NATIONAL INSTITUTE OF TRANSPORT

DEPARTMENT OF COMPUTING AND COMMUNICATION TECHNOLOGY

PROGRAM : HIGHER DIPLOMA IN INFORMATION TECHNOLOGY

MODULE NAME : DISTRIBUTED SYSTEMS

MODULE CODE : ITU 08106

LECTURE’S NAME : SARAH FLORENCE

ACADEMIC YEAR : 2025/2026

GROUP NUMBER : 7

TASK : GROUP ASSGINMENT 02

S/N	NAME	REG NUMBER	SIGNATURE
1	COLLETHA PAULO	NIT/BIT/2023/2089	
2	WAHABI BUKURU	NIT/BIT/2023/2115	
3	FATUMA YUSUPH	NIT/BIT/2023/2224	
4	JEREMIAH MASABURI	NIT/BIT/2023/2041	
5	ASTERIA MWALYENGA	NIT/BIT/2023/2046	
6	AYUBU MSOKOLO	NIT/BIT/2023/2225	
7	FLORIDA MAYALLA	NIT/BIT/2023/2208	
8	GEOFREY KANDEA	NIT/BIT/2023/2243	
9	EMANUEL DAKHO	NIT/BIT/2023/2158	
10	SHADRACK MWASAMYA	NIT/BIT/2023/2199	
11	DIDAS MALISAWA	NIT/BIT/2023/2240	
12	AVINUS R ADABERT	NIT/BIT/2023/2351	
13	NASIBU ATHUMANI	NIT/BIT/2023/2273	
14	FRANK GEOFREY	NIT/BIT/2023/2328	
15	REVOCATUS KAJANA	NIT/BIT/2023/2330	
16	WILIAM MVUNGI	NIT/BIT/2023/2226	

1. A university wants students to share large research datasets directly without using a central server. Describe how a peer-to-peer system can be designed to support this requirement.

A peer-to-peer system is a distributed system where all computers have equal roles and can directly share resources without relying on a central server.

The following is how a peer-to-peer system can be designed to support this requirement.
Each student's computer acts as both a client and a server, allowing it to store and share research datasets locally. The system uses decentralized peer discovery so that students' machines can locate each other without a central coordinator. Large datasets are split into smaller chunks so that different parts of the data can be downloaded from multiple peers at the same time. This improves download speed and reduces network congestion. Data replication across several peers ensures that datasets remain available even when some students disconnect. As more students join the system, the overall sharing capacity increases naturally.

Example.
A university research department uses a BitTorrent-based academic file-sharing system where postgraduate students exchange large climate or genomic datasets directly from their laptops, enabling fast access without overloading a central university server.

2. A music-sharing application allows users to download files from multiple peers at the same time. Explain how network programming supports this P2P architecture.

Network programming enables computers to communicate and exchange data over a network using protocols such as TCP or UDP.

The following is how network programming supports this P2P architecture.
The application establishes multiple network connections between peers using sockets. Each connection is responsible for transferring a portion of the music file from a different peer. Network programming **allows the use of multithreading or asynchronous communication** so downloads can occur simultaneously. Communication protocols ensure reliable data transfer and error handling. Data integrity is maintained by verifying received file pieces. This approach maximizes bandwidth usage and reduces download time.

Example.
A music-sharing application similar to BitTorrent, where a user downloads different parts of a song from several users at once, resulting in faster and more reliable file transfers.

3. A disaster recovery team uses laptops to exchange data when the internet is unavailable. How would a peer-to-peer system be designed for this scenario?

A peer-to-peer system allows devices to communicate directly without relying on centralized infrastructure or internet connectivity.

The following is how a peer-to-peer system can be designed for this scenario.
The laptops form a local ad-hoc wireless network using Wi-Fi or Bluetooth. Each laptop

discovers nearby peers automatically and establishes direct connections. Data such as maps, reports, and images are shared directly between devices. Files are replicated across multiple laptops to prevent data loss. The system continues functioning even if some devices disconnect. Security mechanisms ensure that only authorized team members exchange data.

Example.
Emergency response teams sharing rescue plans and victim data directly between laptops using Wi-Fi Direct during a natural disaster.

4. A company wants to reduce server load by allowing clients to share updates among themselves. Describe a suitable P2P design approach.

A peer-to-peer design distributes workload among clients, reducing dependency on a central server.

