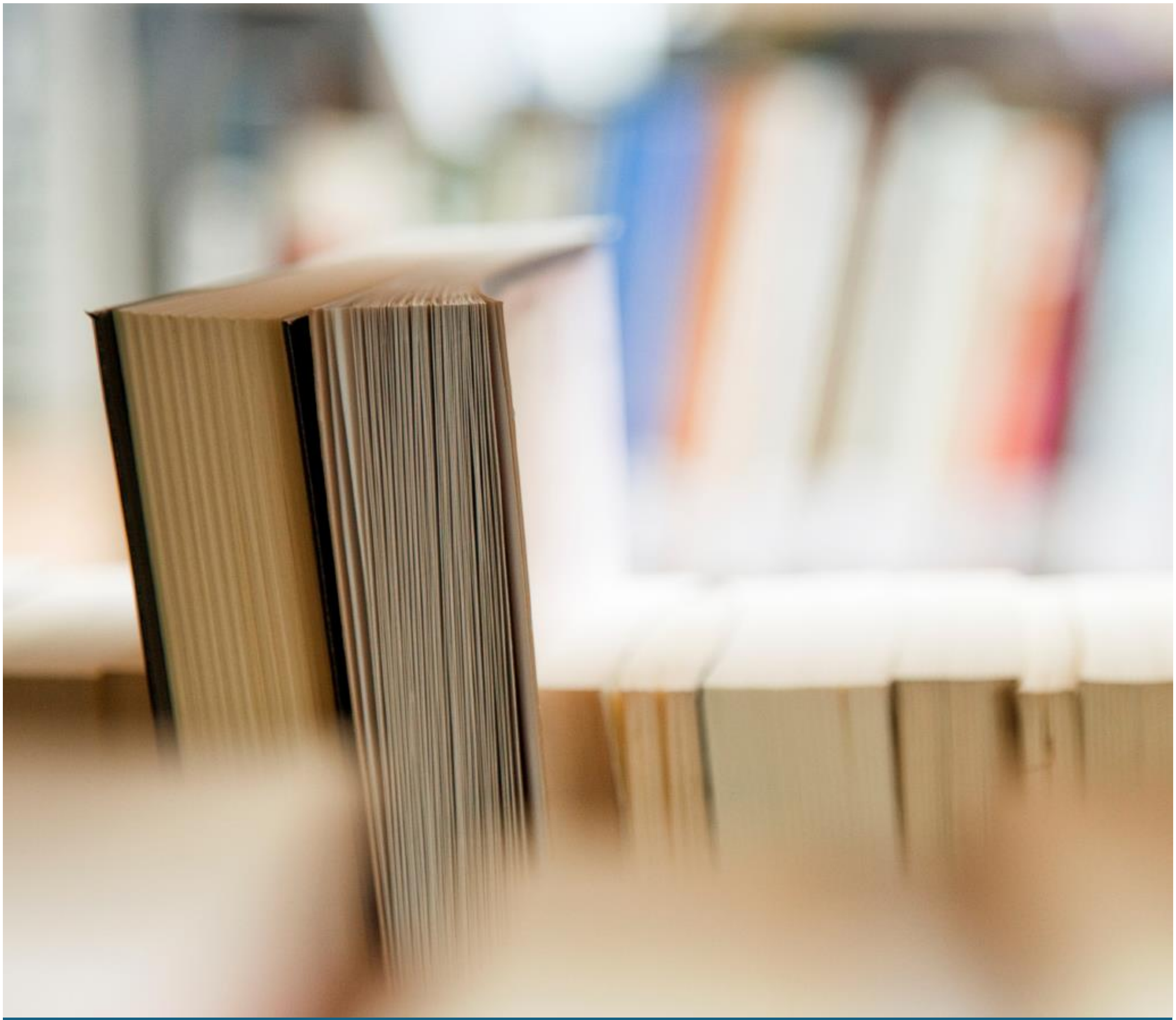

# SDA

Assignment

Muhammad Hashir(FA22-BSE-031) 3/1/25 SDA

**Real-Life Software Failures Due to Architectural Problems**

**1. Twitter: Scalability Issues in Monolithic Architecture**

- **Version with Problem**: Early 2008 (Monolithic architecture).
- **Problems Identified**:
  - High traffic during events (e.g., presidential elections) caused the system to crash.
  - All functionalities (tweeting, user management, notifications) were tightly coupled in a single codebase, creating bottlenecks.
  - Scaling the monolith required expensive hardware upgrades.
  - Rolling out updates required halting the entire system, disrupting services.
- **Solution and Updated Version**: Transitioned to a **microservices architecture** (2010 onward).
  - Split the monolithic system into smaller services (e.g., tweet service, timeline service).
  - Introduced a distributed messaging queue (e.g., Apache Kafka) to handle user requests efficiently.
  - Deployed services independently, reducing downtime during updates.

---

**2. Netflix: Lack of Flexibility in Rigid Systems**

- **Version with Problem**: Pre-2009 (Monolithic backend for DVD rentals).
- **Problems Identified**:
  - The architecture was designed for DVD rentals, limiting its ability to handle the shift to streaming services.
  - Feature deployments were delayed due to interdependencies within the codebase.
  - The system lacked scalability for handling millions of streaming users concurrently.
  - Frequent downtimes frustrated users and hindered growth.
- **Solution and Updated Version**: Migrated to **cloud-based microservices architecture** (2011 onward).
  - Moved the infrastructure to AWS for on-demand scaling.

  - Split the system into over 1,000 independent microservices (e.g., recommendation engine, playback services).
  - Enabled faster feature deployment cycles, improving customer experience.

---

### 3. Microsoft Office Suite: Dependency Issues in Tightly Coupled Architecture

- **Version with Problem**: Early 2000s (Office XP and earlier).
- **Problems Identified**:
  - Tightly coupled applications (Word, Excel, PowerPoint) meant bugs or updates in one application affected others.
