GOPLC: A Universal Gateway for Data Center Infrastructure Management

Author: James Belcher **Date:** January 2026 **Version:** 1.1

Table of Contents

Executive Summary

Data centers run dozens of subsystems from different vendors, speaking different protocols, managed by separate teams. A single data hall may contain Liebert CRAC units on Modbus RTU, Schneider PDUs on SNMP, APC UPS systems on Modbus TCP, Honeywell BMS controllers on BACnet MSTP, SEL protective relays on serial, and Rockwell PLCs on EtherNet/IP. Integrating these into a coherent monitoring fabric typically requires multiple middleware products, custom scripts, and significant ongoing engineering.

GOPLC is an IEC 61131-3 PLC runtime written in Go that functions as a universal protocol gateway with programmable logic. It speaks 12 industrial protocols natively, normalizes raw device data into engineering units through Structured Text programs, and exposes unified data northbound via OPC UA, MQTT, and DNP3. Its boss-minion clustering architecture maps naturally to modular data center pods, and its store-and-forward capability ensures no telemetry loss during network disruptions.

This paper describes the architecture, protocol capabilities, redundancy mechanisms, and deployment model for GOPLC in data center BMS and EPMS applications.

Industry Context

The global data center market reached approximately $430 billion in 2026, projected to exceed $1.1 trillion by 2035. Nearly 100 GW of new capacity is expected between 2026 and 2030, driven by AI workloads demanding higher power density and more sophisticated cooling. This growth creates four challenges relevant to infrastructure management:

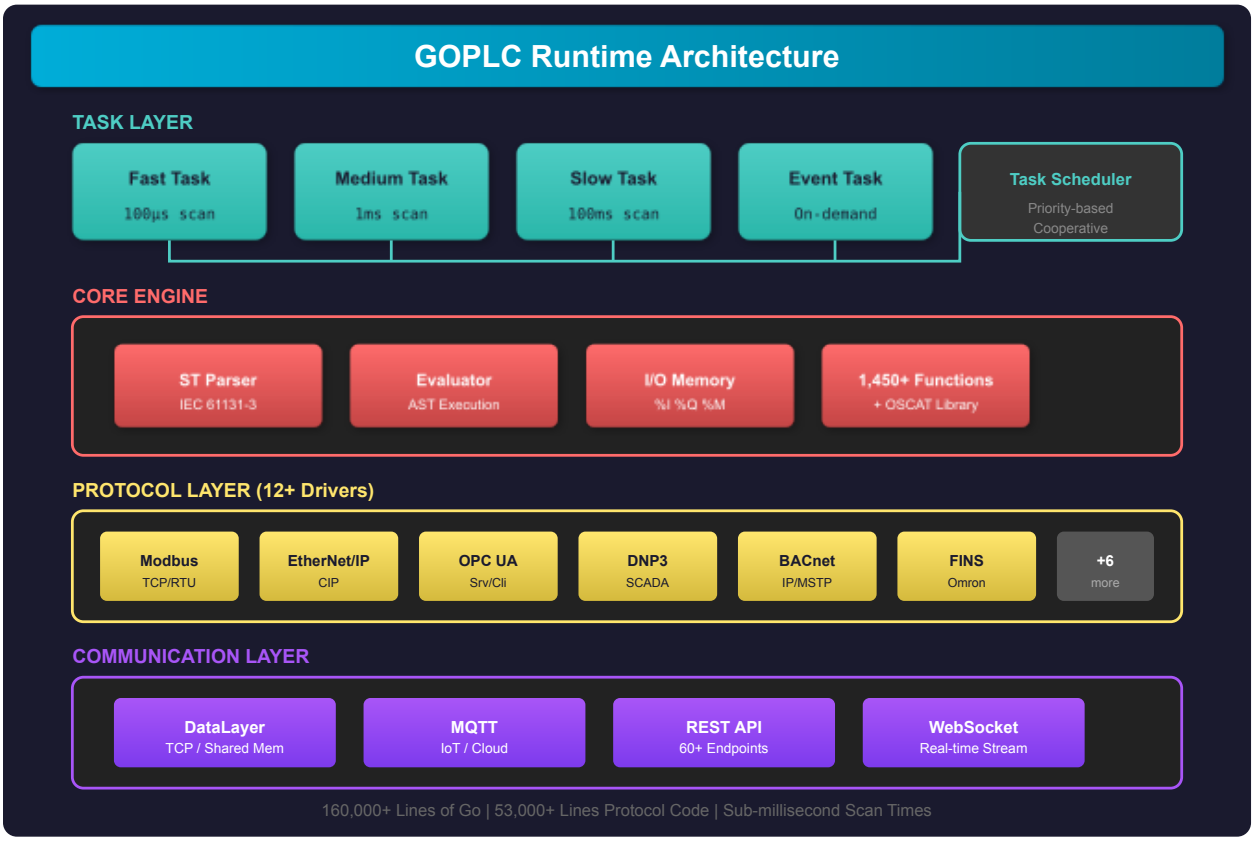

Figure 1: GOPLC Architecture Overview

- **Protocol fragmentation.** Cooling speaks Modbus or BACnet. Power distribution uses DNP3, SNMP, or proprietary protocols. Building automation runs BACnet MSTP over RS-485. No single vendor covers the full stack.
- **Commissioning cost.** Mapping registers, scaling raw values, configuring alarm thresholds, and wiring up SCADA points for a new pod repeats for every pod and every device type.
- **Reliability requirements.** A CRAC failure in a high-density row can cause thermal events within minutes. EPMS data loss during utility-to-generator transfers affects billing accuracy and compliance.
- **Multi-site operations.** Operators need a consistent data model across sites, even when each site uses different vendors and protocols.

Architecture

GOPLC deploys in a three-tier hierarchy mirroring physical data center topology: edge modules, site supervisors, and corporate dashboards.

