

# Heterogeneous Narwhal and Paxos

Tobias Heindel<sup>a</sup>, Aleksandr Karbyshev<sup>a</sup>, and Isaac Sheff<sup>a</sup>

<sup>a</sup>Heliac AG

\* E-Mail: [tobias@heliac.dev](mailto:tobias@heliac.dev), [aleksandr@heliac.dev](mailto:aleksandr@heliac.dev), [isaac@heliac.dev](mailto:isaac@heliac.dev)

## Abstract

Blockchain interoperability often requires a mutually trusted layer (in the case of roll-ups or side-chains), or a third party (in the case of bridges). Heterogeneous Consensus protocols introduced the possibility for more direct, decentralized interoperability: chains can commit transactions together, given “enough” overlap in their trust assumptions. Like a bridge, such protocols ensure atomicity under specific trust conditions. Unlike a bridge, chains never relinquish control over the safety or liveness of their data.

However, consensus is only part of the story: efficient mempool architectures like Narwhal can dramatically improve scalability and increase chain throughput. However, these mempools also rely on the chain’s underlying trust assumptions. We generalize Narwhal for a Heterogeneous Trust model, and explain how to use it with Heterogeneous Paxos to commit cross-chain atomic transactions.

**Keywords:** Consensus; Heterogeneous trust; Interoperability; cross-chain; mempool;

(Received: April 16, 2024; Version: Jun 26, 2024)

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Preliminaries and background</b>	<b>3</b>
2.1	Distributed systems protocols . . . . .	3
2.2	Learners & cross-chain transactions . . . . .	4
2.3	Narwhal Mempool DAG . . . . .	5
<b>3</b>	<b>Connecting quorum systems</b>	<b>7</b>
3.1	Generalities about belief systems . . . . .	7
3.2	Beliefs about liveness . . . . .	8
3.3	Shared beliefs about safety . . . . .	8
3.4	Heterogenizing Consensus . . . . .	9
<b>4</b>	<b>On availability and integrity</b>	<b>10</b>

<b>5</b>	<b>Heterogeneous Narwhal DAG</b>	<b>11</b>
5.1	Superimposing Narwhal DAGs . . . . .	12
5.2	The mempool DAG . . . . .	13
<b>6</b>	<b>HNarwhal protocol</b>	<b>15</b>
6.1	The worker protocol . . . . .	16
6.1.1	Batch completion . . . . .	16
6.2	The primary protocol . . . . .	16
6.2.1	Vertex announcement . . . . .	17
6.2.2	Responses to vertex announcement . . . . .	17
6.2.3	Receiving availability commitments . . . . .	18
6.2.4	Integrity votes . . . . .	18
6.2.5	CI reception . . . . .	18
6.3	Interaction with Paxos . . . . .	18
6.3.1	Safety Guarantees . . . . .	18
6.3.2	Liveness Guarantees . . . . .	19
6.3.3	Byzantine Behavior Evidence . . . . .	20
<b>7</b>	<b>Conclusion</b>	<b>20</b>
	<b>References</b>	<b>20</b>

## 1. Introduction

In essence, blockchains maintain replicated state machines backed by trust in validators or miners. Generally, the larger and more trustworthy the validator set is, the more expensive running a chain becomes. If applications must share a chain in order to interact, we end up trying to push all applications onto a chain trustworthy enough for everyone. Attempts at such a global chain (e.g. Ethereum) are very expensive, and still do not meet every application’s trust demands. For instance, JP Morgan does not trust Ethereum with its accounts [LSS20]. Ideally, applications should pick a chain with a trust model that fits their needs, and be able to connect to applications on other chains with suitable trust models.

We address the question of how to take base chains’ heterogeneous trust assumptions into account in a common ecosystem. To represent trust assumptions, we use Heterogeneous Paxos’ *learner graphs* [SWvRM20]. Heterogeneous Paxos facilitates agreement between chains with overlapping (but not identical) trust models, which are surprisingly common among proof-of-stake chains [pr(2)3]. We complement this heterogeneous consensus with a DAG-based mempool based

on Narwhal. Narwhal facilitates much more scalable chains, substantially improving throughput [DKSS22]. Our adaptation preserves safety and liveness guarantees of the base chains, and adds a cross-chain atomicity guarantee. In some ways, this resembles a bridge, but each “bridged” chain can evolve independently, and atomicity guarantees arise naturally from mutual trust.

Our heterogeneous version of the Narwhal mempool, combined with heterogeneous Paxos [SWvRM20], forms a heterogeneous DAG-based consensus protocol that facilitates cross-chain interoperability. As part of heterogeneous Narwhal, we clearly separate out certificates of availability and integrity, inspired by Charlotte [SWB<sup>+</sup>23]. We apply the idea with a two base chains example, using a Heterogeneous mempool DAG without a trusted third party or bridge.

Atomic cross-chain transactions typically require multiple phases [Her18], or combined proposers for multiple chains [RCG23]. Either solution requires strong liveness or synchrony assumptions. In short, Heterogeneous Narwhal preserves full BFT guarantees, allows base chains to evolve at different speeds, and facilitates cross-chain transactions with only partial synchrony.

The remainder of this paper is structured as follows. We first recall essential concepts from distributed systems and Narwhal in particular. Then, we revisit the learner-graph as a data structure that captures trust-assumptions of base chains and we reiterate the changes to consensus, the guarantees that Heterogeneous Paxos provides for consensus. As preparation for the main developments, we describe availability and integrity certificates in the context of learner-graphs in section 4. We then present the main ideas of a heterogeneous mempool DAG and describe the protocol that the validators are running, including how we interface with Heterogeneous Paxos. We describe how together, these can be used to commit atomic cross-chain transactions with specific guarantees.

## 2. Preliminaries and background

This section summarizes basic concepts, notation, and conventions that we use throughout the paper. We shall use the term *algorithm* for the restricted class of computations that neither involve network communication nor require (geo-)distribution of processors. Otherwise, we prefer *protocol* for descriptions including solutions to the consensus problem.

