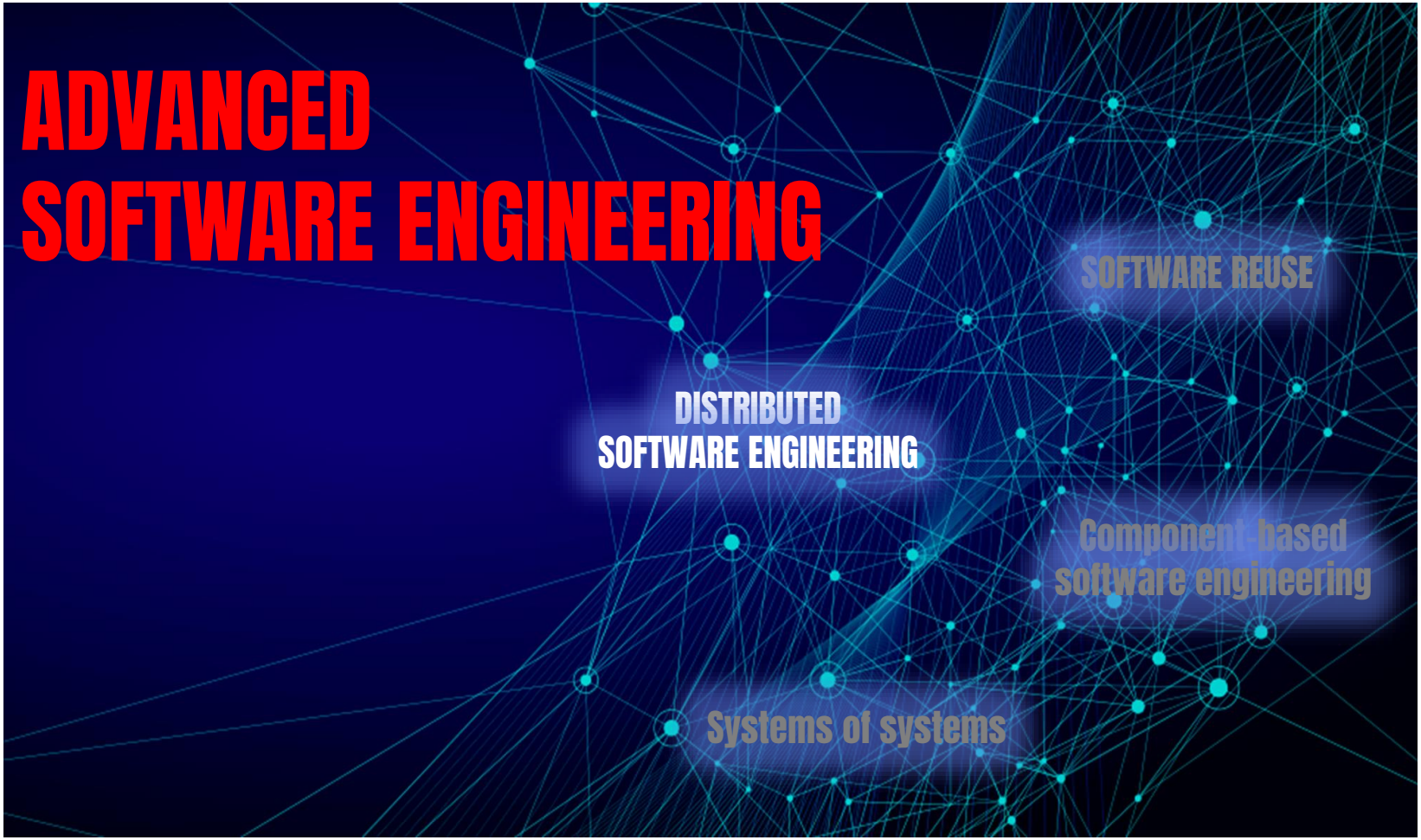
# Class9: Distributed Software Engineering

## Chapter 17

Adeeb Noor, PhD
IT Department
Faculty of Computing and Information Technology
King Abdulaziz University