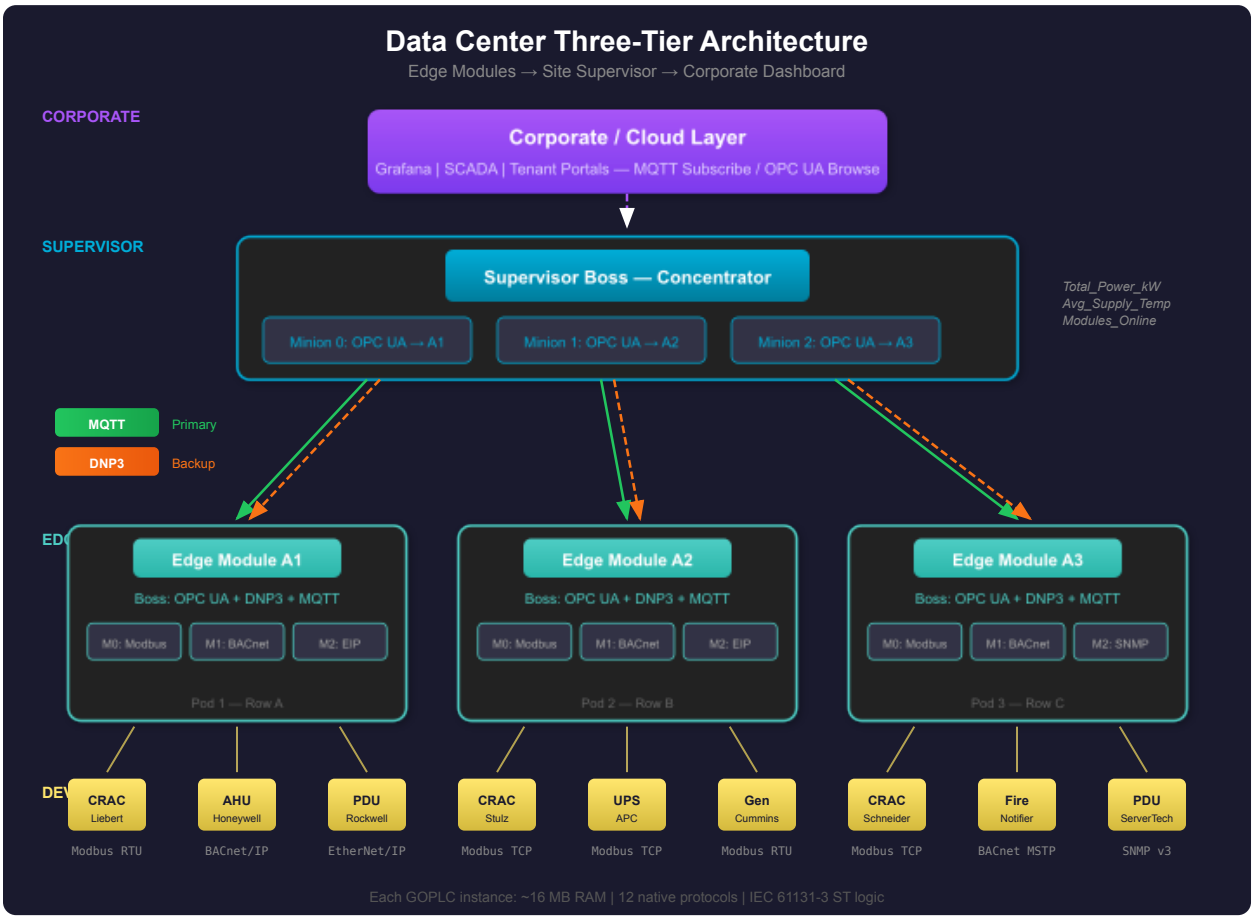

Figure 2: Data Center Three-Tier Architecture

Edge Module Layer

Each pod runs a GOPLC cluster with one boss and 2-4 minions. Each minion handles a single protocol and device type — Minion 0 polls CRAC units via Modbus TCP/RTU, Minion 1 reads AHU data via BACnet/IP COV subscriptions, Minion 2 communicates with power meters via EtherNet/IP. The boss aggregates data through the DataLayer (shared memory within the cluster), then exposes a unified OPC UA namespace and publishes to MQTT. It also runs a DNP3 outstation that automatically buffers events when the supervisor is unreachable.

From the actual edge configuration (`configs/datacenter-edge-cluster.yaml`):

```
PROGRAM EdgeBossAggregator
  VAR_GLOBAL
    CRAC_Supply_Temp : REAL := 0.0;
    CRAC_Return_Temp : REAL := 0.0;
    PDU_Power_kW     : REAL := 0.0;
    Module_Healthy   : BOOL := TRUE;
    Devices_Online   : INT  := 0;
  END_VAR

  Devices_Online := 0;
  IF CRAC_Supply_Temp <> 0.0 THEN Devices_Online := Devices_Online + 1; END_IF;
  IF AHU_Supply_Temp <> 0.0 THEN Devices_Online := Devices_Online + 1; END_IF;
  IF PDU_Voltage_A <> 0.0 THEN Devices_Online := Devices_Online + 1; END_IF;
  Module_Healthy := Devices_Online >= 3;

  IF MQTT_IS_CONNECTED('mqtt_north') THEN
    MQTT_PUBLISH('mqtt_north', 'dc/module-a1/crac/supply_temp', CRAC_Supply_Temp);
    MQTT_PUBLISH('mqtt_north', 'dc/module-a1/pdu/power_kw', PDU_Power_kW);
    MQTT_PUBLISH('mqtt_north', 'dc/module-a1/health', Module_Healthy);
  END_IF;
  (* Backup: DNP3 outstation auto-buffers when supervisor disconnects *)
END_PROGRAM
```

Site Supervisor Layer

The supervisor runs its own GOPLC cluster. Minions act as OPC UA clients to each edge module; the boss provides rollup calculations and a flat address namespace (`A1_CRAC_Supply_Temp`, `A2_PDU_Power_kW`, etc.). Flat naming is deliberate — deep OPC UA hierarchies add browsing overhead and complicate client configuration.

Dual-path ingestion: MQTT for sub-second primary data, with staleness detection (5 seconds without update = offline). When MQTT goes stale, the DNP3 master polls the edge outstation for buffered events with original timestamps intact.