The following is how a suitable P2P design approach can be implemented.
The central server initially provides updates to a small number of clients. These clients cache the updates locally. Other clients request updates from nearby peers instead of the server. Peer discovery mechanisms help clients locate update sources. This reduces repeated requests to the server. As more clients participate, the system becomes more efficient and scalable.

Example:
A corporate software update system where employee computers share application updates locally instead of all downloading directly from the company server.

5. In a P2P chat application, peers frequently join and leave the network. Explain how the system maintains communication.

A dynamic peer-to-peer system is designed to handle frequent changes in network membership.

The following is how the system maintains communication.
The chat application uses peer discovery protocols to identify active users. Routing information is updated whenever peers join or leave. Heartbeat messages are exchanged to detect available peers. Messages are forwarded through currently active connections. Temporary disconnections are tolerated without system failure. Synchronization mechanisms ensure message consistency across peers.

Example.
A decentralized chat application where users can freely join or exit chat rooms while messages continue flowing among active participants.

6. A banking system uses multiple servers to process customer transactions. Explain how Java RMI can be used to invoke remote services.

Java RMI allows a program to call methods on objects located on remote machines as if they were local.

The following is how Java RMI can be used to invoke remote services.
Banking servers expose transaction services as remote objects. Clients locate these services using the RMI registry. When a client invokes a method, the request is sent over the network to the remote server. The server processes the transaction securely. Results are returned to the client transparently. This allows distributed transaction processing across multiple servers.

Example:
A banking application where a customer's account balance is retrieved from a remote transaction server using Java RMI calls.

7. A logistics company exposes shipment data through a Python-based web service. Describe how distributed applications can consume this service.

A web service allows applications to access remote functionality over a network using standard communication protocols.

The following is how distributed applications can consume this service.
The logistics company publishes shipment data through a RESTful API developed in Python. Distributed client applications send HTTP requests to the service endpoints. Requests include parameters such as shipment IDs or tracking numbers. The web service processes the requests and retrieves the required data. Responses are returned in standard formats like JSON. Client applications parse the responses and present shipment information to users. This allows seamless access to shared data across different systems.

Example.
Multiple logistics partner applications accessing real-time shipment status from a central Python-based tracking service using REST APIs.

8. An online learning platform uses different programming languages across servers. Explain how web services enable interoperability.

Web services enable communication between systems developed using different programming languages and platforms.

The following is how web services enable interoperability.
Each server exposes its functionality through standardized web service interfaces. Communication is done using common protocols such as HTTP. Data is exchanged in platform-independent formats like JSON or XML. Clients do not need to know the internal implementation details of servers. This allows systems written in different languages to work together. Changes in one service do not affect others as long as interfaces remain consistent.

Example.
A Java-based learning portal accessing course content from a Python server and assessment results from a PHP server using web services.

9. A healthcare system requires remote access to patient records. Describe how RMI or Python web services support this functionality.

Remote service technologies allow secure access to data stored on distant systems.

The following is how RMI or Python web services support this functionality.
The healthcare system exposes patient record services remotely. Authorized users authenticate before accessing data. Client applications send requests to remote servers. The server processes requests and retrieves patient information. Results are returned securely over the network. Access control and logging ensure data privacy and compliance.

Example.
Doctors accessing patient medical records from a hospital server using a secure Python REST API or Java RMI-based system.

10. A weather monitoring system collects data from remote stations. Explain how web services help integrate the distributed components.

Web services provide a standardized way to integrate distributed system components.

The following is how web services help integrate the distributed components.
Remote weather stations collect temperature and humidity data. Each station sends data to a central system using web service calls. **Standard data formats** ensure consistency across stations. The central server aggregates and analyses incoming data. Results are made available to forecasting applications. The system can easily scale as new stations are added.

Example.
National meteorological departments integrating data from multiple remote weather sensors using RESTful web services.

11. Multiple processors share a single database and must avoid conflicts. Explain how clock synchronization helps manage resource sharing.

Clock synchronization ensures that all processors in a distributed system have a consistent sense of time.

The following is how clock synchronization helps manage resource sharing.
Synchronized clocks allow processors to timestamp database operations consistently. These timestamps help determine the order of read and write operations. Conflicting access requests can be detected and resolved fairly. Transaction ordering becomes predictable across processors. Deadlocks and race conditions are reduced. Overall data consistency is maintained across the system.

