trillion of new value to the world's economy by the year 2030.[1] In the short term, blockchain solutions face several hurdles before the technology can be integrated with existing enterprise infrastructures and business models. In this paper, Insolar outlines a new vision for blockchain, in which key features such as network capacity and consensus are addressed in a way that will enable shared business processes and optimize contractual transactions for enterprise operations.

The Insolar platform offers flexible governance, which will allow users to choose between using an Insolar public network—creating a domain with its own rules—or hybridizing the public network and a private domain. Private domains can be stand-alone private networks with their own servers or permissioned networks with resources provided by ecosystem members. This flexibility enables distributed business networks where everyone can select the configuration and domain features that best meet their needs.

Insolar's smart contracts are designed with enterprise developers' needs in mind. They are streamlined to work with business logic, and they are easy to develop using Golang and, soon, Java and other JVM-based code. Moreover, the Insolar platform provides data safety features that enable businesses to run their operations on public networks if they choose while being compliant with regulations such as the EU's General Data Protection Regulation (GDPR) and other country-specific or company-specific cryptography standards.

Insolar's vision is to facilitate seamless low friction interactions between companies by distributing trust, thus accelerating and opening up new opportunities for innovation and value creation. The Insolar platform is built to satisfy enterprise requirements by combining distributed and cloud technologies and dozens of industry-first features. The Insolar platform is the most secure, scalable, and comprehensive business-ready blockchain toolkit in the world. Insolar's goal is to give businesses access to features and services that enable them to launch new decentralized applications quickly and easily, whether they need a minimum viable product or full-scale production software and to integrate those applications with existing systems.

---

[1] Panetta, K., "The CIO's Guide to Blockchain," *Smarter With Gartner*, July 13, 2018, https://www.gartner.com/smarterwithgartner/the-cios-guide-to-blockchain/

Contents

1. Background

Today's business landscape is becoming increasingly interconnected: 36% of companies are connecting to and sharing data with twice as many stakeholders (customers, partners, vendors, etc.) than they were just two years ago.[2] Existing systems were not built for these demands; as a result, they are severely limited by issues such as interoperability, speed, scalability, governance, and onboarding costs.

In addition, 80% of corporate data resides in heavily guarded silos, meaning that a lot of time and resources are wasted moving, aggregating, cleansing, and verifying data.[3] This creates an informational ecosystem where data is not only fragmented, incomplete, slow, and costly to access, but is also prone to integrity decay. The consequence of this is considerable transactional friction leading to high barriers for efficient business interaction.

Blockchain technology is the most promising solution to this data connectivity problem. Traditional electronic document interchange solutions fall short of real-world enterprise needs. Even when they come close to addressing these needs, there are overly expensive investments required in IT labor, training, and infrastructure.

Even when a company's internal operations are running at near optimal efficiency, external interactions are most certainly not. Blockchain technology can address this problem. For instance, since most of the disputes arising from external interactions are said to be resolvable objectively, using blockchain to execute preagreed smart contract conditions or enable proper and immutable documentation collection and automated verification will lead to efficient and effective automated dispute resolution.

Blockchain's core benefit of dramatically reducing connectivity and interaction friction between organizations is at the core of innovative disruptions such as decentralized and disintermediated business networks, novel asset digitization, and new forms of value exchange.

Although the potential for blockchain is vast, it must be recognized that today's blockchain platforms are still in the early stages of evolution. Too often, they are designed without real-world enterprise needs in mind, lack skilled developers, have a fragile service ecosystem, fall short of key compliance requirements, or are expensive to set up. Therefore, blockchain platforms are less attractive than many traditional data technologies in solving well-defined and understood business problems. Furthermore, many platforms revel in an excess of technical and narrowly

---

[2] Goldberg, M., "Are Data Silos Killing Your Business," Dun & Bradstreet, May 7, 2018, https://www.dnb.com/perspectives/marketing-sales/data-management-strategies-avoiding-data-management-silos.html

[3] Accenture Technology Vision 2018 [https://drive.google.com/file/d/1vWGEDihwleX81spsUQj8FwZTQZoCEjuq/view]

focused features that are not business-oriented, resulting in impractical and misaligned solutions. All these issues lead to an understandable lack of significant adoption by enterprises today, which unfortunately has had the effect of discrediting and undermining some of the market's confidence in the potential of decentralized systems. Figure 1 encapsulates these issues.

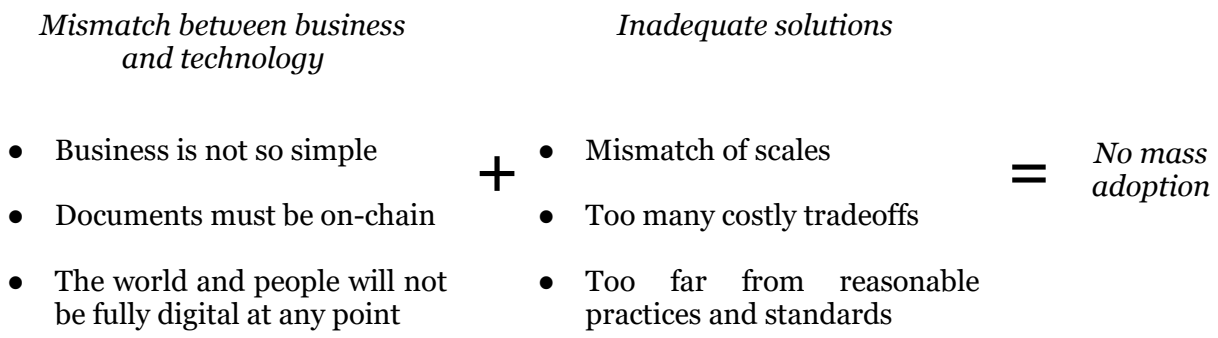


Figure 1. State of blockchain adoption.

Blockchain-based platforms and solutions need to move in a new direction to where technology-purism is balanced by business practicality. Four key realities about today's business context must be considered:

- the increasing complexity and intensive diversification of cross-business interactions;
- the emerging role of decentralization in decision making;
- the lack of resources to expand and maintain large internal IT teams compounded by a deficit of qualified professionals; and
- the inability of current IT systems to simultaneously provide scalability, trust, and security.

Insolar aims to address these four points. The scale-out architecture of the Insolar platform allows business customers to streamline cross-enterprise interactions with mindful risk management and long-term cost of ownership that is both predictable and affordable. It is also compatible with regulations on corporate, national, international levels.

2. Evolution of Blockchain

It is important to recognize how blockchain technology has evolved over the last decade. This evolution can be split into four generations (Table 1):

1. Blockchain 1.0: digital scarcity and distributed ledger (e.g., Bitcoin and other cryptocurrencies)
2. Blockchain 2.0: smart contracts (e.g., Ethereum and Hyperledger)
3. Blockchain 3.0: scaled decentralized applications (e.g., Cardano, Zilliqa, EOS, Aion)
4. Blockchain 4.0: enterprise-ready business network platforms (e.g., Insolar)

Table 1. Generations of blockchain.

	Blockchain 1.0	Blockchain 2.0	Blockchain 3.0	Insolar Blockchain 4.0
Consensus	PoW	PoW, PoS, PBFT	Many	Dynamic
Network cost	High	High	Medium	Low
Hardware cost	Very high	High to medium	Medium	Low
Consensus cost	Very high	High, Dependent on number of nodes	Medium, Dependent on sharding	Adjustable (per transaction) Value/Risk vs. speed/cost
Network	P2P Distributed	P2P Distributed	P2P Distributed	Federation of Clouds
Defining features	Transparent Ledger Hash function Block chaining	Public and permissioned Smart contracts Privacy	Sharding Better scalability Security Governance	High performance High scalability Complexity abstraction Customizable domains Advanced governance Advanced security Interoperability
Transaction cost	Very high	Medium	Medium	Adjustable (SLA)

Cryptocurrencies were the first blockchain-based applications to attract broad attention, mainly due to a decentralized design which allowed for transactions to be completed directly without

third-party involvement. In general, cryptocurrencies rely on a peer-to-peer (P2P) distributed architecture and serve as a form of digital cash. Early implementations relied on proof-of-work (PoW)—a very resource-intensive consensus model. Because of this, they were bottlenecked by the high costs of networking, node hardware, and transaction execution. As for their defining features, first-generation blockchains are characterized by a distributed transparent ledger, a hash function for block chaining, and PoW consensus mechanisms to enforce agreement and thus digital scarcity (solving the double spending problem).

Second-generation blockchains arose with the application of smart contracts. Although this generation saw the introduction of the proof-of-stake (PoS) consensus model, many version 2.0 blockchains still used PoW and, therefore, had high networking and consensus costs. Blockchain 2.0 allowed for the implementation of public and permissioned models, as well as ensuring that privacy requirements were considered. A smart contract itself is an autonomous piece of code that lives on a blockchain and executes automatically in accordance with predefined conditions, such as validation or contract performance enforcement. The smart contract's unique feature is its invulnerability to unfair behavior or unauthorized entities. A well-known example of a smart contract–focused blockchain is Ethereum. The inclusion of smart contracts has allowed for a reduction in general transaction costs, but it has also led to a loss of some of the core values provided by public blockchain networks. Hyperledger Fabric is a permissioned blockchain technology that can execute smart contracts (called *chaincode*) across private channels enabling private transactions between network participants.

Scalable decentralized applications (dApps) shaped the third generation of blockchains. dApps rely on decentralized infrastructure, and their code runs on a P2P network. They can have many different consensus models of medium cost, with dApp frontends able to be hosted via decentralized storage such as Ethereum. Blockchain 3.0 solutions feature better scalability capabilities with sharding (distributed database partitioning) as well as higher transaction speeds. Other defining features include flexible governance and improved security.

Of the approximately 50,000 blockchain-related projects proposed during the first three blockchain generations, less than 10% have been actively maintained. Their average project life span is about 15 months.[4] Although the project failure rate has been high, a critical mass of ideas has been generated. Much was learned from both the successes and failures of proof-of-concept (PoC) studies. Digitalization, Industry 4.0, and decentralization trends have also influenced

[4] Trujillo, J., et al., "The Evolution of Blockchain Technology," *Deloitte Insights*, November 6, 2017, https://www2.deloitte.com/insights/us/en/industry/financial-services/evolution-of-blockchain-github-platform.html

blockchain ecosystems, which has caused the technology to rapidly venture into use in cases beyond cryptocurrency.

The evolution to Blockchain 4.0 is related to simplicity and usability in real industry environments, especially in the case of digitized contexts. This includes sensors, automation, enterprise resource planning, data management, and the digitization of business areas. Increasing task automation means an increased need for safeguards, and this is where blockchain stands apart from other technologies. It strengthens business integration and provides for shared environments for business processes. Machine learning for predictive maintenance, supply chain management, document flows, and industrial Internet of Things (IoT) data collection are all examples of areas that can be empowered by Blockchain 4.0, as proposed by Insolar.

2.1 Comparison of Blockchain Platforms

Table 2 has been prepared by the Insolar Research team as part of its analysis of the performance of more than 20 blockchain platforms to help define the requirements for the Insolar platform. It presents a technical brief for each platform with highlighted strengths and weakness.

Table 2. Comparison of selected blockchain platforms.

Platform Parameters		**Insolar**	**Hyperledger Fabric**	**Corda**	**Quorum**	**Ethereum**	**EOS**	**VeChain**
Governance model	Public networks	Yes	No	No	No	Yes	Yes	Yes
	Private networks	Yes	Yes	Yes	Yes	No	No	No
	Hybrid networks	Yes	No	No	No	No	No	No
	Open source code	Yes	Partial	Yes	No	Yes	Yes	Yes
Smart contracts	Smart contract languages	Java, Golang	Java, Golang	Java, Kotlin	Solidity	Solidity	C++, multiple	Solidity
	Advanced contracts	Yes	Yes	Yes	Partial	No	No	No
	Amendable contracts	Yes	No	No	No	No	Yes	No
Performance	Throughput	Very high	Medium	Medium	Medium	Low	Medium	High
	Block time,[5] s	10	0.5–2	0.5–2	2	15	0.5	10
	Finality time, s	2–30	1–10	1–10	2	180	180	60
	Large transactions (1–100 KB/tx)	Yes	Yes	Yes	Yes	No	No	Yes
	Long transactions (more than a block)	Yes	No	No	No	No	No	No
	Global network	Yes	No	Yes	Yes	Yes	Yes	Yes
	Parties in transaction	>500	<50	>2	2	2	>2	2
	Dynamic consensus	Yes	No	No	No	No	No	No
Data management	Custom governance	Yes	Yes	Yes	Yes	No	No	No
	Data regulations	GDPR, HER, HIPAA, EMR	GDPR, HIPAA	HIPAA, HER, EMR	Partial	No	No	GDPR, Chinese
	Atomic swaps	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Side chains	Yes	No	No	No	Yes	No	No
	Multizone security	Yes	No	No	No	No	No	No
Business features	Primary purpose	Enterprise General	Enterprise General	Enterprise Finance	Enterprise Finance	Value transfer	Value transfer	Enterprise Integration
	Flexibility	Yes	Yes	Yes	Partial	No	Yes	Yes
	Interoperability	Yes	Partial	No	No	No	No	No

Note: EHR, electronic health record; EMR, electronic medical record; GDPR, General Data Protection Regulation; HIPAA, Health Insurance Portability and Accountability Act; KB/tx, kilobytes per transaction; s, seconds.

[5] Insolar block time is configurable (from 1 second to minutes); minimum time depends on network size and average network latency.

2.1.1. Hyperledger Fabric

Hyperledger Fabric has a general enterprise focus, with flexible and customizable add-ons. It can run smart contracts (chaincode) in Golang and Java.[6] In addition, integration with external systems and complex logic can be used inside smart contracts (advanced contracts).[7]

Transaction performance is claimed as 2,250 transactions per second (tx/s) at peak, with average use-case speeds of 400 tx/s.[8] Scalability is channel-based and limited to 50 nodes per channel. This number reduces when the data volume transferred between parties increases, so exchanging 100 kilobyte documents usually limits the size to 10 nodes per channel. Block formation time ranges from between 0.5 and 2 seconds, and finality time is usually between 1 and 10 seconds, but this increases with more channels, nodes, and with higher network latencies between nodes. A channel should have at least one orderer node that receives all channel transactions, thereby adding an aspect of centralization and bottleneck. Hyperledger supports large transactions but does not have long (more than a block time) transaction support, and it is not able to maintain the global network or geographically specialized nodes.

The platform relies on a Practical Byzantine Fault Tolerance (PBFT) consensus model, which is planned to be replaced by Raft in version 1.4. The platform supports dynamic plug-in consensus but lacks sidechain capabilities and requires parties to own or run a server.

Hyperledger supports customizable data governance but does not provide data sharding or isolation of governance and business logic. It is compliant with both the Health Insurance Portability and Accountability Act (HIPAA) and GDPR, and supports pluggable cryptography in signing, but it does not support encryption of transport level security nor in block integrity.

2.1.2. Corda

Corda is focused on the financial industry, with the option to customize add-ons. The solution is open source[9] and built for private permissioned networks only. Corda supports Kotlin Java-based

---

[6] "What is Chaincode?" Hyperledger Fabric, [https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html]

[7] Hyperledger Fabric, "A Blockchain Platform for the Enterprise" https://hyperledger-fabric.readthedocs.io/en/latest/; Thakkar, P., et al., "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," arXiv:1805.11390v1, May 29, 2018, https://arxiv.org/pdf/1805.11390.pdf

[8] Thakkar; "Performance Test of the Hyperledger Fabric," Stack Overflow, https://stackoverflow.com/questions/50334489/performance-test-of-the-hyperledger-fabric

[9] "Corda," GitHub, https://github.com/corda/corda

smart contracts.[10] Contract logic can be executed on several nodes with a Validator checking whether results match.[11]

Regarding performance, Corda has medium throughput of 1,700 tx/s for one node, and 170 tx/s for two nodes.[12] Corda is a scalable solution, but not all nodes see all transactions. It supports large transactions (capable of carrying 1–100 kilobytes), as well as node specializations including general nodes and notaries. Corda can maintain block time of 0.5 to 2 seconds and finality in 1 to 10 seconds.[13,14]

Corda supports two consensus models: validity consensus and uniqueness consensus. It also has customizable data governance and atomic swaps, but it does not support data sharding. It is compliant with HIPAA regulations and is compatible with electronic health records and electronic medical records.