Corporate Layer

Standard tools (Grafana, SCADA, custom dashboards) subscribe to MQTT topics or browse OPC UA namespaces. GOPLC's role ends at the supervisor — it provides clean, normalized, aggregated data that any downstream system can consume. MQTT topic-based access enables colocation

tenants to subscribe to their rack-specific data (`dc/tenantA/rack42/power`) without touching the control network.

Protocol Coverage

GOPLC includes 53,000+ lines of protocol implementation across 12 protocols. Each is a native Go implementation — no external drivers or modules required.

Protocol	Lines	Data Center Subsystem	Key Capability
Modbus TCP/RTU	7,241	CRAC, PDU, UPS, VFD	Connection pooling, heartbeat, RTU serial (RS-485), gateway mode. 89,769 req/sec benchmarked
BACnet/IP & MSTP	7,883	HVAC, fire, access, lighting	COV subscriptions, priority arrays (16 levels), ReadPropertyMultiple, RS-485 MSTP
DNP3	13,354 + 1,862 S&F	EPMS, UPS, breakers	Class 1/2/3 event buffering, unsolicited responses, SBO, store-and-forward (SQLite + AES-256-GCM)
OPC UA	4,496	Northbound SCADA/MES	Server auto-exposes VAR_GLOBAL, client with subscriptions, flat namespace
MQTT	integrated	Pub/sub, tenant access	QoS 0/1/2, TLS, retained messages, wildcard subscriptions
SEL	4,208	Protective relays, EPMS	ASCII metering, Fast Message (<10ms), Fast Operate, Mirrored Bits
SNMP v1/v2c/v3	3,597	Network gear, smart PDUs	GET/SET/WALK/GETBULK, trap receiver, SNMPv3 auth+encryption
EtherNet/IP	5,388	Rockwell PLCs, power dist.	CIP messaging, I/O connections
S7comm	2,441	Siemens controllers	DB read/write, block operations
PROFINET	1,997	Siemens real-time I/O	Cyclic data exchange
FINS	3,565	Omron controllers	Memory area read/write, TCP/UDP
DF1	1,417	Legacy Allen-Bradley	SLC 500/MicroLogix serial

Serial RTU support deserves emphasis: many data center devices still sit on RS-485 daisy chains. GOPLC handles Modbus RTU (CRC-16, configurable inter-frame timing, RTS half-duplex) and BACnet MSTP natively, bridging serial legacy into TCP/IP infrastructure.

Redundancy and High Availability

Redundancy in data center monitoring is not optional. GOPLC provides three complementary mechanisms, all implementable in Structured Text without runtime modifications.

Dual-Path Communication

Every edge module exposes data through MQTT (fast, sub-second) and DNP3 outstation (store-and-forward, event-buffered). When the MQTT path fails, the DNP3 outstation continues buffering events with timestamps in its SQLite-backed queue. On reconnection, the supervisor's DNP3 master retrieves the full event history in order — zero data loss, timestamp integrity preserved, no single point of failure.

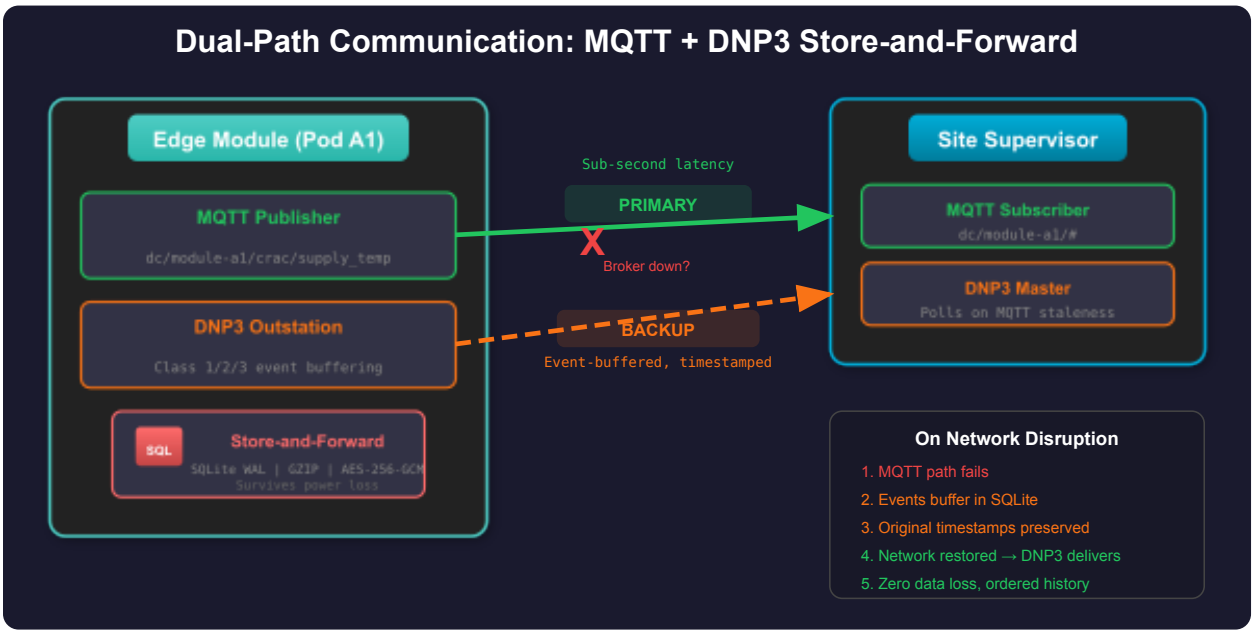

Figure 3: Dual-Path Communication

Redundancy Test Implementations

Three redundancy approaches have been implemented and tested as full 3-cluster scenarios, each with a primary server, supervisor, and backup server monitoring 5 data center subsystems (CRAC, UPS, PDU, Generator, Fire Suppression).

Test 1: API-Driven OPC UA Reconfiguration — Single OPC UA client connects to primary. On heartbeat staleness (5 consecutive stale reads), the supervisor disconnects, deletes the client, creates a new one pointed at the backup, and reconnects. Clean but incurs reconnection latency.

