QDRVM

Polkadot Treasury Proposal
KAGOME – C++ implementation of Polkadot Host
Milestone 4

Proponent:
1544YD9AzZNXq3Bickbk4rGRQ5piRP5AP9b38Nw6boCx58q3 (Quadrivium)
Period: 01.11.2024 – 01.09.2025 (10 months)
Date of submission :
Requested allocation: $981’800 | 981’800USDC
Previous treasury proposal: Milestone 3

Fourth milestone scope:

- Security assessment of new features by SRLabs
- Libp2p coroutines revamp
- Claim queue
- Grid in approvals (retroactive)
- Remove async backing params
- Trie caching
- Traces
- Faster erasure coding (RFC-139)
- Pending code storage key (RFC-123)
- Standardize compressed blob prefixes (RFC-135)
- Reputation system for disputes and statement distribution
- PVF improvements (retroactive):
  - PVF execution parameters
  - PVF priority
  - PVF unix socket
  - PVF clone
- Constrain parachain block validity on a specific core (RFC-103, retroactive)
- Networking improvements (retroactive)
  - QUIC support
  - Audi-v3 (RFC-91)

    - Parallel sync
- Tests & post-security audit improvements (retroactive)

1. Context of the proposal:

KAGOME is a C++ implementation of the Polkadot Host that brings clients diversity to Polkadot and Kusama networks.

Diversifying Polkadot's operating infrastructure is paramount to raising the ecosystems resilience and to open it up to a larger community of innovators. Towards these objectives, the Web3 Foundation aims to have more node operators running KAGOME in the next cohorts of the Decentralized Nodes Program.

By introducing KAGOME and other client implementations, we enhance client diversity within Polkadot, mitigating risks associated with fatal bugs, fostering innovation, and expanding the development community.

During previous work as part of Polkadot treasury proposal 3, the following goals were achieved:

- Grid and cluster topologies
- Elastic scaling
- Disabled validators mechanism
- Validation v3 protocol
- Systematic chunks
- Security assessment (by SRLabs)

These results allowed us to become the first alternative Polkadot Host implementation to join the Polkadot validating set alongside existing validators running Polkadot-SDK. The KAGOME validator is also launched on Kusama and Westend.

This proposal requests funding for the next 4 months, in addition to the previous 6 months of work on features that were retroactively implemented (marked as "Done" in Section 4). This funding is needed to implement new RFCs and introduce new

features to ensure the KAGOME validator's stability and compatibility with Polkadot-SDK.

2. Problem statement

Client diversity is essential for the security and resilience of Polkadot as it helps to mitigate the risk of bugs and exploits. If there is only one implementation then any bug or exploit that is found in that implementation could potentially bring down the entire network. However, with multiple implementations, the risk of a bug or exploit affecting all nodes in the network is greatly reduced.

Another advantage of having multiple clients is that it is opening the way for having another team that already sees the end state of the protocol and therefore has the advantage to implement protocol in the most optimal way.

The importance of having multiple client implementations was highlighted multiple times by Polkadot twitter (source), and Web3 foundation RFP (source). Moreover, W3F recognizes clients diversity importance and allocates 10 million DOT prize for the future multiple implementations of JAM protocol.

To learn more about multi-client philosophy and KAGOME Polkadot Host implementation watch:

- Building alternative clients in Polkadot | Polkadot Decoded 2023
- Polkadot Host architecture in 2024 | Sub0 Asia 2024

3. Alignment with JAM

Quadrivium is also developing a JAM client alongside KAGOME.

However, we recognize the importance of multiple Polkadot Host implementations today, especially since most features are already available and KAGOME is compatible with the Polkadot SDK. Moreover, it's not anticipated that JAM will be launched within the next one or two years.

Also JAM is not the end of the story for Polkadot clients. We will still need parachain clients which will be using Polkadot-SDK even when the JAM chain is live.

