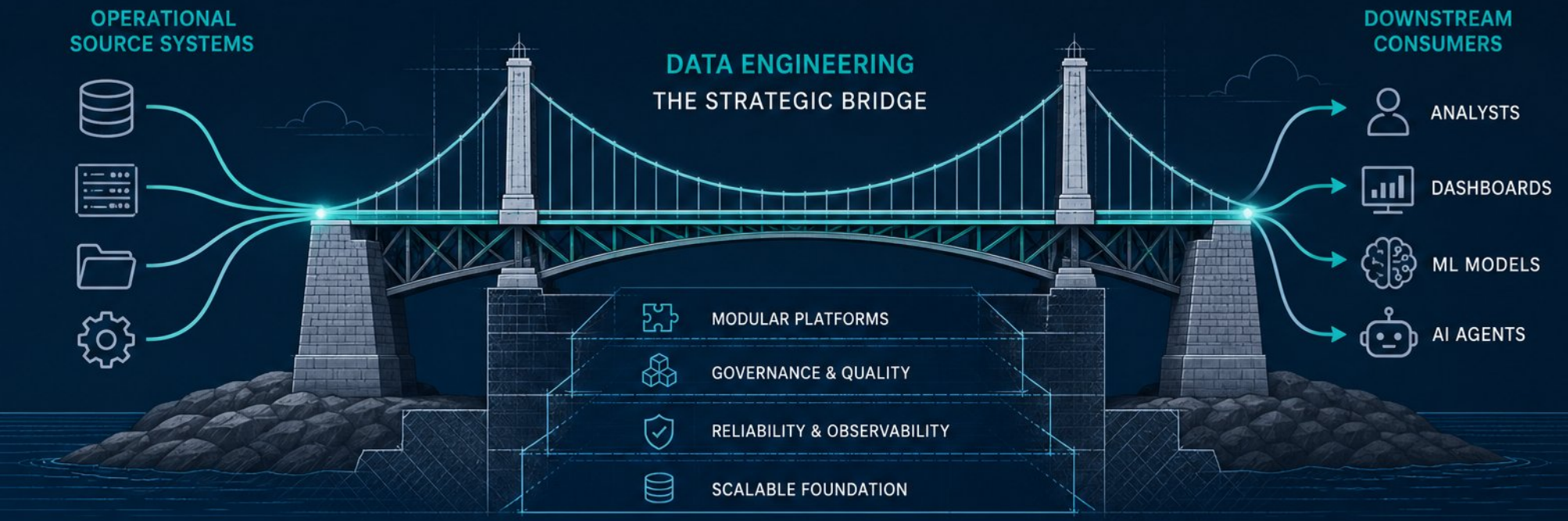
Data Engineering Process: Stages, Roles, and Deliverables

- Structured process ensures reliable, scalable, trustworthy analytics
- Full journey: raw source data to business-ready data products and AI inputs
- Five core stages: planning, ingestion, transformation, storage, serving
- Governed and observed at every step, not as an afterthought
- Connected roles: data architect, data engineer, analytics engineer