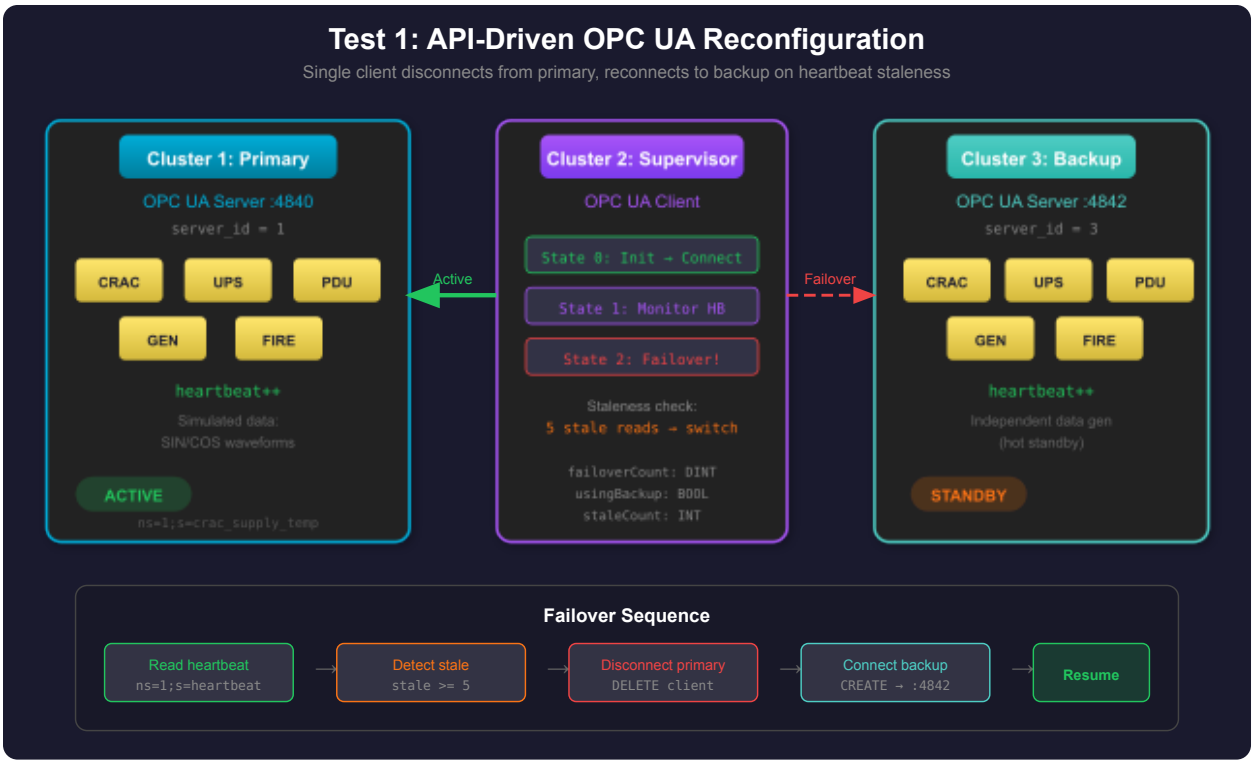

Figure 4: Test 1: API-Driven Reconfiguration

Test 2: Dual-Client Approach — Two OPC UA clients are always connected simultaneously (one to primary, one to backup). Both heartbeats are monitored continuously. Failover is a boolean switch (`useBackup := TRUE`) — zero reconnection delay. Automatic failback when primary recovers.

Test 3: DataLayer Backbone — Primary and backup nodes publish `DL_*` prefixed variables to a TCP DataLayer. The supervisor reads via `DL_GET()` and monitors health via `DL_LATENCY_US()`. Failover triggers an `HTTP_POST` to reconfigure the DataLayer address. Sub-microsecond latency on shared memory, 100-500 us on TCP LAN.

OPC UA Client-Side Failover (Test 1 Detail)

For the supervisor-to-edge path, heartbeat-based failover in ST:

```
PROGRAM Supervisor
  VAR
    state : INT := 0;
    primaryEndpoint : STRING := 'opc.tcp://10.0.0.101:4840';
    backupEndpoint  : STRING := 'opc.tcp://10.0.0.102:4840';
    staleCount      : INT  := 0;
    STALE_THRESHOLD : INT  := 5;
  END_VAR

  CASE state OF
  0:  OPCUA_CLIENT_CREATE('link', primaryEndpoint);
```