2.1.3. Quorum

Quorum is a proprietary solution designed for the financial industry. It supports only financial/reporting businesses and lacks interoperability.[15] As such, it can only be implemented for private permissioned networks.

Quorum supports Solidity smart contracts, as well as Advanced smart contracts. It has a midrange throughput of 2,100 tx/s, and 650 tx/s for private contracts. The platform scales well since consensus is reached among relatively few nodes, with a block and finality time of 2 seconds. It supports large transactions and geodistribution of nodes and their specialization but lacks long transaction support.

---

[10] "Writing a Contract," Corda, [https://docs.corda.net/tutorial-contract.html]

[11] "Welcome to Corda!" Corda, https://docs.corda.net/index.html; Ward, M. "Transactions Per Second (TPS)," Corda, February 9, 2018, https://medium.com/corda/transactions-per-second-tps-de3fb55d60e3]; Carlyle, J., *Corda Performance*, March 7, 2018, https://www.r3.com/wp-content/uploads/2018/04/Corda-Performance-ENG.pdf

[12] Ward.

[13] Brown, R., "Corda Top Ten Facts #8: Pluggable Consensus," Corda, September 7, 2018, https://medium.com/corda/corda-top-ten-facts-8-pluggable-consensus-447545e6cb0d

[14] Rilee, K., "Understanding Corda—Transactions vs Blocks," Corda, February 16, 2018, [https://medium.com/kokster/understanding-corda-transactions-vs-blocks-1f3d42d00cc2]

[15] "quorum-docs," GitHub, https://github.com/jpmorganchase/quorum-docs; Baliga, A., et al., "Performance Evaluation of the Quorum Blockchain Platform," arXiv:1809.03421v1, July 19, 2018, https://arxiv.org/pdf/1809.03421.pdf

The consensus model of Quorum uses Raft-based and Istanbul BFT/PBFT. The solution provides atomic swaps, customization of data governance, but no data sharding.

2.1.4. Ethereum

Ethereum is an open-source,[16] general-value, transfer-focused blockchain platform.[17] It lacks interoperability capabilities and has limited flexibility for practical use. It has low transactional throughput, is unsafe in smaller networks, and its performance degrades with data growth. Ethereum only supports permissionless networks, although there are efforts to build permissioned variants of Ethereum.

Ethereum supports smart contracts on the EVM/Solidity. It has low throughput of 20 tx/s[18] but works actively on state sharing to boost scalability. Block time is 15 seconds,[19] and finality time is 180 seconds. It can handle neither large nor long transactions. From a node perspective, Ethereum has geodistribution and some specialization. Full nodes are responsible for validation, and light nodes can proceed with fast synchronization. Only two parties can take part in transactions.

Ethereum's consensus model relies on PoW, with developments in PoS as well. The atomic swap is based on Altcoin Exchange and side chains using Loom Network.

2.1.5. EOS

EOS is a general dApp blockchain platform. It is designed to be flexible, but not interoperable. It is an open-source[20] solution and works for public permissionless networks only. EOS supports WebAssembly (WASM)-based smart contracts using C++ and other languages.[21]

---

[16] "ethereum," GitHub, https://github.com/ethereum

[17] "Ethereum Homestead Documentation," Ethereum Homestead, http://www.ethdocs.org/en/latest; Manning, J., "The Raiden Network Could Allow Instant Transactions In Ethereum," *ETHNews*, November 4, 2016, https://www.ethnews.com/the-raiden-network-could-allow-instant-transactions-in-ethereum

[18] Njui, J., "Vitalik: Ethereum (ETH) Can Scale to 500 tps Using ZCash's ZK-SNARKs," Bitcasino.io, September 24, 2018, https://ethereumworldnews.com/vitalik-ethereum-eth-can-scale-to-500-tps-using-zcashs-zk-snarks/

[19] Etherscan, https://etherscan.io/

[20] "EOSIO/eos," GitHub, https://github.com/EOSIO/eos

[21] "Required Knowledge," developers, https://developers.eos.io/eosio-cpp/docs/required-knowledge; Xu, B., et al., "EOS: An Architectural, Performance, and Economic Analysis," https://www.whiteblock.io/library/eos-test-report.pdf

The EOS platform has a midrange throughput of 4,000 tx/s and allows horizontal scalability.[22] Its block time is 0.5 seconds,[23] but its finality time is approximately 180 seconds.[24] Twenty-one EOS nodes serve as Validators each round, while the remaining nodes are nonvalidating. The platform supports a delegated PoS consensus model, without dynamic consensus capabilities.

EOS enables atomic swaps via Hashed Timelock Contracts. It does not support national cryptography standards or multizone security at present, nor does the solution offer customized data governance. Moreover, the limited number and relative static allocation of Validator nodes is a major limiting factor for scalability and reliability.

2.1.6. VeChain

VeChain is flexible by design and focused on the supply chain industry and IoT integrations. The solution is open source[25] and can be deployed in public permissioned networks.[26]

VeChain only supports smart contracts written in Solidity and claims a high performance of up to 10,000 tx/s, with 50 tx/s on average.[27] According to preliminary information, it is scalable and has block and finality times of 10 seconds[28] and 60 seconds, respectively. The solution has two node types—authority and economic—with economic nodes maintaining the internal currency.

Although VeChain supports proof-of-authority consensus, it lacks dynamic capabilities. The solution is compliant with GDPR, ISO27001, and the China Cybersecurity Law.

---

[22] EOS Network Monitor.io https://eosnetworkmonitor.io/

[23] "EOS.IO," Wikipedia, https://en.wikipedia.org/wiki/EOS.IO; EOS Network Monitor.io https://eosnetworkmonitor.io/

[24] "How long on average before a transaction included in a block is final and irreversible? (without BFT msgs)," EOS.IO Stack Exchange, https://eosio.stackexchange.com/questions/445/how-long-on-average-before-a-transaction-included-in-a-block-is-final-and-irreve/455#455?newreg=e56eee173831404889d38c06b166a94c

[25] "VeChain," GitHub, [https://github.com/vechain]

[26] bsc44, "VeChain Development Plan and Whitepaper," *Medium,* May 20, 2018, https://medium.com/@bsc44/vechain-development-plan-whitepaper-bccc3c255082; https://www.reddit.com/r/Vechain/comments/97zmoy/in_theory_how_many_tps_is_vechain_built_to_handle/]

[27] "VeChain," reddit, https://www.vechain.org/technology/

[28] "VeChain Explore," VeChain, https://explore.veforge.com/

3. Blockchain 4.0

The main focus of the three previous generations of blockchain was on technological improvements. However, Insolar believes that Blockchain 4.0 should focus on addressing the immediate and future needs of business by making the technology suitable for enterprise integration, particularly by making such integration simple, scalable, and efficient (Table 3).

Table 3. Outcomes of Blockchain 4.0.

Outcome	Description
Connect anyone	● Get network identity for your business for unified addressing and data exchange ● Join others' businesses into your business process(es) or join theirs ● Run uniformly and transparently across on global, hybrid, segmented, and isolated networks
Gain advantage	● Balance transaction-level managed risks with performance and cost ● Match your total cost of ownership model and work with true as-a-service solutions — not a do-it-yourself IT hassle ● Utilize existing IT skills to run Insolar on-premise and to tune it to the needs of your business
Stay focused	● Run business services and rules; work with business logic only ● Package documents together with operations and forget about separate document interchanges ● Scale up automatically with commoditized hardware capacities (CPU, storage, traffic)
Be compliant	● Get technical and regulatory requirements checked and handled on-the-go ● Consider current and future legal and regulatory requirements ● Get anything documented with legally binding signatures of who has handled operations related to your business when, why, and how

To deliver these benefits to various businesses, an open and collaborative environment is required to enable third-party companies to build and maintain templates and services, provide hardware capacities, and adapt services and functions to local practices and legal and regulatory requirements.

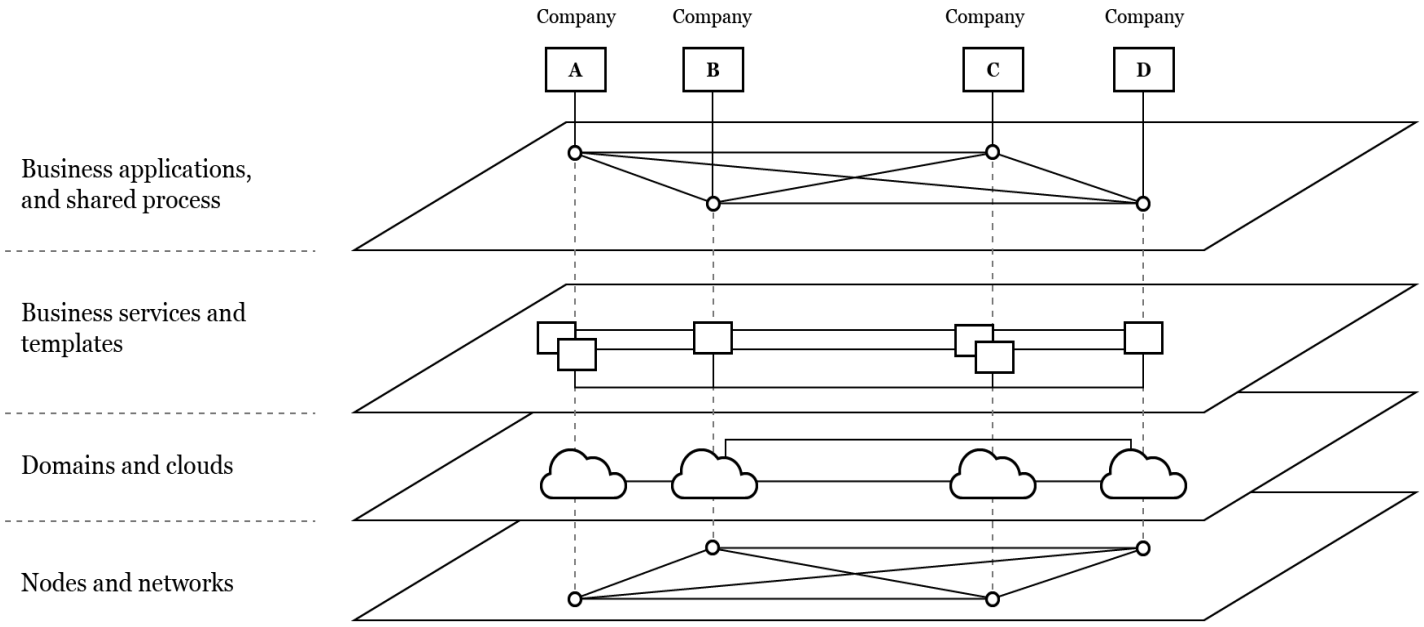


Figure 2. Layers of the Insolar Blockchain 4.0 ecosystem.

Figure 2 is an illustration of the platform architecture that facilitates such a collaborative environment. The architecture is split into four layers. At the bottom layer, there are providers of hardware capacities, which are organized into national and/or industrial domains. At the next layer are clouds. This infrastructure can also be public and offered by governments or even communities as a public good (crowd-sourced computational resources). The next layer represents business services and templates for business applications provided by vendors. Finally, at the top layer, are applications owned by and tailored for companies who serve other companies.

Although this vision of a Blockchain 4.0 ecosystem might appear optimistic, Insolar is proving it is possible. Insolar is currently designing the proposed platform in an incremental fashion allowing it to progressively grow into the ultimate decentralized collaborative environment for various kinds of industries, companies, governments, and communities.

Insolar sees these many features and capabilities as a mandatory part of Blockchain 4.0 technology and implements them as part of the Insolar platform.

3.1. Federation of Clouds

Transparently connects multiple clouds based on Insolar technology, where each cloud runs and is governed independently (e.g., by a community, company, industry consortia, or national agency).

3.2. Cloud

Organizes and unifies software capabilities, hardware capacities, and the financial and legal liability of nodes to ensure transparent and seamless operation of business services (more details provided in Section 4.14).

3.3. Globular Network

The backbone of a cloud. It is a set of protocols enabling the coordination of P2P networks of up to 1,000 nodes and a hierarchical network of up to 100,000 nodes (more details provided in Section 4.9).

3.4. OmniScaling

Is an integral feature that utilizes a multirole model of nodes, a multichain organization of storage, and an innovative approach to distributing work across the network by combining a network-wide membership consensus with deterministic role allocation, and with individual validation groups per each transaction. As a result, OmniScaling enables near-linear and dynamic scalability by CPU, storage capacity, and traffic (network throughput) for domains running within a cloud.

3.5. Domain

Enables different governance models, it defines policies for data and contracts, such as to allow public or permissioned models, or to apply national or industry standards (more details provided in Section 4.13).

3.6. Data Safety

Cost-efficient data safety and leakage prevention for shared cloud solutions through data scattering and density limits, atomic re-encryption, permissioned node access, signatures from nodes that have accessed data, and so on.

3.7. Capacity Marketplace

A special domain within a cloud that defines procedures for business services to do spot and long-term trading for CPU, storage, and traffic as commodities (more details provided in Section 9.3).

3.8. Separation of Business Logic

Helps to focus on what is essential for a business to operate and allows for the deployment of new business services as easy as creating a new mailbox or a website.

3.9. Business Templates

Come from a separation of business logic with the use of domains, making the reuse of business logic possible either as components or as full application templates for easy deployment (more details provided in Section 9.1.3).

3.10. Per-transaction Consensus

Allows validation and finality requirements to be defined as business logic components on a per-transaction basis to match the value and risk of transaction versus the speed and cost of validation.

3.11. Data and Execution Scattering

Together with atomic proxy re-encryption algorithms, data and execution scattering significantly reduce the impact of intrusions and data leakage for off-premise settings.

3.12. Support for Large and Long Transactions

Makes it easy to exchange documents, build complex business services, and use dynamic binding of services via marketplaces. It significantly reduces the complexity of development and deployment for the storage and processing of off-chain documents, while simultaneously increasing the consistency and integrity of blockchain solutions.

3.13. Integration and Compatibility

Provided via contracts and virtual machines, which can be attached to specific nodes and operate as integration gateways.

3.14. Native Support of Industry-standard Languages

Native support of contracts based on industry-standard languages, such as Golang and JVM, as well as support for distributed transactions[29] and microservice-like integrations of contracts to enable the use of existing practices and skills.

[29] In addition to database-distributed transactions with deadlock and runlock avoidance and detection.

4. Key Design Concepts

The architecture of the Insolar platform is multilayer with multiple components and consensuses to address complexity and a variety of requirements. The use of multilayer architecture makes platform design a challenging task, but with proper use it enables building complex solutions with better control of development risks and (later) of ownership costs.

