RChain Architecture Documentation

Release 0.8.1

Ede Eykholt, Lucius Gregory Meredith, Joseph Denman

January 12, 2017

Contents:

CHAPTER 1

Abstract

RChain's platform architecture is a decentralized, economically sustainable public compute infrastructure. The platform design results from the inspirations of earlier blockchains and builds on top of the shoulders of giants in the disciplines of mathematics and programming language design.

Intended audience: This document is written primarily for software designers and developers who want to help see this vision realized, and for others who want to support these efforts.

1.1 Introduction

The RChain Cooperative and its partners are building a public, Sybil-resistant, and censorship-resistant computing utility. This is an open source project. It will be a blockchain-based platform for specifying, verifying, building, and running decentralized protocols ("smart contracts") that form the base for decentralized applications. On top of this technology platform, a broad array of solutions can be built, including financial services, monetized content delivery networks, marketplaces, governance solutions, DAOs, and RChain's own flagship decentralized social platform.

The decentralization movement is ambitious and will spawn solutions that provide awesome opportunities for new social and economic interactions. Decentralization also provides a counter-balance to abuses and corruption that occasionally occur in some organizations where power is concentrated, including large corporations and governments. Decentralization supports self-determination and the rights of individuals to self-organize. Of course, the realities of a more decentralized world will also have its challenges and issues, such as how the needs of international law, public good, and compassion will be honored.

We admire and respect the awesome innovation and intentions of the Bitcoin and Ethereum creators, and other platforms that dramatically advanced the state of decentralized systems and ushered in this new age of cryptocurrency and smart contracts. However, we also see symptoms that those projects didn't use the best engineering and mathematical models for scaling and correctness in order to support mission-critical solutions. The ongoing debates about Bitcoin scaling and the June 2016 issues with The DAO smart contract are symptomatic of foundational architectural issues. As an example question: Is it scalable to insist on an explicit serialization order for all transactions conducted on planet earth?

RChain's requirements, originating from RChain's decentralized social product and its attention & reputation economy, are to provide content delivery at the scale of Facebook along with support for transaction volume and speed at the scale of Visa. After due diligence on the current state of many blockchain projects, after deep collaboration with Ethereum developers, and after understanding their respective roadmaps, the RChain leadership concluded that the current and near-term Blockchain architectures cannot meet these requirements. Therefore, RChain and partners resolved to build a better blockchain architecture. Together with the blockchain industry, we are still at the dawn of this decentralized movement, and it is now the time to lay in a more correct architectural foundation.

The journey ahead for those who share this ambitious vision is as challenging as it is worthwhile, and this document summarizes that vision and how we seek to accomplish it. We compare the blockchains of Bitcoin and Ethereum, outline the RChain architecture, rationale for its creation, and pointers to initial specifications.

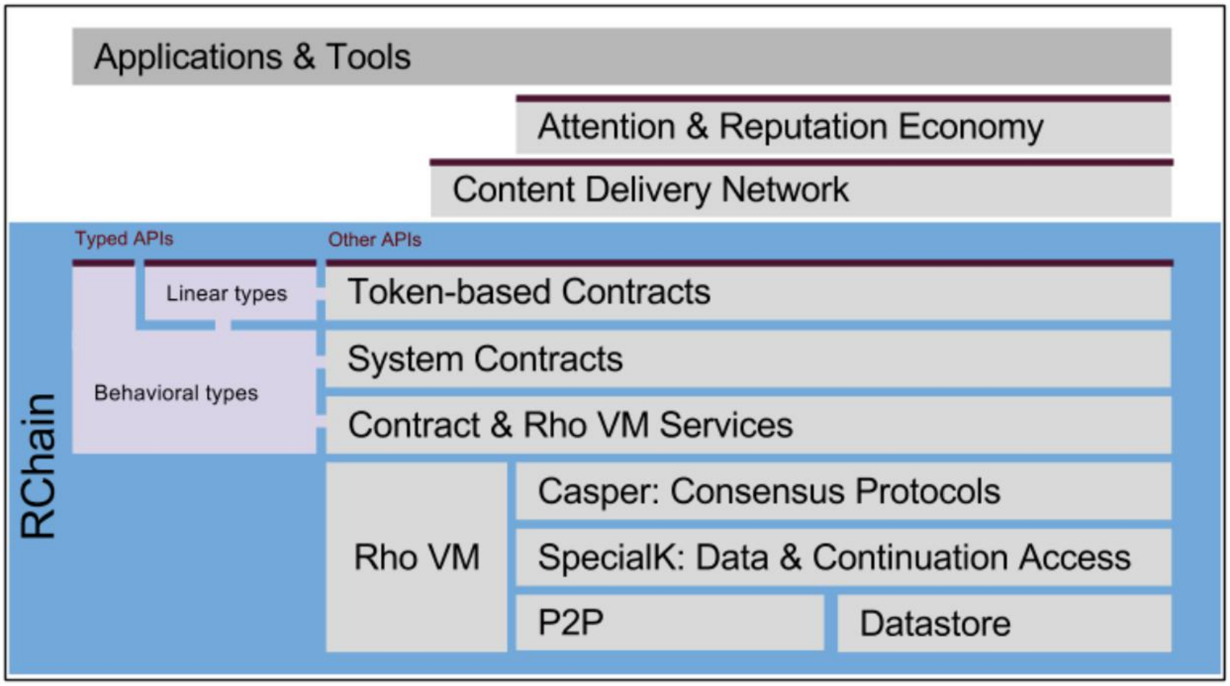
1.2 Comparison of Blockchains

This document assumes the reader is familiar with the basics of Bitcoin and Ethereum. As one approach to introducing the architecture let's compare the characteristics of Bitcoin, Ethereum, and RChain as currently planned.