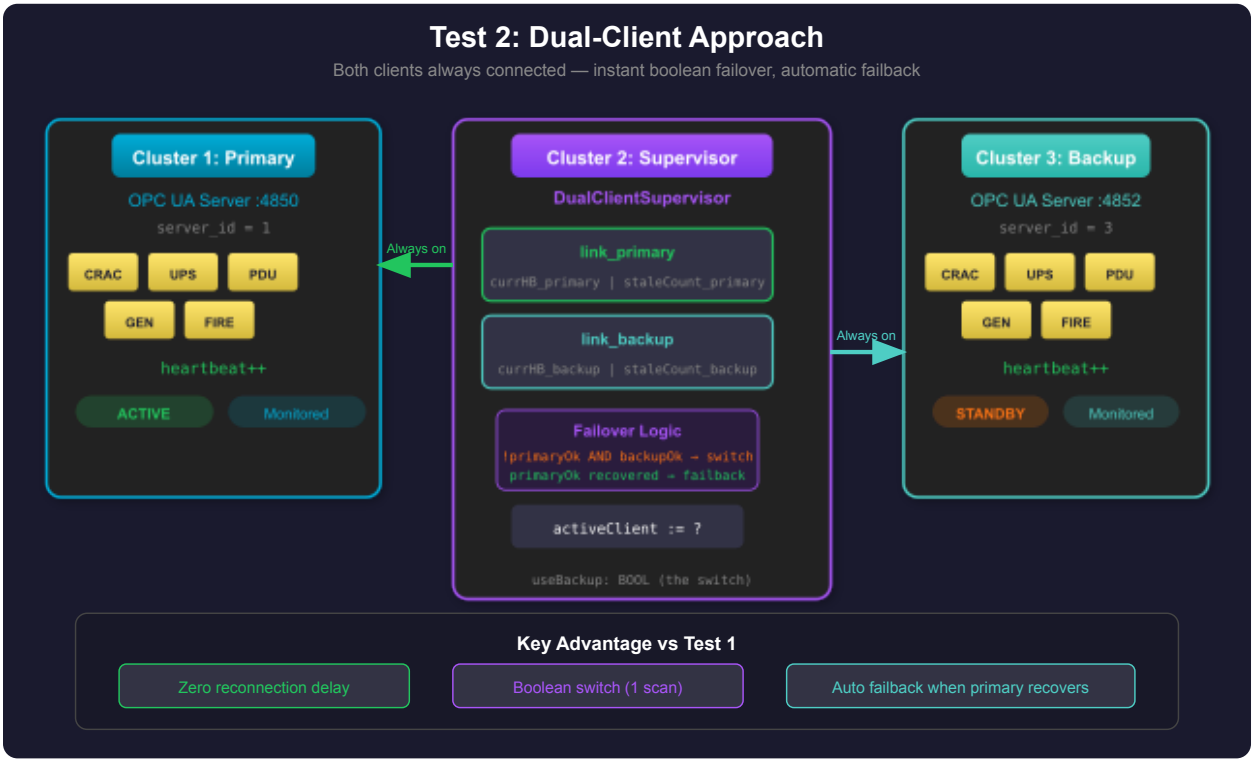

Figure 5: Test 2: Dual-Client

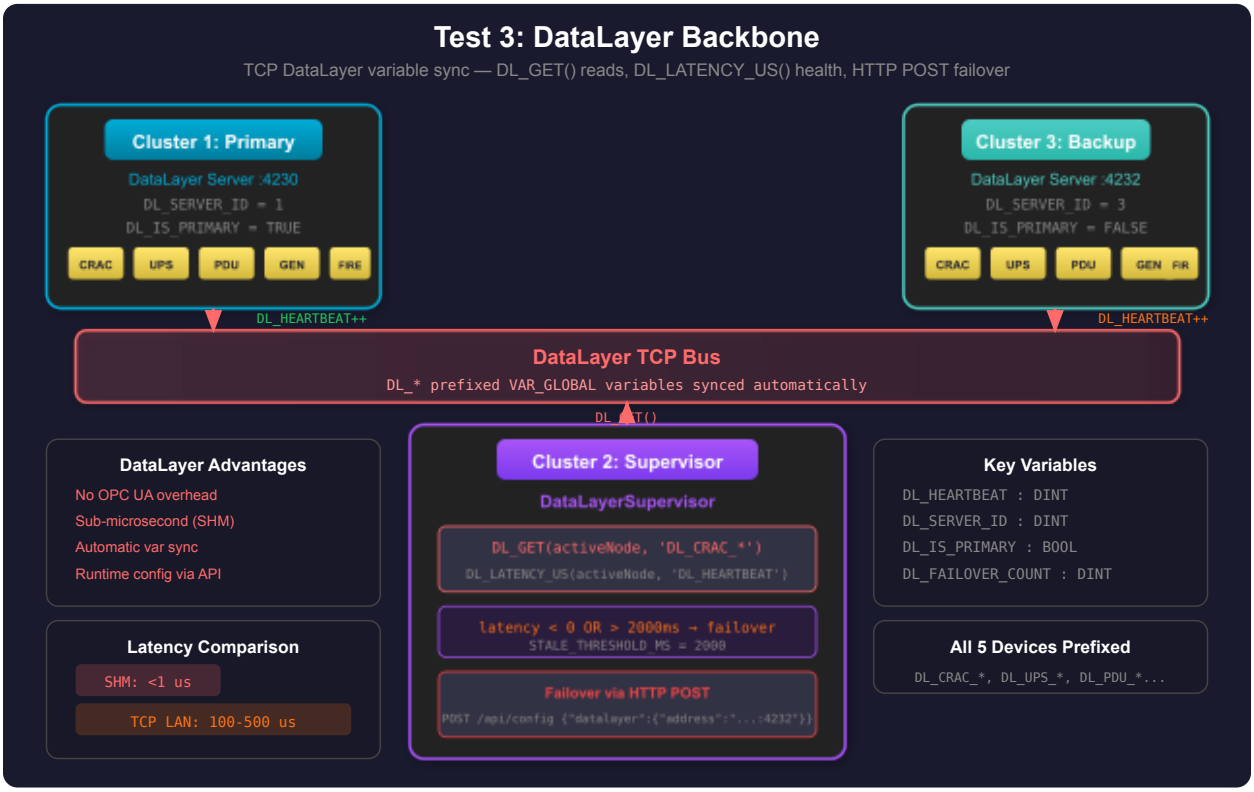

Figure 6: Test 3: DataLayer Backbone

```
      OPCUA_CLIENT_CONNECT('link');
      state := 1;

  1:  currentHeartbeat := OPCUA_CLIENT_READ_INT('link', 'ns=1;s=heartbeat');
      IF currentHeartbeat = lastHeartbeat THEN
        staleCount := staleCount + 1;
      ELSE
        staleCount := 0;
      END_IF;
      lastHeartbeat := currentHeartbeat;
      IF staleCount >= STALE_THRESHOLD THEN state := 2; END_IF;

  2:  OPCUA_CLIENT_DISCONNECT('link');
      OPCUA_CLIENT_DELETE('link');
      OPCUA_CLIENT_CREATE('link', backupEndpoint);
      OPCUA_CLIENT_CONNECT('link');
      failoverCount := failoverCount + 1;
      staleCount := 0;
      state := 1;
  END_CASE;
END_PROGRAM
```

Detection time: 5 scan cycles (500ms at 100ms scan). Switchover: one scan cycle. `failoverCount` and `usingBackup` are exposed as OPC UA nodes for operator visibility.

DataLayer Heartbeat Monitoring

For multi-cluster deployments, each node publishes a heartbeat variable. Remote nodes appear automatically with a `REMOTE_` prefix via the DataLayer. If `REMOTE_PLC1_DL_HEARTBEAT` goes stale for 10 consecutive reads, the local node activates backup logic. DataLayer latency: <1 us (shared memory, same machine) or 100-500 us (TCP, cross-machine).

