UNIT 5: Naming
UNIT 6: Coordination

Yuba Raj Devkota | NCCS

Distributed System

Unit 5. Naming

5.1 Name Identifiers, and Addresses

5.2 Structured Naming

Unit 5. Coordination **7 Hrs.**

6.1 Clock Synchronization

6.2 Logical Clocks

6.3 Mutual Exclusion

6.4 Election Algorithm

6.5 Location System

6.6 Distributed Event Matching

6.7 Gossip-based coordination

5.3 Attribute-based naming

5.4 Case Study: The Global Name Service

Naming Identifiers & Addresses

In distributed systems, the concepts of naming identifiers and addresses are crucial for ensuring that resources can be located and accessed correctly. Here's an overview of these concepts:

Naming Identifiers in Distributed System

- Identifiers in distributed systems are unique names assigned to resources or entities within the system. These resources could be anything from files, services, devices, nodes, or even users. Identifiers ensure that each resource can be uniquely recognized, avoiding conflicts and enabling precise resource management.

Key properties of identifiers include:

- Uniqueness: Each identifier must be unique within its scope.
- Persistence: Identifiers typically remain constant over time, even if the resource they refer to moves or changes state.
- Transparency: Identifiers abstract the physical location of resources, providing a logical name that can be used throughout the system.

Examples of identifiers include:

- UUIDs (Universally Unique Identifiers): Standardized 128-bit identifiers used in many distributed systems.
- Handles: Unique references to resources, often used in file systems and databases.
- URNs (Uniform Resource Names): Persistent, location-independent identifiers for resources.

Addresses in Distributed System

- Addresses in distributed systems refer to the actual locations where resources can be found. Unlike identifiers, which are abstract and persistent, addresses are concrete and may change over time. An address provides the necessary information to locate a resource within the network.

Key properties of addresses include:

- Direct Location: Addresses provide a way to directly access the resource.
- Possibility of Change: Addresses can change if the resource is moved or the network topology changes.
- Network Specificity: Addresses are often specific to the network protocol in use (e.g., IP addresses for TCP/IP networks).

Examples of addresses include:

- IP Addresses: Network addresses used in the Internet Protocol (IP).
- MAC Addresses: Hardware addresses used in local network communications.
- URLs (Uniform Resource Locators): Web addresses used to locate resources on the World Wide Web.

Differences and Relationship

- **Identifiers vs. Addresses:** Identifiers are unique and persistent labels used to unambiguously refer to resources, while addresses are actual network locations that can change over time. Identifiers abstract away the physical location and are often used to maintain a consistent reference to resources even if their addresses change.
- **Mapping:** In distributed systems, there is often a need to map identifiers to addresses. This mapping is managed through various resolution services, such as DNS (Domain Name System), which translates domain names (identifiers) to IP addresses.

Example in Distributed Systems

Consider a distributed file system:

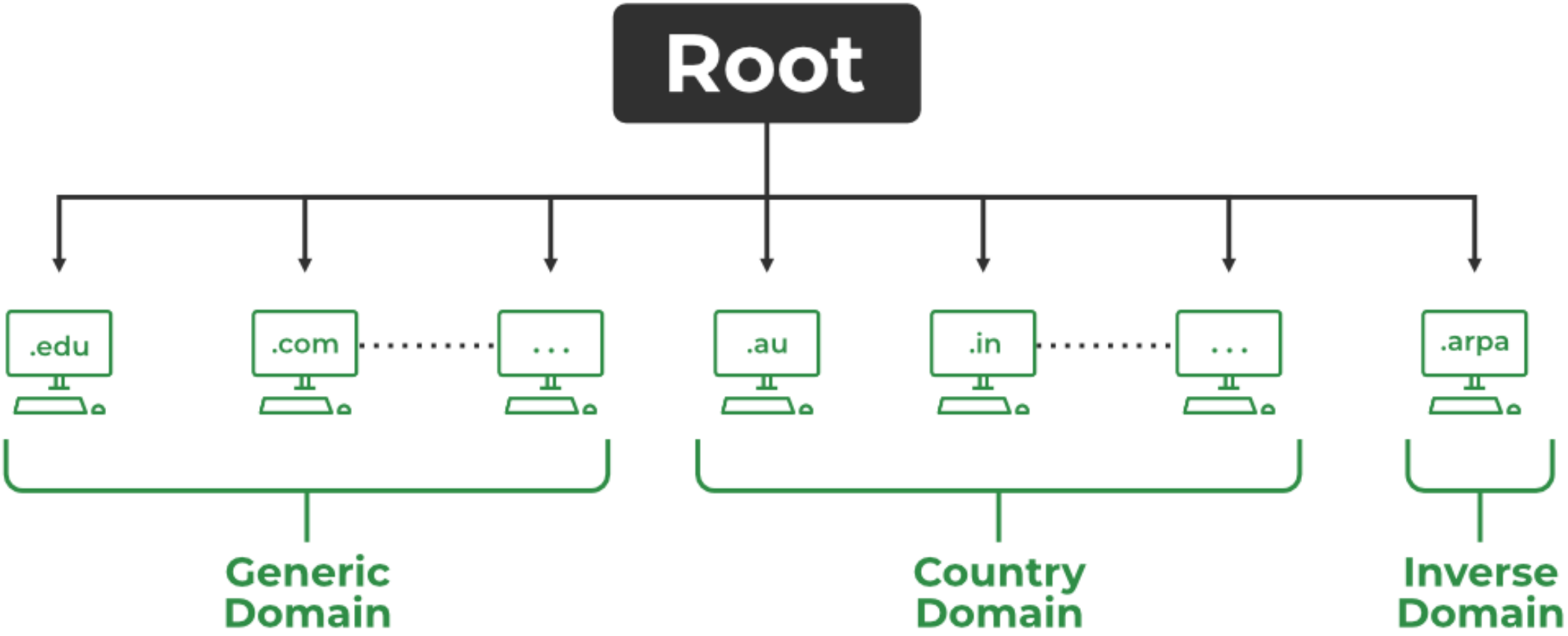
- Identifier: Each file is given a unique identifier (e.g., a file handle or a UUID) that clients use to refer to the file.
- Address: The actual storage location of the file may be an IP address and a path on a particular server.
- When a client requests a file, it uses the identifier. The system then resolves this identifier to the current address of the file, which could involve consulting a directory service or a metadata server that knows the mapping from identifiers to addresses.