Example.
A distributed database system where synchronized clocks ensure correct ordering of simultaneous transaction updates.

Say “Continue” and I’ll proceed with Questions 12–16, then keep going until 50 without changing the format.

continue

Continuing with Questions 12–16 in the same format:

12. A distributed printing system schedules print jobs from different computers. Describe how synchronized clocks improve fairness.

Synchronized clocks in a distributed system ensure that all nodes agree on the timing of events.

The following is how synchronized clocks improve fairness.
Each print request is timestamped using a synchronized clock. The print scheduler orders jobs based on these timestamps. Conflicts between simultaneous requests are resolved in chronological order. Users experience predictable wait times for their jobs. It prevents any computer from monopolizing the printer. Synchronized clocks also simplify logging and monitoring of print tasks. Overall system coordination becomes more efficient and fairer.

Example.
A university computer lab where multiple students send print jobs from different computers, and a synchronized system ensures that jobs are processed in the order they were submitted.

13. A cloud computing platform allocates CPU time to parallel tasks. Explain the role of clock synchronization.

Clock synchronization ensures consistent timing across multiple nodes in a distributed system.

The following is how clock synchronization plays a role in allocating CPU time.
Each node uses synchronized clocks to schedule tasks accurately. Parallel tasks are coordinated to prevent overlaps and resource conflicts. Time-based scheduling ensures fair CPU allocation across different tasks. Task dependencies are respected according to timestamps. Performance monitoring and logging are simplified. Faults caused by timing discrepancies are minimized.

Example.
A cloud platform running parallel video rendering tasks across multiple servers, where synchronized clocks ensure balanced CPU allocation and accurate task execution.

14. A financial trading system needs accurate timestamps across servers. How does clock synchronization improve reliability?

Clock synchronization ensures that all servers have a consistent and accurate sense of time.

The following is how clock synchronization improves reliability.
Transactions are timestamped consistently across servers. Trade orders are executed in the correct sequence. Conflicting operations are resolved based on accurate timing. Auditing and

compliance reporting become reliable. Market data is synchronized in real time. System reliability increases because time discrepancies do not cause errors or disputes.

Example.
A stock exchange system where trades executed on multiple servers are recorded accurately with synchronized timestamps to maintain market integrity.

15. A distributed simulation runs tasks in parallel. Explain how synchronized clocks coordinate processing.

Synchronized clocks provide a common reference for time in a distributed system.

The following is how synchronized clocks coordinate processing.
Each simulation task uses timestamps to determine the order of execution. Dependencies between tasks are respected to maintain correctness. Tasks across different processors can communicate results at the right times. Deadlocks or race conditions caused by timing differences are reduced. Coordinated logging allows accurate tracking of events. Overall, simulation steps occur in proper sequence.

Example.
A physics simulation where particle interactions are computed across multiple processors, and synchronized clocks ensure consistent sequence of events.

16. A distributed chat system must display messages in the correct order. Explain how logical clocks help achieve this.

Logical clocks assign a sequence number to events rather than relying on real-time clocks.

The following is how logical clocks help achieve correct message ordering.
Each message sent is assigned a logical timestamp. Receiving peers use these timestamps to display messages in the correct order. Causality between messages is preserved. Messages that arrive out of order can be reordered using logical timestamps. Conflicts in display order are resolved systematically. This ensures coherent conversation flow across all users. Synchronization does not depend on physical clocks, making it robust in distributed environments.

Example.
A decentralized messaging app where users in different time zones see messages displayed in the order they were sent, even if network delays occur.

17. In a collaborative document editor, multiple users edit the same file. Describe how vector clocks detect conflicting updates.

Vector clocks are a logical clock mechanism that assigns a timestamp vector to track events across multiple nodes in a distributed system.

The following is how vector clocks detect conflicting updates.

Each user's edits are tagged with a vector clock that records the number of operations performed by each participant. When updates are received, the system compares vector clocks to determine the causal relationship between edits. If updates are concurrent, a conflict is detected and flagged for resolution. Sequential edits are automatically merged according to their vector timestamps. This ensures consistency while preserving all user contributions. The system can maintain a complete history of changes for auditing or rollback.

Example.
Google Docs or a collaborative code editor where multiple users can edit the same document simultaneously, and vector clocks ensure conflicting changes are identified and resolved.

