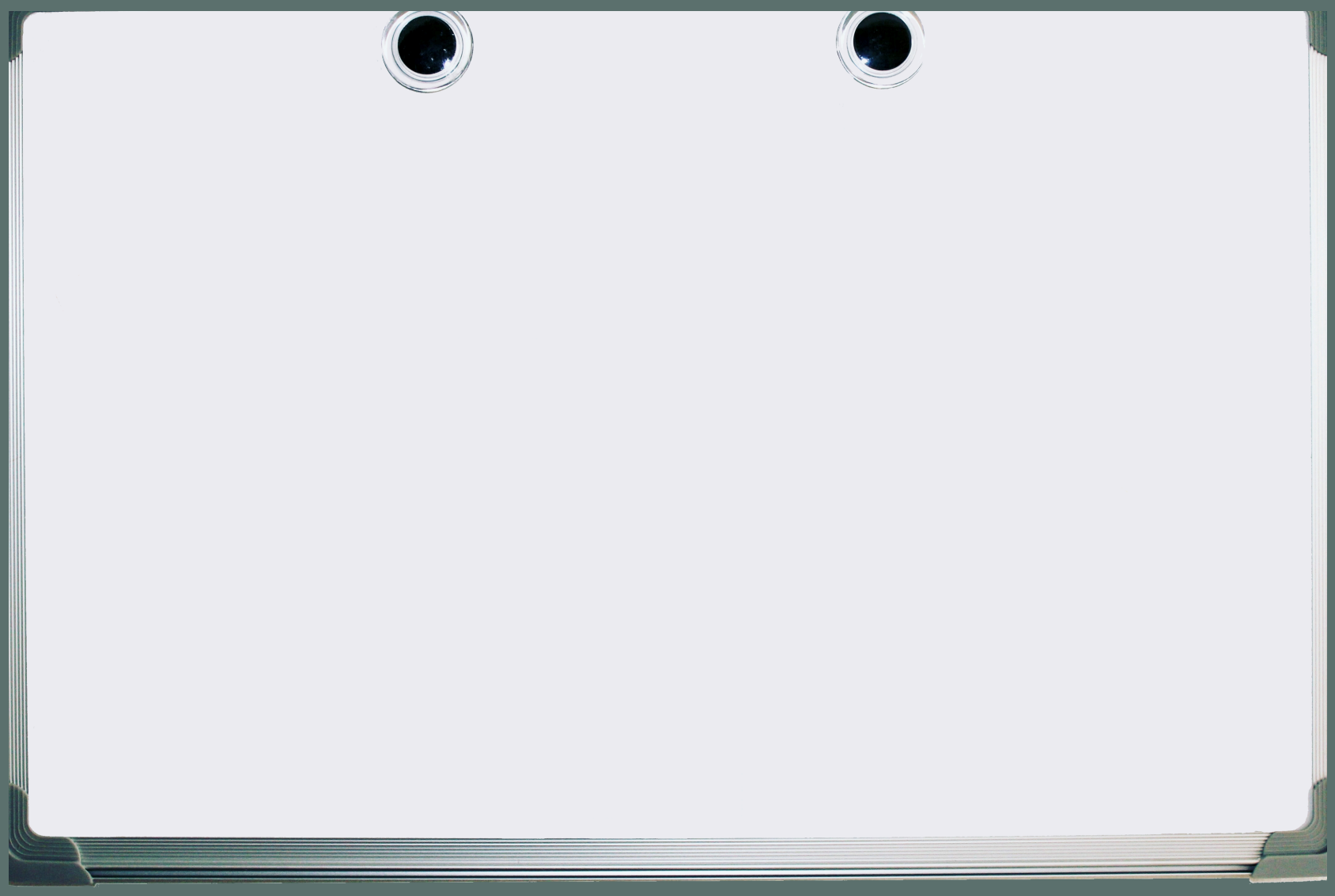
🎨 Systems Design: Petabyte-Scale Data Pipeline



"Ingest **10TB/day**, process it, store it, analyze it—
without breaking the bank"

Where do you start?

