RMI (Remote Method Invocation) is a Java-based technology that allows communication and method invocation between objects in a distributed system. RMI enables developers to create distributed applications where objects on different machines can interact with each other as if they were local objects.

1. Remote Objects: In RMI, objects that can be accessed and invoked remotely are referred to as remote objects. These objects implement remote interfaces that extend the **java.rmi.Remote** interface. Remote interfaces define the methods that can be invoked remotely.

Stub and Skeleton: RMI uses stub and skeleton objects to facilitate communication between the client and the server. The client-side stub acts as a proxy for the remote object and forwards method invocations to the server-side skeleton, which is responsible for dispatching the requests to the actual remote object

Registry: RMI provides a registry service that acts as a central repository for remote object references. The registry allows clients to locate and obtain references to remote objects by specifying their names or other identifiers.

CORBA (Common Object Request Broker Architecture) is a middleware technology and architecture that enables interoperability between distributed objects across different platforms and programming languages. It provides a standardized and language-neutral approach for communication and interaction among distributed components in a networked environment.

Key components and concepts in CORBA include:

1. Object Request Broker (ORB): The ORB is responsible for managing communication and interaction between client and server objects in a CORBA system. It handles requests, marshaling and unmarshaling of parameters, and routing of messages between objects.
2. Interface Definition Language (IDL): IDL is a language used to define the interfaces of CORBA objects. It provides a language-neutral way to describe the methods and attributes of objects, allowing different programming languages to generate language-specific bindings.
3. Object Adapter: The Object Adapter acts as a bridge between the ORB and the objects in the server application. It receives requests from the ORB, dispatches them to the appropriate objects, and marshals and unmarshals data as required.
4. Stub and Skeleton: A Stub represents a client-side proxy object that encapsulates the details of remote method invocations. It communicates with the ORB and transfers method invocations to the server-side Skeleton, which is responsible for handling and executing the actual method calls.

BERKELEY

The goal of the Berkeley algorithm is to ensure that all participating nodes in a distributed system have a consistent notion of time, even if their local clocks are initially unsynchronized or subject to drift. The algorithm works by electing a time coordinator or master node, which is responsible for determining the system time and distributing it to other nodes.

1. Time Synchronization: Each node adjusts its local clock to match the received time value from the time coordinator. The adjustment can be made by advancing or delaying the local clock.
2. Iteration: The above steps are repeated at regular intervals to maintain time synchronization in the distributed system. Nodes continue to send time requests, and the time coordinator calculates and distributes the adjusted time value.

**MPI :**

MPI (Message Passing Interface) is a communication protocol and programming model used in parallel and distributed computing. It enables communication and coordination among multiple processes running on different nodes or processors in a cluster or distributed system.

1. Message Passing: MPI provides a set of functions for sending and receiving messages between processes. Processes can exchange data, instructions, or synchronization signals through point-to-point communication or collective communication operations.
2. Process Coordination: MPI supports process management and coordination functionalities. It allows processes to be dynamically created, terminated, and controlled. It provides mechanisms for process synchronization, collective operations, and process group management.
3. Data Distribution: MPI enables efficient distribution and partitioning of data across multiple processes. It includes data distribution and collection operations such as scatter, gather, broadcast, reduce, and all-to-all communication.
4. Parallel Computing: MPI is widely used in parallel computing scenarios to harness the power of multiple processors or nodes. It allows for parallelization of computations across processes, enabling the execution of complex algorithms and simulations.

**Tokenring :**

Token Ring, in the context of Distributed Systems (DS), refers to a distributed algorithm or protocol that uses a token-based approach for coordination and communication among processes in a distributed system. The concept of Token Ring in distributed systems is similar to the Token Ring network topology in computer networks, but with a different focus and application.

In a distributed system, the Token Ring protocol is used to achieve mutual exclusion or resource access coordination among distributed processes. The token acts as a special message or token object that circulates among the processes in a ring-like fashion. Only the process holding the token is allowed to access or perform certain **operations on shared resources or critical sections of code.**

**BULLY AND RING ALGORITHM :**

The Bully Algorithm and the Ring Algorithm are both leader election algorithms used in distributed systems to elect a leader or coordinator among a group of processes. These algorithms ensure that a single process takes up the role of the leader, responsible for making decisions or coordinating activities within the distributed system.

1. Bully Algorithm:
   * The Bully Algorithm assumes that each process in the system has a unique identifier or process ID.
   * When a process detects the absence of a leader or wishes to initiate a leader election, it sends an election message to all processes with higher IDs.
   * If a process receives an election message, it responds by sending an OK message back to the requesting process.
   * If a process does not receive an OK message within a specified timeout period, it concludes that it has the highest ID and declares itself as the leader.
   * If a process receives an OK message, it acknowledges the presence of a higher ID process and does not participate further in the leader election process.
   * The process that eventually declares itself as the leader sends a coordinator message to inform other processes about its leadership.
2. Ring Algorithm:
   * The Ring Algorithm is based on a logical ring topology, where processes are organized in a ring structure.
   * Each process has a unique identifier or process ID, and the processes are arranged in a predetermined order around the ring.
   * When a process initiates a leader election, it passes a token or a message around the ring.
   * Each process that receives the token compares its own ID with the ID in the token. If it finds that its ID is higher, it updates the token with its ID and continues passing it along the ring.
   * The token continues to circulate around the ring until it reaches the process with the highest ID, which then becomes the leader.
   * The leader sends a coordinator message to inform other processes about its leadership.