Fall 2025

# Setting the Stage



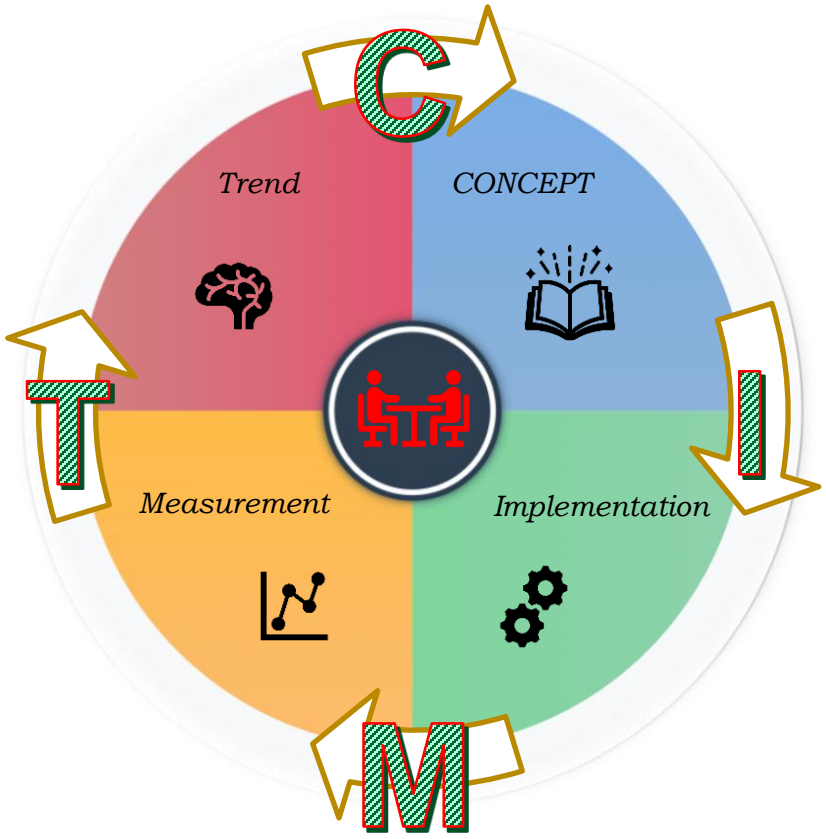
# Our Roadmap

Concept: Principles of distributed systems

Implementation: Client-server models, middleware, security

Measurement: Evaluating system scalability, reliability, security

Trend: Cloud computing, Software as a Service (SaaS)



# The Core Idea

| Standalone system | Distributed system |
| --- | --- |
| All the components are executed within a single device | The components are distributed and executed in multiple devices |
| Do not need a network | Need a network |
| Usually one or tightly coupled set of technologies are used to develop (JAVA, .NET) | Multiple and loosely coupled set of technologies are used to develop (HTML +CSS + JS + PHP) |

*https://medium.com/@gmnchamikara/everything-about-distributed-systems-1d952dcc3118*