### 2.1. Distributed systems protocols

Protocol participants are called *processes*, in general; a possible synonym of processes is *server* [MR98]. In the context of specific protocols, there might be sub-sets of processes that have a specific *role*, e.g., *acceptor* is a role in Paxos. In

the context of the Narwhal mempool protocol [DKSS22], a *validator* is a group of processes that one may want to think of as a single trust domain. (We sometimes refer to a validator as if it is a single process.)

The set of subsets of a set  $M$  is denoted by  $\wp(M)$ , and a partial map  $f: M \rightarrow N$  between sets  $M$  and  $N$  is a function  $f: \text{df}(f) \rightarrow N$  from the domain of definition  $\text{df}(f) \in \wp(M)$ . For the purposes of the present paper, for each protocol, we assume a fixed set of participating processes  $P = \{p_1, \dots, p_n\}$ .

## 2.2. Learners & cross-chain transactions

We concern ourselves with blockchains featuring *finality* in a partially synchronous network: each has some sets of processes (*quorums of validators*) that a user can hear from, and consider a decision *final*, or *decided*.

Each chain maintains a replicated state machine [AGMS18, Lam78, Sch90] by totally ordering user-submitted transactions, and executing them. For historical reasons, we refer to each base chain as a *learner*: it wants to learn the decided total order of transactions. When chains have exactly the same trust model (*i.e.*, the same validators decide transaction order), they are identical *learners*, and can use a *shared sequencer* [MS23], but this naturally does not work for independent chains.

Agreeing on transaction order can be very useful. In particular, it can allow cross-chain transactions. We can represent a cross-chain transaction as set of transactions, one for each constituent chain. For such a transaction to be atomic, we need some guarantee that either all the constituent transactions will be ordered into their respective chains, or none of them will. For more detailed transactions (*e.g.*, ones where the precise state changed on one chain depends on the “current” state of another chain), each chain must learn the order decided on the others: they require full agreement. Our cross-chain agreement solution builds on Heterogeneous Paxos’ *learner graphs*: expressions of conditions under which learners (chains) must agree, and must be able to make progress. In general, the more similar learners’ trust models are, the stronger their agreement guarantees (at the extremes, they are either fully independent or exactly a shared sequencer).

Recall that the consensus problem for processes  $P$  requires deciding on a common value. We recall the definition of a solution to the consensus problem in terms of validity, agreement, and termination.

**Definition 1** (Consensus requirements). *For a fixed set of processes  $P$ , consensus requires:*

- *termination: every correct process eventually decides on a value,*

- agreement: no two correct processes decide different values,
- validity: processes only decide a value that was proposed.

We use the following terminology concerning the behavior of protocol participants, e.g., in the context of consensus.

**Definition 2** (Liveness, safety, and correctness). *A process that participates in a protocol is*

**safe** if every message it sends conforms to the protocol,

**live** if it will always send messages according to the protocol eventually (even if it also sends other messages that violate the protocol), and

**correct** if it is safe and live.

Byzantine processes may be neither live nor safe; however, we consider the possibility of live processes that exhibit Byzantine behavior: a live and Byzantine process is one whose behavior can be corrected by suppressing some of the messages that it is sending, i.e., it has a correct sub-behavior.

Byzantine fault tolerance is often stated in terms of the assumption that less than  $\frac{1}{3}$  of the processes exhibit Byzantine behavior, i.e., more than  $\frac{2}{3}$  are correct. Quorum systems generalize this approach:

**Definition 3** (Quorum system, quorum [MR98]). *Let  $P$  be a fixed, arbitrary finite set. A quorum system  $Q \subseteq \wp(P)$  is a non-empty set of subsets of  $P$ , any pair of which intersect, i.e.,  $\forall q, q' \in Q . q \cap q' \neq \emptyset$ . Each element of a quorum system is a quorum.*

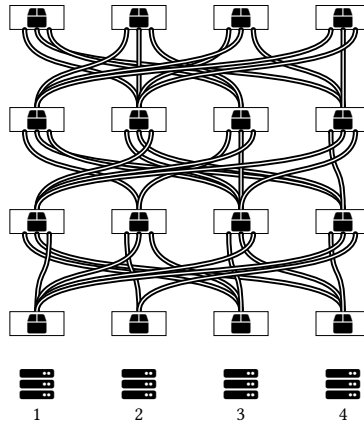
Quorum systems are usually ranged over by the letter  $q$ , possibly with decorations and subscripts.

In the context of a consensus protocol, liveness alone cannot guarantee agreement. We need a minimum number of safe processes. We come back to this topic when discussing the learner-graph in the context of heterogeneous consensus in [section 3](#).

### 2.3. Narwhal Mempool DAG

The Narwhal mempool protocol [DKSS22], conceptually, builds a *global* DAG of block headers (illustrated in Fig. 1). This is the maximal DAG such that each local view of any process is a sub-graph, which one may think of “intersection” of all views. We are mainly concerned with the following properties:

- each validator has proposed at most one block at each height;



**Figure 1.** Mempool DAG of four validators where any three validators form a quorum. Edges correspond to certificates of availability and integrity.

- each block header after genesis references a quorum (*i.e.*, at least  $2f + 1$ ) of blocks headers of the *previous* height.

The edges in Narwhal mempool DAG are induced by certificates of availability and integrity for the referenced block headers. These are special case of availability and integrity attestations as introduced by Charlotte [SWB<sup>+</sup>23]. In the specific context of the mempool DAG, a certificate of integrity attests that a certain block header is *the* block header that some validator has proposed at some height (*i.e.*, no other block header has a certificate of integrity for the same validator and height). Moreover, as block headers only contain hash references to transaction data, Narwhal combines certificates of integrity with certificates of availability, which ensure that all block data (transactions in the mempool DAG) remain available. Certificates of integrity require signatures from a quorum of validators while certificates of availability only require certificates from a *weak quorum*, *i.e.*, a set of validators that intersects with every quorum, also known as a blocking set. We come back to this topic for the heterogeneous setting after reviewing learner graphs as representations of heterogeneous trust assumptions.

While the Narwhal-DAG is being built, validators periodically run consensus, selecting a sequence of *anchor blocks* in the DAG. All blocks (and transactions therein) can then be totally ordered as a function of this anchor block sequence and the mempool DAG.