		Bitcoin	Ethereum	RChain
Semantic Data Structure		Blockchain – a chain of blocks. Each block contains a header that points at the previous block, a list of transactions, and other data.	Blockchain – a chain of blocks. Each block contains a header that points at the previous block, a transaction list, and a ommers (uncles) list.	Blockchain – a graph of blo Each block contains a heade points at one or more previo blocks, a list of transactions other data. For details, see Blockchain Data Semantics
Consensus	*Algorithm*	Proof of work	Current: Proof of work.Future: Proof of Stake – stake-based betting on blocks.	Proof of Stake: Stake-based on logical propositions.
	Finality	Probability of transaction reversal diminishes over time, at each new block confirmation.	Probability of transaction reversal diminishes over time, at each new block confirmation.	Probability of transaction re diminishes over time, at eac block confirmation.
	Visibility	Global	Private, consortium, or public depending on deployed nodes.	Private, consortium, or publ depending on namespace an deployed nodes.
	Revision Mechanism	Soft and hard forks	Current: Soft and hard forks. Future: Block revisions in case of temporary network isolation.	Block revisions in case of temporary network isolation
Sharding	*Heterogeneity*	Homogeneous, i.e., not sharded	Current: Homogeneous, i.e., not shardedFuture: two-level	Sharding of address space a clients to subscribe to select address namespaces withou downloading the entire bloc Able to impose different po different address namespace
	Basis for sharding	N/A	Address range	Dynamic composable shard based on namespace interac
	Number of levels	N/A	Future: two levels: cluster + leaves	Unbounded number of level
	Concurrency	N/A	Current: No Future: Yes	Yes. Allows for concurrent on propositions and commit blocks that don't conflict.
Contracts	*Computational power*	Stack-based language with few instructions	Turing complete	Turing complete
	Runtime architecture	Script runs on Bitcoin Core, Libbitcoin, and other native implementations	Ethereum Virtual Machine implemented on multiple platforms	RhoVM implemented on m platforms
	Programming language	Script	Solidity, Serpent, LLL and any other languages that get implemented on the EVM.	Rholang and any other lang that get implemented on the RhoVM.
Block Size		1MB	Dynamic	Dynamic

1.3 Architecture Overview

The primary components of the architecture are depicted below:



Like all "layer cake" views of architecture, this diagram is a simplification of the actual architecture. At first glance, you'll notice there are components expected in blockchain architectures, but also components that might not be as expected All data managed by the platform requires some associated payment. Of course, an application could also manage its own data, and that data could be referenced via a pointer stored on the blockchain.

In addition to the datastore at the base of the architecture, a consensus protocol and peer-to-peer gossip network form the foundation.

Above that, the SpecialK Data & Continuation Access and Cache layer is an evolution of the existing SpecialK technology (including its decentralized content delivery, key-value database, inter-node messaging, data access patterns, and privacy protecting agent model).

The Casper consensus protocols assure that nodes reach agreement about the contracts, contract state, and transactions for which each node is interested.

Blockchain contracts (aka smart contracts, protocols, or programs) will be written in a new domain-specific language for contracts called Rholang (or in contract languages that compile to Rholang) and then executed on the Rho Virtual Machine on a number of native platforms.

Smart Contracts include some essential system contracts as well as those providing capabilities for tokens and application-supplied contracts.

A metered and monetized content delivery network (CDN) is enabled through token and micro-payment contracts, accessing a mix of blockchain and off-chain data.

The Attention & Reputation Economy provides a model and set of interactions for motivating respectful and economic creation and dissemination of information within social networks.

In the architecture, there will be several APIs, especially at the top layers. Typed APIs will provide access to the RhoVM, Contract Services, and individual contracts. In addition other APIs (including RESTful APIs) will be provided for accessing the CDN, and the Attention & Reputation Economy.

We'll detail these components in the sections below, from the bottom-up. But first, let's discuss the requirements and software architecture approach motivating this platform solution.

1.4 Requirements

Let's look at the requirements for the platform from the vantage point of the developers building applications on top of it. Then, let's look at what is required of the platform itself in order to achieve those requirements.

Requirements for Decentralized Application Developers

- Fully decentralized
- Tamper-proof blockchain for "immutable" history
- Smart contract state (conserved quantities and VM state) reliably replicated
- Support for multiple tokens
- Ability to write predictably secure software contracts
- Scalable

Requirements of the Architecture

- Design with provably correct approaches
- Data separation using namespace addressing to reduce unnecessary data replication of otherwise independent tokens and contracts
- Support for concurrent protocol execution
- Distributed and decentralized
- Minimal external dependencies
- Peer-to-peer and discoverable nodes
- Consensus protocol that is computationally efficient and not resource-intensive

Non-Requirements

- There is a long list of items the architecture will not address, but let's list a few to dispel what might otherwise be common misperceptions. For example, the architecture will not address:
- Compatibility with smart contracts or scripts written on other blockchain technologies
- Automated coin conversion within the platform, since this can be better handled at the application level

1.5 Architectural Approach

Building quality software is challenging. It is easier to hand-craft clever software; however, the resulting software is often of poor quality, riddled with bugs, difficult to maintain, and difficult to evolve. Inheriting and working on such software can be hellish for development teams. When building an open-source system to support a mission-critical economy, we reject a minimal-success mindset in favor of end-to-end correctness.