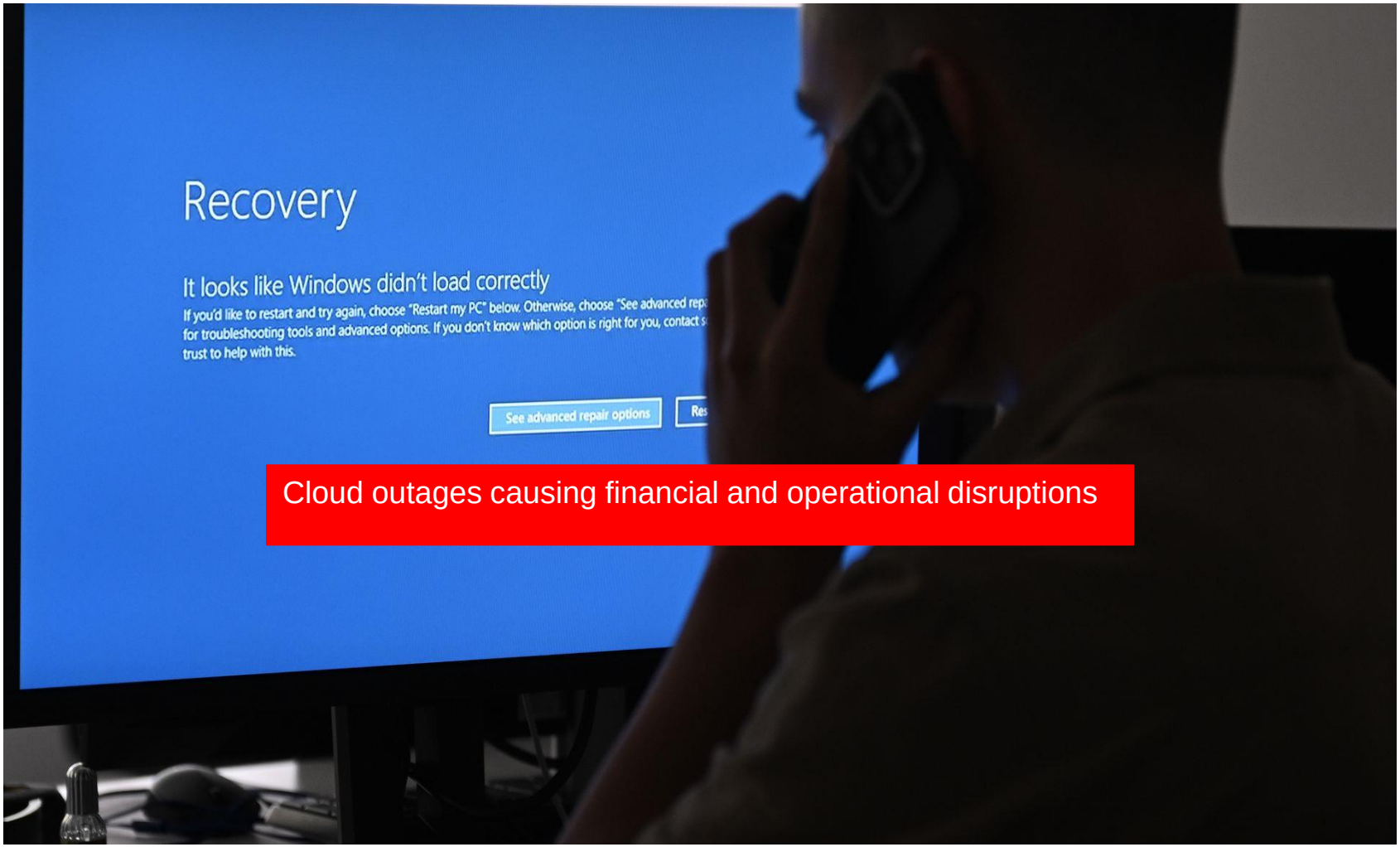
# Key Takeaways

- The Key CLO: Explain distributed systems engineering and distributed systems architectures
- Transparency: Users should not need to know the system is distributed
- Openness: Uses standard protocols for interoperability
- Scalability: Must handle increasing users and workload efficiently
- Security: Managing risks across multiple connected systems

# The Big Why

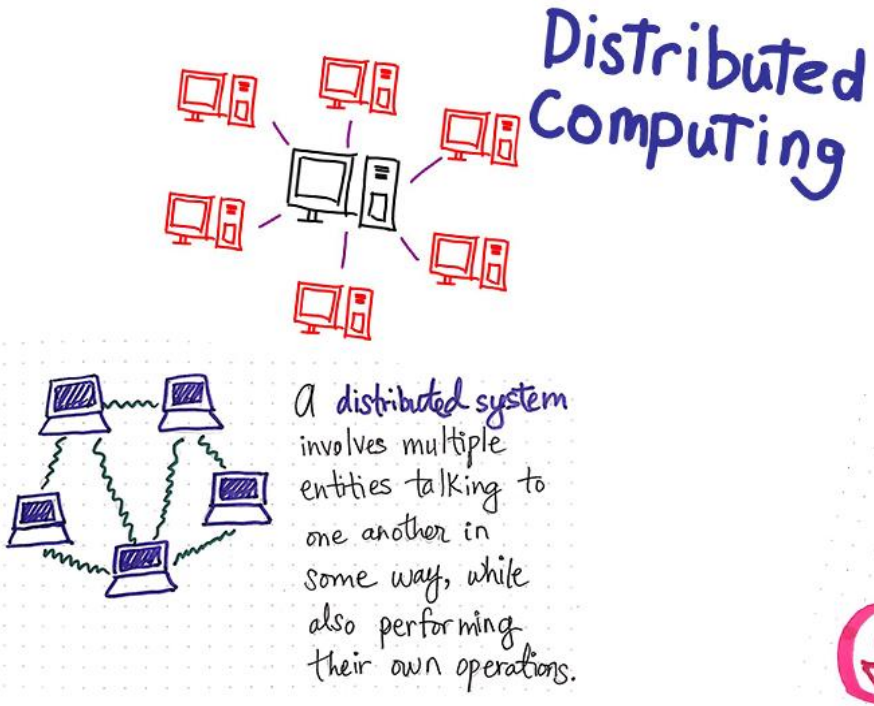
Distributed systems enable high-performance computing, cloud services, and global connectivity but require robust design to balance performance and security

