Date of publication: January 10, 2026.

Centered Remainder Representation for Ring Buffer Correction in System-Level Code

Rudolf Stepan[1]
[1]Independent Researcher, Vienna, Austria

Corresponding author: Rudolf Stepan (e-mail: rstepan@outlook.at).
ORCID: 0009-0004-2842-2579

ABSTRACT — Ring buffers are a fundamental data structure in system-level software, widely used in audio pipelines, network stacks, device drivers, and real-time processing systems. Their state update logic is typically implemented using modulo-based correction, which constrains indices to a non-negative remainder interval. While functionally correct, this approach introduces an implicit asymmetry at the wrap boundary and separates the notion of state from its correction.

This paper presents a practical reformulation of ring buffer index correction based on centered remainder interpretation. Instead of treating wrap-around as a reset into a non-negative interval, deviations are interpreted as signed correction terms within a centered range. The proposed formulation does not alter the underlying data structure, memory layout, or semantics of the ring buffer.

It solely changes how index correction is expressed and reasoned about. Using production-style code patterns, we show that centered correction yields a more symmetric, local, and conceptually uniform update rule. The approach reduces boundary special cases and provides a clearer mental model for reasoning about state evolution in circular structures. The formulation is grounded in the general framework of centered remainder arithmetic as described in REIST Division, but is presented here as a concrete, standalone implementation pattern for system-level software.

INDEX TERMS — Circular buffers, centered remainder representation, modulo correction, system-level software, ring buffer implementation.

I. INTRODUCTION

Ring buffers, also referred to as circular buffers, are among the most widely used data structures in low-level and performance-critical software. They appear in operating systems, device drivers, interrupt handlers, audio and video processing pipelines, network stacks, and lock-free communication queues. Despite their conceptual simplicity, ring buffers frequently contain subtle correction logic that is both error-prone and difficult to reason about under non-trivial update patterns.

In most implementations, the buffer index is constrained to a non-negative interval using modulo arithmetic. This formulation enforces a wrap-around behavior that resets the index to the beginning of the buffer once the upper bound is exceeded. While this approach is well established and functionally correct, it introduces an implicit asymmetry at the wrap boundary and treats correction as an exceptional operation rather than an integral part of the state update.

This paper argues that many of these issues stem not from the data structure itself, but from how the remainder of index updates is interpreted. By adopting a centered remainder representation, the correction can be expressed symmetrically around zero, making wrap-around a signed adjustment rather than a directional reset.

This interpretation aligns naturally with how developers reason about deviations, offsets, and error terms in system code.

The contribution of this work is not a new data structure or algorithm. Instead, it provides a practical, production-oriented formulation of ring buffer correction using centered remainders. The approach is compatible with existing implementations and requires no changes to memory layout or access patterns.

II. Classical Ring Buffer Correction

A ring buffer of size N typically maintains an index variable that identifies the current position within the buffer. Updates to this index are often expressed as an increment or decrement by some offset Δ, followed by a correction step that ensures the index remains within valid bounds.

A common implementation relies on modulo arithmetic:

```
head = (head + delta) % N;
if (head < 0)
    head += N;
```

In performance-critical code, the modulo operator is often avoided, leading to manual correction logic:

```
head += delta;
while (head >= N)
    head -= N;
while (head < 0)
    head += N;
```

These patterns are ubiquitous in production systems. They are correct, well understood, and widely deployed.
However, they encode several implicit assumptions:

1. *The valid state space is the half-open interval* $[0, N)$.
2. *Correction is directional, folding all deviations back into this interval.*
3. *The wrap boundary at 0 and* N *is a privileged location in the state space.*

As a result, the index variable represents both the logical position in the buffer and the corrected remainder, conflating state and correction into a single value. While this is acceptable for simple update patterns, it complicates reasoning when offsets vary in magnitude or sign, and it introduces boundary-centric logic that must be handled explicitly.