Luckily, we can repurpose much of the KAGOME codebase for the future JAM client implementation. For instance:

1. Grandpa – fully implemented and audited.
2. SASSAFRAS – partially implemented.
3. SCALE – fully implemented.

But since we didn't initially intend to reuse these components, some refactoring of KAGOME is required. This will enable these components to be reusable and easily integrated into the future JAM client implementation.

4. Proposed feature set

I. Libp2p coroutines revamp

Libp2p, a modular networking stack enables peer-to-peer communication and interaction between Polkadot nodes. Quadrivium actively maintains a C++ implementation of libp2p.

C++ libp2p library was written in 2019 and since then was actively maintained by our team for the needs of the Polkadot Host implementation.

Today we recognize the need of substantial refactoring aimed at simplifying its asynchronous code which was the place where we spent most of the time during troubleshooting KAGOME. Namely, we plan to take advantage of coroutines features which we didn't have 6 years ago. These changes will be introduced into:

- Reading/Writing operations: update of *read, readSome, writeSome* operations that take callback that is invoked upon completion
- Connections lifecycle: establishing connection, accepting connections, and upgrading connections all involve sequence of asynchronous steps currently managed via callbacks
- Protocol handling: protocol implementations like Ping and Identify involve asynchronous request/response cycles over streams, managed with callbacks
- Muxing: Multiplexers like Yamux and Mplex manage multiple logical streams asynchronously over one physical connection, involving event handling and state management often suited for callbacks or state machines.

With coroutines asynchronous network operations could be written in a more linear, sequential fashion. Instead of:

```C/C++
connection->read(buffer, size, [=](outcome::result<size_t> res) {
  if (!res) return;  // handle error
    connection->write(data_to_write,  size,  [=](outcome::result<size_t>
w_res) {
     if (!w_res) return; // handle error
  });
});
```

We can potentially have:

```C/C++
Task<> handleConnection() {
  try {
    size_t bytes_read = co_await connection->readAsync(buffer, size);
    // Process data...
    co_await connection->writeAsync(data_to_write, size);
    // Continue logic...
  } catch (const std::exception& e) {
    // Handle error
  }
  co_return;
}
```

Status: Not started

II. Grid topology for approvals

To ensure fast propagation of Assignment and Approval messages Grid topology could be used. With grid any message reaches its destination within two hops and number of connections per validator is limited to sqrt{n}

Status: ✅Done

Links:

- https://github.com/qdrvm/kagome/issues/2404

III. Fair claim queue

The claim queue is a structure used in Polkadot to manage the scheduling and allocation of core time among parachains. It keeps track of the claims each parachain has to core time slots, ensuring that each parachain gets a fair share of core time based on its assignments.
However the current implementation doesn't guarantee that each parachain gets a fair share of core time, potentially allowing more aggressive parachains to starve others.
This concern was recently addressed by Polkadot-SDK and should be developed in KAGOME as well.

Status: Not started

Links:

- https://github.com/paritytech/polkadot-sdk/pull/4880
- https://github.com/paritytech/polkadot-sdk/pull/7114

IV. Removal of async backing params

We can deprecate and remove AsyncBackingParameters, specifically max_candidate_depth and allowed_ancestry_len, as they have been superseded by the claim queue. The claim queue already provides mechanisms to enforce limits more accurately, including accounting for parachains sharing a core and elastic scaling.

Status: Done

Links:

- https://github.com/qdrvm/kagome/issues/2352
- https://github.com/paritytech/polkadot-sdk/issues/5079

V. Tracing

Tracing systems like Jaeger/Tempo can help improving KAGOME in multiple ways:

- **Performance Bottleneck Identification**: Identify slow operations in the consensus mechanism, block validation, or networking components.
- **Behavior Analysis:** Understand how different parts of the system behave under various conditions or load scenarios.
- **Root Cause Analysis:** When issues occur in production, tracing helps pinpoint the exact component or function causing the failure.
- **Seamless Integration**: Integrate well with other observability tools like logging and metrics systems, providing a comprehensive view of application's health.
- **Enhanced Visualization**: Visual representation simplifies identifying performance issues, errors, and unusual patterns, making it easier to analyze system behavior.