The main points are three: an mempool DAG is build independently of consensus, consensus is used to choose anchor blocks, and ordering is a function of the sequence of chosen anchor blocks.

### 3. Connecting quorum systems

We provide a short introduction for reasoning about distributed systems with several interdependent quorum systems. One may want to think of these as pre-existing quorum system of some proof-of-stake blockchain and moreover associate each quorum system with a *learner*, an archetypal member of one specific community.<sup>1</sup>

In more detail, this section explains how we can consolidate learner-indexed families of quorum systems into a single structure, the *learner graph* (introduced in [SWvRM20]).<sup>2</sup> We also review if and how the notions of consensus agreement, termination, and validity need to be adapted. Moreover, we describe why learner graphs can be used without modifications to address questions of availability and integrity of transaction data in a heterogeneous version of the Narwhal mempool DAG [DKSS22]. However, we start out with some general remarks about explicit representations of (assumptions about) learner beliefs, very much in the spirit of (dynamic) epistemic logic [BR16].

#### 3.1. Generalities about belief systems

In essence, a belief representation is (an encoding of) a predicate that classifies any given behavior of a validator of interest as possible or forbidden—according to the belief system of one or several interdependent learners.

In this paper, we only consider *static* belief change, *i.e.*,

the objects of agent belief are fixed external truths that do not change, though the agent’s beliefs about these truths may change [BR16].

As an example for belief change, learners may lose trust in the correctness of validators in view of evidence that they have violated the protocol (*e.g.*, double spending). The case of dynamic belief change, which amounts to changeable truths, remains future work.

We consider a finite set of *learners*  $\mathbb{L}$ .

*Note 1:* We do not address the question of how one would gather the required information from learners and what the incentives are for reporting them truthfully. However, one way to understand the learner graph approach is to imagine that each learner (representing its blockchain community) is committed to a publicly known set of beliefs.

<sup>1</sup>Technically, learner is a specific role in the Paxos protocol, *viz.* a process whose goal it is to *learn* about the value that the deciding consensus participants, called *acceptors* in Paxos, have agreed upon.

<sup>2</sup>We have opted to give an alternative, more succinct presentation of the ideas behind learner graphs.

### 3.2. Beliefs about liveness

For the case of beliefs or opinions of learners about liveness of (sets of) validators, we use *learner-indexed quorum systems* (LIQS). Each of these quorum systems consists of sets of validators and the respective learner is assumed to believe that it has named at least one set of validators that consists of live validators only.

**Definition 4** (Learner-indexed quorum system). A learner-indexed quorum system (LIQS) is a function

$$Q : \mathbb{L} \rightarrow \wp(\wp(V))$$

from learners to sets of learner-specific quorums: the learner-specific quorums  $Q_a$  of any learner  $a \in \mathbb{L}$  are required to be a quorum system.

For a learner-indexed quorum system  $Q$  and a learner  $a$ , we denote the learner's set of quorums by  $Q_a$ , whose elements are the learner-specific quorums of  $a$ . We use  $q_a, q'_a$ , etc. to range over learner-specific quorums of some learner  $a$ .

**Example 5** (Proof of stake quorum systems). One important example are validator sets of cosmos zones. Each zone has a different token for staking. The learner-specific quorums for each zone are the sets of validators backed by more than  $\frac{2}{3}$  of stake. This means that the learner believes that there is at least one set of live validators backed by more than  $\frac{2}{3}$  of stake.

### 3.3. Shared beliefs about safety

To consolidate a LIQS to a learner graph, we must express *shared* beliefs of learners about safety: if two learners want to agree (decide only on the same value) given that a specific set of validators are all safe, these validators are a *safe set* shared by those learners.

**Definition 6** (Safe Set System). In a safe set system for a LIQS  $Q$ , each (unordered) pair of learners (including pairs of a learner and itself), map to a set of sets of validators, called their safe sets. We write:

$$a -s- b$$

to mean that  $s$  (a set of validators) is a safe set for learners  $a$  and  $b$  (which can be the same).

In general, if all the validators in one of their safe sets are indeed safe, then a pair of learners must agree (they only decide on the same value as each other).

A Safe Set System defines the *edges* of a *learner graph*, our full representation of heterogeneous trust:



**Definition 7** (Learner Graph). A learner graph is an undirected graph, with vertices labeled with learner-specific quorum systems, and edges labeled with sets of safe sets. Together, the vertices form a LIQS, and the edge labels form a corresponding Safe Set System.

We assume that learner-specific quorums and safe sets are upward-closed, *i.e.*,  $q_a \subseteq q'_a \subseteq P$  and  $q_a \in Q_a$  imply  $q'_a \in Q_a$  for learner-specific quorums and  $a-s-b$  and  $s \subseteq \bar{s} \subseteq P$  imply  $a-\bar{s}-b$  for shared safe sets. This assumption is consistent with intuition, does not impose new restrictions, and allows for an easier way to explain what it means for a learner graph to be condensed [SWvRM20, Lemma 8]. The agreement requirement of our safe set system carries a transitive implication: if  $a$  must agree with  $b$ , and  $b$  must agree with  $c$ , then  $a$  must agree with  $c$ . We call learner graphs that respect this requirement *condensed*.

**Definition 8** (Condensed Learner Graph). A learner graph is condensed iff

$$a-s-b \wedge b-s-c \Rightarrow a-s-c \quad (1)$$

holds for all learners  $a, b, c \in \mathbb{L}$  and sets  $s \in \wp(V)$ .

Heterogeneous Paxos can only solve consensus for *valid* learner graphs, where learners' quorums intersect on honest nodes whenever they must agree:

**Definition 9** (Valid Learner Graph). A learner graph is valid iff

$$a-s-b \Rightarrow q_a \cap q_b \cap s \neq \emptyset \quad (2)$$

holds for all learners  $a, b \in \mathbb{L}$ , sets  $s \in \wp(V)$ , and every pair of learner-specific quorums  $q_a \in Q_a, q_b \in Q_b$  of learner  $a$  and  $b$ , respectively.