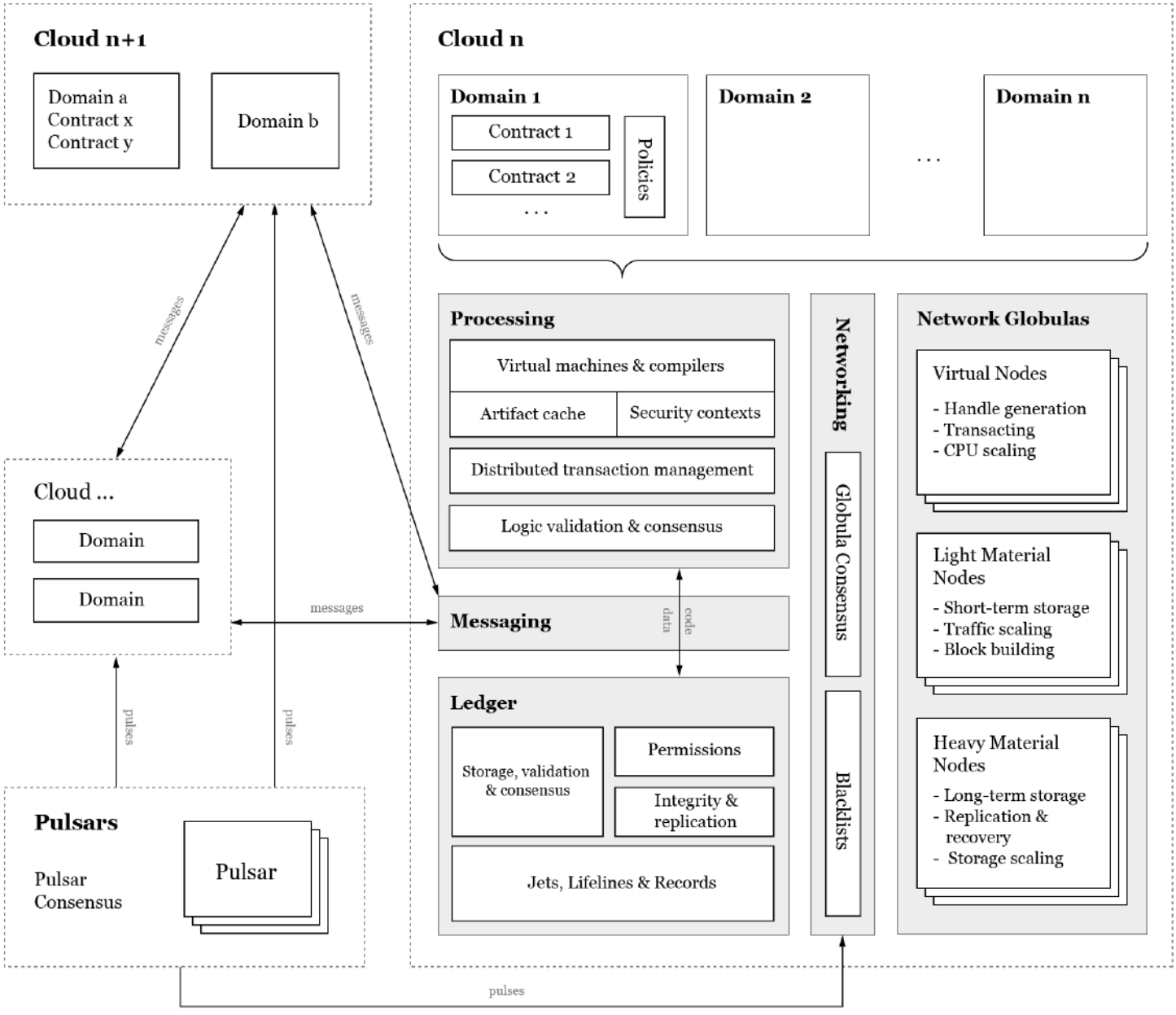


Figure 3. Insolar platform architecture.

4.1. Multichain Ledger

As a platform, Insolar differs in its blockchain implementation, with the architecture designed to be organized as multiple blockchains, which ensures chronological and immutable storage of data. As such, there are three types of chains: Lifelines, Sidelines, and Jets (see Section 7.6 for details on Jets and their usage). Use of multiple kinds of chains enables dynamic reconfiguration of storage by dynamic splitting and merging of Jets without compromising data immutability. It also enables dynamic reallocation of storage nodes on Jets and is a part of the OmniScale feature of the Insolar platform.

To avoid confusion, any Lifeline can have logic and code associated with it, so the terms Lifeline, object, and (smart) contract are interchangeable.

4.2. Lifelines

Lifelines are individual chains in which each data object and all its states are stored. Objects are treated as individual chains.

4.3. Sidelines

Sidelines are utility lifelines used to store temporary or auxiliary data such as indexes, pending operations, or debug logs. Lifelines can have several Sidelines to store information.

4.4. Jets

Jets are equivalent to shards or partitions which make up storage blocks (Jet Drops) and form shard chains. Records in Jets are first produced by Lifelines, then packaged into blocks and placed in a sequence to form a chain of Jets. Replication and distribution of data is managed individually by blocks and Jets.

4.5. Messaging

Communication between Lifelines in the form of messages is the only way interaction takes place on the Insolar platform. This is by design, since the Lifelines of varying objects may reside on different shards and nodes within the network. Moreover, based on messaging, the Insolar platform enables the use of coding practices such as remote procedure calls, remote method invocation, microservices, and enterprise service bus architecture. Although messaging as a singular method of communication limits determinism across the Insolar network, it should be noted that the people and the yet-to-be-integrated systems are not fully deterministic in any case.

In handling re-entrances, and detecting and preventing deadlocks and runlocks, Insolar supports distributed transactions, which allows for atomic swaps and other complex transaction scenarios.

4.6. Validation

The Insolar platform works on the principle of transactions *executed by one node, validated by many*. Therefore, processing power is used more efficiently since only a single node across the network serves a Lifeline to facilitate the execution of calls to an object, while multiple nodes validate transactions. In this process, a specific node is elected to become an Executor and receives calls, collects the results of outgoing calls, and provides updates for validation by other nodes. Validator Nodes are elected once an Executor's status expires, meaning Executor Nodes cannot predict which nodes will validate transactions, thereby avoiding a scenario where nodes can collude. Furthermore, the number of nodes elected as Validators can be determined in accordance with the business process at hand and, since Validators in shared enterprise networks will have liability and legal guarantees, this works as insurance for transactions.

This structure provides for a balance in accordance with client needs. Processing costs are traded off against uninsured risks (suitable for situations where a cheaper transaction is executed, but fewer Validators verify said transaction, meaning greater risk of loss), or processing speed is increased to the detriment of operational risk (which would mean that frequent transactions could be processed without awaiting validation, or validations may be batched together and processed following some delay, leading to the possibility of resource-consuming rollbacks).

It is important to highlight that the previously mentioned node elections are not based on voting; instead, they are part of the OmniScale feature. Insolar uses the active node list and entropy generated by consensus of the Globula Network Protocol (see Section 5.3), and then applies deterministic allocation functions (see Section 6.2.5). As such, this avoids wasting efforts on numerous per-transaction and network-wide consensuses.

4.7. Consensus

Consensus rules among Validators may vary with purpose on the Insolar platform. Examples of this variation include Majority Voting consensus for public networks in which changes introduced by transactions are confirmed by a majority within a single round of voting, and All or Escalate consensus for cross-enterprise, hybrid, and private networks, in which all assigned Validators must agree or another round of voting will begin with a different set of Validators. This process is repeated until a decision is reached or the set number of rounds is exceeded,

which in turn may trigger (internal or external) conflict resolution to provide a satisfactory outcome.

4.8. Pulses

The consistency of the view of network nodes and the distribution of Pulses are dealt with by the network layer of Insolar. A Pulse is a signal carrying entropy (randomness) which acts as a trigger for the production of a new block. The consistency of the entropy and the set of active nodes on the network are vital for the previously mentioned methodology of *executed by one node, validated by many*. Nodes are elected from the active node list, while entropy and consistency ensure behavioral consensus across all nodes. Validator Nodes are elected only on a new Pulse to ensure that Executor Nodes cannot collude with Validators.

The generation of entropy and new Pulses occurs using Pulsars running on the Pulsar Protocol (see Section 5.2), which can either be run on the same network or an entirely separate one. Cases of the former include private networks, which can implement a dedicated server; cross-enterprise and hybrid networks, which can use a shared network of Pulsars yet run individual installations of Insolar networks; and public networks, which can use trusted Pulsar nodes or run the Pulsar function on other nodes.

4.9. Globulas

A network of up to 1,000 nodes is called a Globula. It can run as a truly decentralized network with consistency established by a BFT-based consensus mechanism, implemented as the Globula Network Protocol (GNP). Insolar also supports larger node networks of up to 100 Globulas (a total of 100,000 nodes), which behave transparently across such networks in accordance with whichever contract logic is in place. Such networks rely on the InterGlobula Network Protocol, which implements leader-based consensus.

4.10. Multirole Nodes

Insolar utilizes a multirole model for nodes, in which each node has a single primary purpose and a set of dynamically assigned roles: primary, dynamic, and delegated and utility.

4.10.1. Primary Roles

A node's purpose is defined as a singular primary role and assigning a new primary role requires registering a new node.

There are four categories of primary role nodes:

1. *Virtual Node:* performs calculations;
2. *Light Material Node:* performs hort-term data storage and network traffic;
3. *Heavy Material Node:* performs long-term data storage; and
4. *Neutral Node:* participates in the network for consensus (not in the distribution of work) and has at least one utility role.

The primary role correlates to the type of resource the node can provide to the cloud, and is a part of the OmniScale feature of the Insolar platform.

4.10.2. Dynamic Roles

Nodes can be equipped with dynamic roles—roles able to change—but the dynamic roles that a node can perform depend on the node's primary function.

Dynamic roles for each primary role include, but are not limited to:

- *Executor Node:* handling operations on a Lifeline and building new states (Virtual Nodes), blocking, or granting access to previous blocks (Material Nodes);
- *Validator Node:* verifying Executor Node actions from previous Pulses (time intervals).

A node can have multiple dynamic roles, such as elected to be an Executor to one Lifeline and a Validator of another.

4.10.3. Delegated and Utility Roles

In addition to Primary and Dynamic roles, nodes can take on delegated and utility roles, which serve additional functions, such as caching, InterGlobula coordination, and node joining.

4.11. Permissions

All requests and operations inside Insolar are strictly validated to ensure that only nodes currently elected as a Virtual Executor for a specific Lifeline are permitted to send calls (requests) from, generate new states for, and receive the current state of said Lifeline. Meanwhile, only nodes currently elected as Light Material Executors for a specific Jet are permitted to generate a

new block for, confirm that they have received a call to a Lifeline which belongs to said Jet, and to retrieve the Jet state from the previous Light Material Executor.

4.12. Contracts

In line with the principle that everything is a contract on the Insolar platform, contracts can be considered as Lifelines and are based on general-purpose programming languages such as Golang or Java. As the platform already reduces determinism by using messaging, Insolar applies relatively relaxed requirements regarding the determinism of contracts. As such, invocation of a method on the same state, with the same parameters, and on the same Pulse should produce exactly the same results and consume roughly the same amount of CPU resources, while contract execution methods that run longer than one full Pulse must be explicitly declared with an "execution duration" policy.

A contract that does not produce the same results under the given conditions will not pass validation, and all efforts expended will be at the cost of the party that deploys the contract (as opposed to the caller). Insolar records information on spent effort as Sidelines and can track assigned limits, but actual billing and payment execution must be handled by governance logic (i.e., by other contracts). Although virtual machines are used to isolate contracts incompatible with security or governance rules, new code for a contract can only be introduced to Insolar as source code, with compilation and static inspection performed by nodes in accordance with an applicable governance model.

4.13. Domains

Domains establish governance of contracts and nodes, thus acting as "super contracts," which can contain other Lifelines and can apply varying policies to the Lifelines contained within. Policy can differ with regards to rules for changing the domain itself, access from/to other domains for Lifelines, validation rules for logic (e.g., consensus, number of voters), mutability of code (is it possible for the code to change and what is the procedure to do so?), mutability of Lifeline history (e.g., to implement GDPR or legal action via authorization requirements defined by the domain), and applicability of custom cryptography schemes.

4.14. Clouds

Clouds also establish governance of contracts, but although a domain applies rules to logic, a cloud manages the infrastructure, including nodes, network operations, and layout, storage, cryptography. Therefore, a cloud is a dual entity. On one side, it is a special domain that is stored by the cloud itself and carries static configuration and rules such as procedures for registering

and deregistering nodes, postexecution fraud detection procedures, compensation and penalization procedures, and processing Capacity Marketplace rules. Moreover, clouds run the network and components deployed during node setup, such as bootstrap configuration, Globula discovery and split-protection protocols, node activation and deactivation protocols with the list of currently active nodes and blacklisted nodes, and real-time execution of fraud detection protocols.

4.15. Consensuses

Consensus procedures vary in their degree of control by business logic, with two consensus procedures available:

- *Domain-defined Consensus:* procedures that are a set of Raft-like protocols with entropy-controlled voter selection. These protocols are applied to an object/smart contract after a series of changes. Such protocols can be chosen at the domain level and configured at the transaction level.

- *Utility Consensus:* procedures that are a set of protocols that cover various platform operations that are not directly operated or required by business logic, including Network Consensus, Pulsar Consensus, and Traffic Cascade.

There is a set of consensus procedures which affect every action applied to a Lifeline: logic, storage, network, and pulsar consensuses.

4.15.1. Logic Consensus

Ensures that actions applied to an object were performed correctly, considering the object's state, input parameters, and external dependencies (calls).

4.15.2. Storage Consensus

Ensures that nodes that participated in the Logical Consensus had allocated roles and that records generated by nodes are structurally and referentially valid.

4.15.3. Network Consensus

The Network Consensus ensures node availability and synchronization of time and state among nodes and provides consistent allocation of dynamic roles to nodes. There are two consensus protocols behind Network Consensus:

- *Globula Network Protocol*: a BFT-like protocol that establishes the consistency of a Globula (a smaller network of up to 1,000 nodes).

- *InterGlobula Network Protocol*: a leader-based protocol that extends the GNP and establishes consistency among Globulas of an Insolar Cloud Network (up to 100 Globulas, or 100,000 nodes).

4.15.4. Pulsar Consensus and Pulsars

An additional consensus procedure can be present: Pulsar Consensus. This consensus generates Pulses that trigger time intervals and provide entropy. Pulses are generated by Pulsars only, and Pulsars can be configured differently for different cases (e.g., Pulsars can be a part of the Insolar network or be an external network where Pulsar Consensus and Protocol are running). Moreover, for enterprise installations, it can be just a single isolated or a shared server.

5. Network and Entropy

5.1. Pulses

Pulses and Pulsars represent a separate logical layer that is responsible for network synchronization and provides a source of randomness. A Pulse is both a signal indicating the beginning of the next period and a portion of data that carries information about time and entropy (randomness).

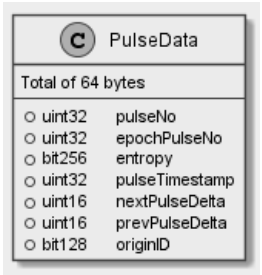
Every Pulse is identified by unique, nonmonotonically increasing integers called Pulse Numbers. Interoperability of nodes within a single cloud is dependent on Pulses, and all nodes must be on the same Pulse to process new requests or operations.

A Pulse Number increase is proportional to the number of seconds passed. This proportion (k) must be less than or equal to 1 (Pulses can go slower than seconds) and must be known to all nodes.

Data related to Pulses includes the attributes presented in Table 4.

Table 4. Description of a Pulse.

Name	*Description*
pulseNo	A Pulse Number is a unique, always-increasing integer. Increases in Pulse Numbers should be proportional to the number of seconds passed (Pulses can go *slower* than seconds).
epochPulseNo	The Pulse Number of the first Pulse of the epoch the Pulse is in.
entropy	A random number.
pulseTimestamp	Reference time when Pulse Data was generated. The time indicated is not directly used by nodes, but for Enterprise and Private installations, it can be used as trusted timestamping.
nextPulseDelta	Delta to receive the next Pulse Number. Nodes must not accept any Pulse which arrives within less than (*2/3 * nextPulseDelta * k*) seconds.
prevPulseDelta	Delta to receive the previous Pulse Number.
originID	Hash of Pulse Proof data.



Distribution of Pulses by Pulsars to nodes includes sending PulseData signed by the "Winning Pulsar" together with Pulse Proof data. The whole packet is signed by the sending Pulsar.

5.2. Pulsar Protocol

The rule for selecting Pulsars is defined by clouds and can vary significantly. For the Enterprise Network, the selection will be managed separately by servers that complete no other operations, whereas for the public network it will be performed by a random subset of 10 to 50 nodes with high uptime. Other configurations are also possible for different network types.

Default Pulse generation is based on BFT consensus among Pulsars, where each member contributes to entropy.

The Pulsar Protocol enables the generation of entropy in such a way that individual nodes are unable to predictably manipulate the entropy through vote withdrawals. This protocol does not include negotiations related to Pulsar membership or Pulse duration—such parameters are considered as preconfigured or preagreed.

The BFT-like Pulsar Protocol includes four phases: pre-prepare, prepare, pre-commit, and commit and emit.

