# SQL Database Monitoring Tools: Selection and Workflow Design

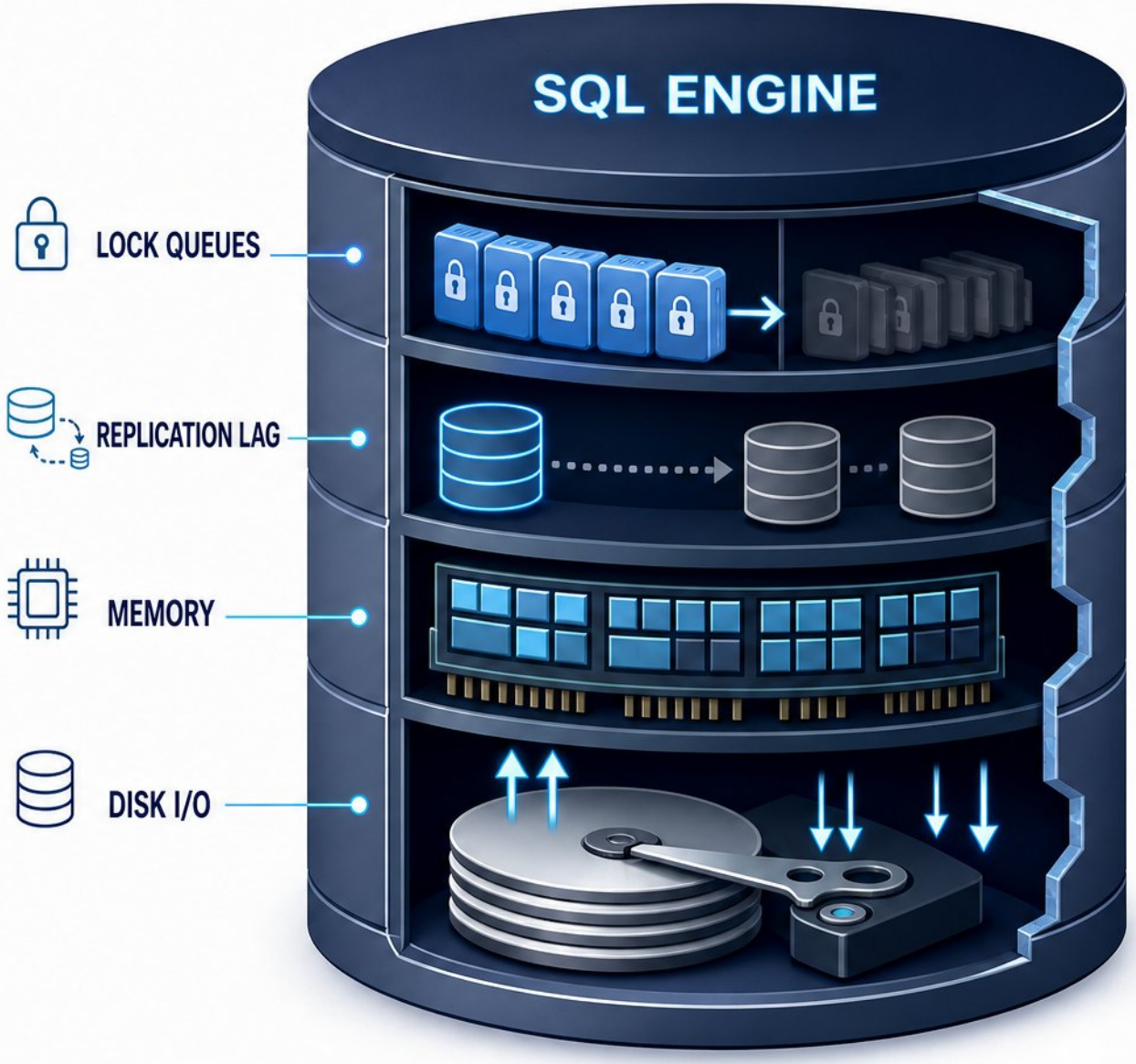
- Define scope: selecting tooling and designing monitoring workflows
- Core skills for DBAs, platform engineers, and data engineers
- We cover: vocabulary, signals, tool landscape, and selection framework
- Also workflow design, cost control, and future observability trends