Learner graph validity generalizes the quorum overlap requirement of Byzantine quorum systems. Intuitively, whether or not two learners must agree depends on what failures actually happen at runtime. We call this property *entanglement*:

**Definition 10** (Entangled learners). Two learners are entangled if one of their safe sets is composed entirely of safe acceptors.

### 3.4. Heterogenizing Consensus

We summarize the heterogeneous consensus problem [SWvRM20]. The rough idea is that the properties of validity, agreement, and termination become relative to whether learner beliefs are true.

**Definition 11** (Heterogeneous termination [SWvRM20]). A learner terminates if it eventually decides. A heterogeneous consensus protocol satisfies termination if every learner  $a \in \mathbb{L}$  terminates if it has a learner-specific quorum  $q_a \in Q_a$  composed entirely of correct validators.

As a result, a heterogeneous consensus protocol has to deal with the possibility that some learners will fail to finish an ongoing execution. This is a crucial property for cross-chain transactions: no involved chain should be in danger if the other chains get stuck.

Agreement hinges on learners being entangled:

**Definition 12** ([Heterogeneous agreement [SWvRM20]).

*Within an execution, a pair of learners agrees if all decisions for either learner have the same value.*

*A heterogeneous consensus protocol satisfies agreement if, for all possible executions, all entangled pairs of learners agree.*

Finally, validity remains unchanged.

**Definition 13** (Heterogeneous Validity [SWvRM20]). *A consensus execution is valid if all decided values were proposed in that execution. A consensus protocol is valid if all possible executions are valid.*

## 4. On availability and integrity

We define certificates of availability (CA), e.g., for data that is referenced by hashes, and certificates of integrity (CI), which specify a unique datum with certain properties. We now make this precise relative to a learner graph.

We assume a valid and condensed learner graph. Data (identified by hash) can be made available to all learners if sufficiently many validators store a copy.

**Definition 14** (Global weak quorum). *A global weak quorum is a set of validators  $X$  that intersects each learner-specific quorum:*

$$\forall a \in \mathbb{L}. \forall q_a \in Q_a. X \cap q_a \neq \emptyset$$

The archetypal example for global weak quorums are sets of validators holding more than  $\frac{1}{3}$  of the stake of every base chain. For a valid learner graph, every learner-specific quorum is a global weak quorum<sup>3</sup>. Global weak quorums are useful for *certificates of availability*.

**Definition 15** (Certificate of availability (CA)). *For a fixed hash function  $\#$ , a certificate of availability for some data  $d$  is a set of cryptographic signatures over  $\#(d)$  such that the signers form a global weak quorum. Each signature is an availability commitment: a correct signer will keep  $d$  available in their local storage.*

<sup>3</sup> Assuming all edges have some safe set, i.e., all learners agree when all validators are safe

A process issues an availability commitment when it stores data, and can make it available on request. Since data is referenced by hash, they cannot lie about content, only fail to be available. A CA proves that, for any learner, so long as one of their quorums is live, they can retrieve the data.

For integrity, we are mainly interested in the uniqueness. In Narwhal, each validator may at most produce one block at a certain “height”. The idea however generalizes to all settings of where something is required to be the unique  $x$  with a certain property  $\phi(x)$ .

What counts as proof of integrity may vary for learners. However, for a specific learner, signatures from a learner-specific quorum of validators suffice.

**Definition 16** (Certificate of integrity (CI)). *Given a fixed predicate  $\phi[\_]$ , applicable to blobs of data, a certificate of integrity is a set of signatures over the pair of the data and the predicate  $\langle x, \phi \rangle$  such that the signers form a learner-specific quorum. Each signature is called an integrity vote.*

A safe process does not sign integrity votes for both  $\langle x, \phi \rangle$  and  $\langle y, \phi' \rangle$  if  $y$  satisfies  $\phi$ : the whole point is to guarantee uniqueness. Thus, for the original Narwhal, the predicate in question is “ $x$  is the  $n^{\text{th}}$  block of validator  $v$ ”.

Note that, for both integrity and availability, certificates can use aggregated or threshold signatures [BNN07] to save space while still proving the same guarantees. Now, all preparations are in place to describe the global heterogeneous mempool DAG.

## 5. Heterogeneous Narwhal DAG

We now present the main novelty of the paper: a heterogeneous version of the Narwhal mempool DAG [DKSS22]. Recall that the core properties of the Narwhal are *availability* of transactions referenced via hashes, *uniqueness* of each validator’s vertex at a certain height, and the referencing of a quorum of vertices from a previous height—ensuring that sufficiently many vertices are created by correct validators—(see [subsection 2.3](#) for more details).<sup>4</sup>

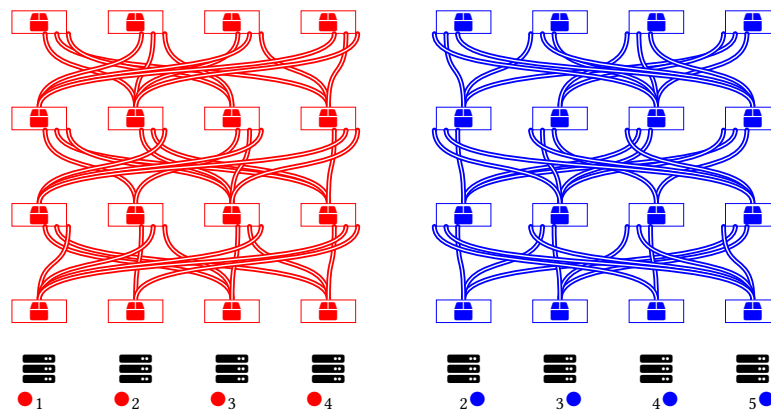
We describe the graph structure in this section (defining vertices, worker hashes, signed quorums, etc.) and describe the actual protocol subsequently, separating out the data structure of interest from the operational context. Concerning the mempool DAG, we put the focus on the global DAG. In view of the complexity of the subject, we kick off with a short synopsis of the main ideas.

<sup>4</sup>See Charlotte [SWB<sup>+</sup>23] for a general approach integrity and availability in the context of blockchains.