18. A transaction processing system needs to determine which event happened first. Explain the use of logical clocks.

Logical clocks provide a method to order events in a distributed system without relying on synchronized physical clocks.

The following is how logical clocks are used in this scenario.

Each transaction is assigned a logical timestamp when it occurs. The system compares timestamps to determine the order of events. Concurrent transactions are identified when timestamps are incomparable. Event ordering ensures proper sequencing for database operations. This prevents inconsistencies caused by simultaneous operations. Logical clocks also simplify debugging and auditing of transaction flows.

Example.
A distributed banking system that processes multiple account transactions on separate servers, using logical clocks to determine which transaction should be applied first to maintain consistency.

19. A distributed debugging tool tracks event causality. How do vector clocks assist in this scenario?

Vector clocks track the partial ordering of events across multiple nodes in a distributed system.

The following is how vector clocks assist in debugging.
Each process in the system maintains a vector clock that increments with every event. When messages are sent between processes, vector clocks are updated to reflect causality. The debugging tool can then reconstruct the exact sequence of dependent events. Conflicting or concurrent events are clearly identified. This enables developers to pinpoint the source of errors or unexpected behaviours. Causal relationships are preserved across distributed logs.
Example.
A distributed system debugger that uses vector clocks to analyse and reconstruct event sequences in a multi-server application to detect race conditions.

20. A social media platform processes likes and comments across servers. Explain how event ordering is maintained.

Event ordering in distributed systems ensures that operations are applied in a consistent sequence across all nodes.

The following is how event ordering is maintained.
Each action, such as a like or comment, is timestamped using logical or vector clocks. Servers exchange these timestamps to determine causal relationships. Concurrent actions are handled by conflict-resolution policies. Updates are applied in a consistent order across all nodes. Users see a coherent and chronological sequence of interactions. This prevents confusion and maintains data integrity in the platform.
Example.
A social media application like Facebook where likes and comments from users in different regions appear in the correct order on everyone's news feed.

21. An e-commerce system transmits sensitive customer data. Explain how encryption secures distributed communication.

Encryption is the process of converting data into a secure format that can only be read by authorized parties.

The following is how encryption secures distributed communication.
Customer data is encrypted before it is transmitted over the network. Only the intended recipient with the decryption key can access the information. Communication protocols such as TLS ensure data integrity and confidentiality. Even if intercepted, encrypted data cannot be understood. Encryption protects sensitive details like credit card numbers and personal information. It also helps prevent tampering during transmission. Secure key management ensures ongoing protection.
Example.
An online shopping website using HTTPS to encrypt credit card and personal data between customer browsers and the backend servers.

22. A distributed voting system must prevent vote tampering. Describe the role of cryptography.

Cryptography is the practice of securing information through encryption, ensuring confidentiality, integrity, and authenticity.

The following is how cryptography prevents vote tampering.
Votes are encrypted before being submitted to the system, making them unreadable to unauthorized parties. Digital signatures authenticate voters and verify that each vote is legitimate. The system can detect any unauthorized modification of votes during transmission or storage. Secure hashing ensures that recorded votes cannot be altered without detection. Encryption protects both the vote content and the voter's identity. This provides a reliable and tamper-proof voting process.

Example.
An online election platform where each voter's ballot is encrypted and digitally signed, ensuring votes are secure, authentic, and verifiable.

23. A file-sharing system requires authentication of users. Explain how security techniques ensure authorized access.

Authentication is the process of verifying the identity of users before granting system access.

The following is how security techniques ensure authorized access.
Users provide credentials such as usernames and passwords or use multi-factor authentication. The system verifies these credentials against stored records. Access control policies determine which resources each user can access. Unauthorized users are denied access to sensitive files. Session tokens or certificates maintain authenticated sessions. Logging and monitoring detect suspicious access attempts. These measures prevent misuse of the file-sharing system.
Example.
A corporate document repository requiring employees to log in with secure passwords and one-time codes before accessing shared company files.

24. A cloud-based payroll system stores confidential data. Describe how cryptography protects stored and transmitted data.

Cryptography secures data by transforming it into a format that can only be accessed by authorized users.

The following is how cryptography protects payroll data.
Sensitive employee information is encrypted before storage in cloud databases. Data transmitted between clients and servers is encrypted using protocols like TLS. Only users with appropriate decryption keys can access the information. Hashing ensures that stored records are not altered without detection. Encryption safeguards against unauthorized access and data breaches. Key management policies maintain long-term security. Overall, cryptography ensures confidentiality and integrity of payroll information.
Example.
A cloud payroll service encrypting employee salaries and tax information, ensuring that only HR personnel can read or modify the data.