Summary (Identifier & Address)

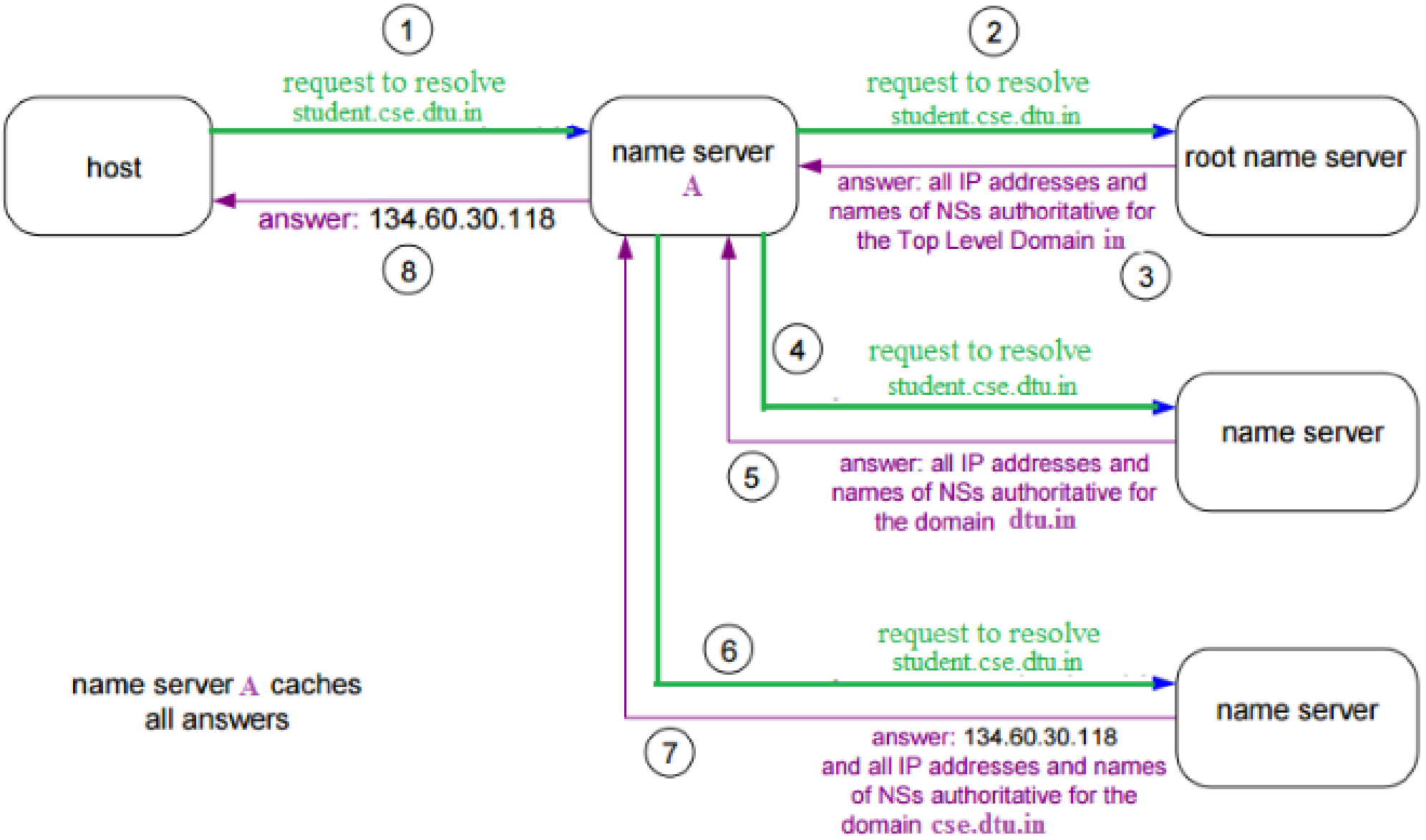
- Identifiers: Unique, persistent names for resources.
- Addresses: Specific locations of resources, which may change.
- Mapping: Systems use services like DNS to map identifiers to addresses, enabling location transparency and resource mobility.
- Understanding and managing these concepts is essential for the design and operation of effective distributed systems, ensuring resources are accessible and locatable despite the complexities of the network.