## 5.1. Superimposing Narwhal DAGs

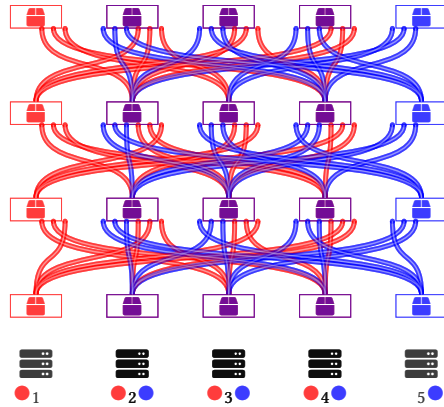
As a running example, let us consider two base chains running Narwhal with proof-of-stake. Instead of building a separate Narwhal DAG for each base chain, the validators will build a single *shared* Narwhal DAG that comprises the information of both. The motivation is that a shared mempool DAG facilitates cross-chain transactions.

A first idea would be a direct “superposition” of DAGs, illustrated in [Figure 3](#): the colored circles next to the validator icons ( $\equiv$ ) represent the stake they hold, namely blue, red, or both. Now, we could simply restrict anchor blocks to those that satisfy the conditions for both base chains. This would lead to *safe* implementation of a shared Narwhal mempool DAG. Unfortunately, the sketched superposition sacrifices liveness: roughly, superimposing Narwhal DAGs leads to a “less live” DAG, relative to *each* of the base chains. In terms of the learner graph, the “superposition” could only produce anchor candidates if a “superquorum” including *both* a red and a blue quorum were live. Thus, the learners of the two base chains would have to have full trust in each other’s chains. Instead, we aim for cooperation with only the minimum number of strings attached.



**Figure 2.** Two “global” Narwhal DAG of five validators with stakes

Before we delve into the details of how we can do better—in general and for this particular example—let us quickly give a picture of how the heterogeneous mempool DAG will look. [Figure 4](#) illustrates a heterogeneous mempool DAG. The first obvious point is that some mempool vertices do not link to anything other than the previous vertex from the same validator. This allows validators to produce vertices as long as they have certificates for availability for transaction data. However, the key feature is that validators can have vertices with a quorum of references for each base chain, independently of the other base chain. The top



**Figure 3.** Naïve superposition of two Narwhal DAGs

left area thus amounts to a vertex with a new height for the red chain, but not a new height for blue chain.

### 5.2. The mempool DAG

The original Narwhal (resp. Bullshark) has a height (or *round*) number for each new block (resp. *vertex*). This generalizes naturally to a number for each learner.

**Definition 17** (Height map). A height map is a partial function  $h: \mathbb{L} \rightarrow \mathbb{N}$  that maps each learner  $x \in \mathbb{L}$  on which it is to defined to  $h(x)$ , the learner-specific height.

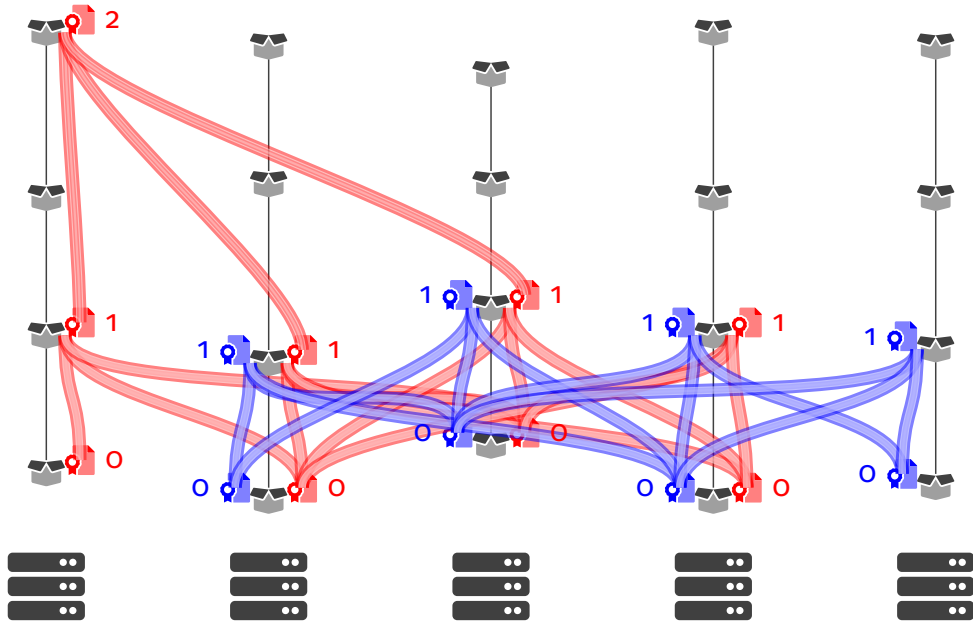
Genesis vertices have a height map that maps all learners to 0. In our running example, we have two learners, namely **red** and **blue**. In Figure 4, the top left vertex will have a height map [**red**  $\mapsto$  2, **blue**  $\mapsto$  1], or [2, 1] (for short). Genesis vertices have [0, 0].


In the original Narwhal, in order to create a vertex, a validator must gather references for a quorum of vertices at the previous height, and include them in the new vertex at the next height. We separate the notion of creating a vertex from incrementing height. In particular, a vertex can increment height for any number of learners (including 0), by gathering references for a quorum of vertices.

**Definition 18** (Signed quorum<sup>5</sup>). A signed quorum at height  $n \in \mathbb{N}$  for a learner  $l \in \mathbb{L}$  is a *CI* and a *CA* for each of a set of vertices, such that the *a-specific* height of each vertex is  $n$ , and the creators of the vertices form an *a-quorum*.

Using signed quorums, we can define our vertices as a generalized version of original Narwhal's:

**Definition 19** (Vertex). A vertex, then, is a data structure that includes:



**Figure 4.** Heterogeneous mempool DAG where certificates of integrity are rendered as  next to the respective vertex, together with the learner-specific height

- $i$ : Public Key, the creator ID (public key)
- $n$ :  $\mathbb{N}$ , sequence number
- $c$ : CA referencing the vertex with the same creator and sequence number  $n - 1$
- $S$ : a (possibly empty) set of signed quorums
- $H$ : a non-empty set of worker hashes

Note that genesis vertices have sequence number 0, and safe validators do not issue multiple vertices with the same sequence number. Thus, safe validators issue a chain of vertices.