Latency Distribution (2ms scan, DataLayer TCP)

Direction	Avg	P50	P95	P99	Max
Client → Server	1.04ms	1.26ms	2.82ms	3.08ms	3.30ms
Server → Client	1.08ms	1.29ms	2.76ms	3.14ms	5.08ms

P50/P95/P99 = 50th/95th/99th percentile latency | All values sub-5ms

Figure 7: Latency Distribution

Resilience Functions

Beyond protocol-level redundancy, 39 built-in resilience functions provide circuit breaker, rate limiter, retry with backoff, cache (last-known-good values), bulkhead isolation, and throttle/debounce

patterns — all callable from ST.

Current Limitations

- **No automatic primary election.** No built-in Raft/Paxos. Primary/backup roles are configured statically; failover decisions live in ST logic or external orchestrators.
- **No shared state transfer.** Backup starts with its own state. The DataLayer shares variables, but task-internal state (FB instances, locals) does not transfer.
- **No quorum detection.** Split-brain resolution depends on application-level logic.

These are deliberate trade-offs. Consensus algorithms add complexity and non-deterministic failure modes that may not be appropriate for a PLC runtime where deterministic behavior matters more than distributed coordination.

Real-Time Performance

GOPLC provides deterministic scan timing through OS-level mechanisms: `mlockall()` to prevent page faults, CPU affinity to isolate from the OS scheduler, `SCHED_FIFO` for real-time priority on PREEMPT_RT kernels, and GC tuning (GOGC=500) to reduce garbage collection frequency.

Mode	Scan Time	Jitter	Notes
Busy-loop + SCHED_FIFO (PREEMPT_RT)	100 us	<10 us	Baremetal, dedicated core
Busy-loop (container)	100 us	<100 us	Docker with CPU pinning
Ticker (standard)	1-100 ms	<500 us	Typical for data center monitoring

Each task has an independent watchdog. If a scan exceeds the watchdog threshold, the runtime logs the overrun, sets the task's fault flag, and optionally halts (configurable per task). Watchdog statistics are exposed via the REST API.

Benchmarks

All measurements on Intel i9-13900KS, 62 GB RAM, January 2026:

Metric	Result
Scan execution (5000 Modbus servers)	20-50 us avg
Modbus throughput	89,769 req/sec
DataLayer latency (TCP LAN)	<1 ms P50, <3 ms P99
DataLayer latency (shared memory)	<1 us
24-hour soak test	0 missed scans, 0 memory leaks