Domain Name System (DNS)

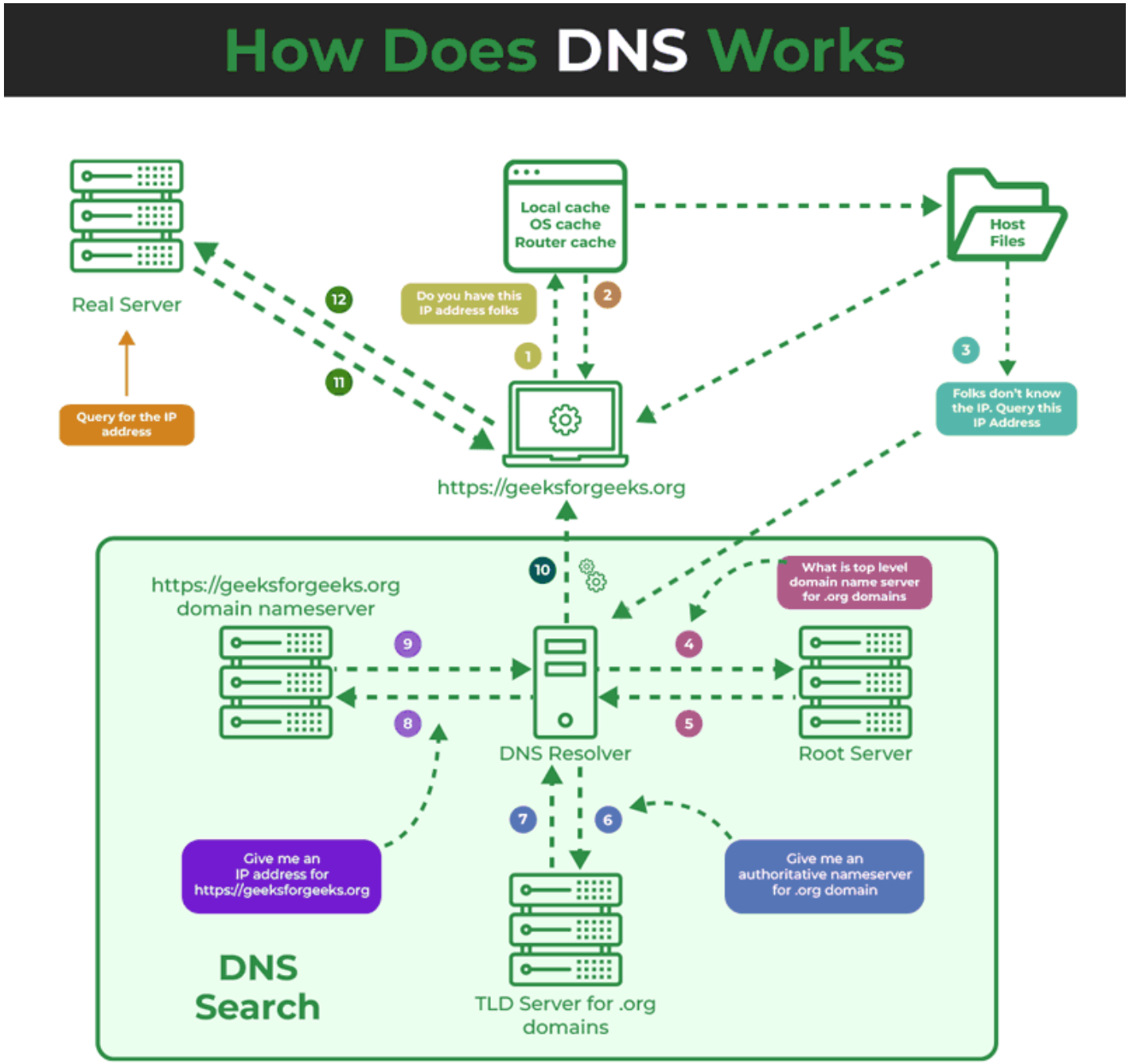
- The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.
- Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).
- DNS Lookup is a browser-based network tool used to find the IP address of a certain domain name. It displays DNS records showing publicly for the domain name being queried.
- DNS Lookup allows you to use public DNS servers such as **Google, Cloudflare, Quad9, OpenDNS**, etc. Specify name server, Top-level domain name server, root name server, Authoritative name server, and other DNS servers for the query. Therefore, if you changed your web hosting or DNS records, those changes will reflect instantly. These DNS server IP addresses support **IPv4** and **IPv6**.