Therefore, we resolved to meet the requirements stated in the earlier section, and to:

- Build quality software that implements well-specified protocols.
- Build software based on software architecture patterns and other correct-by-construction approaches.

- Take cues from mathematics. Use formal verification of protocols, leveraging model checking and theorem proving.
- Make evidence-based decisions with supporting rationale for design decisions.
- Choose functional programming paradigm, since it better enables distributed and parallel processing.
- Apply best practices of software patterns, including compositionality.

1.6 Pseudonymous Identity and Cryptography

Like other Blockchains, RChain will use elliptic curve cryptography (ECC). The exact curve and address formats have not yet been selected.

There are several areas in which cryptography is employed, including:

- Transaction signing
- Data encryption per channel
  - based on Diffie–Hellman key exchange,
  - within and across nodes, and
  - in datastores
- Obscurity of keys and data in DHT

1.7 Blockchain Data

1.7.1 Data Semantics

Like Ethereum, the RChain blockchain will store contracts and their serialized state. UTXO-style transactions will be implemented with simpler system-level contracts. Like Bitcoin and Ethereum, tamper-proof blockchain semantics will be used to create a history of blocks. The blockchain's main purpose is to efficiently store essential state, any necessary sequencing, and timestamping.

Note that the math underlying this blockchain semantic structure is known as a Traced Monoidal Category. For more detail see Masahito Hasegawa's paper on this topic, Recursion from Cyclic Sharing - Traced Monoidal Categories and Models of Cyclic Lambda Calculi.

The RChain design considers all storage "conserved", although not all data will be conserved forever. Instead, data storage will be leased and will cost producers of that data in proportion to its size, contract complexity, and lease duration. Unlike Bitcoin and Ethereum, immutable data is not promised to be truly forever; however, a very long lease duration is equivalent.

The simple economic reason justifying leasing is that storage must be paid by someone or it cannot be maintained. We've chosen to make the economic mechanism direct. It is really an environmentally unfriendly idea that storage is made "free" only to subsidize it by an unrelated process. A small part of the real cost is measurable in the heat signatures of the data centers that are growing to staggering size. This charging for data as it is accessed also helps reduce "attack" storage, the storage of illegal content to discredit the technology.

A variety of data is supported, including public unencrypted json, encrypted blobs, or a mix. This data can also reference off-platform data stored in private, consortium, public, or obscure locations and formats.

1.7.2 Data Storage

Data will be accessed using the SpecialK semantics and physically stored in a key-value database. A given node can choose which address namespaces it cares about, so not all data needs to be replicated in every node.

1.7.3 Addresses and Sharding/Compositionality

In contrast to other blockchains, where addresses are public keys (or hashes thereof), RChain's address space will be structured. This is similar to how both the Internet and the web works, with IP addresses and URLs, respectively. A structured addressing approach allows programs to talk about "location" in a much more nuanced and fine-grained way. This design choice enables fast datalog queries based on those namespaces and better system performance by analyzing communication patterns to optimize the sharding solution.

This sharding solution allows:

- Dynamic composable sharding based on namespace interaction
- Concurrent betting on and committing of blocks that don't conflict.
- Clients to subscribe to select address spaces without downloading the entire blockchain. Able to impose different policies such as maximum transaction size on different address ranges.
- Arbitrary number of levels of address namespace.

For additional information, see Linear Types Can Change The Blockchain (pdf, lex, hangout video), which describes the inspirational math and thinking in this area. Linear Types provide a nice way to decompose the blockchain in a scalable fashion. It already has sharding semantics in it, that is in the type system.

1.7.4 Namespace Definition and Policy

In order to support many of the use cases that users of Bitcoin find valuable as well as broader use cases, namespace definitions will have a corresponding policy set that constrains its use, for example by setting:

- maximum contract code size,
- maximum data size,
- minimum lease time,
- maximum lease time, and
- other parameters

With policies such as these, a namespace can be defined to provide better guarantees of fast transaction speed and immutability, for example.

1.7.5 Contract Ownership, Transactions, and Messages

RChain's contract accounts, transactions, and messages are analogous to those in Ethereum.

1.7.6 Rate-limiting Mechanism

RChain's VM will implement a rate-limiting mechanism that is related to some calculation of processing, memory, storage, and bandwidth resources. This mechanism is needed in order to recover costs for the hardware and related operations. Although Bitcoin and Ethereum (gas) have similar needs, the mechanisms are different. Specifically, the metering will not be done at the VM level, but will be injected in the contract code (via source-to-source translation that is part of the compilation process [1]).

1.7.7 Tokens

Somewhat similar to Omni Layer, multiple types of tokens will be supported. These tokens will have different properties depending on their type, including parameters such as:

- supply (initial supply, supply growth function, and final supply),
- fungibility, and
- other properties

For each type of token, there will be a link between its class (i.e., its set of distinguishing properties) and the rate-limiting mechanism.

1.8 Contracts

An RChain contract is a well-specified and well-behaved program that interacts with others. Contract interaction with clients or other contracts is via transactions.

When the contract at a given state needs to be evaluated, it is read from the blockchain and deserialized into RhoVM intermediate representation (IR) of the contract with its state parameters. This is via a delimited continuation pattern. The RhoVM IR is compiled into another VM format that is then executed. After the contract is run to its next transaction state, the resulting state is serialized and again stored on the blockchain.