Most people start with "big data tools."

I start with data flow and cost constraints.

Here's the thinking process: 👇

📝 Step 1: What Problem Are We Solving?

The Challenge:

10TB/day incoming data
→ **3.65 PB/year**
→ Multiple sources (APIs, databases, streams, files)
→ Different formats (JSON, CSV, Parquet, logs)
→ Need to: clean, transform, analyze, store

Questions that determine everything:
→ Real-time or batch? (Latency requirements?)
→ How long to retain? (Cost implications?)
→ Query patterns? (Analytics vs operational?)
→ Compliance? (GDPR, data residency?)

Wrong architecture = millions wasted on storage you don't need.
Right architecture = only pay for what you use.

🔄 Step 2: Map the Data Flow

Ingestion → Storage → Processing → Analytics

Fill in the [**?**]:
→ Where does **data** come from? (APIs, DBs, streams?)
→ Where does **raw data** land? (Data lake?)
→ How do we **transform** it? (Spark, Airflow?)
→ Where does **clean data** go? (Data warehouse?)
→ How do we **query** it? (SQL, BI tools?)

This is the skeleton.
Now add the medallion architecture that production needs.

🥇 Step 3: Bronze → Silver → Gold

Bronze Layer (Raw):
→ Data lands exactly as received
→ No transformation, no validation
→ Cheap storage (S3, GCS, Azure Blob)
→ Append-only, never delete

Silver Layer (Cleaned):
→ Validated, deduplicated
→ Schema enforcement
→ Type casting, null handling
→ Partition by date/source

Gold Layer (Analytics-Ready):
→ Aggregated, joined
→ Business logic applied
→ Optimized for queries
→ Served to BI tools, ML models

Why three layers?
→ Bronze = audit trail (what arrived?)
→ Silver = reliable source (cleaned)
→ Gold = fast queries (optimized)

Most teams skip Bronze. Then they can't debug.

📥 Step 4: Data Ingestion

Sources:
→ **Streaming** (Kafka, Kinesis, Pub/Sub)
→ **APIs** (REST, webhooks)
→ **Databases** (CDC - Change Data Capture)
→ **Files** (S3, SFTP, object storage)
→ **Logs** (Fluentd, Logstash)

Ingestion Tools:
→ **Airbyte** (pre-built connectors)
→ **Fivetran** (managed, expensive)
→ **Custom** scripts (Python, Go)
→ **Kafka Connect** (streaming)

Questions to answer:
→ Batch or streaming? (Latency vs cost)
→ Schema on read or write? (Flexibility vs validation)
→ Retry logic? (What if source fails?)
→ Rate limiting? (Don't overwhelm APIs)

Bronze layer = everything lands here first.
Raw, unprocessed, exactly as received.

⚙️ Step 5: Processing at Scale

Processing Engine:

→ Databricks (managed Spark, Delta Lake native)
→ Apache Spark (open-source)
→ Google Dataflow (serverless)
→ AWS Glue (managed ETL)

Orchestration:

→ Databricks Workflows (native scheduling)
→ Airflow (DAG-based scheduling)
→ Prefect (modern Airflow alternative)

Bronze → Silver transformation:

→ Schema validation (reject bad data)
→ Deduplication (same event twice?)
→ Type casting (strings → dates → integers)
→ Partitioning (by date, by source)
→ Delta Lake (ACID transactions, versioning)

Silver → Gold transformation:

→ Joins (combine datasets)
→ Aggregations (hourly, daily rollups)
→ Business logic (metrics, KPIs)
→ Denormalization (optimize for reads)

Processing = where you spend compute $.
Optimize here or burn budget.