# Core Vocabulary: Monitoring, Observability, and Telemetry

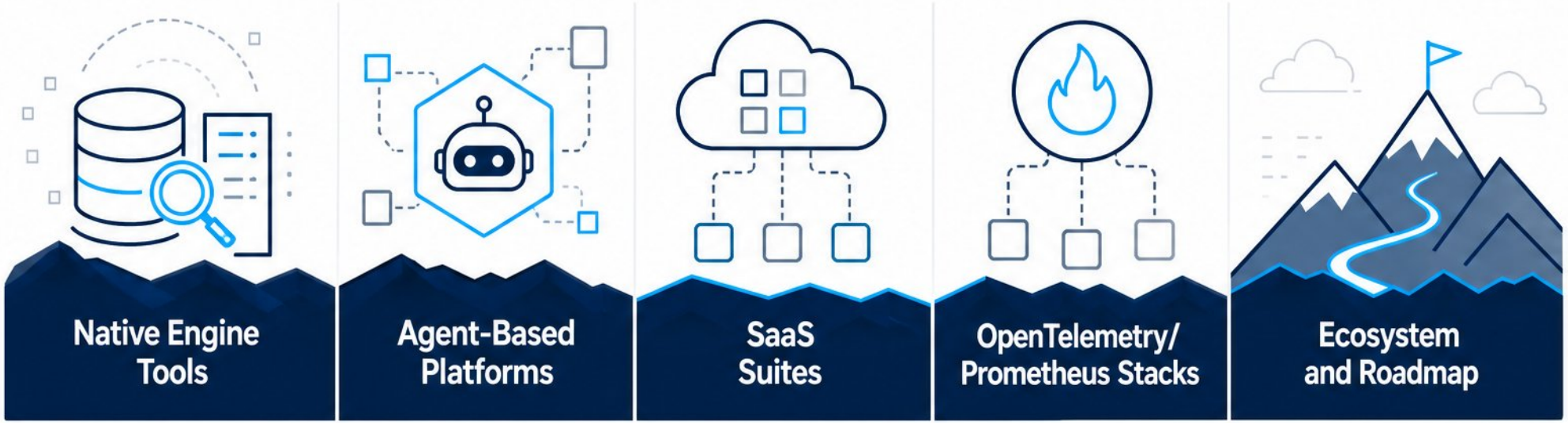
- **Monitoring:** threshold-based alerts for known failure modes
- **Observability:** ad-hoc querying to investigate novel failures
- **Telemetry pillars:** metrics, logs, and traces
- SLOs, SLIs, and error budgets drive reliability discussions
- Monitoring detects; observability explains root cause

# What SQL Databases Require Monitoring

- Track query latency, throughput, locks, disk I/O, memory, connections, replication lag, and index health
- Map signals to user impact: timeouts, split-brain replication, and data corruption
- Engine-specific views: pg_stat_*, SQL Server Query Store, MySQL Performance Schema
- Managed SQL limits: RDS, Aurora, Cloud SQL, Azure SQL expose telemetry via cloud APIs