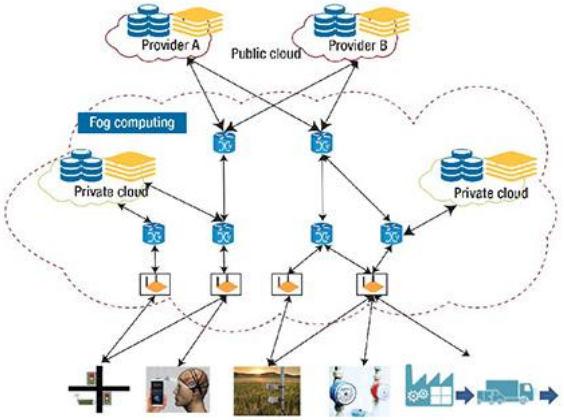
# A Cautionary Tale

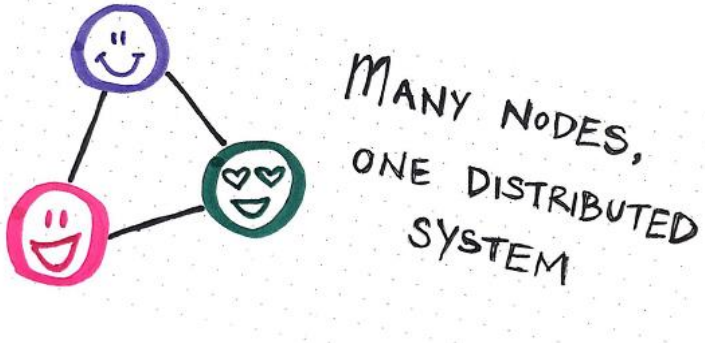


# 17.1 Distributed Software

- **Distributed Software (DS)** a set of computers which are independent and connected with an interconnection network; it is more complex than systems that run on a single processor