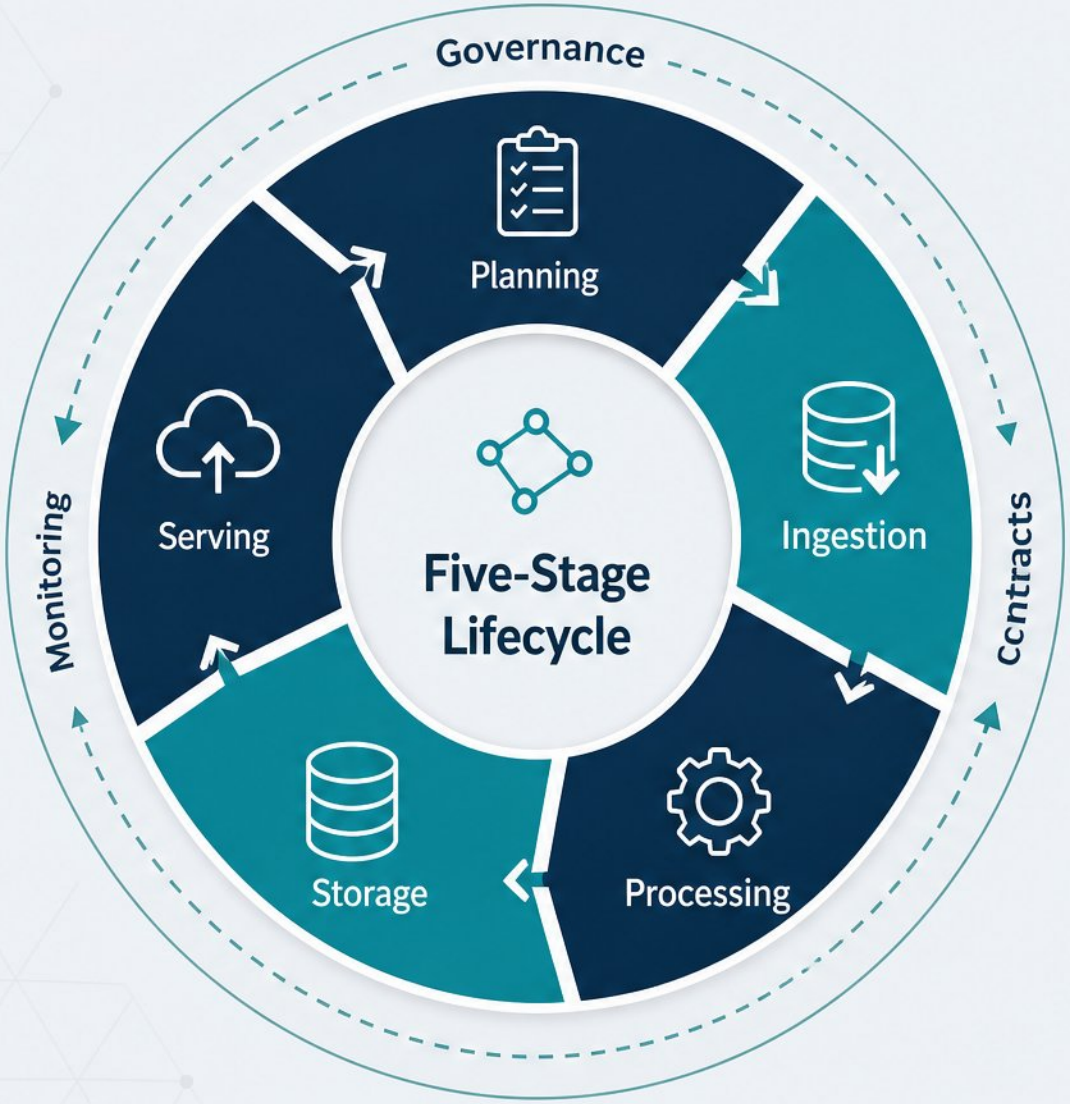
CONNECTED ROLES

Why Data Engineering Is a Strategic Discipline



- Bridges operational source systems and downstream consumers: analysts, dashboards, ML models, and AI agents
- Shift from traditional ETL to ELT and modern, modular platforms changes where compute and ownership live
- Upstream failures cascade silently into inaccurate analytics, bad AI outputs, and eroded business trust
- In 2026, data engineering is platform engineering for data, not merely service desk or pipeline plumbing