Root
.edu
.com
. . .
.au
.in
. . .
.arpa
Generic Domain
Country Domain
Inverse Domain

Types of DNS


1
request to resolve
student.cse.dtu.in
host
name server
A
2
request to resolve
student.cse.dtu.in
root name server
answer: all IP addresses and
names of NSs authoritative for
the Top Level Domain in
3
answer: 134.60.30.118
8
4
request to resolve
student.cse.dtu.in
name server
5
answer: all IP addresses and
names of NSs authoritative for
the domain dtu.in
6
request to resolve
student.cse.dtu.in
name server
name server A caches
all answers
7
answer: 134.60.30.118
and all IP addresses and names
of NSs authoritative for the
domain cse.dtu.in

- The working of DNS starts with converting a hostname into an IP Address. A domain name serves as a distinctive identification for a website. It is used in place of an IP address to make it simpler for consumers to visit websites.
- Domain Name System works by executing the database whose work is to store the name of hosts which are available on the Internet.
- The top-level domain server stores address information for top-level domains such as .com and .net, .org, and so on.
- If the Client sends the request, then the DNS resolver sends a request to DNS Server to fetch the IP Address.
- In case, when it does not contain that particular IP Address with a hostname, it forwards the request to another DNS Server. When IP Address has arrived at the resolver, it completes the request over Internet Protocol.



Structured naming

- Structured naming systems use a hierarchical or systematic structure to organize and identify resources.
- This approach relies on a predefined schema or hierarchy, where names are composed of multiple components, each representing a different level or category within the system.
- The structure often resembles a tree, with each node representing a domain or sub-domain, leading to a clear and organized naming convention.

Key Characteristics:

1. **Hierarchy**: Names are organized in a hierarchical manner. For example, in DNS (Domain Name System), names like `www.example.com` represent a hierarchical structure where `com` is the top-level domain, `example` is a sub-domain, and `www` is a specific host within that sub-domain.

2. **Predictability**: Due to its structured nature, the names are predictable and easier to manage. Each level in the hierarchy provides context about the resource.

3. **Delegation**: The hierarchical structure allows for delegation of naming authority. For instance, a company may control the `example.com` domain and can further delegate the naming within this domain (e.g., `hr.example.com`).

4. **Scalability**: It scales well because the hierarchical structure can expand at each level without much difficulty.

5. **Uniqueness**: Ensures unique names within a given namespace, reducing conflicts.

Examples:

- **DNS (Domain Name System)**: Uses a hierarchical structure to manage names of devices and services on the internet.
- **File Systems**: File paths in file systems (e.g., `` `/home/user/documents/file.txt` ``) follow a hierarchical structure.

Attribute-based Naming

Attribute-based naming, also known as content-based or property-based naming, identifies resources based on a set of attributes rather than a hierarchical structure. Each resource is described by a set of key-value pairs, and queries for resources can be based on these attributes.

Key Characteristics:

1. **Flexibility**: Names are not confined to a strict hierarchical structure. Resources can be identified and located based on various attributes.

2. **Expressiveness**: Allows more complex and descriptive queries. For example, you can search for a printer by specifying attributes like `location=buildingA` and `type=color`.

3. **Discovery**: Facilitates resource discovery since users can search for resources based on the properties they need, without knowing the exact name.

4. **Adaptability**: Suitable for dynamic environments where resources frequently change or need to be described by different properties.

5. **Scalability**: Can handle large and diverse sets of resources effectively.

Examples:

- **LDAP (Lightweight Directory Access Protocol)**: Uses attributes to describe and search for directory entries.
- **Peer-to-Peer Networks**: Nodes and resources are often identified by attributes (e.g., file metadata) rather than a hierarchical structure.

Comparison:

- **Structure**: Structured naming follows a predefined hierarchical structure, while attribute-based naming is more flexible and dynamic.
- **Query Mechanism**: Structured naming uses a fixed path to locate resources, whereas attribute-based naming relies on matching attributes to queries.
- **Use Case Suitability**: Structured naming is suitable for static and well-defined systems (like DNS and file systems), while attribute-based naming is ideal for dynamic environments where resource properties are more relevant than their hierarchical position.

Design of Domain Name Spaces

1. **1. Hierarchical Structure:**
   - The domain name space is organized hierarchically. This hierarchy is visually represented as an inverted tree.
   - At the top of the hierarchy is the root domain, represented by a dot (".").

2. **Levels of Domains:**
   - **Top-Level Domains (TLDs):** These are directly under the root domain. Examples include generic TLDs (gTLDs) like .com, .org, .net, and country-code TLDs (ccTLDs) like .us, .uk, .jp.
   - **Second-Level Domains:** These are directly below TLDs and are typically what people register as their domain names, e.g., example.com.
   - **Subdomains:** These are further subdivisions of second-level domains. For instance, blog.example.com or shop.example.com.

3. **Naming Conventions:**
   - Domain names must be unique within their level and are case-insensitive.
   - They can contain letters, numbers, and hyphens but cannot begin or end with a hyphen.

Implementation of Domain Name Spaces

1. **DNS Servers:**
   - **Root Name Servers:** These handle requests for information about TLDs. They know the authoritative servers for each TLD.
   - **TLD Name Servers:** These handle requests for information about second-level domains. They know the authoritative servers for each second-level domain under their TLD.
   - **Authoritative Name Servers:** These provide authoritative answers to queries about domains within their zone.

2. **DNS Resolution Process:**
   - When a user enters a URL, their computer sends a query to a local DNS resolver.
   - If the resolver does not have the information cached, it queries the root name server.
   - The root server directs the resolver to the appropriate TLD server.
   - The TLD server then directs the resolver to the authoritative server for the second-level domain.
   - Finally, the authoritative server provides the IP address associated with the domain name.

Example

Consider the domain www.example.com:

1.Root Domain ("."):

- The query starts at the root name server.