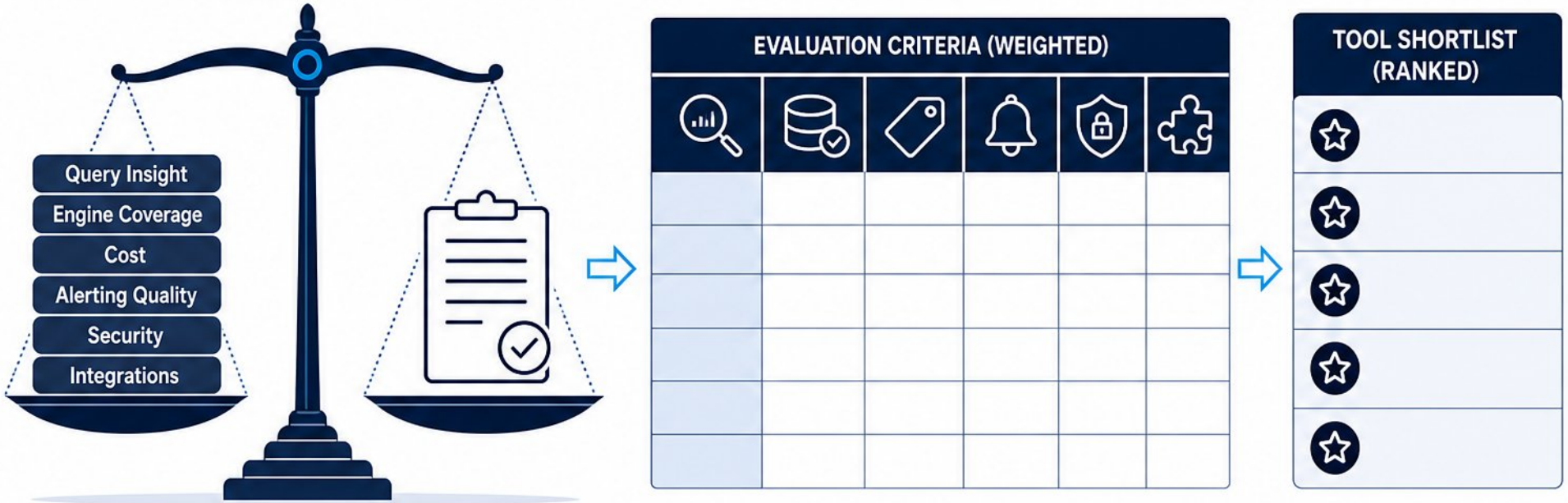
# Monitoring Tool Categories and the 2026 Landscape



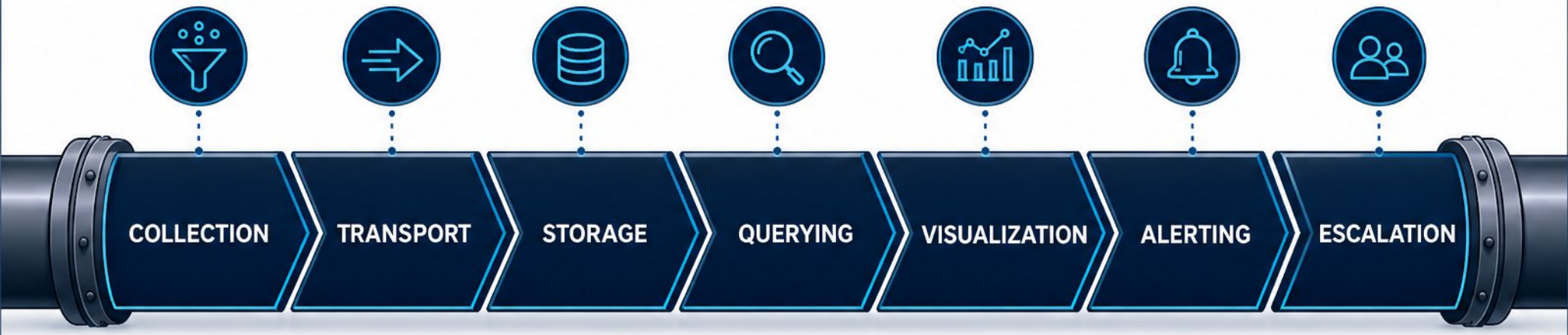

- Native engine tools and agent-based platforms offer granular visibility
- SaaS suites and OpenTelemetry/Prometheus stacks trade depth for breadth
- Agent overhead, cloud support, and cost shape tool selection
- Leaders: Datadog, Dynatrace, Grafana Cloud, SolarWinds, Redgate Monitor
- Open-source depth: Percona PMM, pganalyze; plus native views like pg_stat_*