1.8.1 Contract Execution Model, Rholang, and RhoVM

This section describes the essential requirement for decentralized concurrency in Internet-scale applications, along with the compute models and programming languages that best suit that requirement. Rholang is introduced, which is a behaviorally typed, reflective, higher-order process language.

1.8.2 Concurrency Requirements

A platform supporting a global, decentralized compute utility that supports a wide variety of applications must scale, and concurrency is essential to achieve that. Transactions that do not interact must be able to complete at the same time, because to enforce a sequencing constraint forces all nodes to process all transactions. Such a sequencing constraint is essentially what causes blockchains in their current form to be fundamentally unscalable.

When we say “concurrency”, we’re not just talking about multi-threaded implementation of functions, but handling of non-blocking I/O and concurrent processes within and across nodes. The Internet itself is built out of billions of autonomous computing devices each of which is executing programs concurrently with respect to the other devices, but also concurrently on the devices themselves, as most modern hardware supports native multi-threading capability. Decentralization places special emphasis on the autonomy and independence of devices and programs running on them. The APIs of centralized trusted third parties, which programmers could pretend were part of a giant sequential computer, will become a thing of the past. Even inside those organizations sequential architecture is giving way to lots and lots of autonomously executing microservices.

[1] Ethereum 2.0 is intending on following the same technique.

1.8.3 Mobile Process Calculi

There are relatively few programming paradigms and languages that handle concurrent processes in their core model. Instead, they bolt some kind of threading-based concurrency model on the side to address being able to scale by doing more than one thing at a time. Mobile process calculi provides one model, which we've chosen. They provide a fundamentally different notion of what computing is. In these models, computing arises primarily from the interaction of processes.

The family of mobile process calculi provides an optimal foundation for a system of interacting processes. Among these models of computation, the applied π-calculus stands out. It models processes that send queries over channels. This approach maps very well onto today's Internet and has been used as the tool of choice for reasoning about a wide variety of concerns essential for distributed protocols.

Beyond this basic fit with the way the Internet computes, the mobile process calculi have something else going for them: behavioral types. Behavioral types represent a new kind of typing discipline that constrains not only the shape of input and output, but the permitted order of inputs and outputs among communicating processes. Getting concurrency right is hard, and support from this kind of typing discipline will be extremely valuable to ensure end-to-end correctness of a large system of communicating processes.

1.8.4 Rho-calculus

Even a model based on the applied π-calculus and equipped with a behavioral typing discipline is still not quite the best fit for a programming language for the decentralized Internet, let alone a contracting language for the blockchain. There's another key ingredient: The rho-calculus, a variant of the π-calculus, was introduced in 2004 and provided the first model of concurrent computation with reflection. Reflection is now widely recognized as a key feature of practical programming languages. Java, C#, Scala, have eventually adopted reflection as a core feature, and even OCaml and Haskell have ultimately developed reflective versions. The reason is simple: at industrial scale, human agency is at the end of a very long chain of programs operating on programs. Programmers use programs to write programs, because without the computational leverage it would take too long to write them at industrial scale. Reflection is one of the key features that enables programs to write programs, providing a disciplined way to turn programs into data that programs can operate on and then turn the modified data back into programs. Lisp programmers have known for decades how powerful this feature is and it took the modern languages some time to catch up to that understanding. The rho-calculus is the first computational model to combine all of these core requirements: behaviorally typed, fundamentally concurrent, message-passing model, with reflection. For details, see A Reflective Higher-order Calculus.

1.8.5 Rholang

Rholang is a fully featured, general purpose, Turing complete programming language built from the rho-calculus. Rholang is RChain's smart contract language. To get a taste of Rholang, here's a contract named Cell that holds a value and allows clients to get and set it:

```
data Request[a] = Get(Ch[a]) | Set(a)

contract Cell(client: Ch[Request[a]], state: Ch[a]) = {
  select {
    case(Get(rtn) << client; value := state) {
      rtn!(value)
    }
    case(Set(newValue) << client; value <- state) {
      state!(newValue)
    }
  }
}
```

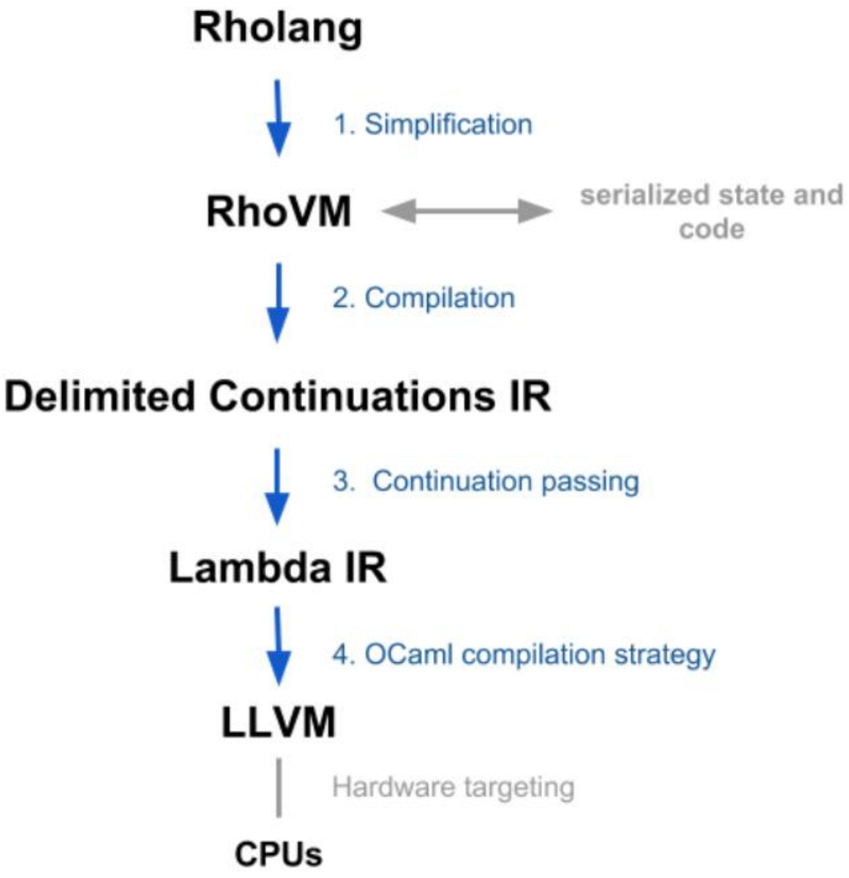
The language is concurrency-oriented, with a focus on message-passing through channels. Channels are statically typed and can be used as single message-pipes, streams or used to store data. Similarly to typed functional languages, it supports algebraic data types and deals with immutable values. It supports formal verification through the use of behavioral types.