5.2.1. Phase 1: Pre-prepare

Each node independently generates and distributes their proposal of PulseData ($PD_{pulsarX}$) and also generates a temporary symmetric encryption key, the security key for PulseData ($SKpd_{pulsarX}$). A network packet sent to all other Pulsars includes:

- Masked proposal ($PD'_{pulsarX}$): a copy of $PD_{pulsarX}$ with Entropy field encrypted by $SKpd_{pulsarX}$;
- Pulsar signature S1: $S1_{pulsarX} = sign(PD_{pulsarX}, PK_{pulsarX})$;
- Pulsar signature S2: $S2_{pulsarX} = sign(PD'_{pulsarX}, PK_{pulsarX})$; and
- Pulsar ID: either $PK_{pulsarX}$ or hash of it.

A Pulsar only accepts a proposal for PulseData when all fields other than entropy are identical to its own PulseData proposal. The S2 signature is used to ensure transport integrity and for fraud-proofing when a Pulsar sends different proposals to other Pulsars (this will be covered in further documents). Following a timeout, the next phase begins.

5.2.2. Phase 2: Prepare

In phase 2, each node distributes the vector containing the Pulsar ID, and Pulsar signatures S1 collected from others. Submissions from Pulsars with mismatching PulseData are excluded, as are Pulsars for which fraudulent activity has been proven. Following a timeout, the next phase begins.

5.2.3. Phase 3: Pre-commit

Phase 3 involves the selection of the PulseData proposal. As in phase 2, Pulsars which have proven fraudulent activity are excluded. The mutually matching PulseData proposal set (PDPSET) subset of PulseData proposals is then selected according to a calculation of *2/3f+1* across all Pulsars, where *f* is a number of Pulsars. The network split and minimum consensus conditions are checked on the PDPSET, and the PDPSET is placed in order by Pulsar IDs, with the Winning Pulsar selected as PDPSET[n], where *n = hash(PDPSET) mod count(PDPSET)*. The Winning Pulsar then distributes its $SKpd_{pulsarX}$ to other Pulsars (referred to subsequently as $SKpd_{pulsarWinner}$).

5.2.4. Phase 4: Commit and Emit

During phase 4, a consensus for PulseData is agreed upon. This is done by Pulsars transforming of $PD'_{pulsarWinner}$ into $PD_{pulsarWinner}$ (decryption of entropy with the $SKpd_{pulsarWinner}$ distributed on phase 3) and validating $PD_{pulsarWinner}$ by checking $S1_{pulsarWinner}$ issued by the Winning Pulsar. If validation succeeds, each Pulsar sends to others $S3_{pulsarX} = sign(PD_{pulsarWinner}, PK_{pulsarX})$, and the signature validating PulseData from the Winning Pulsar with decrypted entropy, along with the Pulsar's public key. If validation fails, the Pulsar will distribute the original PulseData proposal as proof of fraudulent activity. When a Pulsar receives *2/3f+1* S3 signatures and validates them successfully, said Pulsar begins to distribute PulseData with a set of S3 signatures as proof. If this is not possible, the protocol restarts at phase 3.

5.3. Globula Network Protocol

The GNP is a BFT-like, truly decentralized protocol without any kind of consensus leader. The protocol is responsible for providing a consistent view of active nodes within a network of up to 1,000 nodes (Globula). A consistent view of active nodes means that a node is considered active when other nodes it sees as active recognize this node as active.

The design of the GNP primarily relies on pure user datagram protocol transport without additional mechanics for sessions and with guaranteed delivery. For segmented networks where

nodes can be behind different firewalls, GNP communications will pass through gateway nodes and can be wrapped up into TCP/HTTP between the gateway nodes to improve compatibility with existing firewall/inspection practices and infrastructure.

To ensure the efficient use of network datagrams, the GNP implements a set of primary and secondary features related to the inner workings of Globulas:

- Distribution of a new Pulse across a Globula's active nodes;
- Collection of Node State Proofs (NSP) of active nodes to lock down Node State;
- Handling of joining and exiting of nodes from the network;
- Suspension and exclusion of nodes that are unable to provide their NSP to at least *2/3f+1* of active nodes; and
- Detection of fraudulent node behavior and distribution of proof of fraudulent activity to synchronize blacklists.

To avoid doubt, the GNP only provides network consistency and node proofs. For example, the GNP does not provide operational functions, such as remote procedure calls or block distribution, neither does it directly address Sybil attacks and Network split issues; these are managed by additional protocols and higher-layer components of the Insolar platform, which will be covered in further technical documents about Insolar platform.

It should be noted that the collection of NSPs is the most critical element to build a shared state across all nodes of a Globula. Each NSP carries a node state hash and *nodeSignature = sign(hash(nodeStateHash, pulseHash), SK_{node})* and proves that: (1) a Pulse is received by the node, and (2) what was the node state when the Pulse has arrived.

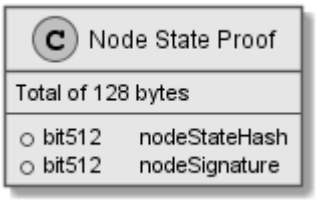


Figure 4. Proof of Node State.

NSP presence is mandatory for node operation during the next Pulse and requires the node to share its node state, which will be used to restrict retroactive changes by the node. An attempt to distribute more than one NSP for a single Pulse will result in the immediate and permanent ban of a node on the network level. Malicious nodes will also be reported to Validators, which may

result in the invalidation of all operations performed by the malicious node on the previous Pulse.

The value for node state depends both on the node's Primary and dynamic roles. For example, Virtual Nodes will build a Merkle tree using all received requests and generated results, while Light Material Nodes will build a Merkle tree of the Jet Drop(s) compiled for the last Pulse.

The GNP occurs in phases like BFT and runs over all the nodes of a Globula when any of the Globula's nodes receives a new Pulse. This protocol has three phases with all-to-all communications, and one additional phase when only a subset of nodes needs to communicate. The additional phase was introduced to mitigate BFT epidemic effects of fraudulent nodes and reduce traffic generated under normal operation.

Integrity and completeness of a Globula's state across all active nodes are controlled by a Globula State Hash (GSH), a Merkle tree root built of NSPs and additional information, such as node indexes. There is no functional difference in using a hash of node list or Merkle tree root to calculate GSH, but the operation of larger networks consisting of multiple Globulas and the InterGlobula Protocol is dependent on Merkle proofs.

The operation of the GNP relies on a dynamic categorization of nodes on the network level (Figure 5 and Table 5).

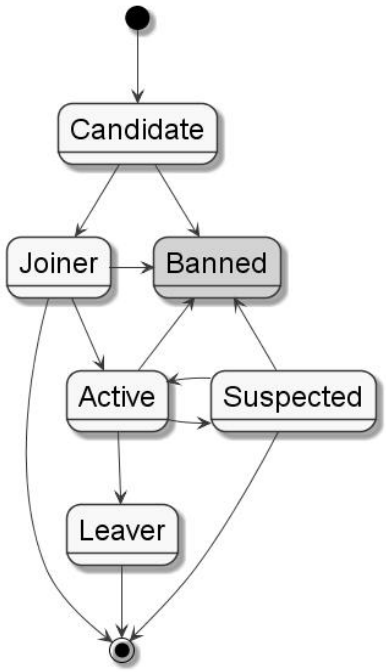


Figure 5. Node network state diagram.

Table 5. Node network state descriptions.

State	*Description*
Active	The node has provided NSP and is known by other active nodes.
Candidate	Node N1 has submitted a join request to an active node (N2), was checked by N2 (via higher-layer components of the platform), and listed locally by N2. N2 will announce N1 as a Joiner on future rounds of GNP.
Joiner	Candidate node N1 broadcasted during phase 1 by one or more active nodes and has had no objections or proof of fraudulent activity published by any active nodes.
Leaver	An active node that has broadcasted a leave request during phase 1.
Suspected	A node that was active on the previous Pulse but had multiple timeouts or was unable to be GSH-consistent with the rest of the Globula. A suspected node is not active, but active nodes will attempt to reconnect with it on the next Pulse. A node that remains Suspected over the course of two consecutive Pulses will be excluded.
Banned	A node performed a malicious action, and at least one proof of fraud exists on one or more active nodes. Banned status is never lifted by the network layer itself—it can only be done via higher-level components of the Insolar platform.
Nonactive	Any other nodes. The network layer does not keep records of unconnected nodes.

Note: GNP, Globula Network Protocol; NSP: Node State Proofs.

Each of the categories listed in Table 5, except for Nonactive, are presented as local lists, maintained by the network layer. All lists, except the list of Banned nodes, are emptied after a node is restarted or is no longer classified as active or Suspended.

GNP consensus is triggered by receiving a Pulse, or by a phase 1 message which carries the new Pulse (Figure 6). A temporary Consensus Node List is then formed, including nodes from active, Joiner, and Suspected categories from the previous Pulse and excluding Leaver nodes. This list is ordered and identical across all active and Suspended nodes as these nodes were participants of the previous consensus. This list will be empty for Joiners, with such nodes composing the list during GNP phases.

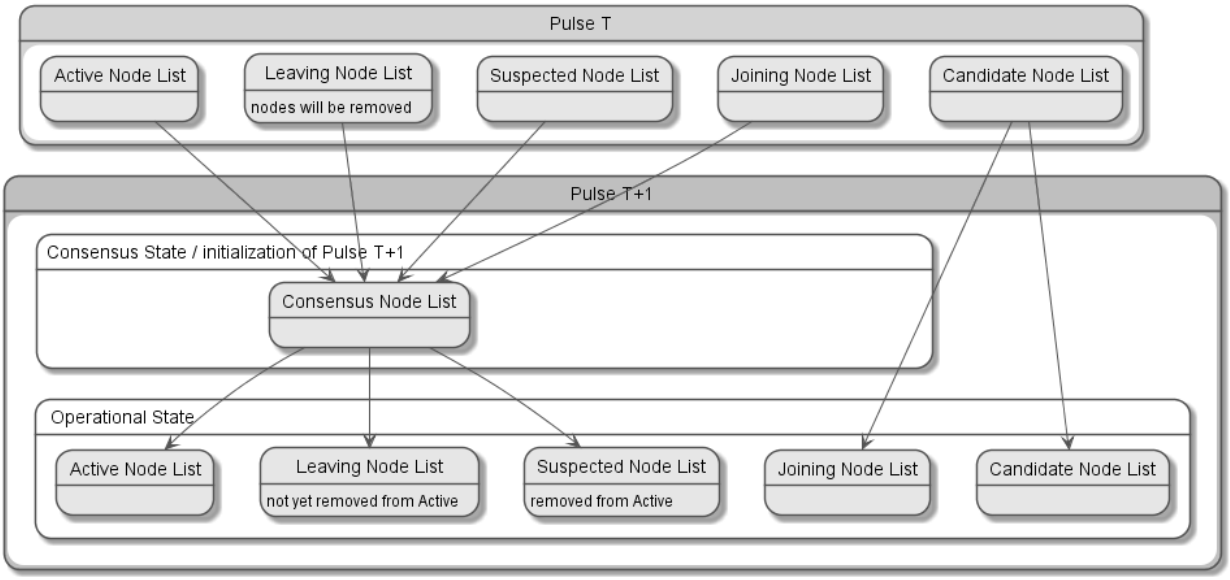


Figure 6. Transformation of Node lists during Pulse transition.

The Consensus Node List is considered immutable for the duration of consensus. Each node will contact every node on this list, and network messages from nodes outside of this list will not be accepted.

Execution of GNP starts with receiving a Pulse or GNP phase 1 message and goes through the phases outlined in Figure 7 and Table 6.

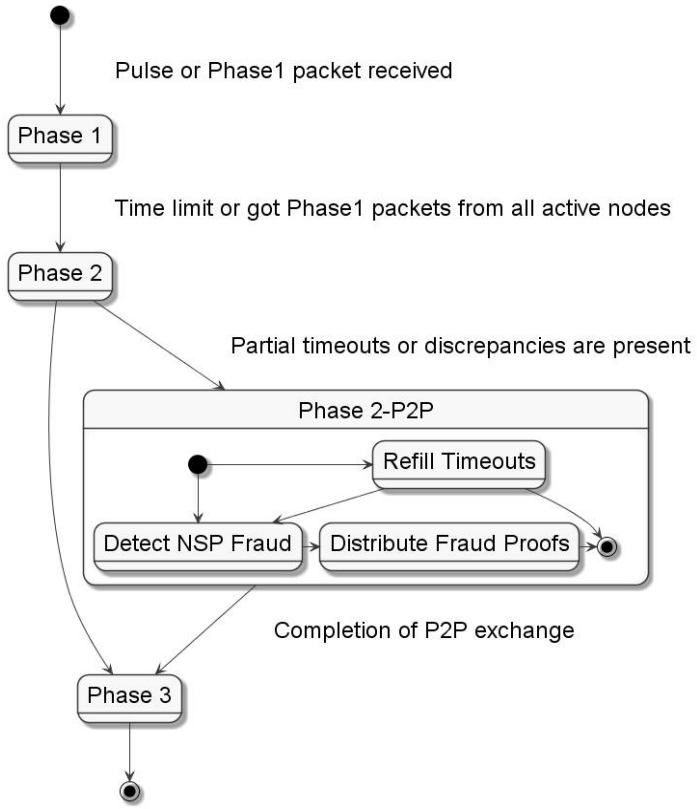


Figure 7. Globula Network Protocol phases.

Table 6. Description of Globula Network Protocol phases.

Phase Name	*Description*
Phase 1	Each node sends a Pulse and its NSP to every node in the Consensus Node List and receives the same from others. Phase 1 also distributes candidates elected to become Joiners, and node details, to Joiners.
Phase 2	Each node sends a bitmap that indicates timeouts (absence of NSPs) and malicious activity by other nodes, in addition to a proof of calculated GSH: $gshProof_{node1} = sign(GSH, SK_{node1})$ In accordance with responses collected, the node will decide to execute phase 2 P2P Extension and identify other nodes who may enter phase 2 P2P Extension.

Phase 2 P2P Extension	Consists of up to three rounds of P2P interactions between a limited set of nodes with *O(n)* complexity. In the first round, any node which previously received timeouts for less than one-third of nodes will request the NSP for the timed-out nodes from other nodes that indicate having received the NSP this node is missing. Second and third rounds are completed to identify node(s) which provide different NSPs to other nodes and to share mismatching NSPs as proof of fraudulent activity. Proving NSP fraud is based on distribution of contradictory NSPs or Merkle proofs based on the NSP signed by the same node. These types of violations can be detected immediately, whereas other fraudulent activities will be recorded and analyzed statistically through higher-level contract logic.
Phase 3	Same message format as phase 2. Using the collected answers, the largest set of mutually agreeing nodes is calculated to form the active Node List. All nodes indicated as malicious by a majority of nodes on the active Node List are immediately excluded from the Globula. Other nodes will be included into the Suspected Node List, unless a node was in the Suspected Node List on the previous Pulse, in which case it will be immediately excluded from the Globula.

Note: GSH, Globula State Hash; NSP, Node State Proofs; P2P, peer-to-peer.

5.4. InterGlobula Network Protocol

As previously mentioned, the GSH is generated as a Merkle tree root. There is no functional difference in using a hash or Merkle tree for GNP, but it is key to the InterGlobula Network Protocol, which uses leader-based consensus among Globulas to build a higher-level Merkle tree over GSH to enable large-scale networks while using similar Merkle proofs for consistency checks.

The use of the InterGlobula Network Protocol for Insolar setups will enable networks of up to 100,000 nodes in exchange of additional phases of Network Consensus and additional synchronization requests among Globulas, thereby increasing the minimum block (Pulse) time to 10 seconds.

6. Nodes and Contracts

Insolar design supports various contract languages and virtual machines (VMs), including Golang and Java/JVM. This is an important feature of Insolar contracts: they allow existing practices, libraries, and development environments to be used straightforwardly.

A contract developer may focus solely on the contract logic and calls of other contracts, while such details as the location and implementation specifics of other contracts are managed transparently by the platform. Every contract has domain-level managed rules that define how contracts in that domain are handled, including policies for changing code, validation requirements, and inbound or outbound call permissions.

In addition to governance with logical rules, domains can also be deployed on separate clouds for stronger network security and data inspection on network edges, while contract/business logic can dynamically tune validation performed by the Insolar platform to balance costs, risks, and performance, adjusting quantity and quality (stake or liability levels) of Validators involved. To the extent of our knowledge, this is a unique feature of Insolar.