The Five-Stage Lifecycle at a Glance

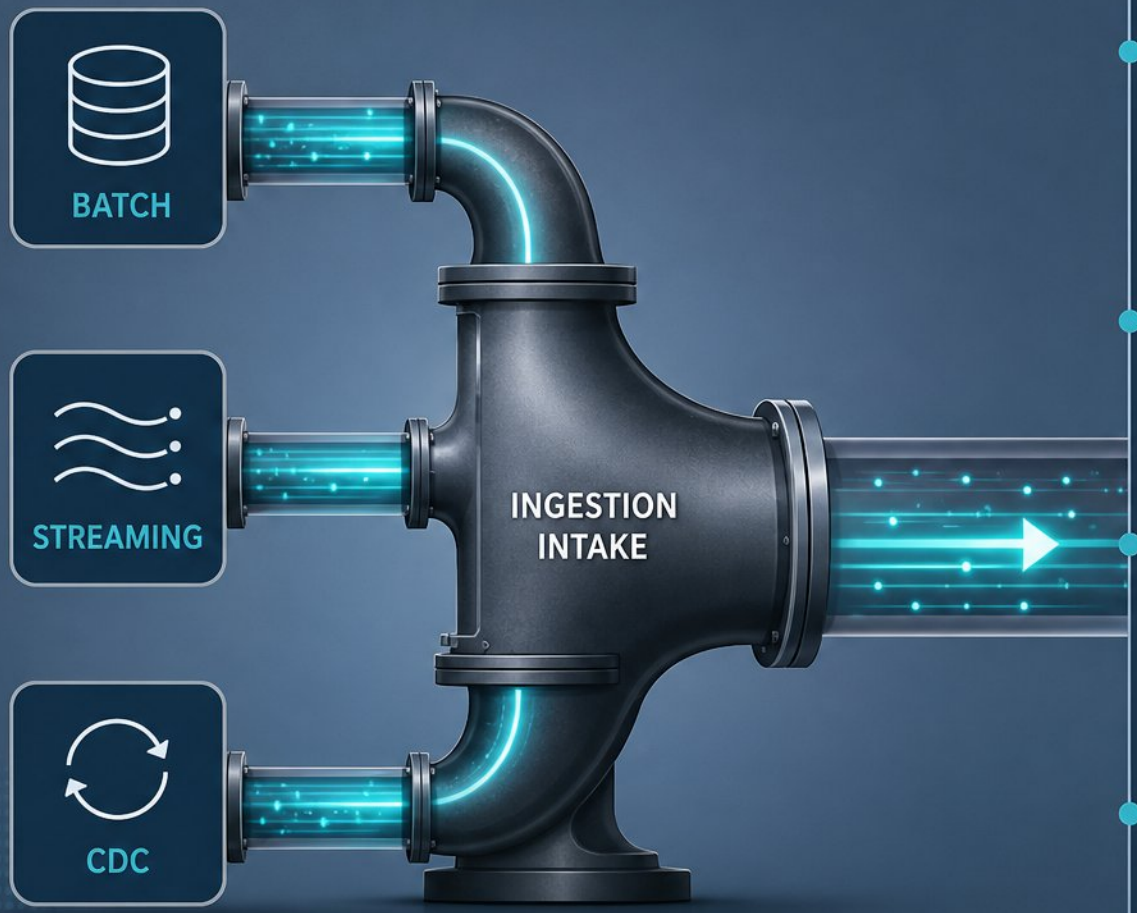


- A practical map: planning, ingestion, processing, storage, and serving
- Governance, monitoring, and contracts wrap around every stage
- Key deliverables: requirements docs, certified datasets, and operational SLAs
- Common terms: source systems, staging, marts, data products, lineage