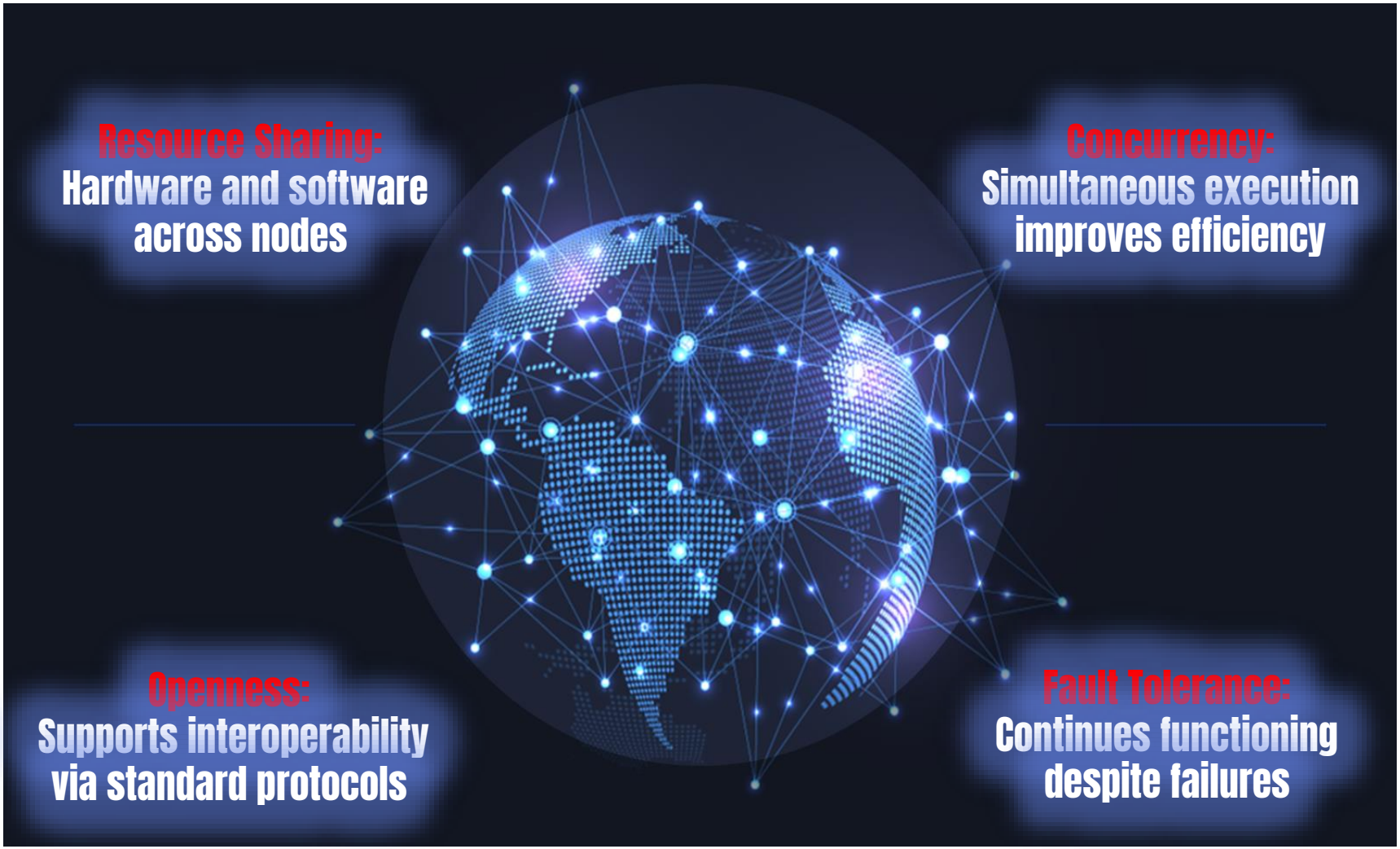


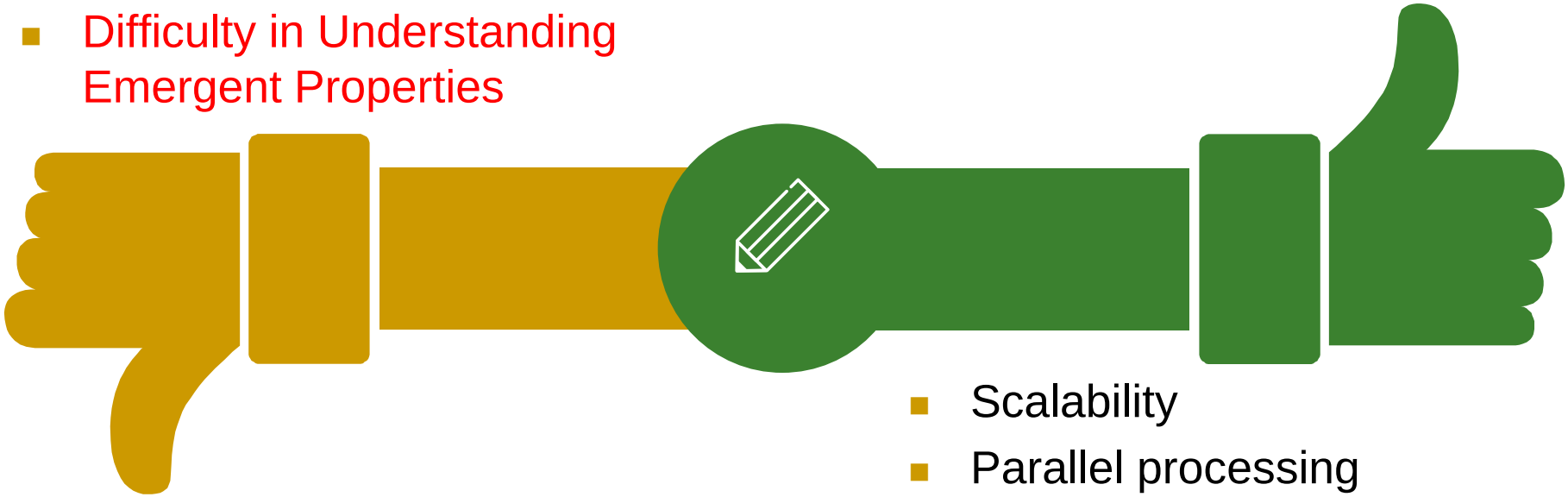

*https://medium.com/@gmnchamikara/everything-about-distributed-systems-1d952dcc3118*