Contracts also have individual time tracking and resources, which can subsequently be connected to custom billing procedures and prepaid or on-spot allocation of hardware capacities. Moreover, the ledger that stores contract data applies strict controls for data access, such as requiring signatures from nodes that need to access data and scattering of versioned data across multiple storage nodes to significantly reduce the risk of fraud, intrusions, or data leaks. Furthermore, any contract is guaranteed to be executed, as well as ensuring duplicate calls will not emerge in case of hardware, system, or network failure.

As for practical enterprise use, Insolar contracts can store and transfer large data objects on-chain, without the need for additional systems integrations, and with algorithms to provide scalability of network traffic as well as for CPU and storage. To provide such features and to ensure the consistency of the distributed network without sacrificing performance or scalability, the Insolar platform applies some unique design decisions (described subsequently).

6.1. Network Consistency

As previously mentioned, Insolar uses the network layer to ensure view consistency across the whole network. The next step is to facilitate the efficient and secure execution of contracts across nodes, by utilizing the consistency when all nodes have the same entropy value and list of active nodes. This is done by setting apart the functionality requiring different resources and

permissions, and by distributing work across all available/active nodes of the Insolar network using entropy.

Workload statistics of nodes are not used for balancing, but instead for implementing pseudorandom work distribution, because a trustworthy workload factor on distributed systems requires full visibility and aggregation of operations. Even then, it still does not guarantee smooth workload distribution when workloads fluctuate faster than the average duration of a workload control cycle (aggregate stats—balance—execute). Therefore, the use of a pseudorandom workload distribution can also cause anomalies within a workload control cycle, but it provides a relatively smooth distribution on longer timescales, without the need for full visibility and aggregation of operations.

Network consistency and the allocation functions for dynamic roles (one of which is illustrated in Figure 8) are the core instruments that enable the OmniScale feature of the Insolar Platform.

6.2. Node Roles and Allocation of Nodes

A node's Primary role defines what kind of resource and functionality are delivered by that node to the network, and how the network uses those nodes. As such, there are four types of nodes: Neutral, Virtual, Light Material, and Heavy Material.

6.2.1. Neutral Nodes

These are nodes which participate in the Network Consensus but do not receive any workload automatically distributed by the Insolar network. Such nodes serve particular functions, such as API gateways, Block explorer support, Discovery support, and key management.

6.2.2. Virtual Nodes

These nodes execute contracts and enable CPU scalability. Nodes in this role are stateless, fast, easy to join and leave, and do not need data recovery. Moreover, Virtual Nodes receive and handle requests to execute contracts, read the latest contract state and generate updates (i.e., new records) via Material Nodes, and handle contract-related data encryption, with access to relevant key storage.

6.2.3. Light Material Nodes

Light Material Nodes build blocks, manage access to data, provide caching for recent data, and enable scalability of network throughput. Nodes in this role are stateful and require recovery on

restart, but for recent data only. Light Material Nodes also perform data retrieval and storage operations for Virtual Nodes, while they are also able to redirect requests to relevant Material Nodes when the required data is not available.

Material Nodes can add technical records and Dust (see Section 7.4), but where Lifelines are concerned, Material Nodes can only add records on behalf of relevant Virtual Nodes, which is enforced by signatures and checks during the validation of new blocks.

Moreover, these nodes maintain indexes of the most recent records for Lifelines, attribute indexes, and other functions, along with deduplicating and recovering requests in case of failures of Virtual Nodes. Furthermore, Light Material Nodes assist Heavy Nodes (described subsequently) in their role as temporary backup and cache for individual blocks, integrity validator and recovery source, proof-of-storage approver, and handover voter. Lastly, these nodes collect and register Dust (e.g., service inconsistency reports, long operations, logs).

6.2.4. Heavy Material Nodes

Heavy Material Nodes provide long-term data storage and scalability of storage capacity. There is an additional network protocol to maintain additional backup and archival storage nodes without burdening the main Insolar Network Consensus. Nodes in this role are stateful and require recovery and revalidation of content (proof-of-storage), both periodically and on rejoining the network. They also store all data from Light Material Nodes (and, as such, also from Virtual Nodes) and check data integrity, but are unable to introduce or change data or form a block. Heavy Material Nodes ensure the minimum required level of block replication and the maximum density of data (i.e., the scattering of data) to reduce the impact of data leakage from a single Material Node (Heavy or Light).

Heavy Material Nodes differ significantly from the other nodes in that they store lots of data and must take additional measures to mitigate risks of either losing (or corrupting) data but not having enough copies, or leaking data caused by the accumulation of too much data on a single node. Implementation of Heavy Material Nodes will be simplified for the TestNet and will be gradually extended during the development of the enterprise version of Insolar.

6.2.5. Dynamic Roles of Nodes

Virtual and Light Material Nodes are designed to enable dynamic and straightforward scaling of the network, as well as to require minimal preparation to become operational and to get new workload allocations, while the dynamic roles of these nodes change with every Pulse.

The selection and allocation of dynamic roles for Virtual and Light Material Nodes for a specific object are based on a role allocation function and can be expressed as shown in Figure 8 and Table 7. This code is guaranteed to return the same result on the same Pulse for any active Node of a Cloud because active nodes always have consistent network configuration.

```
function (NodeID) Func_VEA(LocalRecordID objRef, int pulseNo) {
       excludes = { Func_VEA (objRef, getPrevPulseNo(pulseNo)) };
       listOfGlobs = Network_getOrderedListOfGlobs(pulseNo);
       selectionHash = objRef;
       while (!listOfGlobs.isEmpty()) {
              selectionHash = hash(selectionHash, getPulseEntropy(pulseNo));
              glob = listOfGlobs.remove(selectionHash mod listOfGlobs.getCount());
              node = glob.findNode(VirtualNodes, selectionHash, 1, excludes);
              if (node != null) return node;
       }
       return (null or error);
}
```

Figure 8. Allocation function of Dynamic role “Executor” for Virtual Nodes.

Table 7. Parameters and functions for Figure 8.

Parameter/function	*Description*
objRef	Static address of an object/Lifeline
pulseNo	Current Pulse number
getPrevPulseNo(pulseNo)	Returns the Pulse number occurring immediately before pulseNo
Network_getOrderedListOfGlobs(pulseNo)	Returns network layout at pulseNo
getPulseEntropy(pulseNo)	Returns entropy / random seed, distributed with the given Pulse
glob.findNode(role, selector, count, exclude)	Finds a setup to the given count of nodes of role within Globula that is the best match for selector, excluding nodes—listed as exclude

It is important to note that this allocation is only applied when there is a workload associated with an object; only then will the allocated nodes handle the workload. When there is no workload, they will do nothing, just participate in Network Consensus and produce empty blocks.

6.3. Contract Execution Scenario

A simple example of contract execution on Insolar can be illustrated as follows. Let there be two contracts: contract A, with a function *A.fn(X) { return B.fn(X+1) * 2; }*; and contract B, with a function *B.fn(X) { return X * X; }*. Contract A depends on the execution of contract B, whereas contract B's execution has no dependencies. The execution of contract A is initiated by a call that comes from another contract, which is not considered within the scope of this example. Contracts can only be called by other contracts, but there is a special category of integration (interactive) contracts which capture and translate external API calls into calls of other regular contracts.

Each contract is an individual Lifeline, and it is likely that a different node will execute another contract with message exchange between. Even when the execution of both contracts is assigned to the same node, messaging between VMs will still take place. Contracts running on the same machine may be isolated into separate VMs, depending on the security settings of the relevant contract domains.

The running of a contract takes place over the course of several stages for both execution and validation. Execution is carried out by the Virtual Node and Light Material Node assigned as Executors. A request is received and registered within the same Pulse, but for busy objects (due to state locks or multiple updates), execution may be delayed and handed over to another Executor. Moreover, multiple requests can be executed within the same Pulse when opportunistic execution/validation is allowed by a caller or by a called object. Validation is carried out by a set of Virtual Nodes and by a set of Light Material Nodes assigned as Executors and Validators. It is performed in the Pulse after receiving results produced by execution to ensure that the Executor cannot predict the selection of Validators before the results are submitted. Lastly, the validation of outbound calls is stacked into a single validation round as Validators use signed results collected by previous Executors.

Table 8 and Figure 9 and 14 rely on the following notation to identify a role–node combination:

Role DynamicRole (*Lifeline*, *Pulse*)

where:

· *Role* is the Primary role: V(irtual) | L(ight) | H(eavy)

· *DynamicRole* is the Dynamic role: E(xecutor) | V(alidator)[n] | S(tash) | ...

· *Lifeline* is the identity of a Lifeline (object, contract)

· *Pulse* is the identity of a Pulse and related Entropy. This can be relative (e.g., *P+1* means the next Pulse after *P*)

Table 8. Role–node notation examples.

Example	*Description*
VE(A, P)	Virtual Node (Primary role) as Executor (Dynamic role) for Object A at Pulse *P*
VV1(A, P+1)	Virtual Node (Primary role) as Validator (Dynamic role) for Object A at Pulse following *P*
VVn(A, P)	All Virtual Nodes with Validator Dynamic role for Object A at Pulse *P*
LE(A, P)	Light Material Node (Primary role) as Executor (Dynamic role) for Object A at Pulse *P*

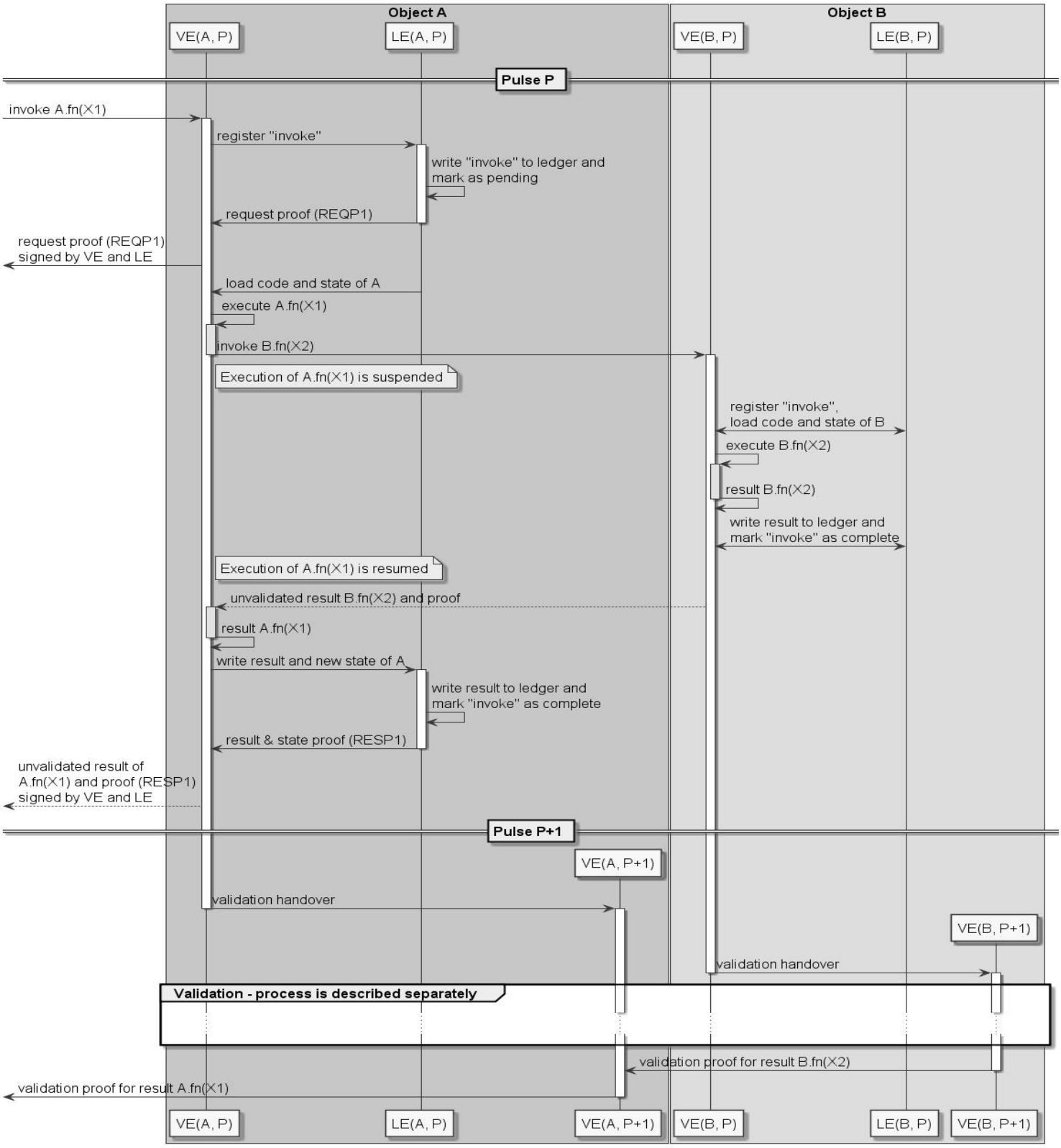


Figure 9. Execution sequence diagram.

6.4. Execution and Validation Process

Execution usually begins at the same Pulse when the request has arrived, but it can be postponed or delegated based on the current load of node and SLA applied to the request. This is done by Virtual Nodes and Light Material Nodes assigned as Executors.

First, the Virtual Node *VE(A, P)* receives an invocation request and verifies this upon the given Pulse. This node is allocated as Executor for contract A, and the sender of the request is the Virtual Node allocated as the Executor for the calling object. *VE(A, P)* then identifies the Light Material Executor Node *LE(A, P)* for object A upon Pulse *P* and registers the request with it. Upon this, *LE(A, P)* makes similar validations as *VE(A, P)* plus validates *VE(A, P)* itself. Then it checks if the request was already registered but not yet completed, stores the request to the ledger, and gives registration proof *(RP1)* to the *LE(A, P)*, subsequently providing code to *VE(A, P)*, along with the last state of object A when needed.

VE(A, P) then deserializes contract A state and runs its code, with the outgoing call made to *B.fn(X2)* recorded to the Execution Tape,[30] including the results returned and signed by *VE(B, P)*. Following this and upon receiving the result of *B.fn(X2)*, the *VE(A, P)* node calculates the result of *A.fn(X1)*, stores the result and possibly new state of object A into *LE(A, P)*, which will effectively mark the incoming *request* as completed, and finally returns nonvalidated results to the caller to enable further processing without awaiting validation.

Validation occurs on the Pulse that follows the one that generated the result. Dynamic roles such as Executor will change, and the validation process of the result will begin. The number of Validators can vary depending on contract logic but cannot supersede the domain or cloud predefined minimums. The new entropy of Pulse P+1 allocates a new Executor VE(A, P+1) and Validators VVn(A, P+1); the new Executor is always one of the Validators, but the previous Executor cannot be a Validator to itself. *VE(A, P)* sends the prepared Execution Tape to Validators, which check its contents and reproduce the recorded execution. (We note that Validators of *A.fn(X1)* do not repeat the implicit *B.fn(X2)* call; instead, the result signed by *VE(B, P)* is used.) When validation is complete, *VE(A, P+1)* sends a validated result and validation proofs (if requested) to the caller node—the Executor of the caller object at Pulse *P*. Validation proofs here are a collection of signatures produced by Validators, which can be validated by any node by knowing P and the list of active nodes at Pulse P. The active node and P can also be validated with a Merkle proof and GSH.

---

[30] Trace of execution that references exact states/version of Lifelines, outgoing calls, and their results.