# Evaluation Criteria and a Repeatable Selection Framework

- Weight criteria: query insight, engine coverage, cost, alerting quality, security, integrations
- Check baseline needs of mixed SQL estates: PostgreSQL, MySQL, SQL Server, managed services
- Score tools objectively; avoid bias toward brand or familiarity



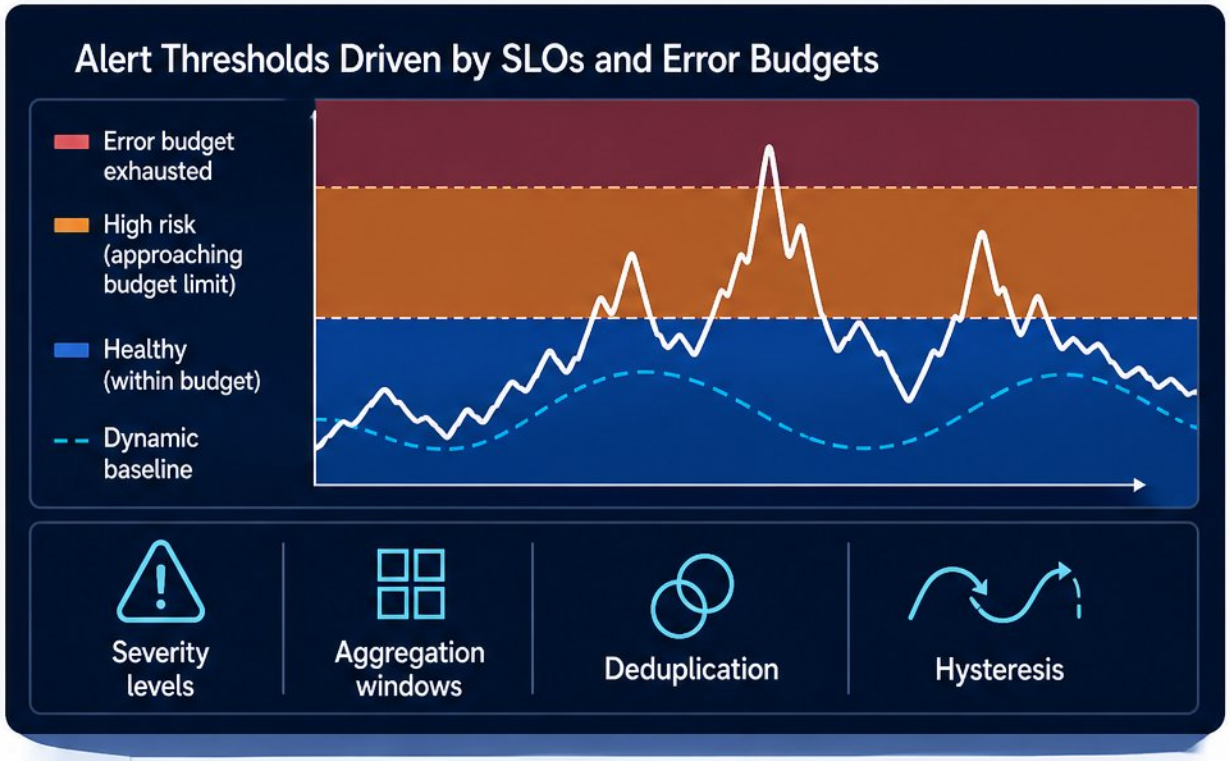
# Architecting the Monitoring Workflow




- Define workflow stages: collection, transport, storage, querying, visualization, alerting, escalation
- Map ownership across DBA, platform, and data engineering roles
- Set ingestion cadence, cardinality controls, and sampling per metric
- Balance fidelity and cost with tiered retention policies
- Preserve high-fidelity traces for high-value templates; sample the rest