Here, worker hashes are much the same as they are in original Narwhal: they uniquely identify a batch of data (transactions) distributed in parallel by this validator (and its worker processes).

An availability commitment for a vertex must attest that not only is the signer storing (and willing to supply) the vertex itself, but all the data represented in all of the worker hashes. As in original Narwhal, the signer can use multiple worker processes to store and supply these in parallel.

**Definition 20** (Vertex Parentage and Ancestry). *A vertex's parentage is a set containing the previous vertex from the same validator (referenced by  $c$ ), and any vertices referenced in signed quorums in the vertex. A vertex's ancestry is a set consisting of itself, its parentage, and all of their ancestry.*

Note that a vertex includes *CAS* for each of its parentage: a validator can *always* fetch and read all of the ancestry of a vertex.

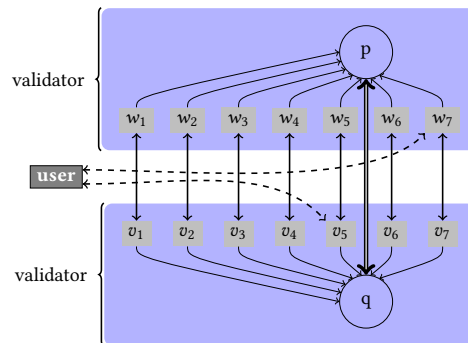
**Definition 21** (Vertex Height). *The height of a vertex is the height map that maps each learner to 1+(the maximum height of any  $l$ -signed-quorum in any vertex in the ancestry), or to 0, if there is no such signed quorum.*

A vertex *must not* include a signed quorum if removing that signed quorum wouldn't change the vertex's height (we don't want redundant signed quorums). Note that it is easy to calculate a vertex's height given only its parentage and their heights. Just as in original Narwhal, heights increment when a vertex's parentage includes a quorum of vertices from the previous height. One might imagine "projecting" a heterogeneous Narwhal DAG into a traditional Narwhal DAG for a specific learner  $l$  by combining all vertices from each validator that have the same  $l$ -specific height.

Next we proceed to the actual protocol, and its connection to Heterogeneous Paxos.

## 6. HNarwhal protocol

The Heterogeneous Narwhal protocol (HNarwhal) assumes an actor model, in which processes send messages as a reaction to events such as other messages arriving, or local conditions becoming true (including local timers).<sup>6</sup> Like original



**Figure 5.** Communication patterns of validators

Narwhal, each validator has one primary and several workers. Each worker has a unique corresponding *mirror worker* on every validator. In Figure 5, we have primaries  $p$  and  $q$ , each of which has seven workers.

We first describe the worker protocol, which in principle could run independently. Then, we describe the primary protocol, which relies on the worker

<sup>6</sup>This roughly amounts to the pattern of event-driven state machines in Erlang.



protocol. Finally, we describe how we interact with Heterogeneous Paxos for choosing anchor vertices.

Workers keep transaction data available<sup>7</sup> such that primaries can “build” the mempool DAG using hash references instead of full transaction data. Heterogeneous Paxos, in turn, chooses a sequence of anchor vertices in order to establish a total order of transactions (refining the partial order from the DAG), very much like in the original Narwhal protocol.

### 6.1. The worker protocol

Users submit transactions (to be ordered) directly to workers. Workers also handle most of the bandwidth between validators. In particular, primaries do not transmit transaction data, because the workers keep it available so that primaries can reference transactions via hash.

The Heterogeneous Narwhal worker protocol is similar to the original Narwhal worker protocol. Each worker copies incoming transactions to its mirror workers. Once it has completed a batch (a set of transactions to be included in a vertex), it distributes a signed hash of the complete batch (a *worker hash*) to both its primary and its mirror workers. Mirror workers communicate to their primaries that they have stored (and can make available) all the transactions involved with this worker hash. Primaries use this information for vertex CAS.

The worker protocol is illustrated in Fig. 6. New transactions ( $\text{TX}$ ) are submitted to the worker. Copies of those transactions ( $\text{TX}$ ) are forwarded to mirror workers.

#### 6.1.1. Batch completion

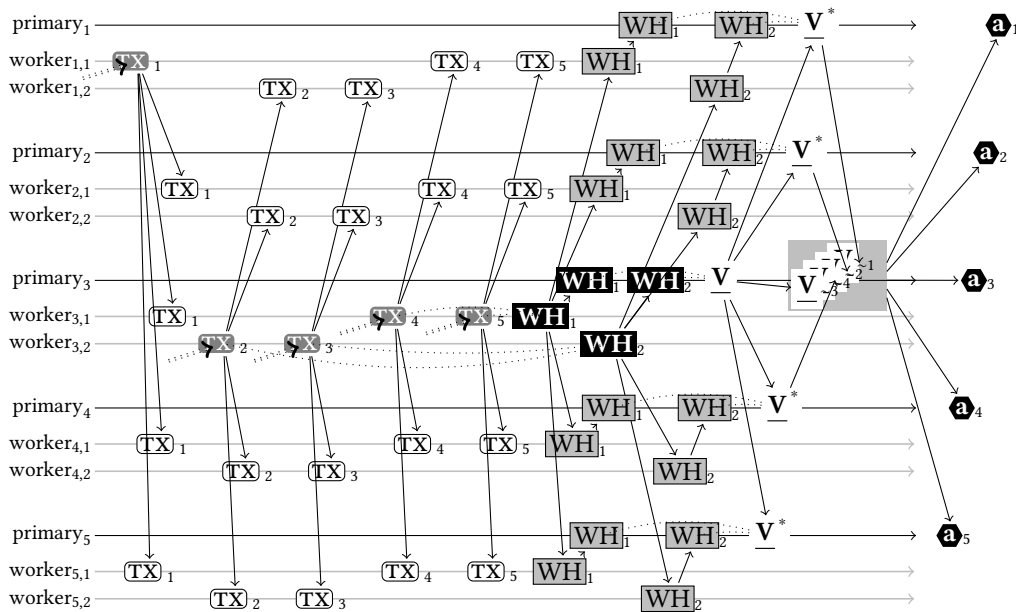
Batch completion may be triggered by a timer that has elapsed or when a batch reaches a target metric (e.g., maximum size). Worker hashes ( $\text{WH}$ ) are formed and broadcast to mirror workers. This mechanism is essentially the same as in the original Narwhal.