A document introducing Rholang in more detail is being produced.

1.8.6 RhoVM

The compiled RhoLang contract is executed in a Rho virtual machine (RhoVM). This virtual machine is derived from the computational model of the language, similar to other programming languages such as Scala and Haskell. In other words, there will be a tight coupling between Rholang and its VM, ensuring correctness. This VM is the machine that will be executed by the compute utility, and we call it RhoVM. To allow clients to execute the VM, we'll build a compiler pipeline that starts with VM code that is compiled into intermediate representations (IRs) that are progressively closer to the metal, with each translation step being either provably correct, commercially tested in production systems, or both.

This pipeline is illustrated in the figure below:



Let's describe these steps in more detail:

- **Simplification.** From programs written in the Rholang contracting language or from another contract language, this step includes: a) injection of code for the rate-limiting mechanism, b) desugaring of syntax, and c) simplification for functional equivalencies. The result targets the RhoVM IR. Note, the state of the RhoVM can be serialized/deserialized to/from storage such as the blockchain.

- **Compilation.** From the RhoVM IR to a Delimited Continuations IR.
- **Continuation Passing.** From Delimited Continuations IR to a Lambda IR. This compilation follows a translation pattern from delimited continuations to a traditional continuation-passing style that has been proven correct.
- **OCaml Compilation Strategy.** From code on a Lambda IR to LLVM, as in the OCaml compiler. Note that LLVM Core libraries provide a modern source- and target-independent optimizer, along with code generation support for many popular CPUs.

For more details see the #rho-lang channel on the RChain Slack (here or join). Early compiler work can be seen on GitHub and discussion on Gitter.

1.8.7 Formal Specification

Rholang will be formally specified, and we are investigating a few frameworks such as K-Framework to achieve this.

1.8.8 Model Checking, Theorem Proving, and Composition of Contracts

In the RhoVM and potentially in upstream contracting languages, there are a variety of techniques and checks that will be applied during compile-time and runtime. These help address requirements such as how a developer and the system itself can know a priori that contracts that are well-typed will terminate.

Formal verification will assure end-to-end correctness via model checking (such as in SLMC) and theorem proving (such as in Pro Verif). Additionally, these same checks can be applied during runtime as newly proposed assemblies of contracts are evaluated.

1.8.9 Discovery Service

An advanced discovery feature that will ultimately be implemented enables searching for compatible contracts and assembling a new composite contract from of other contracts. With the formal verification techniques, the author of the new contract can be guaranteed that when working contracts are plugged together they will also work together.

1.8.10 Validation and Casper Consensus Protocol

Nodes that take on the validation role have the function to achieve consensus on the blockchain state. Validators also assure a blockchain is self-consistent and hasn’t been tampered with and protect against Sybil attack.

The Casper consensus protocol includes stake-based bonding, unbonding, and betting cycles that result in consensus. The purpose of a decentralized consensus protocol is to assure consistency of blockchains or partial blockchains (based on shards), across multiple nodes. To achieve this any consensus protocol should produce an outcome that is a proof of the safety and termination properties of class of consensus protocols, under a wide class of fault and network conditions.

RChain’s consensus protocol uses stake-based betting, similar to Ethereum’s Casper design. This is called a “proof-of-stake” protocol by the broader blockchain community, but that label leads to some misperceptions including overstated centralization risks. Validators are bonded with a stake, which is a security deposit placed in an escrow-like contract. Unlike Ethereum’s betting on a whole blocks, RChain’s betting is on logical propositions. A proposition is a set of statements about the blockchain, for example: which transactions (i.e. proposed state transitions) must be included, in which order, which transactions should not be included, or other properties. A concrete example of a proposition is: “transaction t should occur before transaction s” and “transaction r should not be included”. For more information, see the draft specification Logic for Betting on Propositions (v0.7).

At certain rendezvous points validators compute a maximally consistent subset of propositions. In some cases, this can be computationally hard and take a long time. Because of this a time-out will exist, which, if reached forces validators to submit smaller propositions. Once there is consensus among the validators on the maximally consistent subset of propositions, the next block can easily be materialized by finding a minimal model under which the propositions are valid.

Because of this design and because of the concurrency enabled by sharding of the address space, consensus can be reached for a huge number of transactions at a time.