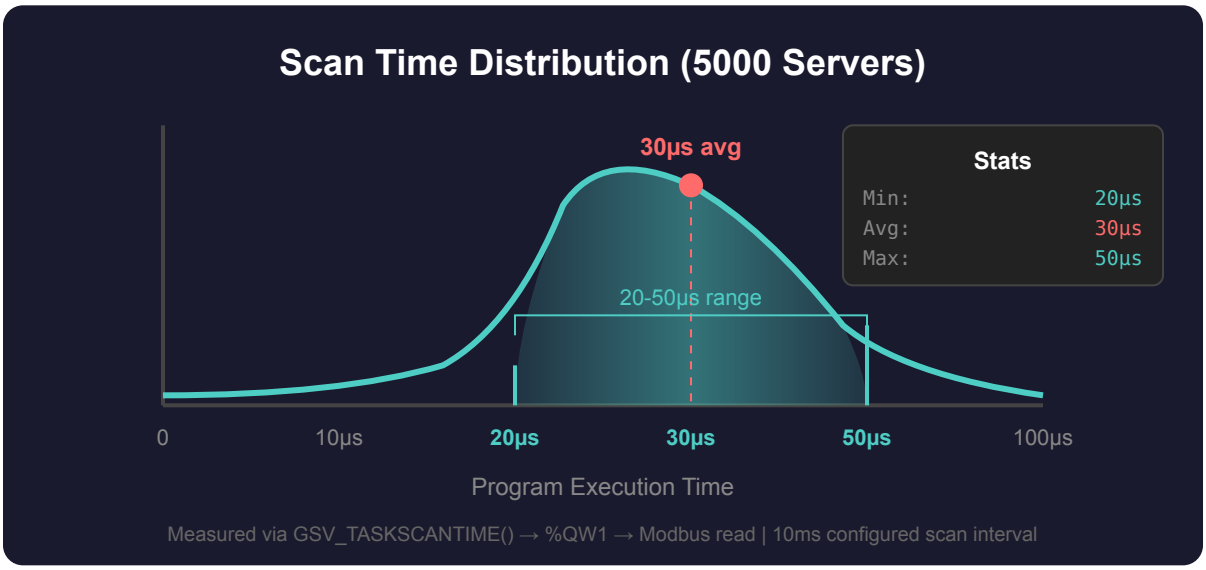

Figure 8: Scan Time Distribution

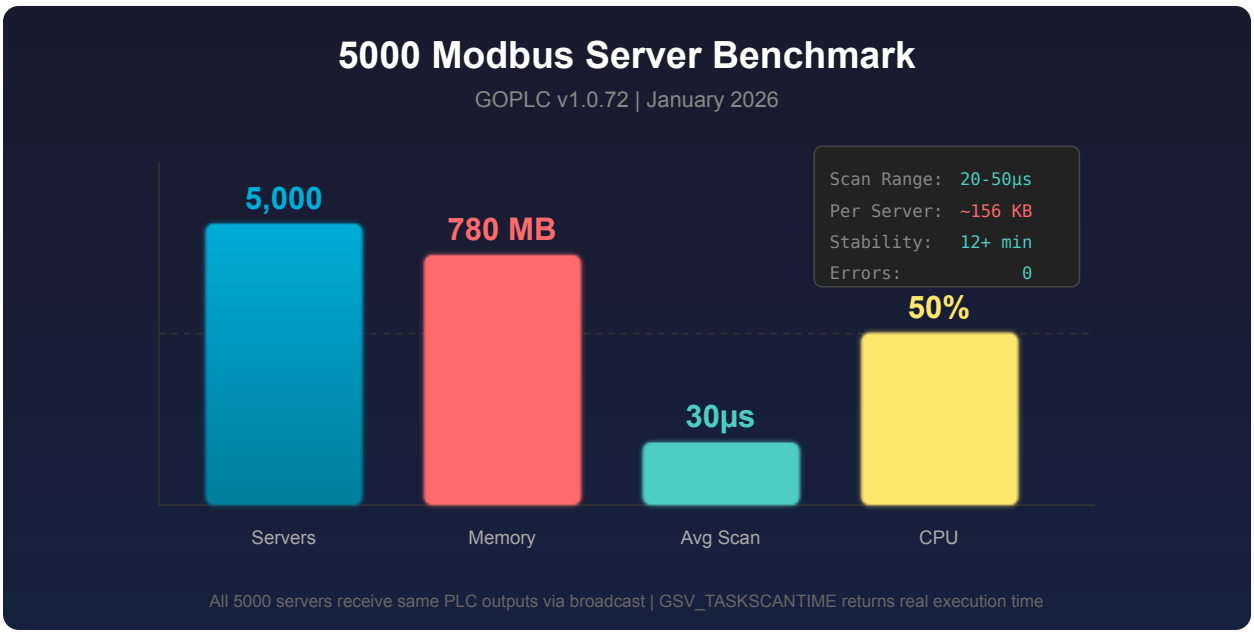

Figure 9: 5000 Server Benchmark

AI-Assisted Commissioning

GOPLC includes a Claude-powered AI assistant in the Web IDE. The assistant's system prompt contains the complete function registry (1,450+ functions with parameter signatures), OSCAT library reference (557 functions/FBs), and protocol-specific knowledge.

Workflow: Operator provides a device manual (Modbus register map, BACnet object list). The AI generates a complete ST program with correct register addresses, scaling factors, and engineering unit conversions. The operator uploads via Web IDE and deploys with hot reload — no restart required.

Example prompt: *"Generate a program to read a Liebert DS CRAC via Modbus TCP at 10.0.0.50, registers 100-103 for supply temp, return temp, humidity, fan speed, coils 0-1 for compressor and alarm. Temps are raw x10."*

The AI generates deployable code with correct `MB_CLIENT_CREATE`, `MB_CLIENT_READ_INT`, and scaling logic because it has exact function signatures in context. This eliminates the manual register-mapping work that dominates commissioning time. For closed-loop control, the AI generates OSCAT `CTRL_PID` code with exact parameter names (`ACT`, `SETO`, `KP`, `TN`, `TV`) parsed from the library source — getting these wrong causes silent failures.

Fleet orchestration: At scale, the AI assists with fleet-wide configuration: generating programs for each device type, validating consistency across edge modules, and producing supervisor rollup logic from edge variable lists.

Deployment

Bosch Rexroth ctrlX CORE X3

The primary deployment target for data center edge modules is the Bosch Rexroth ctrlX CORE X3 industrial controller. GOPLC packages as a Snapcraft snap containing the runtime, all 12 protocol drivers, and the Web IDE. On ctrlX, GOPLC communicates with the ctrlX Data Layer via shared memory for zero-copy data exchange with other automation apps running on the same controller.

The ctrlX platform provides what a DIY gateway cannot:

- **Hardware watchdog** — automatic restart on hang, critical for unattended edge modules
- **Dual Gigabit Ethernet** — separate OT and IT network segments without external switches
- **DIN-rail mounting** — standard industrial form factor for electrical panel installation
- **Vendor support and certification** — Bosch Rexroth global service network, CE/UL markings
- **App ecosystem** — ctrlX Data Layer exposes GOPLC variables to any other ctrlX app (Node-RED, OPC UA aggregation, custom analytics) without integration code

This transforms the edge module from a software project into a certified industrial appliance. Each pod gets a ctrlX CORE X3 running the GOPLC snap, with minions polling local devices and the boss publishing northbound via OPC UA, MQTT, and DNP3.

Boss-Minion Clustering

A single GOPLC process spawns isolated PLC instances as goroutines. Each minion gets its own `PLCContext` with independent protocol registries, a Unix socket API, and its own task scheduler. The boss proxies all minion APIs and health status.

Configuration	Minions	RAM	Per Minion
Idle (scheduler only)	1,000	3.4 GB	3.4 MB
Idle	10,000	37.4 GB	3.9 MB
Running ST programs	1,000	15.8 GB	15.8 MB