# Designing Alerts for SLOs, Not Noise

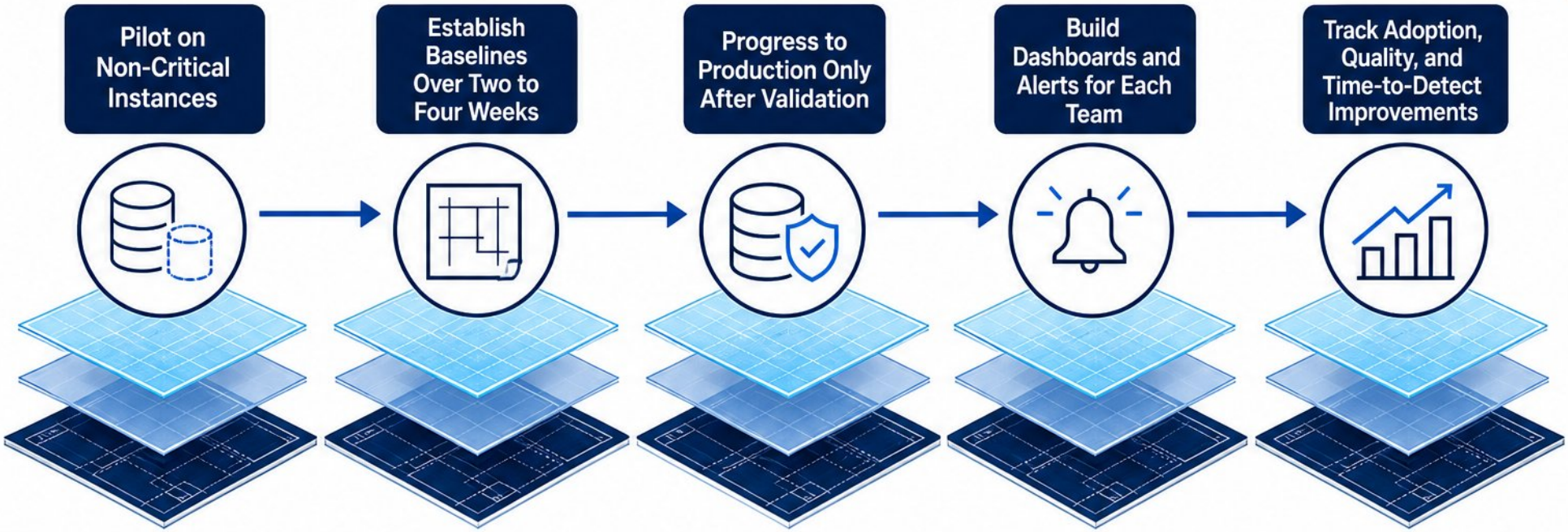
- Thresholds derived from SLOs and error budgets, not raw metric spikes
- Severity levels, aggregation windows, deduplication, and hysteresis reduce alert fatigue
- Connect alerts to runbooks, on-call rotations, and post-incident reviews
- Dynamic baselines beat static thresholds for cyclical workloads
- Closed-loop learning: incidents refine thresholds and dashboards




# Security, Compliance, and Access Controls for Telemetry

- Encrypt telemetry in transit and at rest
- Limit exposure of query text, credentials, and customer data
- Meet GDPR, HIPAA, SOC 2, and audit requirements
- Apply least-privilege access to dashboards and raw logs
- Mask and redact query text in shared observability stores