25. A distributed system is vulnerable to man-in-the-middle attacks. Explain security measures to prevent this.

Man-in-the-middle attacks occur when an attacker intercepts communication between two parties without their knowledge.

The following is how security measures prevent these attacks.
Encrypted communication using TLS ensures that intercepted messages cannot be read. Digital certificates authenticate servers and clients to each other. Strong key exchange protocols prevent unauthorized access to encryption keys. Regularly updating software closes vulnerabilities exploited by attackers. Monitoring network traffic helps detect suspicious activity. Security

measures maintain data integrity and confidentiality during transmission. Together, these techniques make man-in-the-middle attacks ineffective.

Example.

Online banking applications using HTTPS with verified server certificates to protect customer transactions from interception.

26. A server processes payroll, backups, and reporting tasks simultaneously. Explain how OS services manage multiple jobs.

Operating system services provide mechanisms to manage and coordinate multiple processes running on a computer.

The following is how OS services manage multiple jobs.

The OS schedules CPU time for each task to ensure all processes progress efficiently. It allocates memory and I/O resources to prevent conflicts. Multitasking allows tasks such as payroll processing, backups, and report generation to run concurrently. Priority levels can ensure critical tasks complete first. The OS handles synchronization and communication between processes. It monitors resource usage to prevent overload and maintain system stability. Overall, OS services enable smooth and efficient execution of multiple jobs.

Example.

A payroll server that calculates employee salaries, backs up data, and generates monthly financial reports at the same time without performance degradation.

27. A research lab runs simulations in parallel. Describe how the operating system schedules these jobs.

The **operating system** manages multiple processes to ensure efficient use of computational resources.

The following is how the operating system schedules these jobs.

Each simulation task is treated as a separate process by the OS. The scheduler allocates CPU time to tasks based on priorities and scheduling policies such as round-robin or priority-based scheduling. Memory and I/O resources are distributed to prevent conflicts and ensure smooth execution. The OS monitors task progress and switches between tasks to maximize processor utilization. Dependencies between tasks are respected to avoid race conditions. Overall, scheduling ensures parallel simulations run efficiently without overloading the system.

Example.

A physics research lab running multiple particle simulations simultaneously, where the OS schedules each simulation process to utilize all available CPU cores effectively.

28. A web server handles many client requests at once. Explain how OS multitasking supports this.

Multitasking allows an operating system to manage multiple processes simultaneously.

The following is how OS multitasking supports this.

The web server spawns separate processes or threads for each client request. The OS allocates CPU time to handle these processes concurrently. Input/output operations, such as reading files or accessing databases, are managed efficiently. Context switching ensures no single request blocks others from execution. Resource allocation prevents conflicts between requests. Multitasking allows the server to maintain high responsiveness even under heavy load.

Example.
A web server hosting an online retail site processing thousands of user requests simultaneously without delays due to OS-managed multitasking.

29. A data centre executes batch jobs overnight. How does the OS manage job execution?

The **operating system** coordinates the execution of multiple scheduled jobs efficiently.

The following is how the OS manages job execution.
Batch jobs are queued and scheduled based on priority or arrival time. The OS allocates CPU, memory, and I/O resources for each job to prevent interference. Jobs may be executed in parallel if multiple processors are available. The OS handles synchronization and ensures that dependencies between jobs are respected. Logging and monitoring track job completion and errors. Resource usage is optimized to ensure all jobs complete successfully overnight.
Example.
A data centre performing nightly backup, report generation, and analytics batch jobs, all coordinated by the OS scheduler.

30. A student computer lab runs multiple applications concurrently. Explain the OS role in resource allocation.

The **operating system** manages hardware and software resources to allow multiple applications to run smoothly.

The following is how the OS allocates resources.
CPU time is distributed among applications using scheduling algorithms. Memory is assigned to each application to prevent overlap or conflicts. I/O devices such as printers and network interfaces are shared efficiently. The OS monitors application performance and prevents any single program from monopolizing resources. Synchronization ensures coordinated access to shared files or databases. Overall, the OS maintains stability and responsiveness in a multi-user environment.
Example.
A university lab where students simultaneously use word processors, web browsers, and programming IDEs without interference, thanks to OS resource management.