2.Top-Level Domain (".com"):

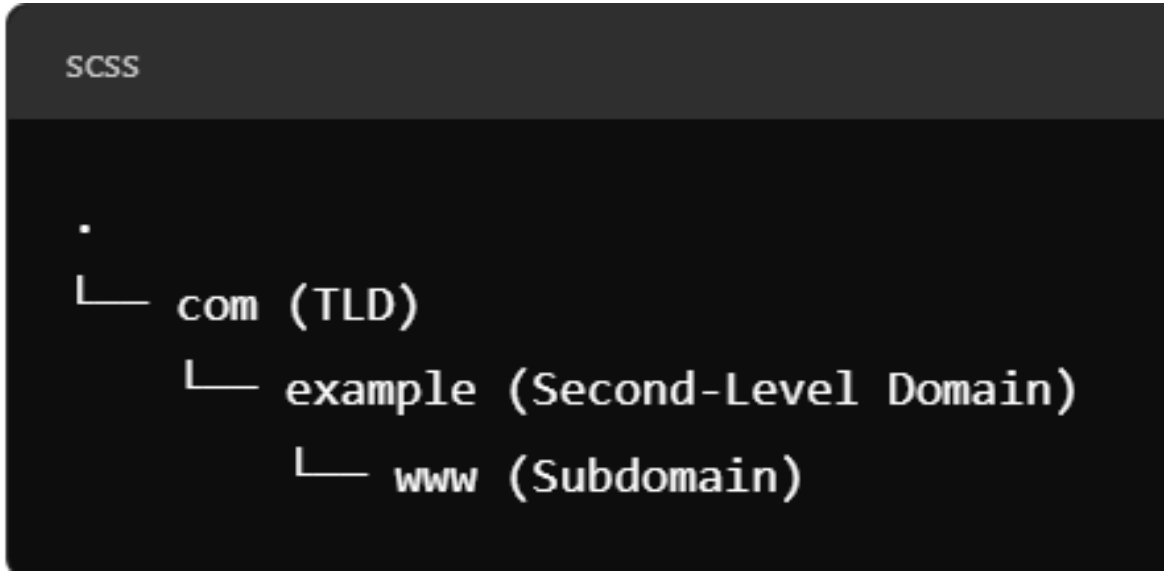
- The root server directs the query to the .com TLD name server.

3.Second-Level Domain ("example"):

- The .com TLD name server directs the query to the authoritative server for example.com.

4.Subdomain ("www"):

- The authoritative server for example.com provides the IP address for www.example.com.



Global Name Service (GNS)

- The Global Name Service (GNS) is a distributed naming service that provides a way to translate human-readable names into network addresses and other resources in a distributed system. It helps manage the naming and addressing of resources in a global network, allowing users and applications to refer to resources using simple, memorable names rather than complex network addresses.
- Example: Consider a large organization with multiple branches around the world. Each branch has its own local network with various resources like printers, servers, and databases. Managing the names and addresses of these resources can be challenging. The GNS can be used to provide a consistent naming scheme across the entire organization.

Scenario:

1. **Naming Convention:** The organization decides on a naming convention where each resource name includes the branch location, the type of resource, and a unique identifier. For example:
   - `nyc-printer-001` for a printer in the New York City branch.
   - `london-server-002` for a server in the London branch.
   - `tokyo-db-003` for a database in the Tokyo branch.

2. **Resolution:** When a user or application wants to access a resource, they use the GNS to resolve the human-readable name to the actual network address. For instance:

   - The name `nyc-printer-001` is resolved to the IP address `192.168.1.10`.
   - The name `london-server-002` is resolved to the IP address `192.168.2.20`.

3. **Consistency and Management:** The GNS ensures that the names are consistent and unique across the organization. It also allows for easy updates and management of the names and addresses. If the IP address of `nyc-printer-001` changes, the GNS can be updated without affecting users who will still use the same name to access the printer.

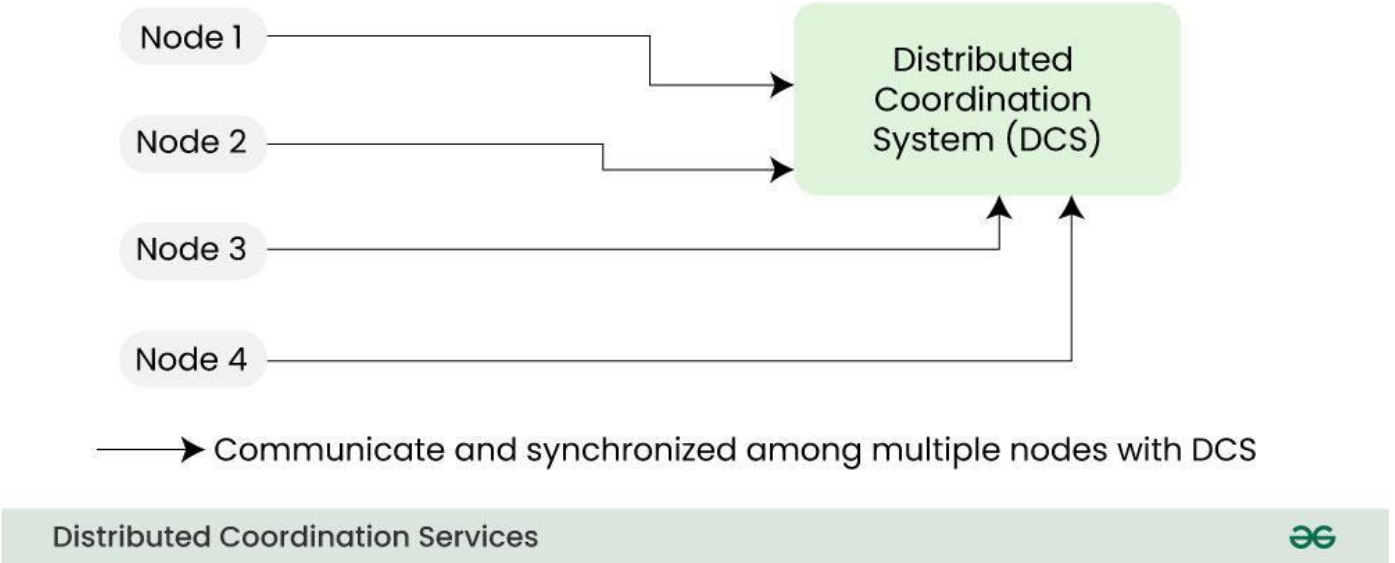
In summary, the Global Name Service simplifies the process of accessing and managing network resources by providing a consistent and human-readable naming system that can be used across a distributed network.