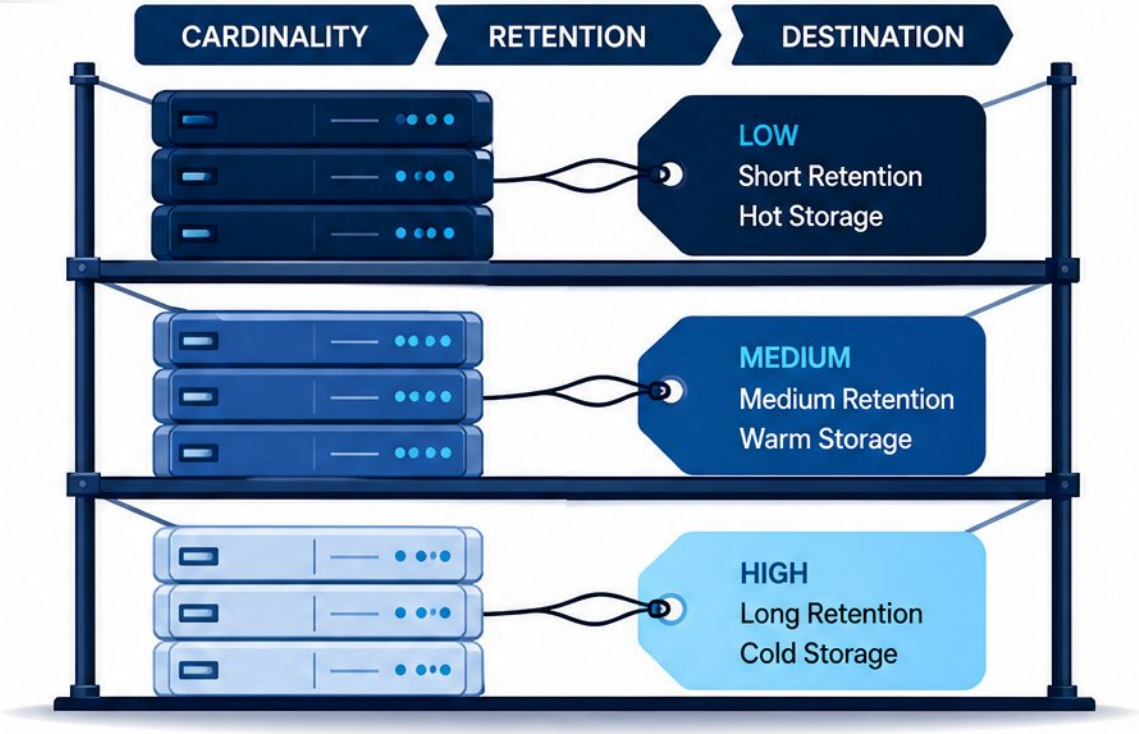
# Implementation and Rollout Strategy




- Start with pilot on non-critical instances
- Establish baselines over two to four weeks
- Progress to production only after validation
- Build dashboards and alerts for each team
- Track adoption, quality, and time-to-detect improvements

# Cost Control and Cardinality Management

- Control costs with cardinality limits, sampling, and retention tiers
- Keep telemetry that detects, explains, or validates incidents
- Attribute cost per database, telemetry type, and owner
- Total cost formula: license + implementation + triage over 3 years
- Right-size instances and decommission idle resources to reduce spend