Stage 1: Planning and Requirements

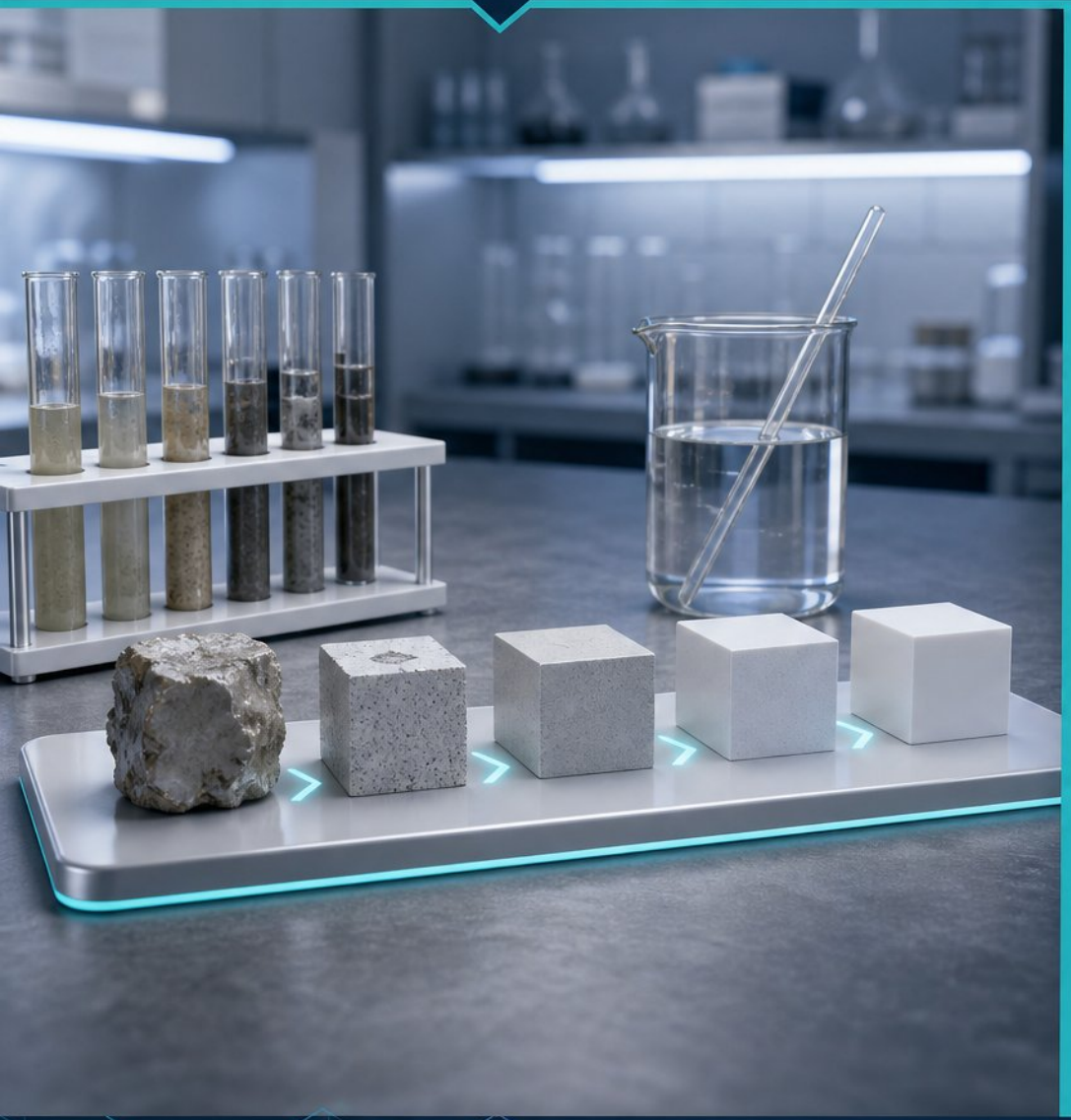
- Define consumers, use cases, freshness needs, and SLAs before building
- Profile source systems: assess quality, volume, velocity, and variety
- Draft source-to-target mappings capturing explicit business rules and defaults
- Deliverables: requirements doc, mapping, and a draft data contract with ownership

Stage 2: Ingestion and Integration



- Match batch, streaming, micro-batch, or CDC to source behavior and freshness SLAs
- Pull vs. push integration patterns: understand operational trade-offs
- Handle schema drift, late data, backfills, and idempotent processing by design
- Deliverables: pipelines, raw landing zones, runbooks, and source-near quality gates