Benefits:

- **Ease of Use:** Users and applications can refer to resources by simple, memorable names instead of complex IP addresses.
- **Scalability:** The GNS can handle a large number of names and resources across multiple locations.
- **Consistency:** Ensures that names are unique and consistently formatted across the entire network.
- **Flexibility:** Allows for easy updates and changes to resource addresses without affecting the names used by users.

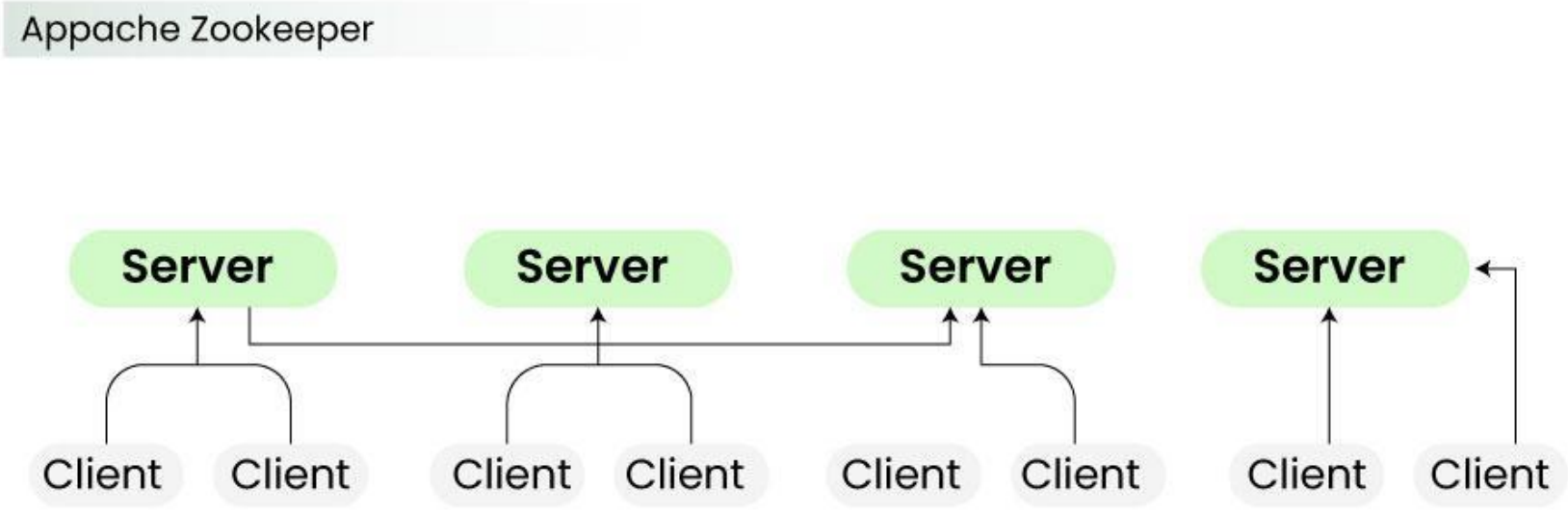
Distributed Coordination services

- *Distributed Coordination Service (DCS) is a service that allows multiple nodes to operate and coordinate in a distributed system.*
- They provide tools to manage all nodes and integrate everything to ensure consistency and prevent errors. The primary goal of a DCS is to ensure that different components or nodes can work together coherently in a distributed environment.
- We can think of Distributed Coordination Services (DCS) as teachers who guide students on projects and ensure collaboration between students. DCS also bootstraps nodes and manages multiple nodes in parallel.