We require additional research on KAGOME, but we have had positive experience using Tempo and OpenTelemetry, and we will primarily focus in that direction.

Status: 🤦Not started

Links:

- https://grafana.com/oss/tempo/
- https://opentelemetry.io/

VI. State trie caching

KAGOME implements Merkle-patricia trie that stores the state of the blockchain in a secure manner allowing efficient storage proof generation. However, trie construction by itself has certain drawbacks, which do not allow it to perform key-value access operations faster than O(log(n)) complexity, where n is the number of entries in a state database. This problem is very well explained by Basti from Parity during Sub- 2022 (link).

According to our benchmarks storage access operations consume up to 42% of total execution time of the block. Therefore, we may gain drastic improvement in speed if we boost read operations.
However, to perform write and proof operations on trie, we still need to read its nodes which have O(log(n)) complexity. We will address this issue as well during our implementation.
To boost performance state cache will be introduced:
State cache is a copy of key-value state stored in the trie, that is put into a fast hashmap. Whenever there is a read operation, we will read entries from the cache instead of going into the trie
We should also take into consideration that since the state trie operating in a blockchain environment there is a possibility of forks, which we should take care of. A possible approach to achieve this is storing a diffs corresponding to unique forks that gather the writes on top of cached state cache.

Status: In progress

VII. Faster erasure coding (RFC-139)

Erasure coding is a crucial part of Polkadot's Data availability. RFC-139 proposes changes to the erasure coding algorithm and the method for computing the erasure root to improve performance. The changes involve switching to a new erasure coding algorithm described in the Graypaper and replacing the Merkle Patricia Trie with a Binary Merkle Tree for computing the erasure root. These changes are expected to significantly improve encoding and decoding speeds, reduce proof sizes, and enhance verification efficiency.

Status: Not started

Links:

- https://github.com/polkadot-fellows/RFCs/pull/139

VIII. Pending code storage key (RFC-123)

RFC-123 introduces a new intermediate storage key, `:pending_code`, for runtime code in the Polkadot ecosystem. The goal is to ensure that the new runtime code is

stored under a different key before being moved to its final location, reducing the risk of incorrect state decoding and improving the reliability of runtime upgrades.

Status: Started

Links:

- https://github.com/polkadot-fellows/RFCs/pull/123
- https://github.com/qdrvm/kagome/pull/2427

IX. Standardize compressed blob prefixes (RFC-135)

RFC-135 aims to standardize the identification of compressed blob types without decompression. It proposes introducing a set of unique prefixes for different blob types and compression methods, improving the handling and routing of various blob types, and supporting future work involving non-Wasm parachain runtimes.

Status: Not started

Links:

- https://github.com/paritytech/polkadot-sdk/issues/784
- https://github.com/qdrvm/kagome/issues/2068
- https://github.com/qdrvm/kagome/issues/2060
- https://github.com/qdrvm/kagome/issues/2005

X. Reputation system for disputes and statement distribution

As part of security assessment KAGOME team introduced a reputation system for managing peer interactions within the dispute coordinator and statement distribution modules. This system aims to track and update the reputation of peers based on their behavior, applying penalties for actions such as invalid signatures, unauthorized requests, and exceeding rate limits.
This enhances the system's ability to handle misbehaving peers, thereby improving network reliability.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2407

XI. PVF improvements (retroactive)

PVF execution parameters

This change aims to update the PVF execution timeout to be derived from runtime parameters rather than configuration files. This change intends to make the timeout settings more dynamic and adaptable to different execution environments.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2218

PVF priority

To ensure efficiency of the PVF execution process KAGOME team split the PVF queue by job kind, and prioritize the processing of approval jobs over backing jobs.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2305

