### **Data Replication Overview**

* **Why Replicate Data?**
  + **Scalability & High Throughput:** Handle growing data and read/write loads.
  + **Fault Tolerance & High Availability:** Maintain application uptime despite failures.
  + **Latency Reduction:** Serve users in different locations with low response times.
* **Challenges of Data Replication:**
  + **Consistency:** Updates must be propagated across distributed nodes.
  + **Application Complexity:** Distributed systems require additional logic for reads/writes.

### **Scaling Architectures**

* **Vertical Scaling:**
  + **Shared Memory:** Centralized server with some fault tolerance.
  + **Shared Disk:** Machines share a fast network but face scalability limits with high writes.
  + **Cost Considerations:** Cloud pricing for high-scale vertical machines can be expensive.
* **Horizontal Scaling (Shared Nothing):**
  + Each node has its own CPU, memory, and disk.
  + Coordination happens via application layer over the network.
  + Allows geographical distribution and use of commodity hardware.

### **Replication vs. Partitioning**

* **Replication:** Duplicates full datasets across nodes.
* **Partitioning:** Splits data, each partition storing a subset of the whole.

### **Replication Strategies**

1. **Single Leader Model:**
   * **Writes go to a single leader**, which replicates changes to followers.
   * Followers process instructions from the leader and serve read queries.
   * **Examples:** MySQL, PostgreSQL, Oracle, MongoDB, Kafka.
2. **Multiple Leader Model:**
   * Multiple leaders accept writes, requiring conflict resolution.
   * Used when applications span multiple data centers.
3. **Leaderless Model:**
   * Any node can accept writes, and data propagates via consensus mechanisms.
   * **Examples:** DynamoDB, Cassandra.

### **Replication Methods**

* **Statement-Based:** Sends raw SQL statements to replicas (error-prone).
* **Write-Ahead Log (WAL):** Logs changes at a byte level (storage-engine dependent).
* **Logical (Row-Based) Logs:** Tracks row-level changes, decoupled from storage engine.
* **Trigger-Based Replication:** Uses triggers to log changes but adds complexity.

### **Synchronous vs. Asynchronous Replication**

* **Synchronous:** Leader waits for follower confirmation (strong consistency but slower).
* **Asynchronous:** Leader does not wait (higher availability but potential data loss).

### **Handling Leader Failure**

* **Challenges:**
  + How to elect a new leader?
  + How to configure clients to switch leaders?
  + Avoiding "split-brain" situations where multiple leaders emerge.
* **Consensus Strategies:**
  + Choose a new leader based on replication state.
  + Use a controller node to manage failover.

### **Replication Lag & Consistency Guarantees**

* **Replication Lag:** Delay in propagating writes to all followers.
* **Read-After-Write Consistency:**
  + Clients should read **recently modified data** from the leader.
  + Can dynamically switch read behavior for "recently updated" data.
* **Monotonic Read Consistency:**
  + Ensures users **never see older data** after reading newer data.
* **Consistent Prefix Reads:**
  + Guarantees that writes appear **in the correct order** acro