Let's walk through the typical sequence:

1. A validator is a node role. Validators each put up a stake, which is akin to a bond, in order to assure the other validators that they will be good actors. The stake is at risk if they aren't a good actor.
2. Clients send transaction requests to validators.
3. Receiving validators then create a proposition including a recent transaction.
4. There are sets of betting cycles among nodes:
    1. The originating validator prepares a bet, which includes the following:

    - *source* = the origin of the bet
    - *target* = the destination or target for the bet
    - *claim* = the claim of the bet. This is a block, a proposition, or maximally consistent subset of propositions
    - *belief* = the player's confidence in the claim given the evidence in the justification. This is a denotation of the betting strategy used by the validator.
    - *justification.* This is evidence for why it is a reasonable bet.

    2. The validator places the bet.
    3. The receiving validator evaluates the bet. Note, these justification structures can be used to determine various properties of the network. For example, an algorithm can detect equivocation, or create a justification graph, or detect when too much information is in the bet. Note how attack vectors are considered, and how game theory discipline has been applied to the protocol design.
5. The betting cycles continue working toward a proof. Note:

1. The goal of the betting cycle is for the validator nodes to reach consensus on a maximally consistent set of propositions.
2. A prerequisite condition for the proof is that of the validators are behaving in a reasonable fashion.
3. Eventually the betting cycle will and must converge.
4. The processing is partially synchronous during convergence.
5. With by-proposition betting, the design will be able to synthesize much bigger chunks of the blockchain all at once.
6. Cycles can converge quickly when there are no conflicts.
7. The point of the by-proposition approach is that several blocks can be materialized all at once. This proposal gets around block size limits. There's no argument about it because the maximal consistent set of propositions might allow for hundreds or even thousands of blocks to be agreed all at once. This will create a huge speed advantage over existing blockchains.
8. For each betting cycle a given validator node may win or lose their bet amount.
9. Scalability is achieved via a fine-grained sharding of proposals and via nesting (recursion) of the consensus protocol.

6. Blocks are synthesized by the protocol when there is agreement on the set of maximally-consistent propositions, and this occurs when there is a proof of convergence among the bets. The current betting cycle then collapses.

For additional information, see:

- Consensus Games - An Axiomatic Framework for Analyzing and Comparing a Wide Range of Consensus Protocols.
- For more detail on RChain's consensus protocol, see Logic for Betting – On betting on propositions
- To find out more about Ethereum's Casper and discussions in the Ethereum Research Gitter and Reddit/ethereum.
- The math underlying the betting cycle is an Iterated Function System. Convergence corresponds to having attractors (fix-points) to IFS. With this, we can prove things about convergence with awards and punishments. We can give validator-node-betters maximum freedom. The only ones that are left standing are validators that are engaged in convergent betting behavior.

1.9 P2P Node Communications

Similar to other decentralized implementations, this component handles node discovery, inter-node trust, and communication.

A number of other platform-level protocols will be developed, such as those related to security, node trust, and communications.

1.10 SpecialK: Data & Continuation Access, Cache

The current "RChain 1.0" technology stack delivers a decentralized CDN. Its primary component is SpecialK, which sits on top of MongoDB and RabbitMQ to create the decentralized logic for storing and retrieving content, both locally and remotely.

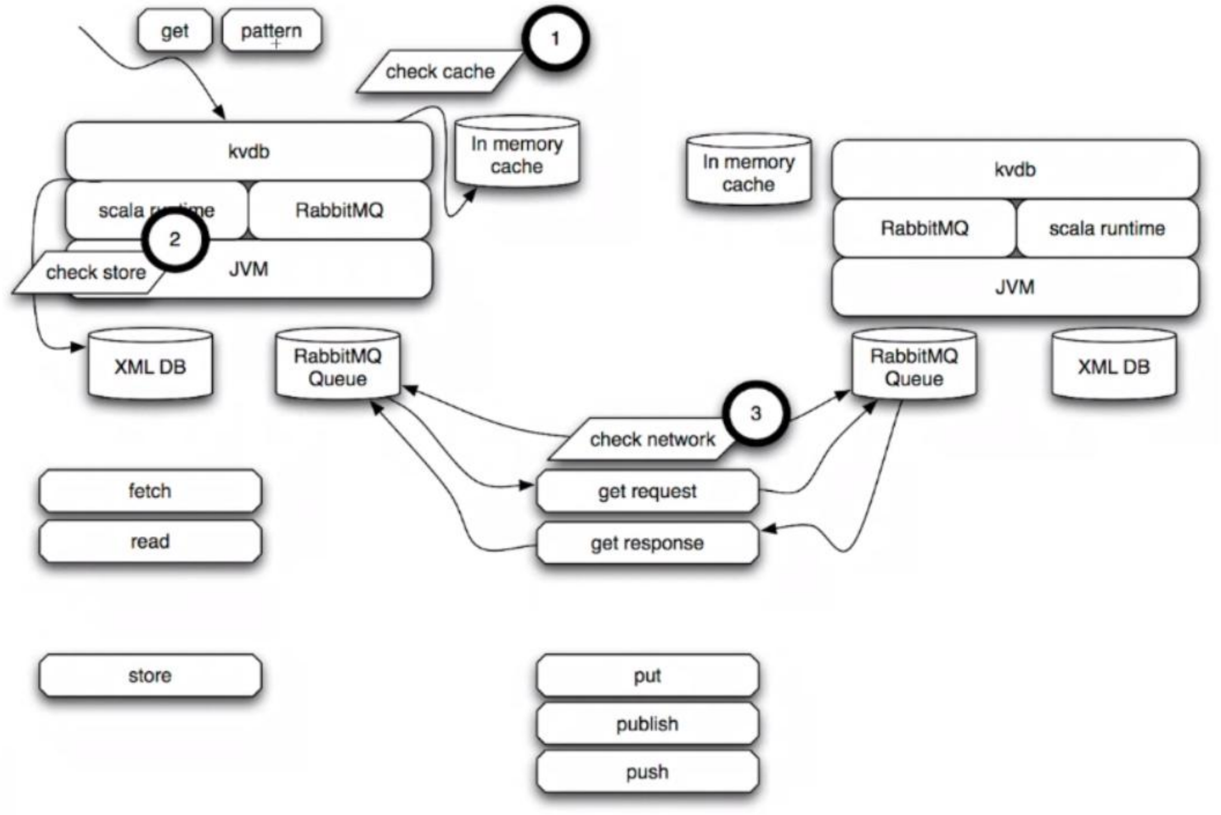
SpecialK implements distributed data-access patterns in a consistent way, as shown below.