PVF unix socket

The Kagome team replaced the use of stdin/stdout with Unix domain sockets for the PVF process in the Kagome project. This change aimed to improve the inter-process communication mechanism by adopting a more robust and efficient method.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2327

PVF clone

As suggested by the security assessment of KAGOME, to enhance the security of the pvf worker process it was important to implement a secure clone mechanism for each job.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2307

XII. Constrain parachain block validity on a specific core (RFC-103, retroactive)

This change introduces core index commitments and a session index field in candidate receipts to secure elastic scaling with open collator sets. This change aims to enhance the security and robustness of the parachain block validation process.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2282
- https://github.com/polkadot-fellows/RFCs/pull/103

XIII. Networking improvements (retroactive)

QUIC support

The KAGOME team implemented QUIC transport into cpp-libp2p using the nexus-lsquic library based on boost::asio.

QUIC is UDP based transport that reduces latency by minimizing the handshake process and enabling faster data transmission. QUIC supports multiple streams within a single connection, reducing the risk of head-of-line blocking and improving data throughput.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2298
- https://github.com/libp2p/cpp-libp2p/pull/254

Audi-v3 (RFC-91)

By implementing RFC-91 KAGOME team added a new creation time field for authority discovery records stored in the Distributed Hash Table (DHT). This enhancement is intended to improve the efficiency and accuracy of authority discovery within the network so that nodes can determine which record is newer and always decide to prefer the newer records to the old ones.

Status: Done

Links:

- https://github.com/polkadot-fellows/RFCs/pull/91
- https://github.com/qdrvm/kagome/pull/2151

Parallel synchronization

For KAGOME as Polkadot validator it is very important to always be on the tip of the chain by downloading the highest blocks as soon as possible. For that purpose sync protocol is used.
Previously KAGOME was requesting a block from the block announce sender. To speed up the process of acquiring the block KAGOME team improved this logic by sending multiple block requests to different peers per each block announcement. Similar improvement was made to the logic of catch up requests that are used to fetch the latest GRANDPA justifications.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2320
- https://github.com/qdrvm/kagome/pull/2317

XIV. Faster state pruner (retroactive)

The Polkadot Host's state pruner is a critical component responsible for removing unused state, which stores account information. Storage can quickly fill up without the state pruner. Simultaneously, it is essential to track state values optimally to ensure fast block execution.

We introduced optimizations by moving caching of computed merkle values inside trie nodes to improve computation reuse. We also refactored the trie pruner to

operate on a separate thread, and enhanced the pruning process, and introduced a benchmark for trie pruning.

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2376

XV. Precompilation strategy

KAGOME utilizes WasmEdge to compile and execute Parachain Validation Functions (PVF). Enabling high optimization during compilation can potentially yield the fastest runtime execution. However, it is crucial to compile PVF code quickly to avoid missing voting slots.

To address this, we will introduce precompilation strategies that allow customization of parachain code precompilation. For instance, we can initially precompile all PVFs using the lowest optimization setting and then recompile in the background with higher optimization levels.

Status: Not started

XVI. Tests and security assessment fixes (retroactive)

As suggested by security assessment to ensure conformance with Polkadot-SDK KAGOME team implemented a lot of unit tests that correspond to existing tests in Polkadot-SDK. Tests cover functionality in backing, chunks recovery, grid topology, and more.

KAGOME team also addressed many issues pointed out by SRLabs team to ensure security of KAGOME

Status: Done

Links:

- https://github.com/qdrvm/kagome/pull/2231
- https://github.com/qdrvm/kagome/pull/2240
- https://github.com/qdrvm/kagome/pull/2413
- https://github.com/qdrvm/kagome/pull/2238

- https://github.com/qdrvm/kagome/pull/2389
- https://github.com/qdrvm/kagome/pull/2321

XVII. DevOps and QA maintenance

In order to constantly ensure the quality of KAGOME as well as keep it compatible with Substrate (and potentially other Host implementations such as Gossamer) Quadrivium maintains, monitors and improves multiple environments for KAGOME.