Stage 3: Processing and Transformation

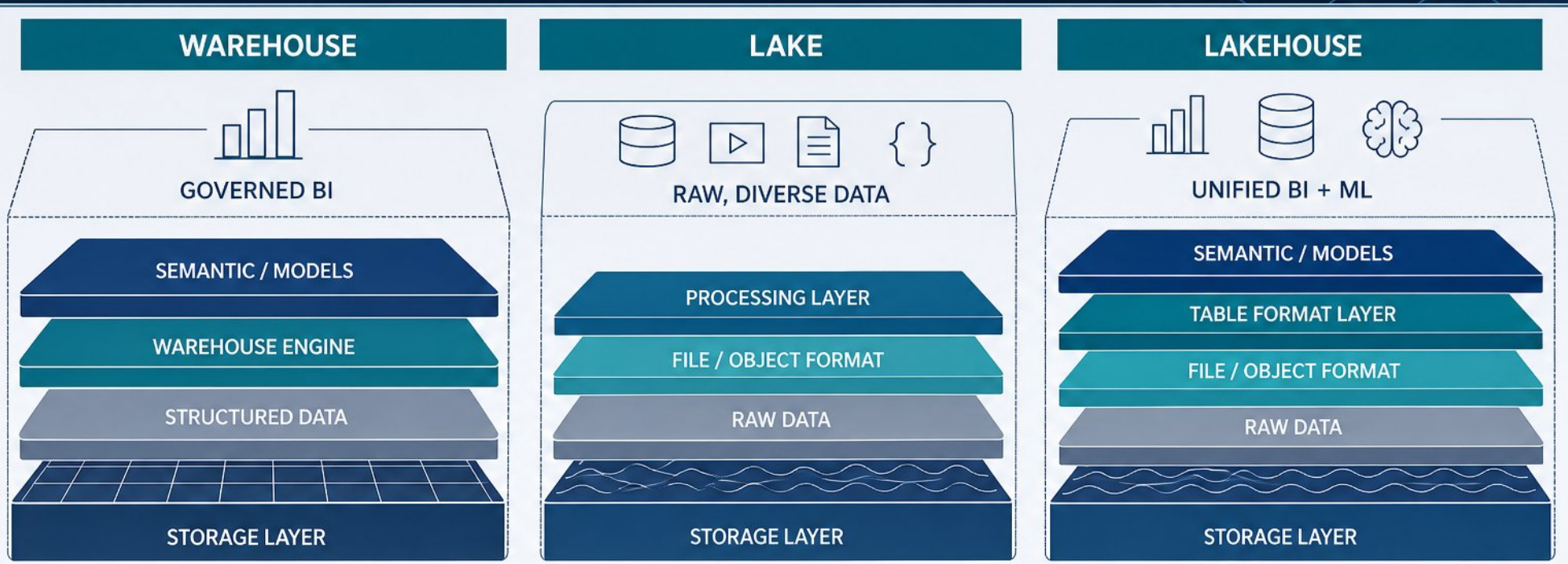


- Clean, standardize, deduplicate, and enrich raw data into source-aligned staging models
- Separate source-centric from business-centric transformations to reduce rework
- Use incremental and full-refresh strategies deliberately, with explicit backfill and replay plans
- Deliverables: transformed datasets, dbt tests or equivalent suites, and documented lineage

Testing, Quality, and Lineage in Processing

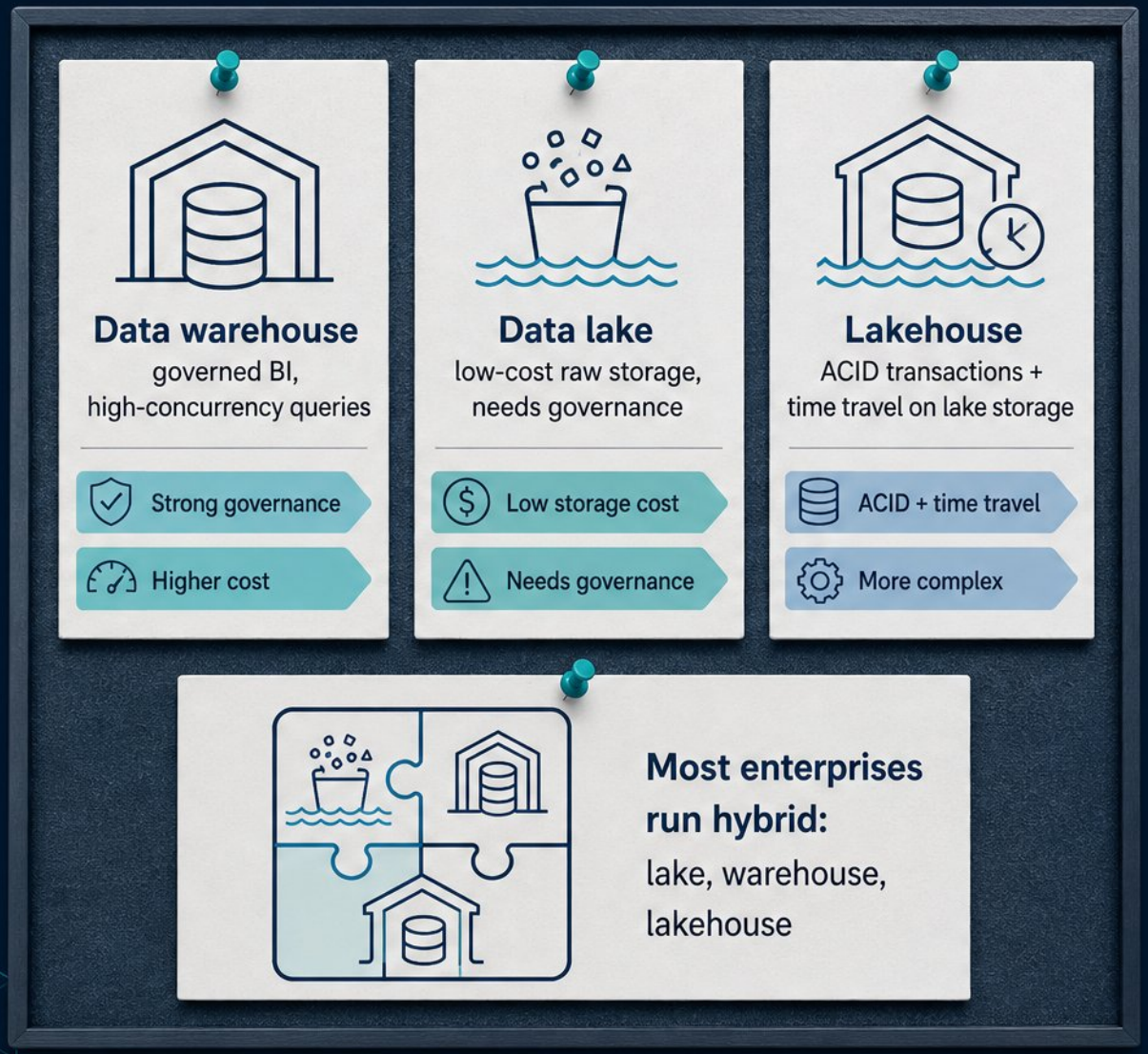
- Treat transformations like software: version control, review, CI tests
- Test primary keys, referential integrity, accepted values, freshness
- Unit-test complex logic with static inputs before materializing
- Lineage from code and DAGs: fast root-cause and safe impact checks
- Observability tracks volume, schema, distribution; catches drift early