# DS Characteristics



# Complexity of DS

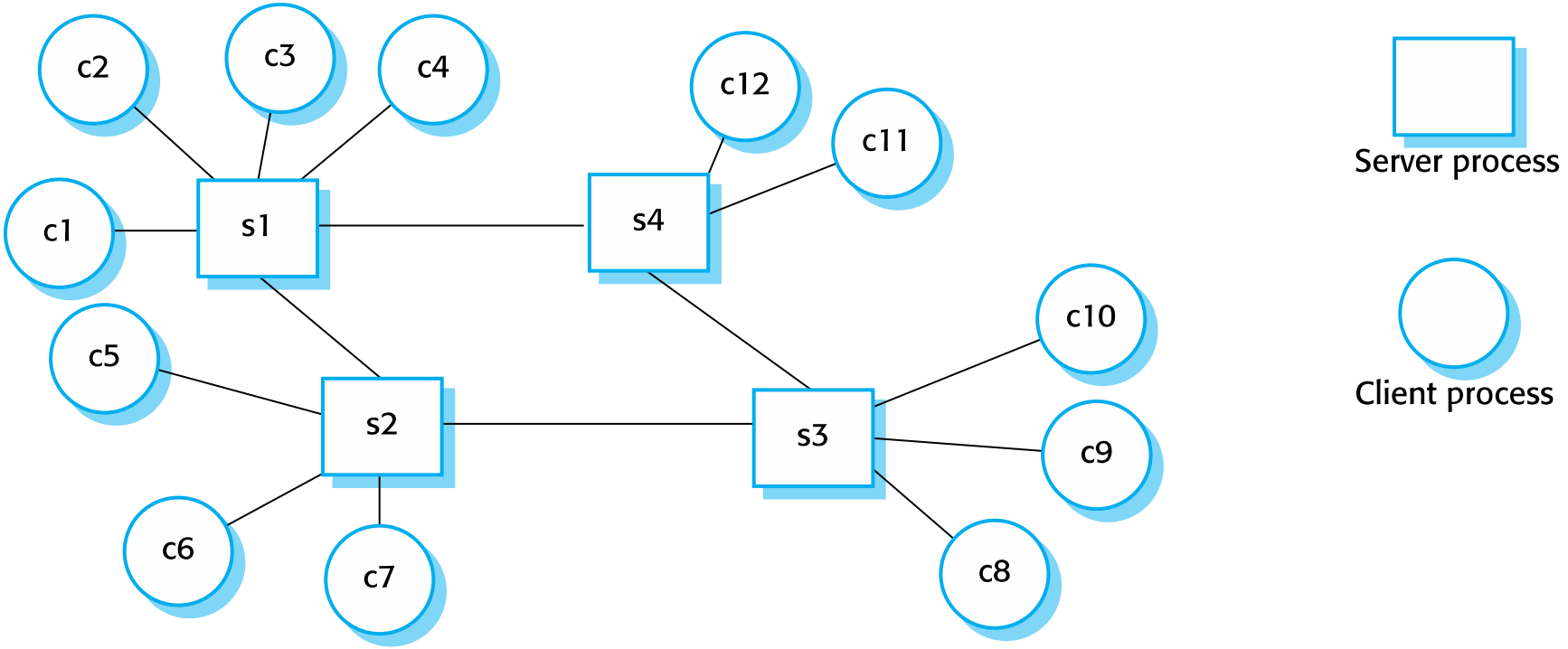
- Lack of Central Authority
- Unpredictable Network Behaviour
- Difficulty in Understanding Emergent Properties



- Scalability
- Parallel processing
- Fault recovery

# 17.2 Client–Server Computing

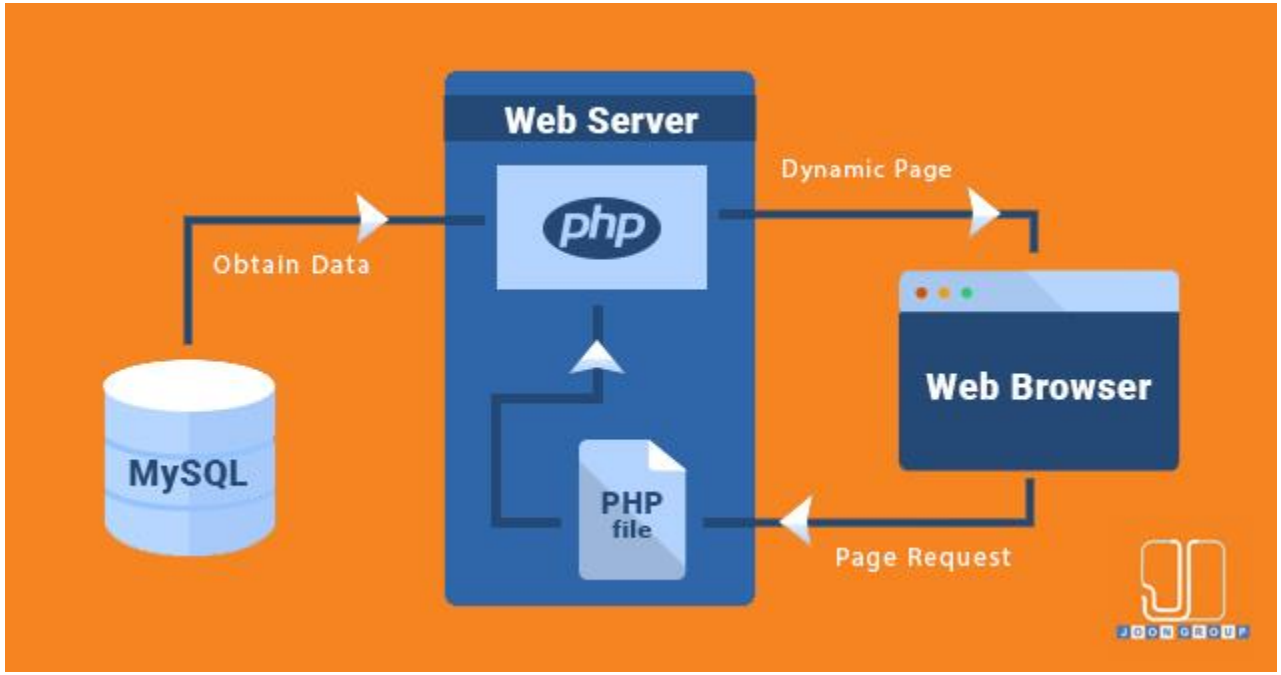
- Clients request services from centralized servers
- Ensures structured communication and data management
- Examples: Web applications, cloud services, online banking