It is important to notice any incompatibilities if any as soon as possible. Therefore we maintain multiple syncing and validating nodes:

- Polkadot validating node
- Kusama validating node x2
- Westend validating node x2

Moreover, we are maintaining the list of zombienet tests and constantly improving our CI to conveniently execute them. During the past 6 months we introduced:

- Introduce ARM builds
- Improved CI
- Migrated some infrastructure to Netcup

In addition, our QA team is constantly ensuring quality of the most recent KAGOME features by running them against different versions of Polkadot-SDK to quickly notice incompatibilities and create bug reports. In addition we conduct regression test scenarios before every release.

5. **Security Assurance** (by SRLabs)

KAGOME's contribution to the Polkadot ecosystem is providing a robust *and secure* alternative client implementation for node operators in the Polkadot ecosystem. Following the initial security audit conducted by SRLabs on KAGOME v0.9.3, followed by the Milestone 3 security audit, it is imperative to assure code changes to continually safeguard the integrity and resilience of KAGOME's implementation.

This section outlines the scope and activities for the next phase of the security assurance, emphasizing proactive measures and collaborative efforts to enhance security over time.

Objectives of the Security Assurance:

- **Assure changes to KAGOME before they go live:**
  Ensure that all modifications to KAGOME are thoroughly reviewed and secured prior to deployment, minimizing the risk of vulnerabilities being introduced in the production environment

- **Ensure code quality and security assurance**
  Implement code review and security assurance processes to maintain high code quality standards, detect potential vulnerabilities early, and ensure robustness of KAGOME's implementation

Support scope for this milestone:

The scope of this security assurance milestone resolves around the following key activities, but is not limited to:

A. Conduct scheduled security audits and follow milestone PRs of KAGOME

a) Perform security audits on the new features developed since the earlier audit, including:

- Pending code storage key (RFC-123)
- Standardize compressed blob prefixes (RFC-135)
- PVF improvements
- Networking improvements
- Audi-v3 (RFC-91)
- Parallel sync
- Faster state pruner
- Constrain parachain block validity in a specific core (RFC-103)

b) In collaboration with the KAGOME development team, monitor pull requests (PRs) submitted to the KAGOME repository. The KAGOME development team will guide which PRs require detailed security audits, ensuring comprehensive review and validation before integration into the main codebase

Please note that this does not include an in-depth review of lilp2p yet so to not delay the completion of this Milestone 4. This additional scope shall be included in Milestone 5 after the community received and accepted Milestone 4.

B. Additional security assurance measures upon request

Provide expert consultation and brainstorming sessions to explore and recommend additional security measures. These sessions will be conducted upon

request and will focus on approaches to enhance the overall security framework of KAGOME

- Conduct brainstorming sessions to identify potential security vulnerabilities in upcoming features or changes
- Engage in discussions on best practices and security measures that can be integrated into KAGOME
- Provide recommendations for
    - Security testing methodologies, such as fuzzing
    - Protocol and architecture enhancements
    - Risk mitigation strategies

Joint alignment sessions will be conducted between Quadrivium and SRLabs to prioritize the support scope during this milestone. This session will ensure that the most critical security needs are addressed promptly and effectively.

The security assurance for KAGOME by SRLabs is designed to be adaptive and responsive, addressing the evolving needs of the project. By ensuring thorough audits of code changes, conducting in-depth reviews and exploring additional security measures, we aim to maintain and enhance the security of KAGOME. This approach will help KAGOME its goal of providing a secure and reliable client implementation for the Polkadot ecosystem.