Stage 4: Storage and Modeling



- Choose warehouse for governed BI; lake for raw, diverse data; lakehouse for unified BI + ML
- Model with dimensional schemas, Data Vault, or wide tables based on query patterns
- Design partitioning, clustering, and access patterns with cost awareness
- Deliver certified datasets, documented models, access policies, retention rules

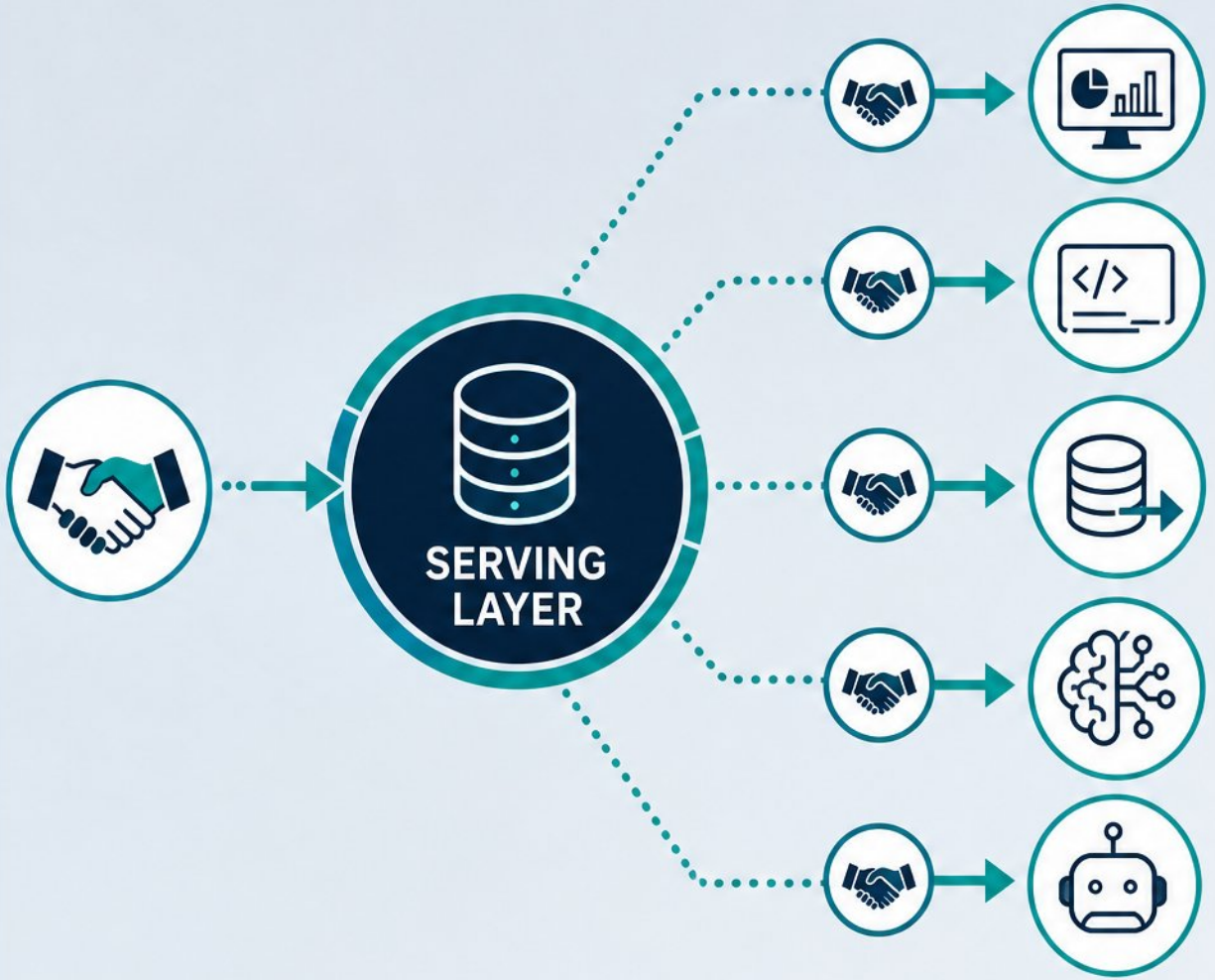
Modern Storage Architectures and Their Trade-offs