# 17.3 Architectural Patterns

- **Master-Slave** Real-time systems needing predictable processing and localized control. **'Master'** manages; **'slaves'** execute. Ex, Factory automation a master controller coordinating robotic arms
- **Two-Tier** Simple client-server. **Thin-client**: easy management, can overload server. **Fat-client**: distributes processing, complicates client management. Ex, Basic online survey with browser UI
- **Multi-Tier** Separates concerns, improves scalability. Complex systems needing flexibility. Ex, E-commerce site (Amazon) with web, application, and database servers.
- **Distributed Component** Flexible service allocation, dynamic reconfiguration. Components are easily added/moved. Ex, Microservices architecture with independent, scalable services
- **Peer-to-Peer** Decentralized systems where nodes share resources/tasks. Efficient for data/computation sharing. Ex, File-sharing systems (BitTorrent)

# Middleware: Connecting Distributed Components




Database middleware acts as an intermediary, allowing a web server (like PHP) to communicate with a database server (like MySQL). The PHP script uses the middleware to connect to the database, send SQL queries to retrieve or modify data, and receive the database server's response. The middleware handles the connection, query formatting, data transfer, and response handling, simplifying database interactions for the PHP script. Sources and related content

# Scaling & Security in DS

- Vertical Scaling: Adding more power to individual machines.
- Horizontal Scaling: Adding more nodes for distributed processing.
- Key Challenge: Maintaining performance as demand grows.

Interception: Unauthorized data access.

- Interruption: System disruption and DoS attacks.
- Modification: Data integrity compromised.
- Fabrication: False transactions introduced.

Failure Management Strategies:

Redundancy - Load balancing – Self-Healing

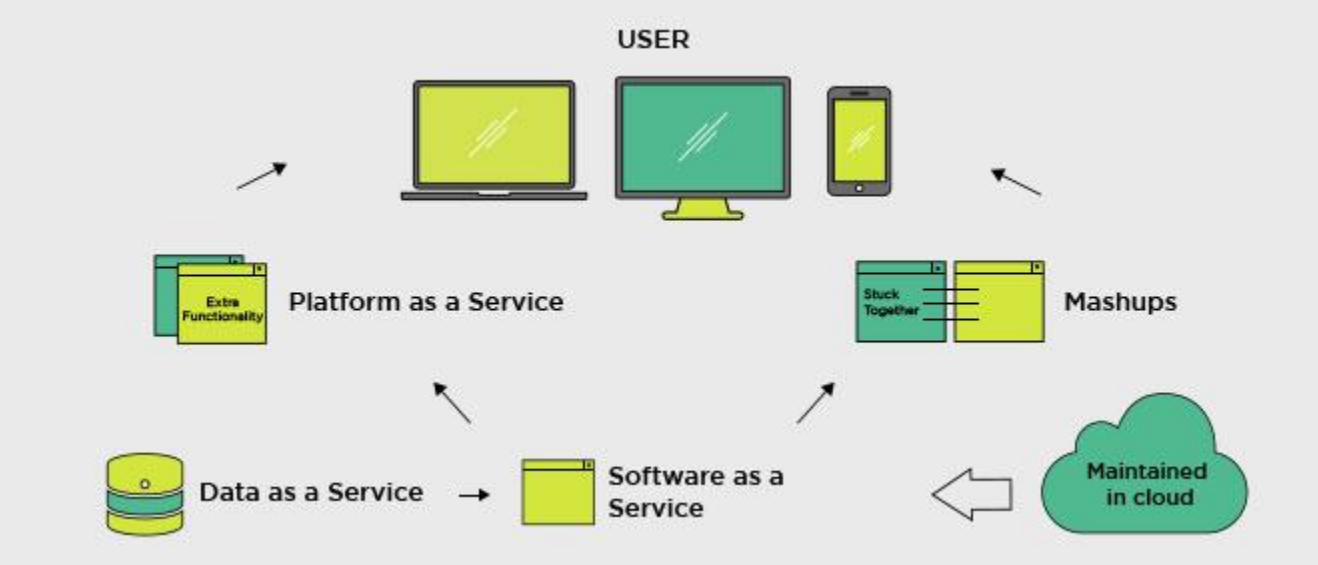
# 17.4 Software as a Service (SaaS)

- Software as a service (or SaaS) is a way of delivering applications over the Internet—as a service. Instead of installing and maintaining software, you simply access it via the Internet, freeing yourself from complex software and hardware management


- Benefits: Scalable, cost-effective, low maintenance
- Examples: Gmail, Office 365, Google Docs