6. Projected task allocation and payment details

Quadrivium development team

To come up with the total requested payment we provide each task with the man-hours estimate and multiply by hourly rate which is $120/h. This rate was selected to ensure that all KAGOME-related costs are covered including:

- Personnel costs
- Taxes & compliance
- Financial & administrative fees
- Business development

1. Engineering manager
2. Senior C++ developer x 4.5
3. DevOps engineer x 0.5
4. QA engineer x 0.5

QDRVM

Release	Epic	Feature	Description	ETA (hours)	Role involved
v1.2.0	Libp2p coroutines revamp	Connection implementations	Update of read/write loops and handshake logic with co_await	100	Senior C++ engineer + QA
		Transport upgrader	Multi-step upgrade process (RawConnection->SecureConnection->MuxedConnection)	50	Senior C++ engineer + QA
		Multiplexers	Managing stream states and handling frames	50	Senior C++ engineer + QA
		Dialing/Listening	Logic for trying multiple addresses to establish best connection	50	Senior C++ engineer + QA
		ProtocolMuxer	Negotiation state machine in MultiselectInstance will be upgraded to a single coroutine	50	Senior C++ engineer + QA
		Integration into KAGOME	Update KAGOME with new interfaces	50	Senior C++ engineer + QA
v1.1.0	Grid topology for approvals		Integration of grid topology for assignment and approval messages	100	Senior C++ engineer + QA
v1.2.0	Fair claim queue		New structure to manage the scheduling and allocation of core time among parachains	400	Senior C++ engineer + QA

v1.1.0	Removal of async backing parameters		Remove max_candidate_depth and allowed_ancestry_len params as superseded by claim queue	100	Senior C++ engineer + QA
v1.2.0	Tracing		Integration of Tempo and Opentelemetry for improved tracing of KAGOME bottlenecks and behaviour analysis	150	Senior C++ engineer + QA
v1.2.0	State trie caching		A state cache—a fast hashmap copy of the key-value state—will be implemented. Read operations will use the cache, while write and proof operations will still involve trie reads	300	Senior C++ engineer + QA
v1.1.0	Faster erasure coding (RFC-139)		Faster EC to improve DA performance	100	Senior C++ engineer + QA
v1.1.0	Pending code storage (RFC-123)		Updated logic for the runtime upgrade	100	Senior C++ engineer + QA
v1.1.0	Standardize compressed blob prefixes (RFC-135)		Handling of different blob types. Crucial for future support of non-WASM runtimes (e.g. PolkaVM)	100	Senior C++ engineer + QA
v1.0.0	Reputation system in parachains protocol	Reputation system in statement distribution	Keeping track of events produced by peers participating in statement distribution	100	Senior C++ engineer + QA

			and adjusting their reputation based on peers activity		
		Reputation system in disputes protocol	Implement penalties for peers producing incorrect requests that may lead to redundant work of KAGOME validator	100	Senior C++ engineer + QA
v1.0.0	PVF improvements	PVF execution params	Dynamically adjust PVF params such as execution timeout	50	Senior C++ engineer + QA
		PVF priority	Split backing and approval tasks PVF queues	50	Senior C++ engineer + QA
		PVF unix socket	Replaces stdin/stdout with unix socket communications for efficient interaction between KAGOME application and PVF workers	50	Senior C++ engineer + QA
		PVF clone	Spin up new PVF worker processes using clone instead of fork calls. New approach is considered more efficient and secure	50	Senior C++ engineer + QA
v1.0.0	Constrain parachain block validity on a specific block (RFC-103)		Introduces core index commitments and a session index field in candidate receipts to secure elastic scaling with open collator sets	300	Senior C++ engineer + QA

v1.0.0	Networking improvements	QUIC Support	New libp2p transport that reduces latency and minimizes handshake process	300	Senior C++ engineer + QA
		Audi-v3 (RFC-91)	Creation time field for authority discovery records stored in the Distributed Hash Table (DHT)	50	Senior C++ engineer + QA
		Parallel sync	Sending multiple requests per each block announcement	150	Senior C++ engineer + QA
v1.1.0	Faster state pruner		Optimized pruning of Polkadot Host state	200	Senior C++ engineer + QA
v1.0.0	Tests & post-security audit improvements		Unit tests ensuring conformance with Polkadot-SDK & post-audit fixes	600	Senior C++ engineer + QA
	DevOps and QA maintenance (10 months)		Maintenance of infrastructure for integration tests, KAGOME nodes in different networks, and CI integration in Github	800	DevOps engineer + QA
	Project management (10 months)		Tasks tracking, team management	800	Engineering manager
	Total			5300	