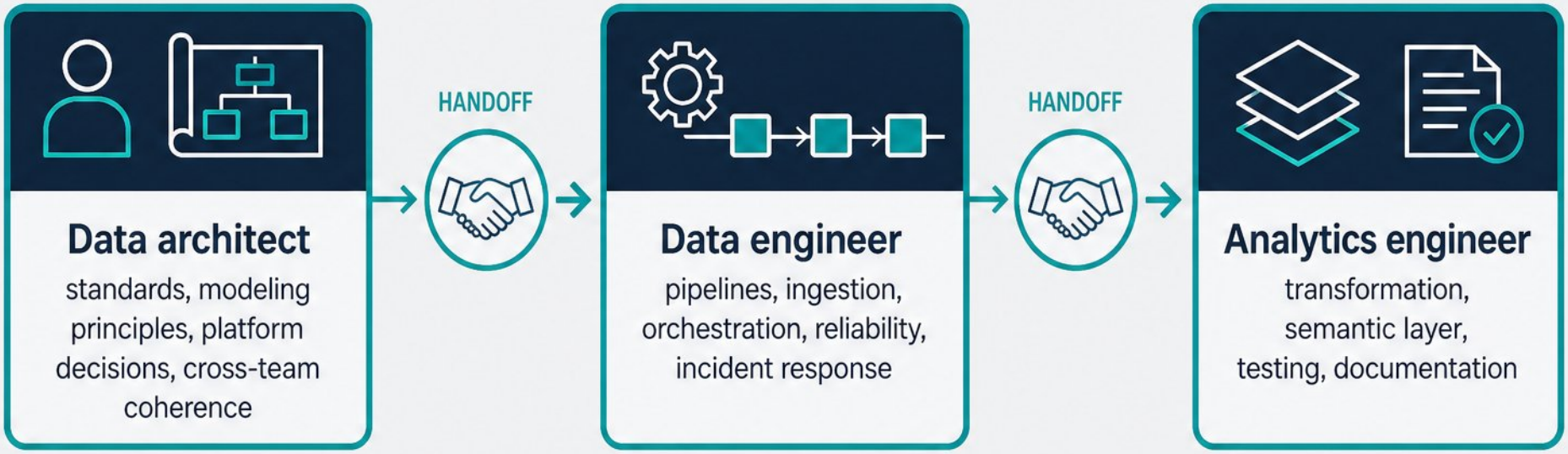
- Data warehouse: governed BI, high-concurrency queries
- Data lake: low-cost raw storage, needs governance
- Lakehouse: ACID transactions + time travel on lake storage
- Most enterprises run hybrid: lake, warehouse, lakehouse
- Match storage architecture to workload, not trend