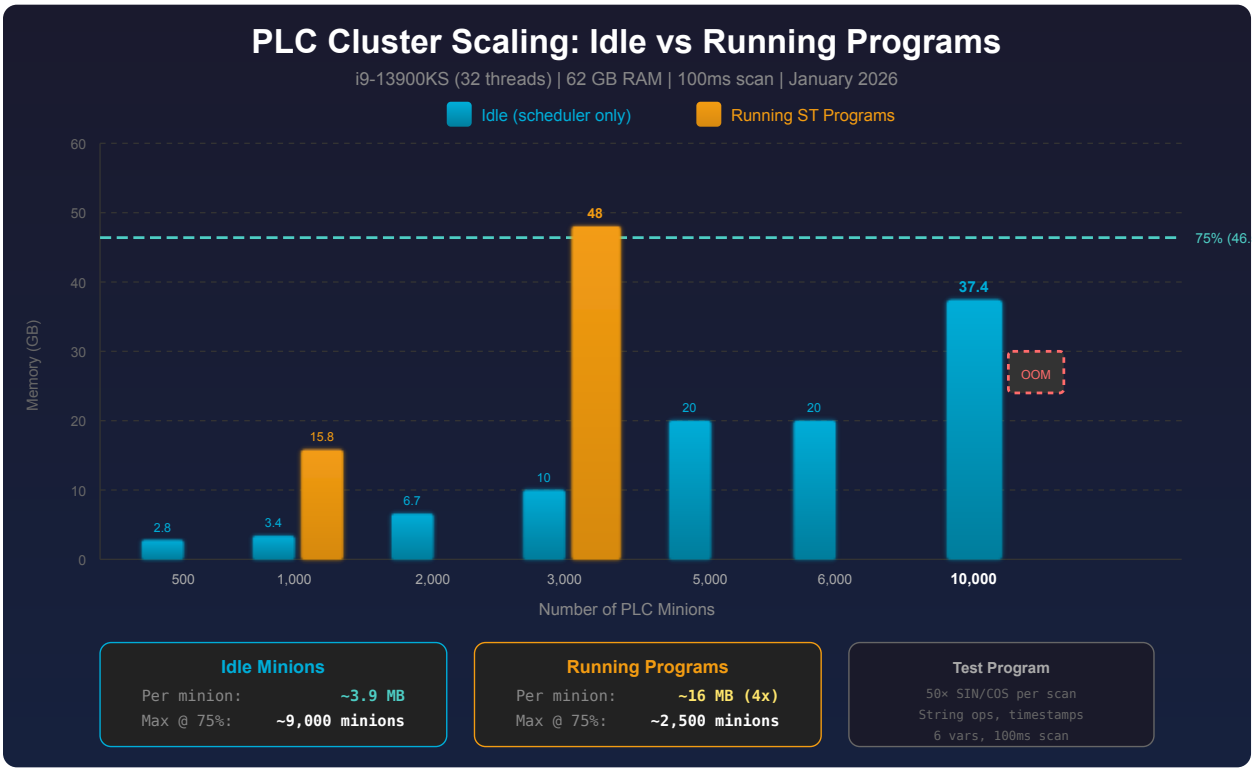

Figure 10: Cluster Benchmark

A typical edge module needs 3-5 minions — well within the capacity of the ctrlX CORE X3. For the supervisor tier, a rack-mount server or VM handles dozens of edge module connections.

Alternative Deployment

GOPLC also runs as a Docker container (Alpine-based, ~20 MB image, Kubernetes health probes), directly on any Linux host including Raspberry Pi 5, or on cloud VMs for the corporate aggregation layer.

Scale Estimates

Instance Counts at Hyperscale

The gateway hierarchy scales to facilities of any size. Using industry-standard equipment density (4–6 CRACs/MW, 40–50 rack PDUs/MW, 2–3 UPS modules/MW):

Facility Size	Device Gateways (practical)	Aggregation Tier	Total Instances
50 MW	~737	13	**~750**
100 MW	~1,385	15	**~1,400**
500 MW	~6,583	17	**~6,600**
1 GW	~12,978	22	**~13,000**

Practical counts use grouping for high-density, low-complexity device types (rack PDUs, environmental sensors): one gateway instance polling 10–20 devices of the same type via SNMP, rather than strict 1:1 mapping.

Hardware vs. Virtualized Deployment Cost

The gateway tier can be deployed on dedicated edge SBCs (ctrlX CORE) or as containerized instances on existing server infrastructure reaching field devices via routed VLANs on the datacenter's existing Cisco distribution switch.

Facility	Edge SBC Model (infra + SW)	Virtualized Model (infra + SW)	Savings
50 MW	$1.01M–$1.43M	$482K–$520K	52–66%
100 MW	$1.89M–$2.66M	$888K–$948K	54–64%
500 MW	$8.91M–$12.54M	$4.17M–$4.44M	50–65%
1 GW	$17.55M–$24.70M	$8.19M–$8.68M	51–53%

ctrlX CORE SBC at $600–700/unit. GOPLC standalone license $400/instance, cluster license $600/instance. Virtual infra = commodity servers + VLAN config. Savings driven by infrastructure elimination; cluster licensing is $200/instance more expensive than standalone.

For a detailed treatment of both architectures see: - *GOPLC DC Simulation: Hardware Gateway Architecture* — edge SBC deployment, tiered hierarchy, standalone vs. cluster configuration - *GOPLC DC Simulation: Virtualized Gateway Architecture* — server VM deployment, Cisco SVI routing, migration path from hardware

---

Comparison with Existing Solutions

Capability	GOPLC	Ignition	Niagara	Custom Scripts
Native protocols	12	10+ (modules)	8+ (drivers)	Per-script
IEC 61131-3 logic	ST, 1,450+ functions	SFC only	Limited	None

Capability	GOPLC	Ignition	Niagara	Custom Scripts
Edge deployment	Pi 5, any Linux, ctrlX	Java/x86	JVM	Varies
Cluster mode	Built-in (10,000+ minions)	Redundancy module ($)	N+1 Supervisor	Manual
Store-and-forward	Built-in (SQLite + AES)	S&F module	Journal	Must build
AI commissioning	Built-in (Claude)	None	None	None
Memory per instance	16 MB running	~512 MB+	~256 MB+	Varies
Licensing	Proprietary	Per-tag	Per-instance	Open source

GOPLC's differentiator is the combination of lightweight deployment (16 MB vs. hundreds of MB for JVM-based solutions), native protocol implementations (no external drivers), and programmable logic on the same runtime as the gateway. This eliminates the middleware layer between edge devices and the monitoring system.

Conclusion

GOPLC addresses data center infrastructure management by collapsing protocol translation, data normalization, programmable logic, and redundancy into a single lightweight runtime. The hierarchical architecture maps directly to physical topology. Dual-path communication (MQTT primary, DNP3 store-and-forward backup) provides zero-data-loss operation without redundancy hardware. AI-assisted commissioning reduces the manual effort of integrating new device types. And Snapcraft packaging for platforms like ctrlX CORE X3 provides a path from software project to certified industrial appliance.

The system is operational with tested configurations for CRAC cooling, PDU power monitoring, AHU air handling, and multi-module aggregation. Cluster mode is benchmarked to 10,000 instances, and 24-hour soak tests demonstrate zero missed scans and zero memory leaks.

For data center operators navigating the 2026 capacity expansion, GOPLC provides a standards-based (IEC 61131-3), protocol-native, and operationally simple path to unified facility monitoring.

Contact: James Belcher – jbelcher@jmbtechnical.com | LinkedIn

---

GOPLC: 160,000+ lines of Go. 12 industrial protocols. 1,450+ ST functions. Built for real-world automation.