Hourly rate: 120$/h
Cost (in USD): **$636’000**
Cost (in USDC): **636’000**

SRLabs security assurance team

1. Code assurance lead
2. Senior code assurance auditors x 3
3. Expert code assurance auditor x 1

Task	Description	Hours	Costs

5A. Conduct scheduled security audits and follow milestone PRs of KAGOME	a) Perform security audits on the new features developed since the initial audit, including:
	• Pending code storage key (RFC-123)
	• Standardize compressed blob prefixes (RFC-135)
	• PVF improvements
	• Networking improvements
	  • Audi-v3 (RFC-91)
	  • Parallel sync
	• Faster state pruner
	• Constrain parachain block validity in a specific core (RFC-103)
	
	b) Security review of pull requests (PRs) submitted to the KAGOME repository, guided and prioritized by KAGOME development team
		1600	$291’200

5B. Additional security assurance measures upon request	Expert consultation on additional security measures (e.g. dynamic testing improvements, security best practice workshops)	300	$54’600

Hourly rate: 182$/h

Cost (in USD): **$345’800**

Cost (in USDC): **345’800**

Total cost (Quadrivium + SRLabs)

Cost (in USD): **$981’800**

Cost (in USDC): **981’800**

Mentoring and technical support

The Web3 Foundation will partner on this proposal as technical advisor and deliverables auditor to ensure the completed milestones are technically sound.
In turn KAGOME team will help W3F improve Polkadot Host specification by reviewing spec changes. Kamil (KAGOME project lead) has already joined the Spec committee to decentralize the process of spec development.

Please note funds will come from the community treasury and Web3 Foundation technical team has no control over these funds and will not be rewarded as a reviewer of the milestones. The team will serve the sole purpose of evaluating deliverables in alignment with the community approval of this proposal and this role is based on their past participation in this project.

QDRVM

Appendix & additional information:

KAGOME presentations:

✅ Building alternative clients | Polkadot Decoded 2023
✅ Polkadot Host architecture in 2024 | Sub0 Asia 2024
✅ Optimizing Polkadot's Data Availability | sub0 reset 2024

About Quadrivium

Quadrivium (https://www.qdrvm.io) is a blockchain infrastructure development company founded in 2023. The company specializes in the development of blockchain clients, peer-to-peer networking tools, and zk-cryptography. Quadrivium develops KAGOME Polkadot Host implementation, in partnership with the Web3 Foundation. The company also maintains the C++ libp2p library.

Quadrivium's mission is to build the infrastructure for a decentralized future. The company believes that blockchain technology has the potential to revolutionize many industries, and it is committed to developing the open-source tools and services that will make this possible.

Teams current and past experience

- https://github.com/libp2p/cpp-libp2p/ – official implementation of libp2p – a modular, upgradable network stack providing convenient interface for networking layer in p2p networks
- https://github.com/filecoin-project/cpp-filecoin – C++ implementation of Filecoin network protocol
- https://github.com/hyperledger/iroha – permissioned blockchain from Hyperledger umbrella, that is currently being used in Cambodian CBDC system

About SRLabs

SRLabs (https://srlabs.de/) is home to knowledge leaders securing critical infrastructures in finance, blockchain, energy, and telecommunications. We focus on hands-on hacking resilience – not compliance –, which we shape by combining our hacking research with impactful consulting work for innovation leaders that have a natural thrive for cutting-edge technologies.

SRLabs is one of the leading blockchain audit companies with experience in many Substrate-based blockchains, including the Polkadot layer-0 relay chain and parachains built on top.