💾 Step 6: Storage Tiering

Bronze (Raw - Cheap):
→ S3 Standard (frequently accessed)
→ After 30 days → S3 Infrequent Access
→ After 90 days → S3 Glacier (archive)
→ Cost: $0.023/GB → $0.01/GB → $0.004/GB

Silver (Cleaned - Medium):
→ S3 Standard (queried regularly)
→ Delta Lake format (ACID transactions, time travel)
→ Partitioned (date, source)

Gold (Analytics - Fast):
→ Databricks SQL Warehouse (interactive queries)
→ Or S3 + Athena (serverless queries)
→ Delta Lake optimized for BI tools

Why tier storage?
→ Bronze: 3.65 PB/year @ **$0.023/GB = $84K/year**
→ Bronze with tiering: ~**$40K/year (50% savings)**

Lifecycle policies = automatic cost optimization.
Set it once, save forever.

📊 Step 7: Query & Analytics

Query Engines:

→ Databricks SQL (serverless, auto-scaling)
→ BigQuery (GCP serverless)
→ Redshift (AWS-native)
→ Athena (query S3 directly, cheapest)

BI Tools:

→ Tableau (dashboards)
→ Looker (embedded analytics)
→ Power BI (Microsoft stack)
→ Databricks SQL Analytics (built-in)

ML Access:

→ Databricks Feature Store (native integration)
→ Direct S3/GCS access (Spark, Python)

Query optimization:

→ Delta Lake (Z-ordering, data skipping)
→ Partition pruning (only scan needed dates)
→ Column pruning (only read needed columns)
→ Liquid clustering (automatic optimization)

Gold layer = optimized for this.
Queries run in seconds, not minutes.

📈 Step 8: Data Observability

Track:

→ Data freshness (is today's data here?)
→ Volume anomalies (sudden spike/drop?)
→ Schema changes (breaking downstream?)
→ Null rates (data quality degrading?)
→ Processing lag (how far behind?)

Tools:

→ Great Expectations (data validation)
→ Monte Carlo (data observability)
→ dbt (data transformation + testing)
→ Custom metrics (Prometheus + Grafana)

Alerts:

→ Data missing (expected 10TB, got 1TB)
→ Pipeline failed (Airflow DAG error)
→ Query slow (Databricks warehouse overloaded)
→ Cost spike (someone left Spark cluster running)

Without monitoring:

→ Bad data propagates to Gold
→ BI dashboards show wrong numbers
→ ML models train on garbage
→ Business makes decisions on bad data

Data quality > data quantity

⚠️ Step 9: Design for Failure

Where does petabyte pipeline break?

Ingestion failures:
→ API rate limits (retry with backoff)
→ Source downtime (queue and replay)
→ Schema changes (alert, don't fail)

Processing failures:
→ OOM errors (tune Spark memory)
→ Skewed partitions (repartition)
→ Corrupt data (skip, log, alert)

Storage failures:
→ S3 throttling (batch smaller)
→ Disk full (auto-scale, lifecycle policies)

Cost explosions:
→ Runaway Spark jobs (set timeouts)
→ Full table scans (partition pruning)

Then design defenses:
→ Idempotency (rerun safely)
→ Checkpointing (resume where failed)
→ Dead letter queues (bad data isolated)
→ Circuit breakers (stop before it gets worse)

Top pipelines fail gracefully. And recover automatically.

💰 Step 10: Cost Control at Scale

Cost optimization isn't optional at **petabyte scale.**
It's survival.

Where money goes:

Storage: 3.65 PB/year
→ Without tiering: **$84K**/year
→ With tiering: **$40K/year (50% savings**)

Compute (Databricks):
→ All-purpose clusters: **$100K/year**
→ Jobs clusters (auto-terminate): **$30K/year** (**70% savings**)

Queries (Databricks SQL):
→ Always-on warehouse: **$50K/year**
→ Serverless SQL (auto-suspend): **$20K/year** (**60% savings**)

Data transfer:
→ Cross-region: **$10K/year**
→ Same region: **$1K/year (90% savings**)

Total without optimization: $244K/year
Total with optimization: $91K/year
Savings: $153K/year (63% reduction)