  - Large and interdependent codebase made testing and deployment time-consuming.
  - Integration with new cloud technologies (e.g., OneDrive) was cumbersome.
- **Solution and Updated Version**: Modularized architecture introduced in **Office 2007**.
  - Broke down the suite into independent modules, each with its own APIs.
  - Allowed independent updates, reducing overall testing and deployment time.
  - Improved integration with cloud-based storage and collaboration tools.

---

### 4. Amazon Prime Video: Scalability and Availability Challenges in Monolithic Architecture

- **Version with Problem**: Pre-2012 (Monolithic system for video streaming).
- **Problems Identified**:
  - High traffic during popular events (e.g., new show releases) caused service interruptions.
  - The monolithic architecture struggled to handle regional demands for content delivery.
  - Updates required shutting down the entire system, leading to downtime.
  - Scaling the system horizontally was difficult due to tightly coupled components.
- **Solution and Updated Version**: Adopted a **microservices architecture** (2012 onward).
  - Split the system into independent microservices (e.g., content delivery, user authentication, playback).

  - Leveraged AWS infrastructure to achieve global scalability and redundancy.
  - Implemented region-specific services to handle localized traffic efficiently.
  - Enabled continuous deployment for individual services, eliminating downtime during updates.

---

**5. eBay: Data Integrity Challenges in Distributed Systems**

- **Version with Problem**: Early 2000s (First-generation distributed architecture).
- **Problems Identified**:
  - Data inconsistency due to synchronization issues between databases.
  - Auction timers and bid placements were delayed during high traffic.
  - Event-driven processes were difficult to manage, resulting in system lags.
- **Solution and Updated Version**: Shifted to an **event-driven architecture** (2008 onward).
  - Introduced eventual consistency models to ensure reliability across distributed databases.
  - Upgraded to a robust messaging system (e.g., RabbitMQ) for real-time event handling.
  - Enhanced scalability and fault tolerance, improving overall user experience.