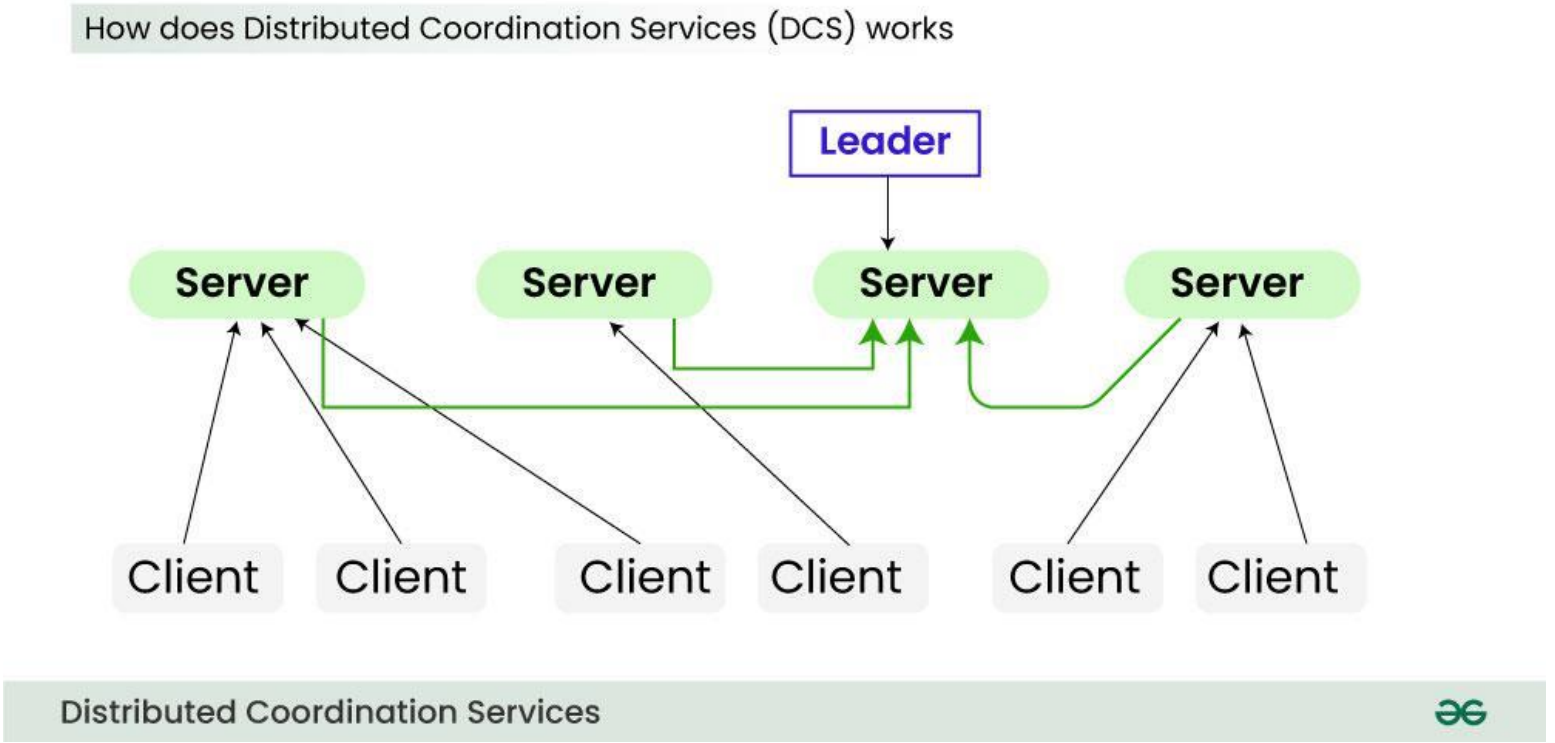


Example of Distributed Coordination System (DCS)

- A popular example of a shared hosting service is **Apache ZooKeeper**. Apache ZooKeeper is an open-source integration service that provides a reliable and powerful way to manage roles in an integrated environment. It is designed to be simple and effective by providing a convenient mechanism that the dispatcher can use to synchronize between nodes.



