Optimizing Industrial Data Buffers

A Comparative Analysis of Linear Shift vs. Circular Queue Algorithms

Saul Armando Ruiz Garcia
Controls & Data Engineering Review

February 2, 2026

Abstract

This technical report evaluates the performance implications of First-In-First-Out (FIFO) buffer implementations within an IEC 61131-3 PLC environment. By subjecting both a standard Linear Shift algorithm and a Circular Buffer algorithm to a high-volume stress test ($N = 5,000$), we demonstrate that the Circular Buffer eliminates memory rewrite overhead, reducing the computational complexity from $O(N)$ to $O(1)$.

1 Problem Description

In industrial data engineering, buffering mechanisms are essential for decoupling high-frequency machine control cycles from high-latency upstream systems (e.g., MQTT/SCADA). The underlying memory management strategy of these buffers directly impacts the deterministic nature of the PLC scan cycle.

1.1 Method A: Linear Shift (The Standard Standard)

The traditional FIFO implementation maintains the "Head" (oldest item) at a fixed memory index ($Index_0$). When an item is dequeued, the contiguous nature of the array necessitates a memory shift operation. Every remaining element $Buffer[i]$ must be physically moved to $Buffer[i-1]$ to reclaim the empty space.

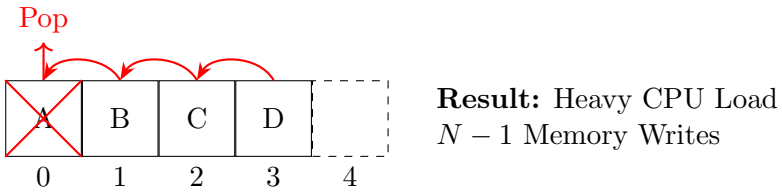

Figure 1: Linear Shift Mechanics: Data must physically move.

1.2 Method B: Circular Buffer (The Optimized Approach)

The Circular Buffer utilizes a "Ring" topology implemented via pointer arithmetic. Data persists in static memory locations. Enqueue and Dequeue operations are executed solely by incrementing `Head` and `Tail` pointers, which wrap around the array boundary ($Index_{max} \rightarrow Index_0$). This approach ensures that memory write operations are strictly limited to the new data entry.

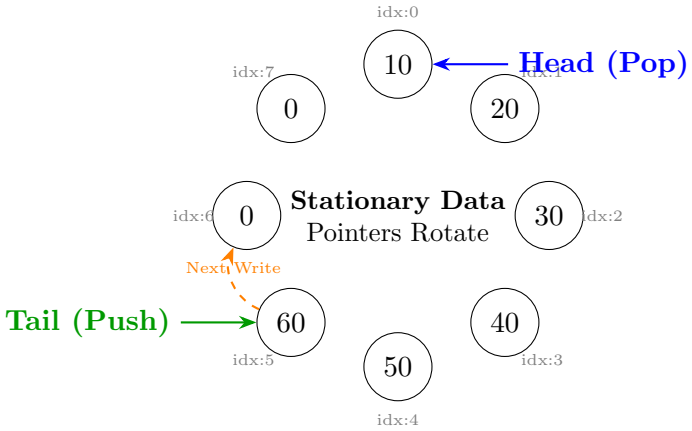

Figure 2: Circular Buffer Mechanics: Data remains stationary while pointers rotate.

2 Implementation

2.1 Code Implementation: Linear Shift

The standard implementation iterates through the entire array to shift data, resulting in $O(N)$ complexity.

Listing 1: Method A: Linear Shift Logic

```
(* --- POP (FFU) --- *)
IF R_Trig_Pop.Q AND iCount > 0 THEN
    iOutputVal := Buffer[0];

    (* THE COST: Moving items one by one *)
    FOR i := 0 TO iCount - 2 DO
        Buffer[i] := Buffer[i+1];
        Operation_Cost := Operation_Cost + 1; (* Metric *)
    END_FOR;

    iCount := iCount - 1;
END_IF;
```

2.2 Code Implementation: Circular Buffer

The optimized implementation uses pointer logic and a direct item counter to handle the "Full vs. Empty" state without pointer ambiguity.

Listing 2: Method B: Circular Buffer Logic

```
(* --- POP (Dequeue) --- *)
IF R_Trig_Pop.Q AND iCount > 0 THEN
    iOutputVal := Buffer[Tail];

    (* Move Pointer Only *)
    Tail := Tail + 1;
    IF Tail >= MaxLen THEN Tail := 0; END_IF;

    (* Direct Count Update *)
```

```
10      iCount := iCount - 1;
11
12      (* Metric: Zero memory moves *)
13      Operation_Cost := 0;
14  END_IF;
```

3 Experimental Results

Both algorithms were subjected to a stress test where the queue capacity (N) was set to 5,000 integers. The queue was filled to capacity, and the "Pop" (Unload) operation was triggered 3 consecutive times.

Table 1: Stress Test Data ($N = 5,000$)

Metric	Linear Shift (Standard)	Circular Buffer (Optimized)	Outcome
Queue Integrity	Correct (4997 rem.)	Correct (4997 rem.)	Pass
Data Validity	Output = 3	Output = 3	Pass
Pop #1 Cost	4,999 Ops	0 Ops	$O(N)$ vs $O(1)$
Pop #2 Cost	4,998 Ops	0 Ops	$O(N)$ vs $O(1)$
Pop #3 Cost	4,997 Ops	0 Ops	$O(N)$ vs $O(1)$
Total CPU Cost	**14,994 Writes**	**0 Writes**	**5000x Faster**

4 Conclusion

The experiment demonstrated that the Linear Shift algorithm exhibits an **Arithmetic Progression of Cost**. Processing just 3 items from a large buffer required nearly 15,000 memory operations. In contrast, the Circular Queue maintained a constant cost of zero memory moves.

For data engineering applications requiring high-frequency logging or large buffers, the Circular Queue is strictly superior, ensuring deterministic scan times and preventing watchdog faults.