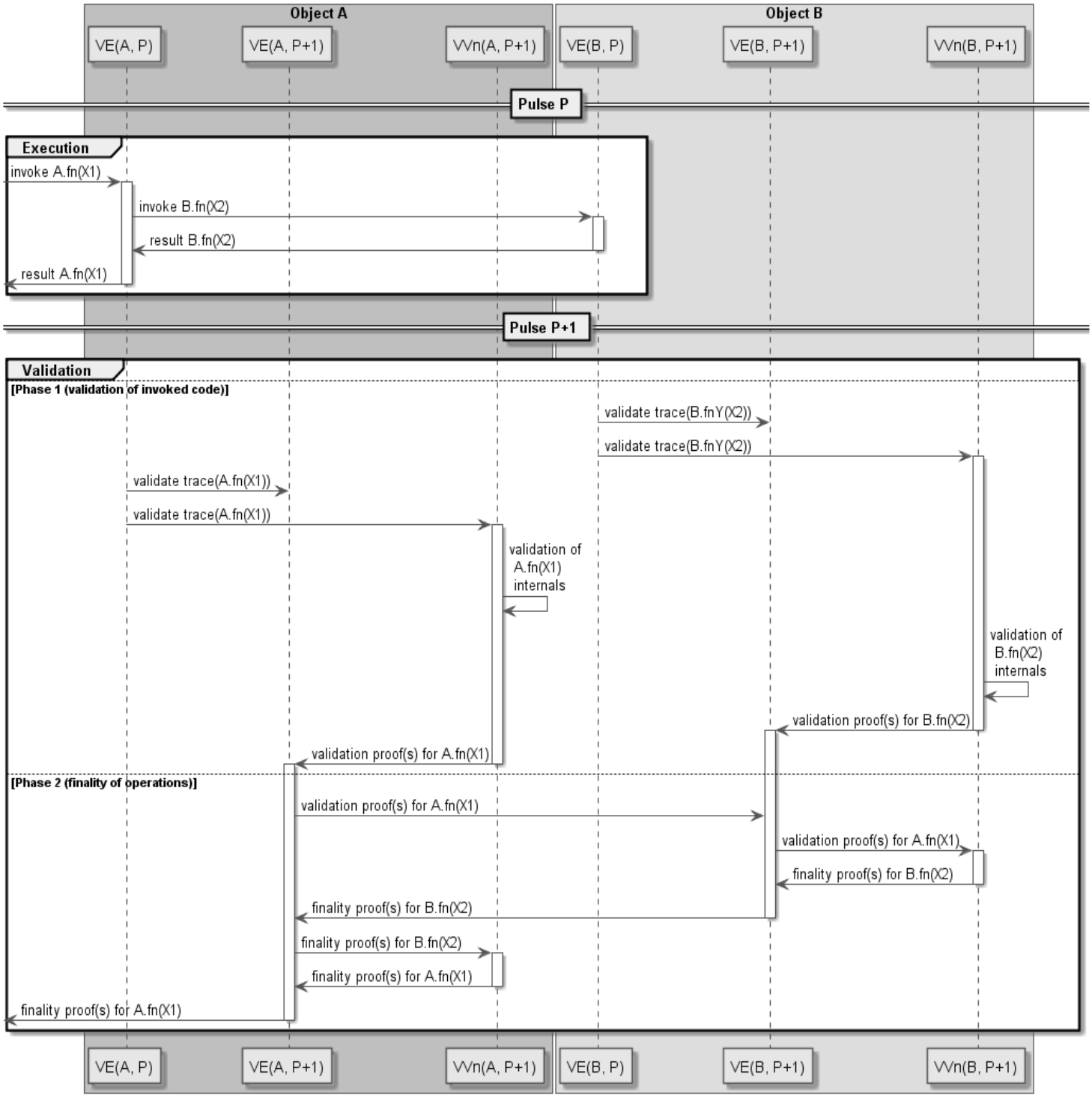


Figure 10. Validation sequence diagram.

The validation process ensures that the code was executed correctly, and produced the same result, and that incoming and outgoing calls were all valid. The key steps are shown in Figure 10.

We note that, for the sake of brevity, this diagram does not show interactions between Virtual Validating nodes and Material Nodes that carry out block validation for blocks generated during the previous Pulse. Validation done by Material Nodes is simpler because it only validates data integrity and allocation of nodes which have generated records. There is no need to check inbound and outbound calls for blocks.

As presented in Figure 10, validation is performed by a number of *VVn(A, P+1)* nodes. One of them is *VE(A, P+1)*, which also works as a consensus leader. The validation process has three subphases: Preparation, Validity/Reproducibility verification, and finality check.

During the Preparation phase, the former *Executor VE(x, P)* provides an execution trace (the so-called Execution Tape) for validation by *VVn(x, P+1)* nodes. The Execution Tape includes information on object state, call parameters, and results. Additionally, the Execution Tape includes results of all outbound calls with Executor signatures for called objects. This enables Validators to avoid repeating outbound calls and only to check signatures and allocations of relevant Executors.

In the Validity/Reproducibility verification phase, each *VVn(x, P+1)* ensures that the Execution Tape has all the results and byproducts reproducible within the given code, initial state, call parameters, and with the results of outbound calls. The result of outbound calls is stored within the Execution Tape; therefore, there is no dependency between callee and caller, and reproducibility checks are performed in parallel by all participating objects. Although not shown in Figure 10, *VVn(x, P+1)* nodes store reproducibility results with *LE(x, P+1)* and then conduct a vote in accordance with the applicable consensus (e.g., by Majority or All or Escalate, as described in Section 4.7). The results of the vote with relevant validation proofs are summarized by *VE(x, P+1)* and sent to callees; for example, *VE(A, P+1)* sends validation proofs to *VE(B, P+1)*, to confirm the validity of the inbound call to B.

During the finality check, each *VVn(x, P+1)* ensures that the inbound call was valid and all outbound calls are final by using validation proofs provided by the caller via its validation leader *VE(x, P+1)*. As soon as this happens, all changes introduced by *B.fn(X2)* are considered final. While this is a precise description for A, this is simpler for B. There are no outbound calls, so B becomes final as soon as it is validated and has validation proofs for its inbound call. Also not shown in Figure 10 is that *VVn(x, P+1)* nodes store finality results with *LE(x, P+1)* and then hold a vote by the applicable consensus (e.g., “Majority” or “All or Escalate”). The results of the vote

with relevant proofs are summarized by *VE(x, P+1)* and sent to callers; for example, *VE(B, P+1)* sends finality proofs to *VE(A, P+1)* to confirm the finality of the outbound call from A to B.

For network security, allocation of Validators requires new entropy to be available for both validation and finality of operations, with validation and finality archived within two Pulses. Operations will usually be executed much faster than the duration of a single Pulse, but when validation and finality are required this process will take one to two Pulses

We note that this algorithm seems complex and is responsible for delayed validation. It also provides several benefits, including that request ordering can be naturally completed by a single Executor, contract logic can manage a number of Validators and the amount of validation efforts, opportunistic and pessimistic execution models are supported, and there is an extension of the validation phase—not only atomic swaps but any complex, atomic transactional schemes can be supported.

7. Ledger

The logical organization of the Insolar ledger (internally referenced as storage) is based on Records, which are base elements that represent a minimal storage unit; Filaments, which are a sequence of Records associated with the same entity or function; Lifelines, which are primary Filaments for object state, with other auxiliary Filaments attached to it; and Jets, which are groups of Lifelines and their records, which will be stored together.

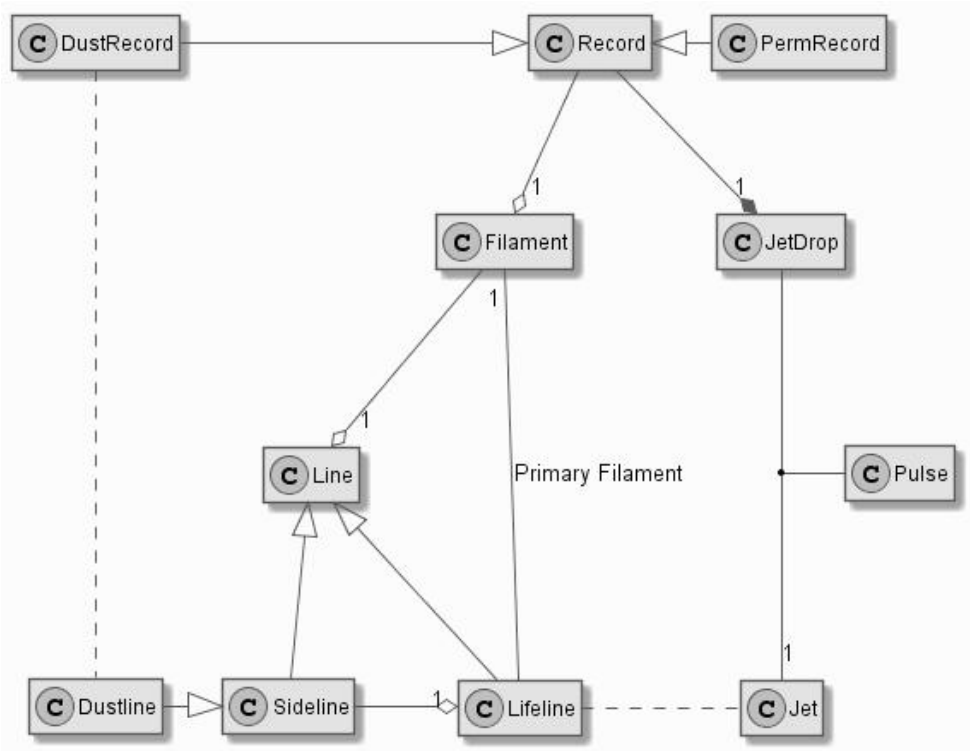


Figure 11. Ledger entities.

7.1. Records

A record contains information such as a request, response, control of state, and maintenance details. All records fall into two main categories in accordance with the usable lifetime of a record, a period during which the record can be used under normal circumstances. These categories are Permanent Records and dust records.

7.1.1. Permanent Records

These records are generated by contract logic, in which the usable lifetime of a record is controlled by contract logic (e.g., legal documents that must be stored for specific periods).

7.1.2. Dust Records

Dust Records are associated with a Permanent Record and are generated either by an application or by system logic, in which maintenance procedures limit the usable lifetime of a record, usually measured in days (e.g., logs or transaction control records). Dust can also be used to identify complex forms of fraud or infringement, and such actions will be registered as Permanent Records, whereas the original Dust Records will be archived or removed.

7.1.3. Record Addressing

A record can be categorized in three ways by the way they can be addressed:

1) Globally addressable records, whereby they can be located using the ID within the distributed storage, without any additional information. Such records are used to create new Lifelines, for instance.

2) Relatively addressable records can be located via their ID and the ID of a globally addressable record.

3) Nonaddressable records can either have a nonunique ID or by design are not intended to be addressable (e.g., block opening and closing records).

7.2. Filament

The next level of storage organization is a Filament, which is a sequence of records connected in a unidirectional linked list, which is in a most-to-less recent order. A Filament is identified by reference to its head (the first record), and every record of a Filament has an affinity field, which refers to the Filament's head.

Most Permanent Records are part of a Filament, but some records are singular (e.g., special wipe out/cleanup records). Dust Records are also a form of Filament, but such Filaments always branch from a Permanent Record. Examples of a Filament include a sequence of records of object state changes (see Section 7.3. for Lifeline), a request's state (received, pending, or answered), and listing (delta and full snapshots of child object listings). It should be noted that Filaments are not allowed to fork and, therefore, neither are all derived entities, such as Lifelines and Sidelines.

7.3. Lifeline

Filaments serve different functions. As previously noted, a Lifeline is a Filament of Permanent Records that is accessible by its head. Lifelines require a head-to-tail index to find the tail (i.e., the most recent record). The head of a Lifeline is its permanent identifier, while the tail is its most recent state, so to locate the tail, one should start with the Lifeline's head. The use of a stable address (head) allows operations to be conducted without requiring knowledge about the latest state and enables affinity-based optimizations. Moreover, a Lifeline consists of a primary Filament that represents the state of an object/contract and can have additional Filaments that represent the state of requests, indexes, and listings related to this object. The presence of additional Filaments is not seen as a state fork.

Figure 12 illustrates relations between Records, Filaments, and a Lifeline. The whole fragment is a Lifeline, and it has one primary Filament for the Lifeline state. There are auxiliary Filaments to track requests, while all Filaments are built of Records.

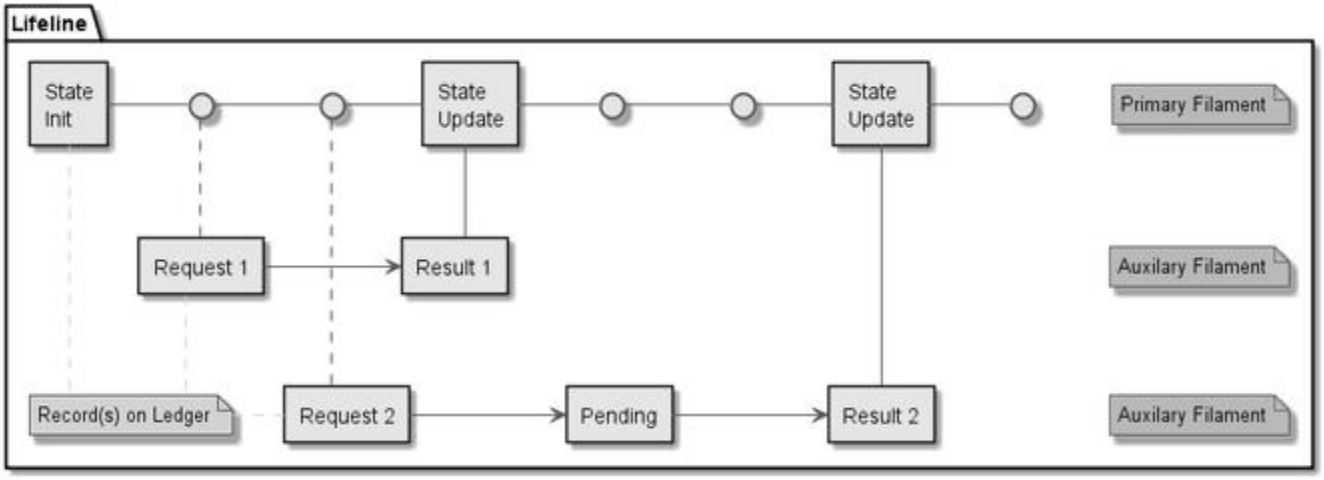


Figure 12. Relations between Records, Filaments, and a Lifeline.

7.3.1. Pseudolifelines

So-called Pseudolifelines also exist in the network, and these are a form of Lifeline for immutable objects—when an object cannot change its contents but can be locked or destroyed. If an example of a Lifeline is a contract or domain, then examples of Pseudolifelines include declared types, compiled code, and policy objects.

7.3.2. Sidelines

Sidelines, on the other hand, are a Filament that is a part of a Lifeline and carry auxiliary information. Sidelines are not head-to-tail indexed for performance reasons, with examples of such including side chains for auxiliary information (e.g., a list of children of an object).

7.3.3. Dustlines

Dustlines are a Filament of Dust Records and can be thought of as a kind of Sideline that can be discarded. Dustlines must start and end within the same block and are stored in a separate section. Examples include logs, billing records, and SLA violation proofs.

7.4. Dust

Dust is a relatively addressable element of maintenance logic or data and resides as a Dustline, which is attached to Lifelines that have triggered operations producing the Dust, and will be stored within the same block as the relevant Lifeline.

Dust is used to carry out postverification of consensus rounds, for operational cost billing, and for handling disputes with larger sets of voters. Dust has a conditional life span: it is processed and marked as “ready for cleanup,” with the actual cleanup taking place several days later. All facts collected from Dust are processed, compressed, and handed over to the relevant contract logic.

7.5. Removals and Cleanups

Storage volume in Insolar can be optimized by excluding expired or unused objects as defined by a domain (e.g., the legally binding period provided for commercial contracts). Furthermore, the rearrangement of blocks does not make the platform less secure even if the changes were made retrospectively because each object is represented by a chain where records cannot be partially deleted. However, in compliance with regulations such as the GDPR, the platform provides for situations in which any Lifeline record can be replaced by a special “wipe out” record, which includes the original record hash and a reference to a record that authorized such an operation.

7.6. Jet

Finally, the uppermost logical element of capacity scalability management is a Jet, which is a partition or shard to which nodes with storage and processing capacities are allocated (Figure 13). All records of a Filament and all Filaments attached to a Lifeline are associated with the same

Jet. This association is carried out by the Record Affinity Function that applies to all records within the storage. Although the Affinity Function is configured for deployment across the Insolar network as a whole, it will be possible to consider per-object and per-transaction storage requirements through a special delegation procedure.