	Item-level read & write (distributed locking)	DB read & write	Pub/Sub messaging	Pub/Sub with history
Data	Ephemeral	Persistent	Ephemeral	Persistent
Continuation [1]	Ephemeral	Ephemeral	Persistent	Persistent
Producer Verb (K) [2]	Put	Store	Publish	Publish with history
Consumer Verb	Get	Read	Subscribe	Subscribe

A view of how two nodes collaborate to respond to a get request is shown below:

[1] By convention a continuation function is represented as a parameter named k.

[2] This is only a subset of the verbs possible under this decomposition of the functionality. The verb fetch, for example, gets the data without leaving a continuation around, if there is no data available.



1. The first node checks its in-memory cache, then if it is not found
2. checks its local store, then if it is not found stores a delimited continuation at that location, and
3. checks the network. When the network returns data, the delimited continuation is brought back in scope with the retrieved data as its parameter.

With the RChain platform, the implementation of the CDN will also evolve, although not in its fundamental design.

1.11 Content Delivery Network

This layer will track access and storage of content. Software clients will be required to pay for creation, storage, and retrieval of all content delivered to/from the CDN, via microtransactions. Since storing and retrieving content is not free, why should a technical solution make it free to users like centralized solutions that subsidize the cost in indirect ways? With the promise of micropayments, the RChain platform can more directly charge for the storage and retrieval of content.

1.12 Attention & Reputation Economy

From a user-centric perspective, this economy aims to directly but unobstructedly allow value to be placed on the content's creation, consumption, and promotion. This applies to many types of content. For example, a short textual post is created, sent to an initial distribution list, read, promoted (liked), and made available to even more readers. Or, a short movie can go through the same workflow. Along these paths, attention is given, and rewards can flow back to the content originator and to promoters. Based on one's own engagement with the content exchanged to/from one's connections, each connection's reputation is computed. The reputation rank can be used subsequently to present content in a manner consistent with how the user has demonstrated attention in the recent past.

For more information, see the original whitepaper, RChain - The Decentralized and Distributed Social Network. The latest thinking about Attention & Reputation Economy will be described in Slack discussions and blog posts.

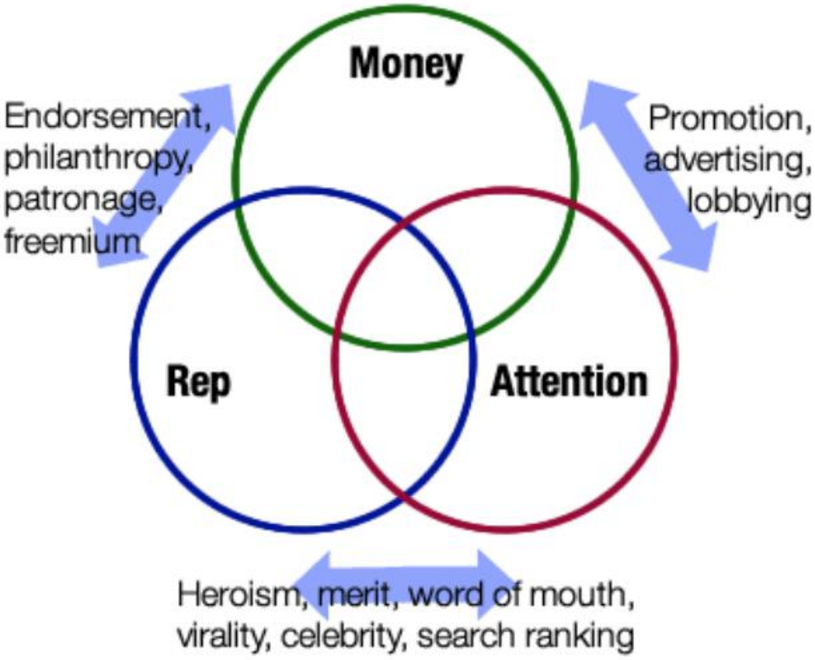

Money
Endorsement, philanthropy, patronage, freemium
Promotion, advertising, lobbying
Rep
Attention
Heroism, merit, word of mouth, virality, celebrity, search ranking

Fig. 1.1: 3economiesvenn.png [1]

1.13 Applications

Any number and variety of applications can be built on top of the RChain Platform that provide a decentralized public compute utility. These include, for example:

- Wallets
- Exchanges
- Oracles & External Adapters
- Custom Protocols
- Smart Contracts
- Smart Properties
- DAOs
- Social Networks
- Marketplaces