31. A computing cluster uses job submission libraries. Explain how these libraries support distributed processing.

Job submission libraries provide tools and APIs to submit, monitor, and manage tasks on a distributed cluster.

The following is how these libraries support distributed processing.
Users submit jobs using standardized library calls instead of directly interacting with each node. The library handles job distribution across cluster nodes based on availability and resource requirements. It tracks the status of each job and collects results upon completion. Dependencies between tasks are managed automatically. Resource allocation is optimized for load balancing. Error handling and retries ensure jobs complete reliably across the cluster.
Example.
An HPC research centre using a Python-based job submission library to distribute climate simulation tasks across multiple cluster nodes efficiently.

32. A weather forecasting system submits analysis jobs automatically. Describe how OS utilities assist.

Operating system utilities provide tools for automating job submission and system management in distributed or local environments.

The following is how OS utilities assist.
Scheduled tasks or cron jobs are used to automatically submit analysis scripts at predefined times. The OS monitors the execution of these tasks and handles resource allocation. Logging utilities track progress and detect errors. Utilities manage file transfers between nodes if data is distributed. Alerts can be generated for failed jobs to prompt corrective action. Overall, OS utilities simplify repetitive job management and ensure consistent execution.
Example.
A meteorological centre automatically running weather pattern analysis scripts overnight using cron jobs and logging utilities on a Linux server.

33. A research institution uses scripts to submit multiple jobs. Explain how libraries simplify job management.

Job submission libraries provide standardized functions to submit, monitor, and manage computational tasks on distributed systems.

The following is how libraries simplify job management.
Researchers write scripts that call library functions to submit jobs instead of manually interacting with cluster nodes. The library handles job distribution, prioritization, and tracking. Dependencies between jobs are resolved automatically. Status updates and results are collected centrally. Error handling and retries are performed transparently. This reduces administrative overhead and minimizes the risk of manual mistakes.
Example.
A university HPC lab where Python scripts using a job library submit genome sequencing analysis tasks to hundreds of cluster nodes efficiently.

34. A cloud platform allows users to submit workloads remotely. How do OS utilities support this?

Operating system utilities provide mechanisms for managing remote execution and resource control on cloud platforms.

The following is how OS utilities support remote workloads.
Command-line tools or APIs allow users to submit jobs to cloud servers from any location. The OS schedules and allocates CPU, memory, and storage resources. Job monitoring and logging utilities track progress and detect errors. Utilities manage input and output data transfers between the user and cloud storage. Notifications alert users of job completion or failures. This ensures reliable and efficient remote execution of workloads.
Example.
A data analytics company submitting large machine learning tasks to cloud servers using OS-managed scheduling and monitoring tools.

35. A university HPC system supports job queues. Explain the role of software libraries.

Software libraries provide APIs and functions to interact with job queue systems for scheduling and managing tasks efficiently.

The following is how software libraries support job queues.
Users submit tasks to the queue using library calls rather than manually. The library communicates with the scheduler to assign jobs to available compute nodes. Job dependencies and priorities are managed automatically. Status updates, completion notifications, and result collection are simplified. Error handling ensures failed jobs are retried. Overall, libraries streamline queue management and reduce user complexity.
Example.
A university HPC cluster using a Python library to submit physics simulations to a SLURM-managed job queue, ensuring fair distribution across nodes.

36. A parallel image-processing application splits tasks across processors. Explain how processors communicate.

Processors in parallel systems communicate to exchange data and coordinate task execution efficiently.

The following is how processors communicate in this scenario.
Each processor works on a portion of the image and exchanges intermediate results with others. Communication occurs through shared memory or message-passing protocols. Synchronization mechanisms ensure that processors wait for required data before continuing. Data consistency is maintained to prevent errors in final image output. Communication overhead is minimized to optimize performance. Proper coordination enables parallel processing to speed up image analysis tasks.
Example.
A medical imaging application that processes large MRI scans across multiple CPU cores, with processors exchanging pixel data to reconstruct the complete image.

37. A distributed database synchronizes updates across CPUs. Describe processor inter-communication.

Processor inter-communication allows multiple CPUs to exchange data and coordinate operations in a distributed system.