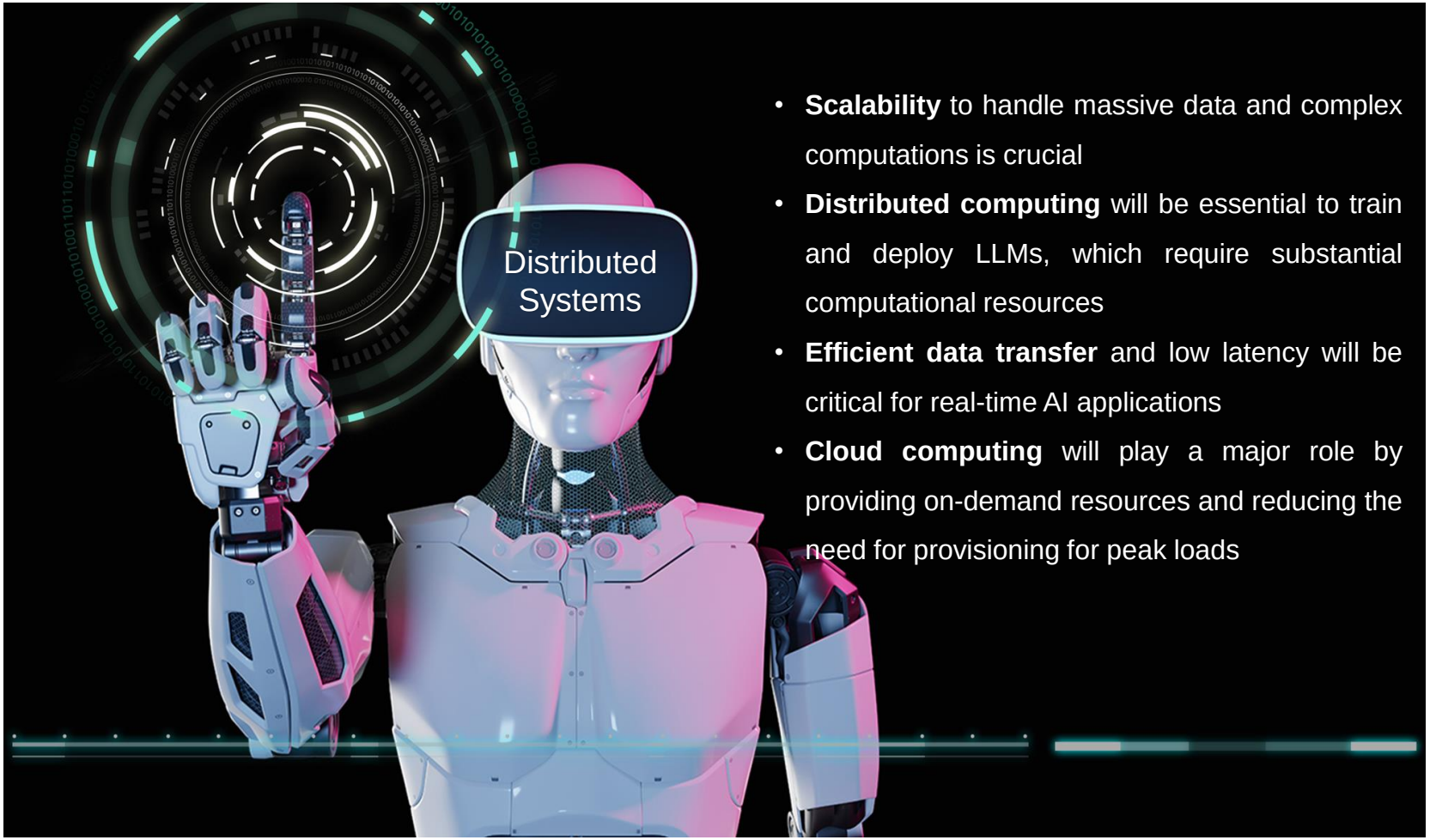
# Service-Oriented Architecture (SOA)

- Service-oriented architecture is an approach to structuring a software system as a set of separate, stateless services. These may be provided by multiple providers and may be distributed.




- Examples: CRM, ERP, PIM

# DS and the AI Revolution




- **Scalability** to handle massive data and complex computations is crucial
- **Distributed computing** will be essential to train and deploy LLMs, which require substantial computational resources
- **Efficient data transfer** and low latency will be critical for real-time AI applications
- **Cloud computing** will play a major role by providing on-demand resources and reducing the need for provisioning for peak loads

# Ex; Amazon Multi-tier client-server architecture and distributed component architecture

# Your Challenge

- How to show this in your project – Iteration 2
  - Design a distributed system considering scalability, security, and reliability
  - Justify architectural choices and middleware selection