### 6.2. The primary protocol

Each validator has one primary process. Each primary produces a sequence of vertices, each carrying a *vertex number*. For a safe validator, these vertices form a chain. The primary protocol builds on the worker protocol, which provides worker hashes. In addition to its sequence number, each vertex has a height map (Definition 21), which generalizes height or “round” in the original Narwhal or Bullshark.

<sup>7</sup>Typically, only until execution, but there can be other reasons to keep data available.





**Figure 6.** Steps towards forming a vertex. Transactions (rendered as **TX**) are submitted to Validator 3, to workers 1 and 2. The vertex is announced and a certificate of availability (**a**) is formed using signatures from Primaries 1,2,3, and 4, and is broadcast.

### 6.2.1. Vertex announcement

Each primary *announces* a sequence of vertices, starting with a genesis vertex. It broadcasts each vertex to all other primaries. In Fig. 6, vertices are rendered as boldface **V**. Each vertex should contain all worker hashes and signed quorums the primary has received so far (but not included in previous vertices).

Finally, (for non-genesis vertices) the primary must create a CA for its previous vertex before it can complete the new vertex. This is done by collecting responses to the vertex announcement (as illustrated on the right in Fig. 6 where CAs are rendered as **a**).

### 6.2.2. Responses to vertex announcement

The first time a primary  $j$  receives a vertex  $v_i^n$  with sequence number  $n$  from another primary  $i$ , it stores it, and responds promptly with an integrity vote  $v_i^n$  (as illustrated in Figure 7 on the left where primaries 1, 2 and 4 send integrity votes to primary 3).

The primary  $j$  ignores other vertices from  $i$  with the sequence number  $n$ .

When it has been notified by its workers that all the referenced worker hashes are available, it also sends an availability commitment for  $v_i^n$  to  $i$ .

### 6.2.3. Receiving availability commitments

Each primary stores availability commitments from other primaries for its own vertices. When it receives enough commitments for a vertex  $v_i^n$ , it creates a CA, and broadcasts it to all other primaries. These may be used in future signed quorums (Definition 18).

### 6.2.4. Integrity votes

Each primary stores integrity votes from other primaries for its own vertices. When it receives enough votes to complete a learner specific CI for a vertex  $v_i^n$ , (if it has not previously sent a CI for  $v_i^n$  for the same learner), it broadcasts the CI to all other primaries (as illustrated left of the double line in Figure 7 where primary 3 is broadcasting a red and a blue certificate of integrity for its vertex). These may be used in future signed quorums (Definition 18).

### 6.2.5. CI reception

Primaries store each received CI, for use in signed quorums (Definition 18). The formation of signed quorums is illustrated right of the dotted double line in Figure 7 where primary 3 has collected enough blue certificates to form a signed quorum.

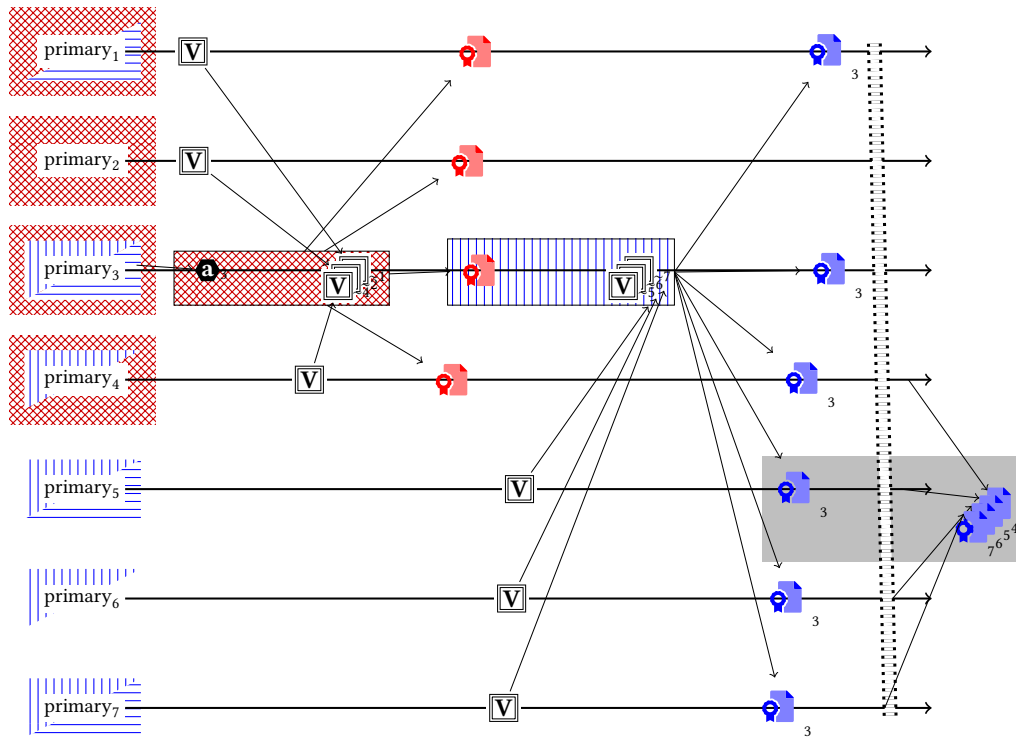
When it announces a new vertex, the primary should form signed quorums of the highest height possible for each learner, and within each, reference the latest vertex possible from each primary.

## 6.3. Interaction with Paxos

Concurrently to the Heterogeneous Narwhal protocol, all validators run a sequence of Heterogeneous Paxos ([SWvRM20]) executions using the same learner graph. These executions determine an ever-growing sequence of anchor vertices. Note that only the vertex (and not the full text of all the transactions) must be transmitted as part of the consensus protocol.