The following is how processor inter-communication works in this scenario.
Each CPU performs updates on its local portion of the database. Updates are shared with other CPUs using message passing or shared memory. Synchronization mechanisms ensure that all changes are applied in the correct order. Conflicts are detected and resolved to maintain data consistency. Timely communication reduces latency and prevents data divergence. Overall, inter-processor communication allows the distributed database to behave as a coherent system.
Example.
A bank's distributed database where transactions from multiple ATMs are synchronized across servers to maintain consistent account balances.

38. A scientific application shares intermediate results between processors. Explain how communication is managed.

Communication between processors enables the sharing of data required for parallel computation tasks.

The following is how communication is managed.
Each processor computes its part of the problem and sends intermediate results to other processors. Message-passing protocols or shared memory interfaces are used to transmit data. Synchronization ensures processors receive data in the correct order before proceeding. Buffering techniques prevent data loss during high traffic. Communication overhead is minimized to maintain performance. Proper coordination allows accurate and efficient computation across all processors.
Example.
A climate simulation where temperature and wind calculations are exchanged between processors to compute global weather patterns.

39. A multiplayer game server distributes tasks across cores. How do processors coordinate?

Processors in a parallel system coordinate to ensure consistent execution of shared tasks and data.

The following is how processors coordinate in this scenario.
The server divides game logic, rendering, and physics calculations across multiple cores. Processors exchange information about player actions and game state. Synchronization mechanisms prevent race conditions and ensure consistent updates. Inter-process communication ensures all cores are aware of critical events in real time. Timing and load balancing optimize performance. Coordination allows the game to run smoothly for multiple players simultaneously.
Example.
An online multiplayer game like Fortnite, where server cores process movement, combat, and environment updates concurrently while maintaining consistency for all players.

40. A parallel search algorithm requires frequent data exchange. Explain inter-processor communication.

Inter-processor communication allows processors to share intermediate results and coordinate their search tasks efficiently.

The following is how inter-processor communication works in this scenario.
Each processor explores part of the search space independently. Results or partial solutions are exchanged with other processors. Synchronization ensures that all processors have the latest information before proceeding. Message passing or shared memory is used for fast data exchange. Communication overhead is minimized to maintain high search efficiency. This enables the algorithm to find solutions faster by leveraging multiple processors concurrently.
Example.
A distributed database search where multiple servers exchange index data to quickly locate relevant records.

41. A client application locates services using DNS. Explain how name services support this process.

Name services provide a mapping between human-readable names and network addresses to locate resources in a distributed system.

The following is how name services support this process.
The client queries the DNS system with a domain name. DNS servers translate the name into the corresponding IP address. The client uses this address to establish a connection with the desired service. Name resolution ensures that services can be accessed reliably without requiring users to know IP addresses. Caching reduces lookup times for frequently accessed services. Name services simplify communication in large and distributed networks.
Example.
A web browser looking up "www.example.com" through DNS to connect to the correct web server hosting the site.

42. A Java RMI application uses an RMI Registry. Describe how it helps clients locate remote objects.

An RMI Registry is a directory service that allows Java clients to look up and connect to remote objects in a distributed system.

The following is how the RMI Registry helps clients locate remote objects.
Servers register their remote objects with the RMI Registry using a unique name. Clients query the registry to obtain a reference to the desired remote object. Once located, clients can invoke methods on the remote object as if it were local. The registry abstracts the network details, providing location transparency. It ensures that clients can reliably find and connect to the services they need. The system can dynamically add or remove remote objects without affecting clients.
Example.
A banking application where client applications use the RMI Registry to locate transaction servers and invoke account operations remotely.

43. An organization uses LDAP to manage user accounts. Explain its role in distributed systems.

LDAP (Lightweight Directory Access Protocol) is a protocol for accessing and managing directory services across a network.

The following is how LDAP supports distributed systems.
LDAP stores information about users, groups, and resources in a centralized directory. Distributed applications query the LDAP server to authenticate users and retrieve account details. Access control policies are enforced consistently across all systems. LDAP provides a standardized interface, allowing different applications and platforms to integrate seamlessly. Replication ensures directory availability across multiple servers. Overall, LDAP centralizes user management while supporting distributed access.
Example.
A university IT system using LDAP to authenticate students and staff across multiple services such as email, learning platforms, and library systems.

44. A distributed file system allows users to access files transparently. Explain how naming is handled.

Naming in distributed file systems maps human-readable file names to locations on multiple servers, ensuring seamless access.