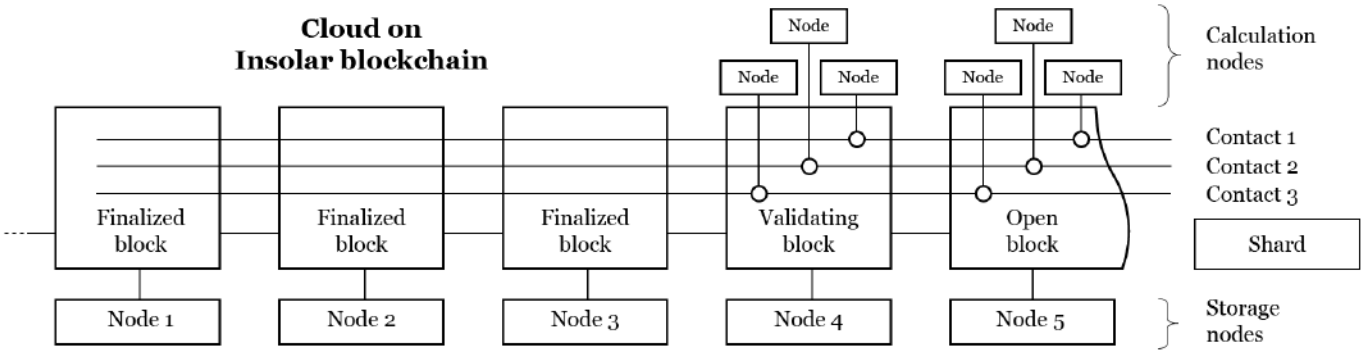


Figure 13. Allocation of nodes in a Jet.

It is important to note that there are two types of Jets: Material Jets (also referred to in this document simply as "Jet") and Virtual Jets. Each of these has different types of nodes allocated for processing.

7.6.1. Material Jets

Material Jets are assigned to Material Nodes and provide storage and network throughput scalability. Each Material Jet is served by Light (Material) Nodes, which focus on throughput and recent caching, and Heavy (Material) Nodes, which focus on long-term storage. A Material Jet carries out data storage and retrieval for Virtual Jets, while it also maintains indexes, ensures integrity, enforces access control, and forms new blocks.

7.6.2. Virtual Jets

Although all persistence (storage) is handled by Material Jets, Virtual Jets have Virtual Nodes assigned to them and enable CPU scalability, while each Lifeline is considered as a separate Virtual Jet. Therefore, Insolar can near-linearly scale up to any number of active objects/Lifelines.

7.7. Jet Drop

In addition to the logical structure of storage, there is a structure called Jet Drop, which is functionally close to a block of earlier blockchains but has a more complex structure (this will be discussed in future documents). Jet Drop is a unit of storage for a management operation, such as balancing, integrity control, and replication. Jet Drops are also units of Material Jets, and all Material Nodes operate with Jet Drops, not with individual records or Filaments. Each Jet Drop is built with a single Light Material Node Executor. After a new Pulse arrives, each Jet Drop is validated by dynamically allocated Validators, in which the quantity and quality of Validators are adjusted by the attributes of domains whose contracts have records in Jet Drop.

Jet Drops contain all records registered within a Pulse for all Lifelines and Filaments of a Jet. In other words, a Jet Drop is a unit of two-dimensional organization of Insolar storage (see Figure 14) with a temporal axis, for which each Drop is associated with a Pulse and an affinity axis, for which each Drop is associated with a Jet; sets of Jets can vary by Pulse due to dynamic sharding—splitting and merging of Jets.

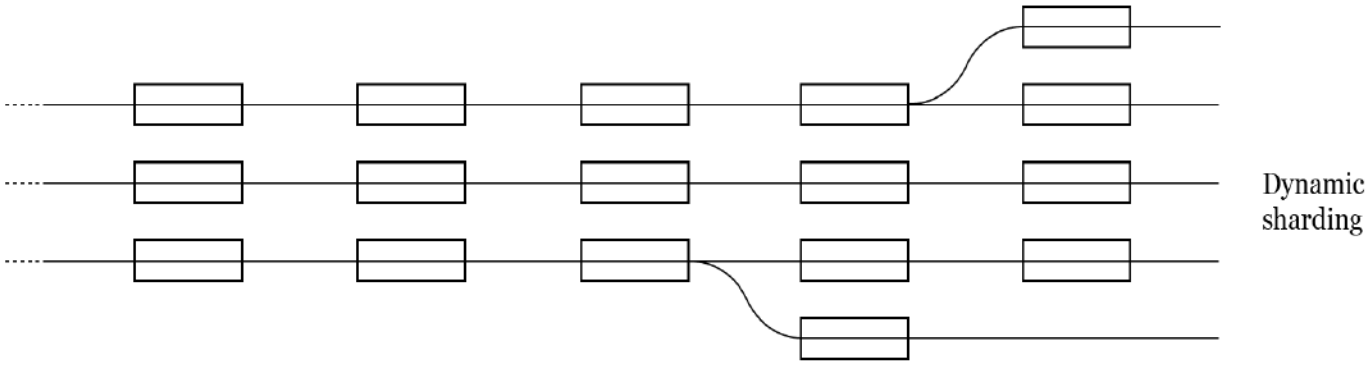


Figure 14. Dynamic sharding.

Both Jet Drops and records of individual objects are chained, while dynamic sharding can cause a Jet to split into two or more Jets, with the appropriate reallocation of Lifelines. A split is initiated on a Jet when the byte size of Jet Drop for 5 to 10 Pulses exceeds 25% of the traffic that an average Light Material Node can receive during one Pulse.

A Jet can also be merged back after a split when both Jets have a Jet Drop byte size for 50 to 100 Pulses of less than 5% of the traffic that an average Light Material Node can receive during one Pulse.

Irrespective of whether Jet Drops split or merge, Lifelines and their Sidelines remain uninterrupted and are always stored within one Jet, and Jet Drops accordingly, forming individual chains (see Figure 15).

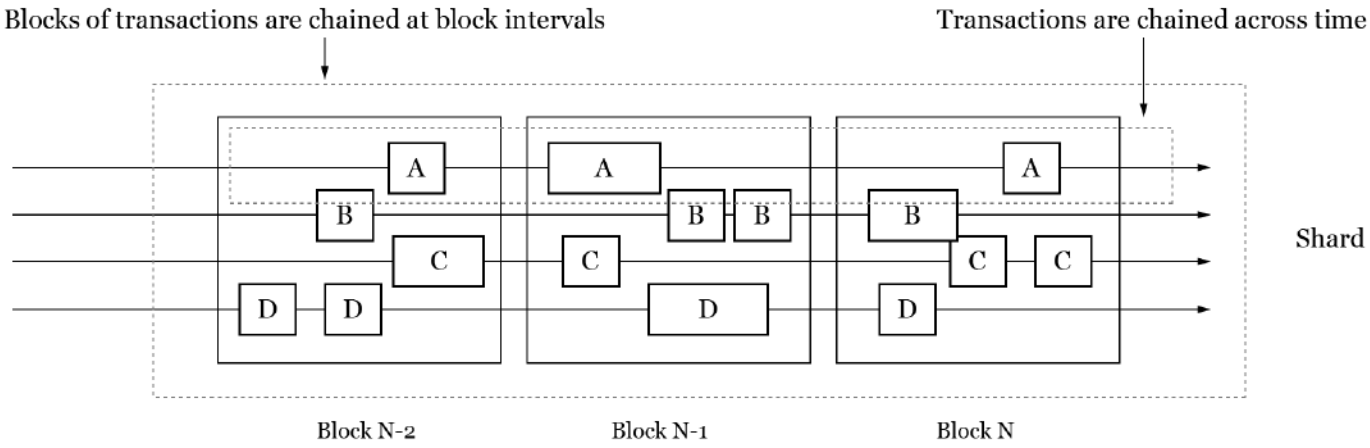


Figure 15. Chains of records and a chain of blocks.

The internal structure of a Jet Drop means records are stored by being split into fixed and variable parts (e.g., parameters of a call or serialized object state), which can be stored separately for a record payload. Records can be selectively wiped out or rearranged, but the relevant proof will be included to keep integrity checks functional, while Dust records can be removed without compromising integrity checks.

Jet Drops also contain artifacts generated by custom cryptography schemes. For example, to satisfy national standards, Jet Drops can include record extensions generated by custom (national) cryptography schemes, while they are also able to reinforce the integrity of a Jet Drop by the custom cryptography scheme without mixing it with Insolar platform cryptography.

Additional details on the structure of Jet Drops will be published in later documents about the Insolar platform.

8. Domains and Clouds

8.1. Domains

Domains play a key role in managing and adjusting the Insolar platform's capabilities for different applications. They have a wide range of features based on declared policies that are imposed on every domain participant and may be applied in several ways.

Domains define generic processes, standards, work rules, data formats, procedures for introducing changes into the standards, and procedures for implementing changes (e.g., ITU standards or legal requirements for processing personal data), in addition to initiating and managing objects such as market reference data, code dictionaries, and company registers. Moreover, they provide and manage storage and transfer policies of protected data (e.g., personal and biometric data, history of clients' financial transactions) and establish access policies to and from objects outside the domain, in addition to cryptography standards, and terms for calculating computing power consumption and storage volume.

The Insolar platform registers domains, enabling interaction between them, and allows almost any kind of governance logic and management procedures defined by each domain individually. For example, each domain can set up a voting procedure to change its rules, policies, and/or participants.

An important feature of the Insolar platform is that it allows domains to define logic that can manage rules and requirements applied to validation, while it also allows to enable business logic and transaction *value* to control consensus type and the number of validating nodes of the transaction; something beneficial for both business and performance. Examples of this include balancing processing costs against uninsured risks, or processing speed against operational risk (see Section 4 for more details).

Furthermore, the difference between public network contracts and legally binding contracts is addressed by the application of Majority Voting consensus for public networks and All or Escalate for enterprise networks (see Section 4 for more details). Application of relevant consensus procedures via domain policy is not limited to blockchain logic: a domain can allow changes to be initiated by legal procedures and court orders, or issues can be escalated to support or arbitrage.

8.2. Clouds

Clouds are the next element for enterprise and business application to tailor Insolar to their needs. A cloud is a set of rules—a tool to manage nodes and provide hardware capacity from hardware providers to services in a unified manner. This also includes the business side of operations such as SLA and node liability; national cryptography, server, and data locality requirements; data isolation and inspection procedures; and KYC of customers, users, and hardware providers.

A cloud is a separate network built and managed by using Insolar platform instruments and should be relatively homogeneous inside regarding security and reliability.

Clouds can be configured in a customized fashion or follow one of several network types: Isolated, Serviced, and Open.

8.2.1. Isolated (Private or Permissioned) Network

A network which consists of trusted participants with their own servers serving each participant individually. By allocating additional servers, a participant can scale up related functions, but not the network.

8.2.2. Serviced (Permissioned) Network

This network consists of participants without their own hardware. Computing power is provided by specific companies that are bound to financial and legal guarantees of liability and confidentiality.

8.2.3. Open (Public) Network

A network in which participants serve as computing power providers if they deposit tokens or cryptocurrency. Unlike isolated and controlled networks that use conventional legal mechanisms, conflicts in public networks are solved using the Majority Voting consensus algorithm.

Moreover, any existing network, such as Ethereum, Corda, or SWIFT, or a stand-alone installation (e.g., a Hyperledger Fabric solution), can be represented in the form of a cloud or a domain. There are two major approaches here:

1) A special VM that executes contracts of other platforms (e.g., Fabric's chaincode) and can:

    a) process platform-specific calls based on destination through Insolar functions, so that a contract can live and operate in both networks; or

    b) use gateway contracts sharing the same code to map operations between Fabric and Insolar networks; or

    c) apply validation consensus rules to reproduce contract execution and confirm the validity of requests coming from Fabric to Insolar.

2) A special gateway that presents functions of another network as contracts of the Insolar network. Initial integration will be for Fabric, as both Insolar and Fabric contracts are based on Golang. Integration with Corda is also under consideration as Insolar has plans to support JVM, but exact details are under evaluation for both directions.

Standard cloud implementation provided by the Insolar platform is based on a blockchain, but a cloud can use alternative storage implementations such as a database management system. However, what makes Insolar unique as both a Blockchain 4.0 and Cloud 2.0 platform is that it ensures interaction and cooperation between differently implemented domains, which is covered in the next section of this document.

8.3. Federation of Clouds

A global network for the ultimate simplification of cross-business interactions will be built on the Insolar platform. Additionally, a group of companies will be able to create their own network tailored to specific tasks, with an option to connect to the global network.

Communicating with other companies on Insolar is as easy as using an email or a messenger app. After joining the network for the first time, a company can immediately start communicating with other businesses on the platform, allowing companies to build both generic and individual interactions quickly and efficiently.

A cloud has to be relatively homogeneous inside in terms of such things as network protocols, encryption schemes, and different regulation procedures; therefore, there will be multiple clouds of different kinds. These are public, industry-specific, nation-specific, and industry consortium clouds.

8.3.1. Public Clouds

These will be configured to enable open membership and majority-based governance on functionality and policies. For instance, to enable Ethereum-like functionality, a cloud should define a root domain that enforces code immutability on contracts, defines mandatory wallets for all objects, and enables anyone to register a new node without KYC or SLA. In such instances, all consensus procedures should be configured as majority-based.

8.3.2. Industry-specific Clouds

Such clouds will provide access to industry standards, procedures, and registries, and they will be governed by industry associations or committees. Policies of domains and objects in these clouds will be based on relevant procedures (e.g., the voting of delegates) and adding nodes will require hardware providers to pass KYC and follow SLA.

8.3.3. Nation-specific Clouds

Nation-specific clouds contain information such as the registration of legal entities, rules, and regulations for personal data, as well as special cryptography standards, while also possibly being restricted by localization of nodes and servers.

8.3.4. Industry Consortium Clouds

Set up by companies or groups of companies, which may or may not be discoverable or only selectively accessible from other clouds, while also possibly implementing alternative, nonblockchain storage.

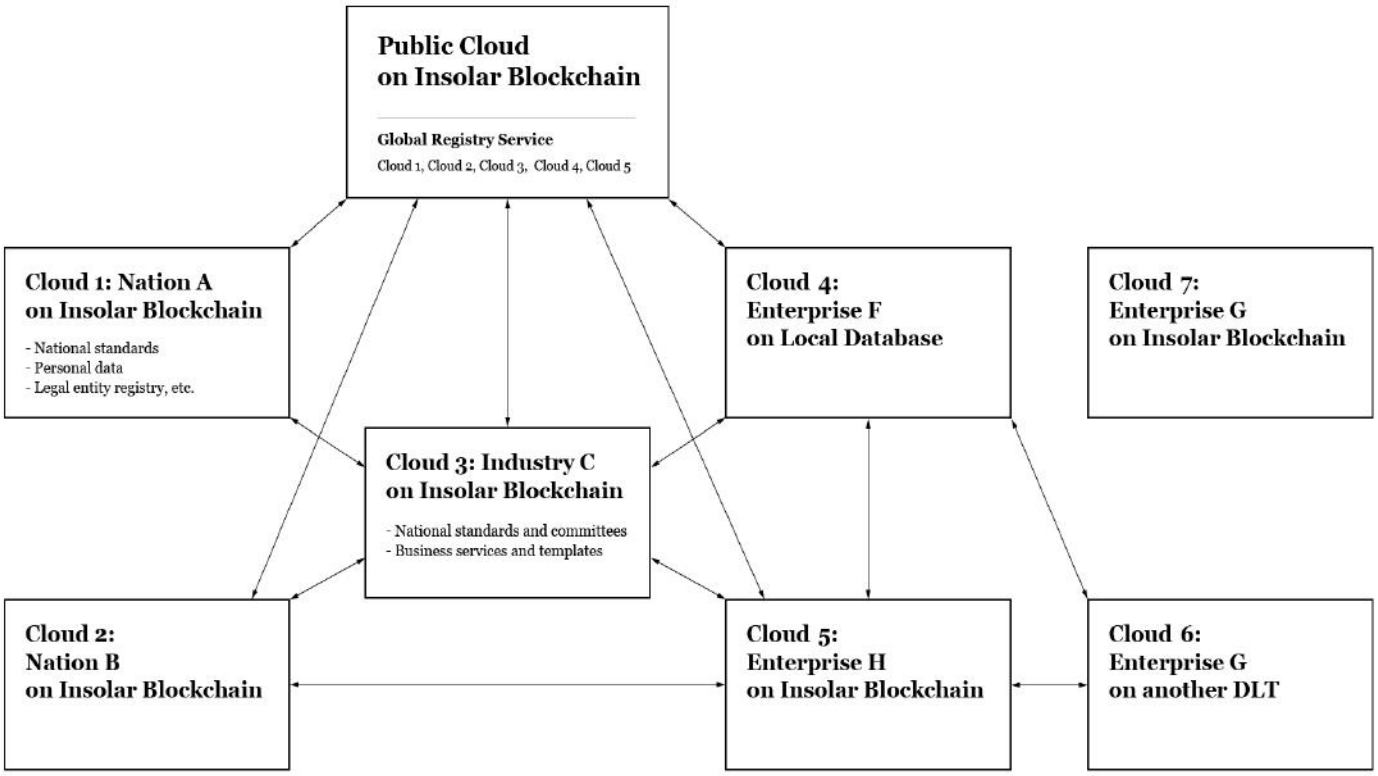


Figure 16. Example of Clouds within a Federation.