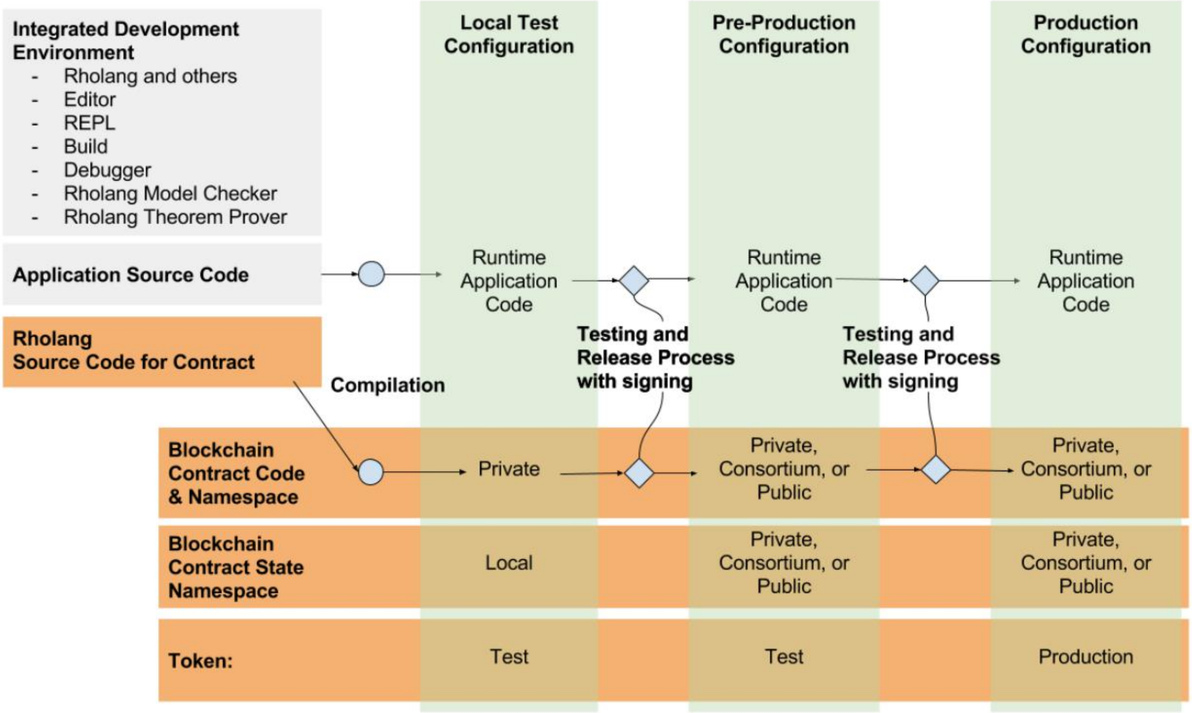
[1] Croll, Alistair (2009) - http://human20.com/free-reputation-for-everyone-the-three-non-traditional-economies/

Several application providers are already committed to this platform, including RChain for its social product, LivelyGig for its marketplaces, weWOWwe for its sports-based social network, and Nobex Radio for a to-be-announced product.

1.14 Contract Development & Deployment

The purpose of this next discussion is to illustrate how namespaces allow for heterogeneous deployment of contracts and contract state. Namespaces is one of the crucial features for sharding, and with that we get the benefits analogous of sidechains, private chains, consortium chains, as well as the distinction between test and production, all under one rubric.

For example, the following diagram depicts some of the possible development, test, and deployment configurations and considerations, and how release management is enabled using namespaces and sharding.


Integrated Development Environment
- Rholang and others
- Editor
- REPL
- Build
- Debugger
- Rholang Model Checker
- Rholang Theorem Prover
Local Test Configuration
Pre-Production Configuration
Production Configuration
Application Source Code
Runtime Application Code
Runtime Application Code
Runtime Application Code
Rholang Source Code for Contract
Compilation
Testing and Release Process with signing
Testing and Release Process with signing
Blockchain Contract Code & Namespace
Private
Private, Consortium, or Public
Private, Consortium, or Public
Blockchain Contract State Namespace
Local
Private, Consortium, or Public
Private, Consortium, or Public
Token:
Test
Test
Production

We'll collaborate with IDE tool vendors to integrate Rholang and validation tools.

1.15 Governance

Like other open source and decentralized projects, and especially those involving money and blockchains, the RChain Platform components will require they be created, tested, released, and evolved with great care. RChain's leadership fully intends to help define these governance processes and to empower a public community to enforce them.

1.16 Implementation Roadmap

The RChain roadmap is currently being developed. Major milestones may include the following:

- Programming model and execution
  - rholang 1.0
  - rhovm 1.0
- Blockchain I
  - bet-by-proposition Casper-style proof of stake
  - blockchain storage
- Blockchain II
  - Metering
  - Token
- Content delivery
  - basic query & update model
- Attention economy
  - post as contract model
  - AMP and REO as stochasticity

1.17 Call for Participation

We invite you to participate in RChain's Slack channels, joining via http://slack.rchain.coop.

We need a variety of talent, but most urgently programmers with solid computer science, formal methods, and ideally experience with mobile process calculi and functional programming. Or, individuals who can demonstrate their ability to quickly learn these disciplines.

We need investors to help fund the building out this architecture. Note that there are many forward-looking statements in this document, and are subject to many risks.

Contact Lucius Gregory Meredith and / or Ed Eykholt for more information.