The following is how naming is handled.
Each file has a unique name that is independent of its physical location. The system maintains a directory service that resolves file names to storage nodes. Users can access files without knowing where they are stored. Updates to file locations are automatically reflected in the naming service. This provides location transparency and simplifies user interaction. Consistent naming ensures all users see the same file structure.
Example.
A corporate distributed file system where employees access documents using familiar folder structures, while files are stored across multiple servers transparently.

45. A cloud service relies on name resolution for scalability. Describe the importance of name services.

Name services provide a mechanism to locate resources efficiently in large-scale distributed environments.

The following is the importance of name services.
They allow clients to resolve human-readable service names to network addresses. This enables dynamic scaling, as new servers can be added without changing client configurations. Load balancing can redirect requests to available servers. Name resolution ensures reliable connectivity even as nodes are added or removed. It simplifies service discovery and enhances system flexibility. Overall, name services are critical for scalable, distributed cloud applications.
Example.
A global cloud storage service using DNS-based name resolution to distribute client requests across multiple data centres.

46. A client submits tasks to a server for parallel execution. Explain how client–server architecture supports this.

Client–server architecture separates the roles of requesters and providers in a distributed system.

The following is how client–server architecture supports parallel execution.
The client submits tasks to the server, which manages scheduling and resource allocation. The server distributes subtasks across multiple processors or worker nodes. Results from parallel processing are aggregated and returned to the client. Communication protocols ensure reliability and data integrity. Centralized control simplifies task monitoring and error handling. This architecture allows clients to leverage server processing power without managing parallel execution themselves.
Example.
A financial analytics application where clients submit complex risk calculation tasks to a central server that executes them in parallel across multiple cores.

47. A web application processes requests using multiple backend servers. Describe the parallel processing design.

Parallel processing in distributed systems allows multiple tasks to be executed simultaneously to improve performance and responsiveness.

The following is how the parallel processing design works in this scenario.
The web application distributes incoming client requests across multiple backend servers using a load balancer. Each server processes its assigned requests independently while sharing data as needed. Requests are executed concurrently to reduce overall response time. Data consistency is maintained through synchronization mechanisms or shared databases. Monitoring tools track server performance and ensure balanced workload distribution. This design allows the system to handle high traffic efficiently.
Example.
An e-commerce platform like Amazon using multiple backend servers to process product searches, shopping cart updates, and payment requests simultaneously for thousands of users.

48. A student registration system uses a centralized server. Explain the client–server interaction.

Client–server architecture separates client applications that request services from a central server that provides them.

The following is how client–server interaction works in this scenario.
Students access a registration interface on their client devices. The client sends requests to the centralized server for course registration, availability checks, or schedule updates. The server processes these requests, updates the database, and returns responses. This ensures that all student data is consistent and centrally managed. Communication is handled using standard network protocols. The server manages multiple client requests concurrently.
Example.
A university registration portal where thousands of students simultaneously enrol in courses via a central server that handles requests and updates schedules.

49. A data analytics system distributes workloads from clients to servers. Explain the architecture.

Distributed client–server architecture enables clients to offload computationally intensive tasks to powerful servers.

The following is how the architecture works in this scenario.
Clients submit analytics jobs to a central or distributed server cluster. Servers divide tasks into smaller units and execute them in parallel. Results are aggregated and returned to the clients. Load balancing ensures efficient use of server resources. Communication protocols ensure data integrity and secure transmission. Centralized control allows monitoring, fault tolerance, and optimized processing.
Example.
A business intelligence system where analysts submit large sales data analysis jobs to a cluster of servers that process them concurrently and return aggregated results.

50. A banking system processes transactions from many clients simultaneously. Describe how client–server architecture ensures efficiency.

Client–server architecture separates transaction requests from processing resources, enabling high concurrency and reliability.

The following is how client–server architecture ensures efficiency.
Client applications submit transaction requests to centralized banking servers. The servers handle multiple requests in parallel using multithreading or distributed processing. Load balancing ensures that no single server becomes a bottleneck. Transaction logs and databases are synchronized to maintain consistency. Security mechanisms protect sensitive financial data during communication. Centralized control simplifies monitoring and error recovery while allowing clients to perform banking operations efficiently.
Example.
An online banking system where thousands of customers perform deposits, transfers, and bill payments simultaneously, and the server cluster efficiently processes all transactions without delays.