### 6.3.1. Safety Guarantees

Heterogeneous Paxos ensures that entangled learners (Definition 10) agree on the anchor vertex sequence. Given a sequence of anchor vertices, an arbitrary (but global) function can refine the mempool DAG's partial order of vertices (and by extension, transactions), into a total order, which defines a blockchain. Because a vertex contains a CA for each element of its parentage, all of these transactions are available (for all learners).



**Figure 7.** Formation of integrity certificates (on the left) and signed quorum (on the right) in the context of one red quorum (red cross-hatch) and two blue quorums (vertical and horizontal lines): blue certificates of integrity are annotated by the number of the validator whose vertex is certified.

### 6.3.2. Liveness Guarantees

Heterogeneous Narwhal retains the guarantees of original narwhal for each learner. One of these is inclusion-fairness: if some vertex from each learner-specific height is eventually chosen as anchor, it will commit new transactions from an entire learner-specific quorum of validators. For this reason, honest validators should favor proposed anchor blocks with a higher height (for as many learners as possible) than previously chosen anchors.

If two learners (say, **red** and **blue**) are entangled, then each of their quorums intersects each other on a safe validator. As a consequence, if there is some vertex with a **red** height of  $r$  and a **blue** height of  $b$ , then any vertex with a **red** height strictly greater than  $r$  must have a **blue** height of at least  $b - 2$ . In other words, if **red** and **blue** are entangled and continue making progress, then any new **red** proposals will include the same new content (modulo a finite delay) as new **blue** proposals. This means that if a learner repeatedly commits new learner-specific height vertices as anchors, then for all entangled learners, an

entire learner-specific quorum of validators eventually commit new transactions.

When learners are not entangled, byzantine behaviour violating this guarantee is detectable, eventually allowing both to make progress:

### 6.3.3. Byzantine Behavior Evidence

If a set of Byzantine failures (unsafe validators) ensures that two learners are *not* entangled, it is possible that separate sub-DAGs will form: one in which **blue**'s height advances, but **red**'s does not, and one in which **red**'s height advances, but **blue**'s height does not. Fortunately, these DAGs include evidence of the Byzantine failures involved: multiple signed vertices from the same validators for the same sequence numbers. This can be integrated directly with Heterogeneous Paxos' *caught* function, which removes the requirement that **red** and **blue** agree. Once this has been proven for an anchor block, the chain can fork: future anchor blocks in the sequence can be decided independently. This means that honest validators can favor anchor proposals that advance **blue** height on the **blue** chain, and **red** height on the **red** chain, preserving the safety and liveness guarantees for each.

We cannot provide much in the way of guarantees for learners who are not entangled with themselves. That would be the equivalent of a base chain forking.

## 7. Conclusion

We have described a heterogeneous mempool DAG protocol that adapts Narwhal [DKSS22]. Making a clear distinction between certificates of availability and integrity, we obtain a protocol description that allows for an interleaving execution of two Narwhal instances, but shared between base chains.

Paired with Heterogeneous Paxos [SWvRM20], Heterogeneous Narwhal can facilitate cross-chain atomic transactions with well-defined guarantees, and without traditional drawbacks. We leave it to future work to explore suitable fee mechanisms for inclusion and ordering fees (besides execution fees) such that the theoretical potential of horizontal scaling of workers is matched with a suitable auction mechanism.

## References

- AGMS18. Karolos Antoniadis, Rachid Guerraoui, Dahlia Malkhi, and Dragos-Adrian Seredinschi. State Machine Replication Is More Expensive Than Consensus. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (cit. on p. 4.)

- BNN07. Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In *Automata, Languages and Programming: 34th International Colloquium, ICALP 2007, Wrocław, Poland, July 9-13, 2007. Proceedings 34*, pages 411–422. Springer, 2007. (cit. on p. 11.)
- BR16. Alexandru Baltag and Bryan Renne. Dynamic Epistemic Logic. In Edward Nouri Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2016 edition, 2016. (cit. on p. 7.)
- DKSS22. George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. (cit. on pp. 3, 4, 5, 7, 11, and 20.)
- Her18. Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 245–254, New York, NY, USA, 2018. Association for Computing Machinery. (cit. on p. 3.)
- Lam78. Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks (1976)*, 2(2):95–114, 1978. (cit. on p. 4.)
- LSSS20. Alexander Lipton, Aetienne Sardon, Fabian Schär, and Christian Schüpbach. Stablecoins, digital currency, and the future of money. *Building the New Economy*, 30, 2020. (cit. on p. 2.)
- MR98. Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Comput.*, 11(4):203–213, 1998. (cit. on pp. 3 and 5.)
- MS23. Akaki Mamageishvili and Jan Christoph Schlegel. Shared sequencing and latency competition as a noisy contest. *arXiv preprint arXiv:2310.02390*, 2023. (cit. on p. 4.)
- pr(2)3. @ObadiaAlex. You might ask – do such slot overlaps actually exist? <https://x.com/ObadiaAlex/status/1626657258200506368>, 2023. Accessed: 15th of April, 2024. (cit. on p. 2.)
- RCG23. Uma Roy, Eugene Chen, and John Guibas. Shared validity sequencing. <https://www.umbraresearch.xyz/writings/shared-validity-sequencing>, 2023. Accessed: 15th of April, 2024. (cit. on p. 3.)
- Sch90. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, dec 1990. (cit. on p. 4.)
- SGSKK22. Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22*, page 2705–2718, New York, NY, USA, 2022. Association for Computing Machinery.
- SWB+23. Isaac C. Sheff, Xinwen Wang, Kushal Babel, Haobin Ni, Robbert van Renesse, and Andrew C. Myers. Charlotte: Reformulating blockchains into a web of composable attested data structures for cross-domain applications. *ACM Trans. Comput. Syst.*, 41:2:1–2:52, 2023. (cit. on pp. 3, 6, and 11.)
- SWvRM20. Isaac C. Sheff, Xinwen Wang, Robbert van Renesse, and Andrew C. Myers. Heterogeneous Paxos: Technical report. *CoRR*, abs/2011.08253, 2020. (cit. on pp. 2, 3, 7, 9, 10, 18, and 20.)