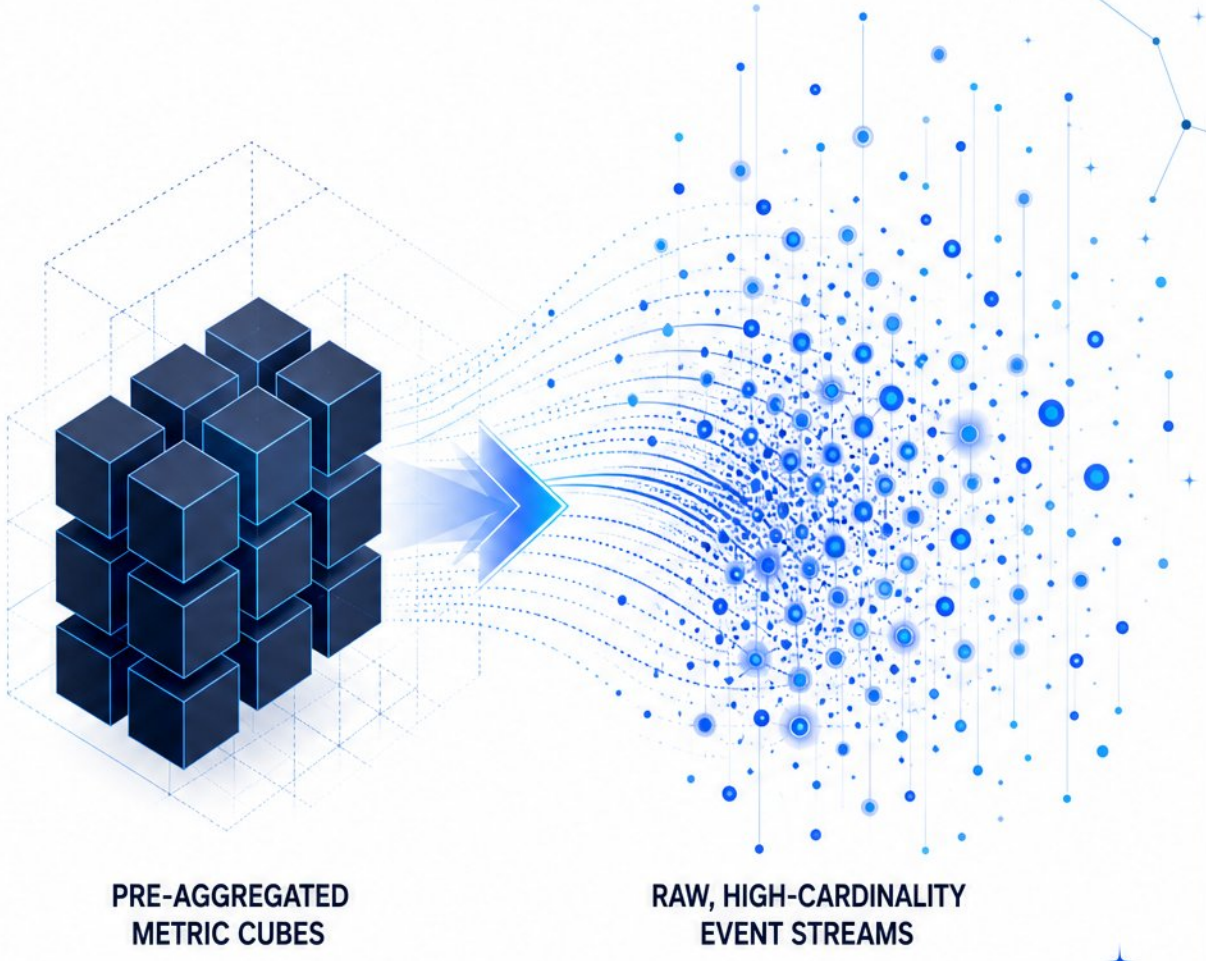



# Operationalizing Monitoring: Dashboards, Runbooks, and Ownership


- Use SRE overview, DBA forensics, engineering self-service dashboards
- Link alerts to diagnostic queries, steps, escalation in runbooks
- Define ownership for collection, storage, alerting, dashboards
- Review incidents to refine thresholds, dashboards, workflows

# Future Trends and the Path to Observability 2.0

- AI-assisted anomaly detection and predictive cost models
- OpenTelemetry as the de facto collection standard
- Observability 2.0: raw, high-cardinality events, query-time slicing
- Shift from pre-aggregated metrics to arbitrary dimension grouping
- Plan for Performance Insights EOL and Database Insights migration
- Consolidate agents and collectors across the SQL estate



# Workshop: Scoring Your Current Monitoring Stack

- Apply the weighted evaluation framework from module 4 to your own SQL estate
- Run a gap analysis: missing signals, alert noise, ownership, compliance, cost overruns
- Prioritize next steps with an impact/effort matrix
- Output: one agreed action per team for the next sprint

# Key Takeaways and Recommended Next Steps

- Monitoring detects known failures; observability diagnoses novel ones; both are complementary
- Tool selection driven by weighted criteria, not vendor familiarity
- Workflow design, alerting, and cost control are ongoing practices
- Next: baseline telemetry, run a tool trial, establish SLO-based alerting on one critical DB

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/448fdaff/public