In any case, all these clouds can only bring value if they can interoperate. This ability is provided by the Insolar platform because all contract interactions are handled on principles similar to Message/Service Bus, providing guaranteed message delivery and transactional integrity, and there will be no difference in contract logic in regard to where the caller and callee are located. This is why Insolar refers to its Network as a Federation of Clouds: Insolar enables multiple clouds (networks) to be deployed and run individually, yet with the capability to work together.

9. Templates, Applications, and Services

The Insolar platform architecture is designed with both technical and business aspects of usage in mind, and its multilayered nature enables assets to be owned and managed in different ways throughout their lifecycle.

For instance, there will be at least five sets of participants whose relevant assets will have different life cycles: private and business customers represented within the platform, but who own no services; businesses running their own (or template-based) business services on the platform; vendors who develop out-of-the-box business service templates for the platform; cloud operators managing the necessary financial and legal liabilities between participants, including running Capacity Marketplace service(s) for a cloud; and hardware capacity providers who provide and own nodes.

To satisfy the requirements of all these participants, Insolar needs a business-oriented ecosystem built around the technical core provided by the Insolar platform. To handle this properly and avoid a shift in the mission of the Insolar platform, Insolar Business Solutions has been created to design and implement the necessary business-oriented toolkits and elements of the outer landscape.

9.1. Ecosystem Layers

The complete outer landscape will include a Business Foundation created by Insolar Business Solutions on top of the Insolar platform and several more layers created via the joint effort of Insolar Business Solutions and its partner companies. These layers will be Business Templates and Whitelabel Solutions (detailed subsequently). Finally, there are Business Applications and Services: B2B & B2C solutions operating in the Insolar ecosystem.

This approach enables the Insolar ecosystem to be extended by different participants, especially at upper layers where strong business expertise is necessary.

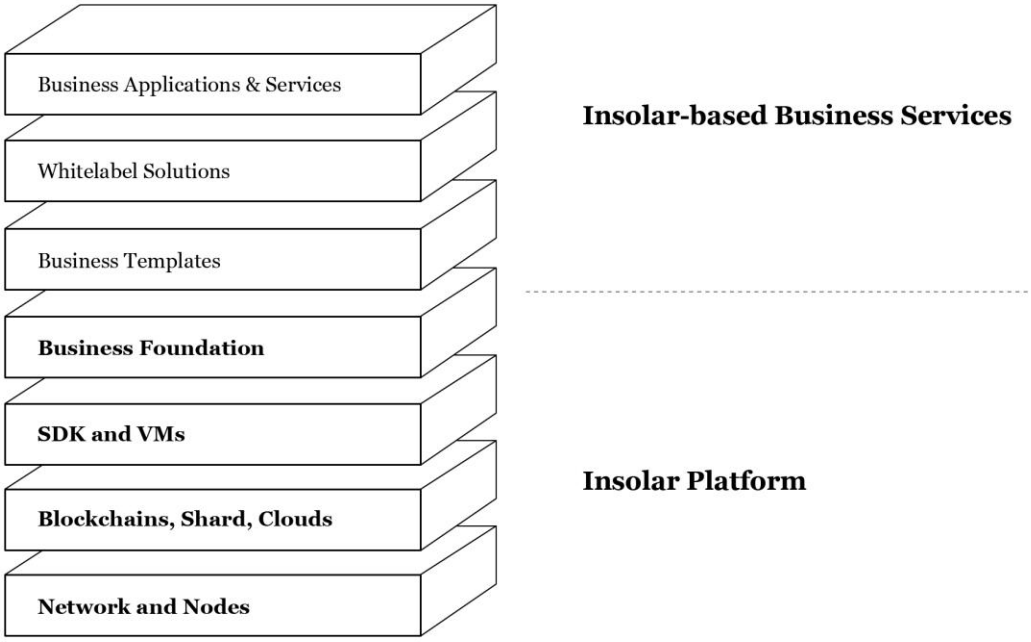


Figure 17. Ecosystem layers diagram.

Table 9. Ecosystem layers.

Layer	*Description*
Business Applications and Services	Business-owned applications and services provided and supported by various companies and vendors.
Whitelabel Solutions	Industry-specific, ready-to-deploy business solutions built by any external vendors and/or by Insolar.
Business Templates	Ready-to-use "blocks" for applications and services (e.g., supply agreements, document workflow).
Business Foundation	Toolkits for business templates to access platform execution layers and external/integration functions.
SDKs and VMs	SDKs to extend Insolar platform capabilities, as well as to run and to maintain the Insolar platform. Enables the use of different VMs with transparent interoperability.
Blockchains, Shards, Clouds	Scalability and rebalancing mechanisms, data integrity, sharding and scattering, execution tracing.
Network and Nodes	Node identification and connectivity, security and inspection of data flows, gateways and firewalls.

9.2. Business Solution Example Using Layers

The following example illustrates a layered business solution built on Insolar:

A logistics service provider, Company A, runs an instance of a white label solution called "Delivery Logistics" on Insolar Enterprise Network. This does not mean Company A needs to own servers; instead, Company A has bought a one-year hardware capacity subscription, which is enough to process 1,000 orders daily. When this order number is exceeded, Company A can purchase additional processing capacities on the spot. All capacities are purchased via the Hardware Capacity Marketplace of the Insolar Enterprise Network.

Additionally, to deploy their "Delivery Logistics" solution, Company A has registered a domain in the Insolar Enterprise Network and agreed to make it discoverable by others. As part of deploying "Delivery Logistics," Company A registered with the relevant payment and IoT services when the solution was deployed.

This white label solution is provided by Vendor B, who charges a support fee to Company A in addition to a charge for the hardware capacities consumed by the solution.

The "Delivery Logistics" solution is using a set of Business Templates, which are components to build specific business solutions. Examples of Business Templates can be templates of contracts implementing GS1 XML documents and processes, templates for smart logistics services agreements, templates and tools for personal data management for target jurisdictions (e.g., GDPR compliant), and templates enabling interactions with various region-specific payment services.

Finally, all these layers rely on services provided by the Business Foundation, which acts as a bridge to the technical side of the Insolar platform. Examples of the Business Foundation's services include generic implementation of balances and accounts (which can uniformly handle both fiat and cryptocurrency balances, fungible asset balances, etc.), billing instruments that allow the collection and conversion of execution tracing details into bills by using various pricing schemes, a Hardware Capacity Marketplace (described in Section 9.3) and a Global Registry Service that provides human-readable names (e.g., DNS is for the Internet), and enables the discovery of domains across different clouds.

9.3. Hardware Capacity Marketplace

One of the key components of the Business Foundation layer is the Hardware Capacity Marketplace (HCM). This component enables users and owners of different services and

contracts running on the platform to trade for hardware capacities such as CPU, network traffic, and storage provided by hardware providers.

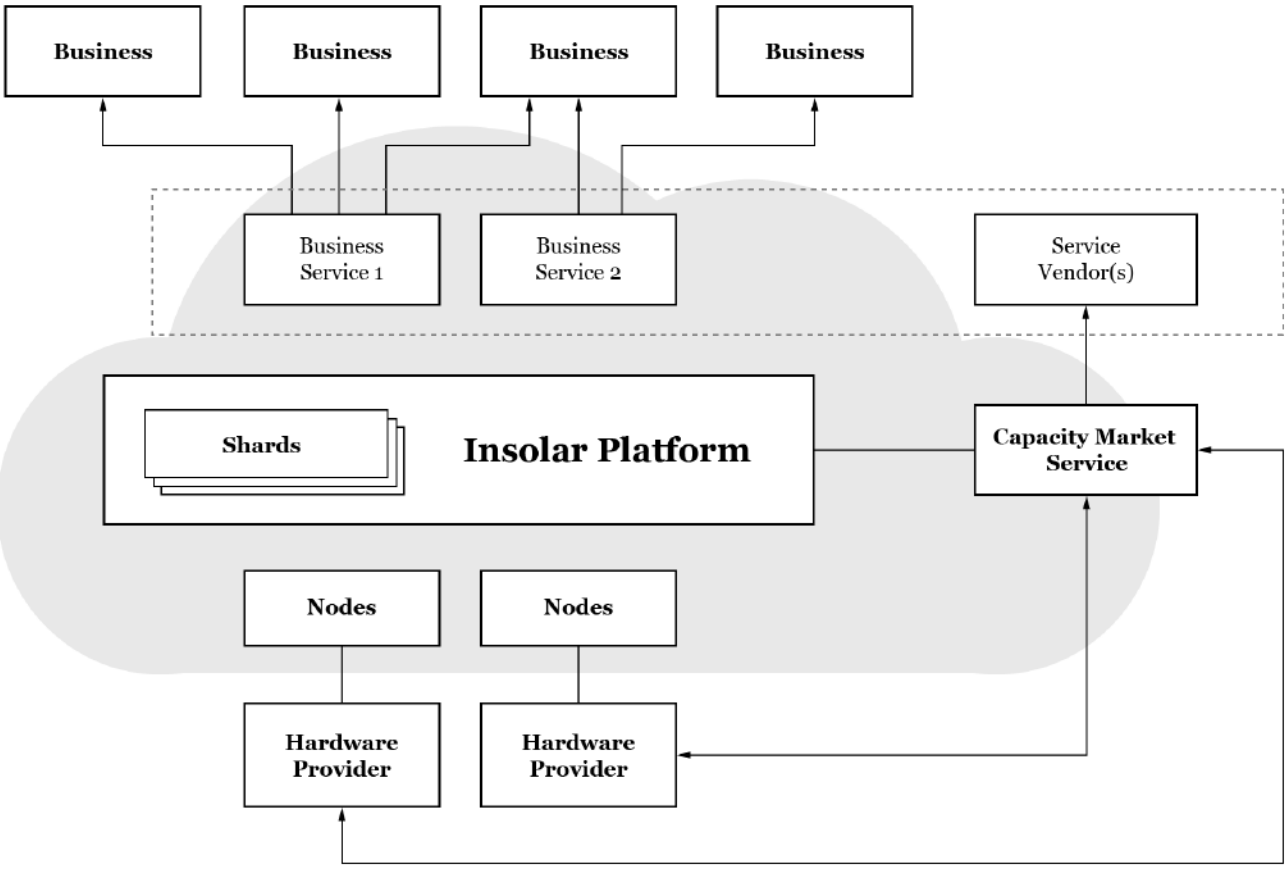


Figure 18. Hardware capacity market.

To serve existing businesses without overburdening IT, the Insolar platform commoditizes hardware capacity, thus making it available as a resource to enterprises in industries such as electricity, coal and other commodities. In Figure 18, this relationship is illustrated from the bottom up, with hardware capacity commoditization shown below the platform and the services made available to businesses shown above. Hardware capacities will be available on both subscription and spot models, with trading of capacities taking place on the HCM.

The HCM will exist in every cloud and will match the nature of the cloud. For instance, for private clouds, there will be no trading, just quotas. For a shared enterprise, cloud hardware providers will have to pass KYC and sign service and liability agreements, and public cloud hardware providers (miners) will need to provide a stake for liability reasons.

The HCM is a place where quotas of hardware capacities and customers using a service will be charged by rules of the service: a customer must have a wallet with INS tokens and pay for service fees and hardware capacities consumed, or the customer can have a prepaid service agreement with the service provider, and the provider absorbs the actual costs.

To allow different pricing models to coexist, the Insolar platform collects quantities of resources consumed by every transaction as Dust and validates them via consensus. Moreover, although the Insolar platform supports the application of limits to different hardware resources for different operations, the Platform itself does not have any in-built pricing, billing, and accounting models.

Instead, each object and transaction carries information about special contracts (so-called Billing and Cost Centers), and when a transaction has been executed and validated, the information collected on the quantity of consumed resources is routed to the Billing and Cost Centers to be processed, invoiced, and settled. This enables the creation and use of a variety of billing and payments schemes, without being dependent on core platform development.

Insolar will publish additional details regarding Billing and Cost Centers and the HCM in later documents.

10. Conclusion

Insolar's fourth-generation blockchain platform represents a significant maturity of blockchain technology. It is designed for enterprise networks and shared business networks. It enables business networks to implement the pragmatic balance of cost, complexity, risk, and business requirements. It hides the complexity of blockchain so that business networks can focus on doing business.

The Insolar platform is designed as general purpose, with an initial focus on running shared processes, handling documents on-chain, and simple reuse of services and solution templates.

In addition to enterprise scalability, Insolar is implementing many advanced features, such as objects and transactions with large on-chain documents, the ability to balance processing costs and speed against transaction value and risks, storage sharding, dynamic workload rebalancing, domains with distinctive governance models, and multiple interconnected networks including efficient support of networks of different scales—from tens to hundreds of thousands of nodes.

Also, the approach described in this document for ledger organization and workload distribution enables easy scaling by storage and a complete audit trail for data access, while reducing the risk of leakage by data scattering. This makes Insolar's solution useful to business networks requiring shared network solutions and decreases the IT burden typical of on-premise or PaaS/IaaS/BaaS solutions.

The Insolar platform has very high scalability, starting with 10,000 transactions per second on networks of 20 to 30 nodes and scaling to millions of transactions per second on networks of thousands of nodes. Near-linear scalability of CPU, network throughput, and storage scaling can be achieved simply by adding more nodes.

Execution of transactions is not tightly coupled with block time and can be extended, thus enabling complex scenarios and dynamic binding of contracts when run time is hard to predict. Block time can be configured as 1 to 120 seconds per network without breaking interoperability, and finality time is double that of block time, with an exception for long-running transactions.

To further increase performance, reduce the impact of network failures, and support varying regional and national regulations, Insolar will support the Federation of Clouds, in which multiple Insolar-based networks (clouds) with varying data, domain, and cryptography policies can interact with one another.

11. Glossary

Term	Definition and Blockchain 1.0–3.0 analogy	Reference
Cloud	Blockchain network under the same node membership policy	Section 3.2
Domain	Decentralized application (dApp) that governs access, consensus, mutability, and other capabilities for other dApps	Section 8.1
Dust	Auxiliary object or record; a log record stored within a sidechain	Section 7.4
Dustline	Removable Sideline for a Dust record	Section 7.3.3
Federation	Interconnected blockchain networks running on the Insolar platform	Section 8.3
Filament	Linked sequence of records	Section 7.2
Jet Drop	Block of a shard chain with adjoined blocks of relevant sidechains that keeps records produced at a specific Pulse	Section 7.7
Lifeline	Chain of records (Filaments) for a single object with relevant Sidelines, representing auxiliary information about the object	Section 7.3
Material Jet	Shard chain that keeps records of a subset of objects contained by a cloud; it is allocated nodes to store related records	Sections 4.4 and 7.6.1
Node	Server that serves hardware capacity to a cloud	Section 6
Object	Smart contract, dApp, application object	Section 4.1
Pulse	Tick that opens up a new block	Section 4.8
Record	Similar to a transaction record; Insolar has a variety of record types	Section 7.1
Sideline	Sidechain (Filament) for auxiliary information, such as logs, etc.	Sections 4.3 and 7.3.2
Virtual Jet	Logical group of affined objects for allocating nodes to process requests related to these objects	Section 7.6.2