III. Centered Remainder Interpretation

An alternative perspective is to treat index correction not as a reset into a non-negative interval, but as a signed adjustment applied to an otherwise continuous state. This can be achieved by representing the remainder in a centered interval:

$$\left(-\frac{N}{2}, \frac{N}{2}\right]$$

Within this interpretation, wrap-around is expressed as a correction of magnitude $\pm N$, rather than a transition to a fixed boundary. Deviations are treated symmetrically, and no boundary is inherently special.

This idea corresponds to the centered remainder framework formalized in REIST Division, where the remainder is explicitly modeled as a signed correction term rather than a passive residue. In the context of ring buffers, this interpretation provides a more uniform model of state evolution without altering the underlying modulo-N semantics.
Importantly, the centered representation does not change the equivalence class of the index modulo N. It merely changes how deviations from the canonical range are expressed and corrected.

IV. Ring Buffer Implementation with Centered Correction

Using the centered remainder interpretation, the ring buffer update logic can be expressed as follows:

```
head += delta;
if (head >  N / 2)
    head -= N;
if (head < -N / 2)
    head += N;
```

Alternatively, the correction can be made explicit:

```
int error = head + delta;
if (error >  N / 2)
    error -= N;
if (error < -N / 2)
    error += N;
head = error;
```

This formulation preserves all essential properties of the ring buffer:

- *The buffer size remains* N.
- *Memory access is still performed modulo* N.
- *No additional state is introduced.*
- *The data structure and semantics are unchanged.*

The only difference lies in how correction is expressed. Instead of forcing the index into a non-negative range, deviations are allowed to exist temporarily as signed values and are corrected symmetrically when they exceed half the buffer size.

V. Practical Implications

The centered correction model offers several practical advantages that are immediately visible in code, without requiring performance measurements.

First, correction becomes symmetric. Positive and negative deviations are handled uniformly, and no special treatment is required for the lower or upper boundary.

Second, the wrap boundary loses its privileged status. There is no distinguished "reset point"; correction is applied purely based on magnitude.

Third, the mental model becomes simpler. The index represents a local deviation within a bounded range, rather than a position that must always be forced into a canonical interval.

Finally, boundary-related special cases are reduced. This is particularly relevant in systems where offsets may vary dynamically or where index updates are composed across multiple stages.

These properties make the centered formulation easier to reason about, review, and maintain in system-level code.

VI. Relation to REIST Division

The formulation presented in this paper is a concrete application of the centered remainder framework described in REIST Division. While REIST Division provides a general treatment of centered remainders and their implementation-oriented interpretation, the present work focuses exclusively on a single, widely used system pattern.

This paper does not introduce new arithmetic rules or extend the formalism. Instead, it demonstrates how the REIST interpretation can be applied directly to existing production code without changing data structures, APIs, or semantics.

Ring buffers serve as a representative example of a broader class of systems that rely on modulo-based correction. Similar reasoning applies to phase accumulators, periodic schedulers, and cyclic counters, but these are beyond the scope of the present discussion.

VII. Conclusion

This paper has shown that ring buffer correction can be reformulated using a centered remainder interpretation without altering the underlying data structure or semantics. By treating wrap-around as a signed correction rather than a directional reset, the update logic becomes symmetric, local, and conceptually uniform.

The contribution is intentionally practical. No benchmarks are required to observe the benefit, as the improvement lies in reasoning clarity and structural simplicity rather than raw performance. The approach integrates naturally with existing system code and aligns with established production patterns.
Centered remainder correction provides a clearer and more robust way to express cyclic state updates in system-level software and represents a practical application of the REIST Division framework.

References

[1] D. E. Knuth, The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[2] R. P. Brent and P. Zimmermann, Modern Computer Arithmetic. Cambridge, U.K.: Cambridge University Press, 2010.

[3] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 6th ed. San Mateo, CA, USA: Morgan Kaufmann, 2019.

[4] B. Lamport, "Concurrent Reading and Writing," Communications of the ACM, vol. 20, no. 11, pp. 806–811, Nov. 1977.

[5] Stepan, R. (2025). REIST Division: An Implementation-Oriented Framing of Centered Remainder Arithmetic for Modular Addition (2.0). Zenodo. https://doi.org/10.5281/zenodo.17897540