Stage 5: Serving and Consumption

- Serving patterns: BI dashboards, ad hoc SQL, reverse ETL, ML features, AI agents
- **Per-consumer SLAs** define freshness, latency, and availability individually
- Semantic layers and caching protect systems from noisy consumers
- **Deliverables:** data products, semantic layer definitions, reverse ETL, SLAs



Roles and Ownership Across the Lifecycle

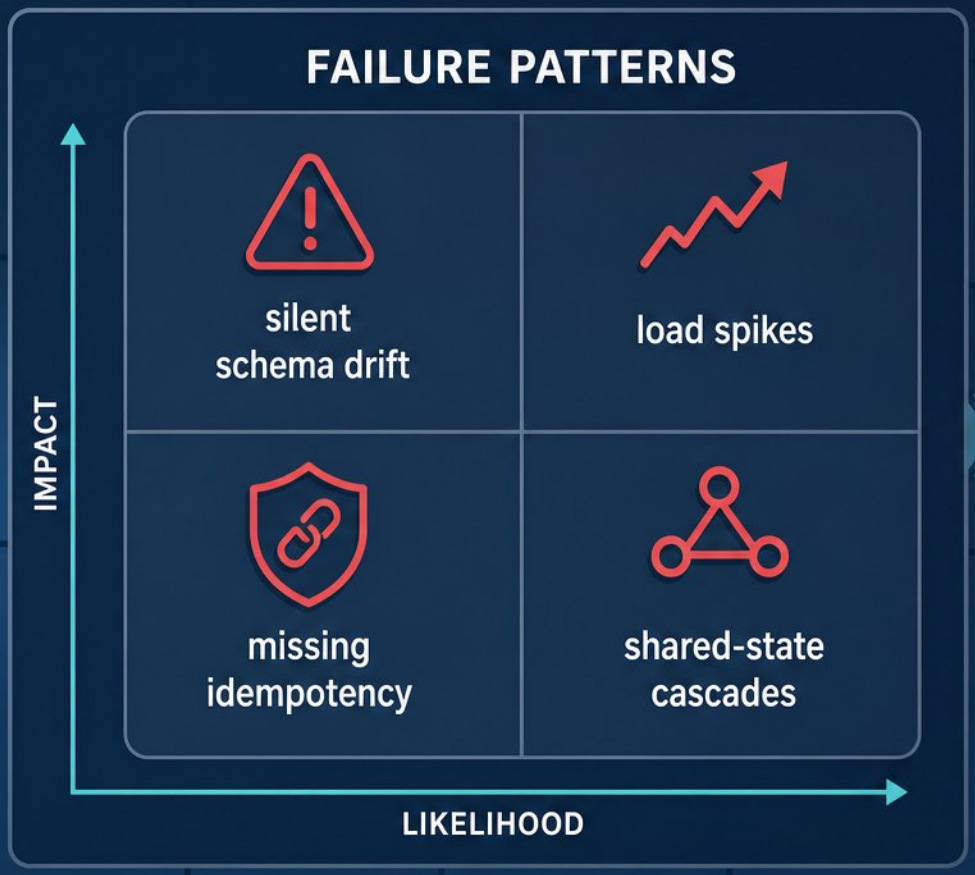


- **Data** architect: standards, modeling principles, platform decisions, cross-team coherence
- **Data engineer:** pipelines, ingestion, orchestration, reliability, incident response
- **Analytics engineer:** transformation, semantic layer, testing, documentation
- Clear handoffs prevent abandoned datasets and ambiguous incident ownership

Artifacts That Keep Data Teams Aligned

- Data contracts: executable agreements enforced in CI and at ingestion
- Runbooks: documented, repeatable failure response
- Lineage: living dependency maps for impact analysis
- Observability: dashboards tracking freshness and health
- SLAs and error budgets with named owners

Common Failure Patterns and High-ROI Improvements



- Failure patterns: silent schema drift, load spikes, missing idempotency, shared-state cascades
- Skipping ownership models leaves untrusted, orphaned datasets
- Prioritize three fixes: contracts on critical datasets, automated tests, observability
- Start small, targeting data with the loudest stakeholder impact

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/8d4907df/public