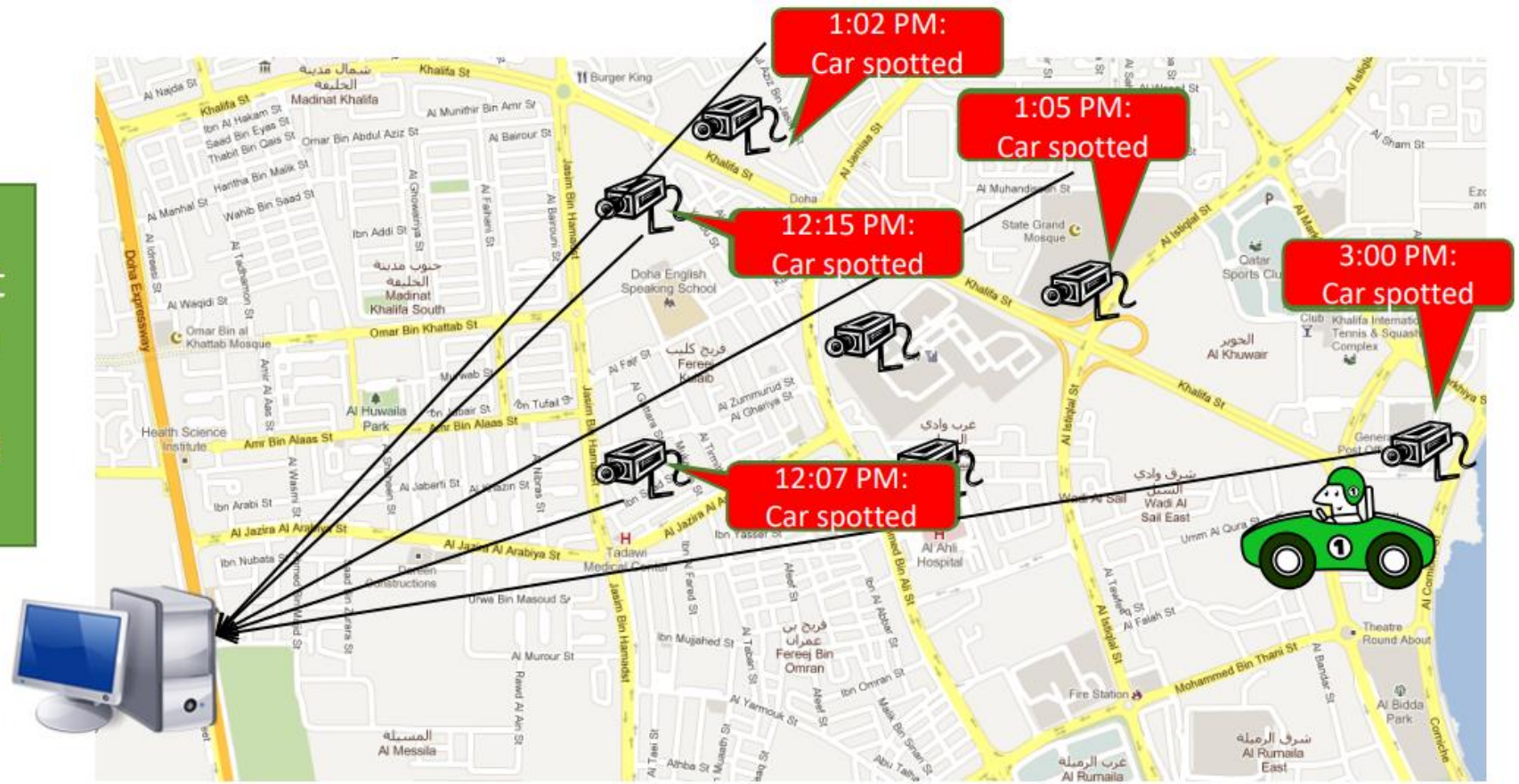
- In this diagram,
- ZooKeeper is widely used in large systems to store configuration information and manage voting managers across different collaborations.
- All servers store a copy of the data in memory.
- Client is connected with only one server, whereas server is connected with other servers as well as other clients.
- In Apache Zookeeper, clients send the request, gets response, watch the events and connection between them is TCP.
- Let suppose if the TCP connection to the server breaks, then client is connected with different servers.



Need for Synchronization – Example 1

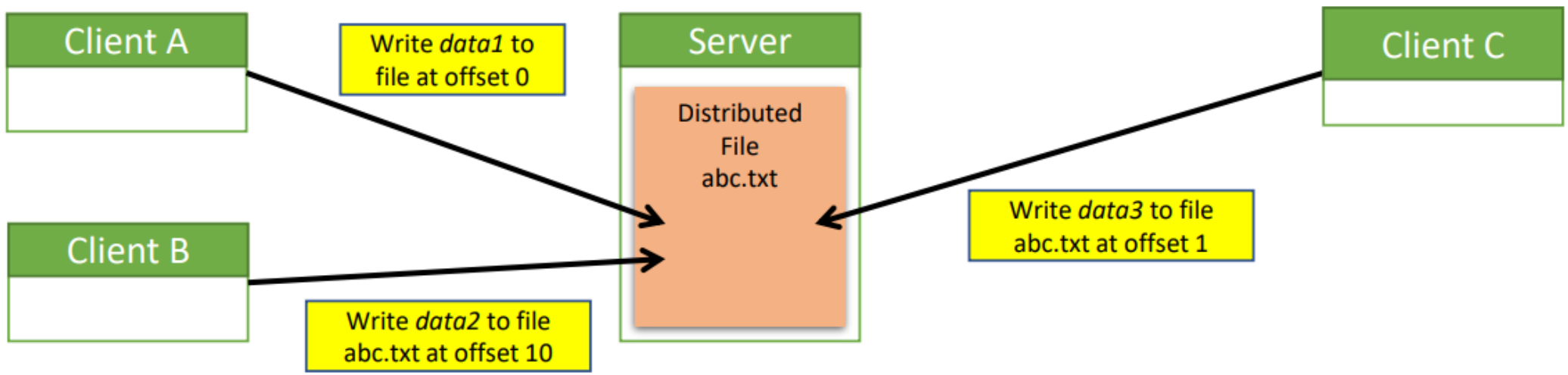
- Vehicle tracking in a City Surveillance System using a Distributed Sensor Network of Cameras
  - *Objective:* To keep track of suspicious vehicles
  - Camera Sensor Nodes are deployed over the city
  - Each Camera Sensor that detects the vehicle reports the time to a central server
  - Server tracks the movement of the suspicious vehicle

If the sensor nodes do not have a consistent version of the time, the vehicle cannot be reliably tracked



Need for Synchronization – Example 2

- Writing a file in a Distributed File System



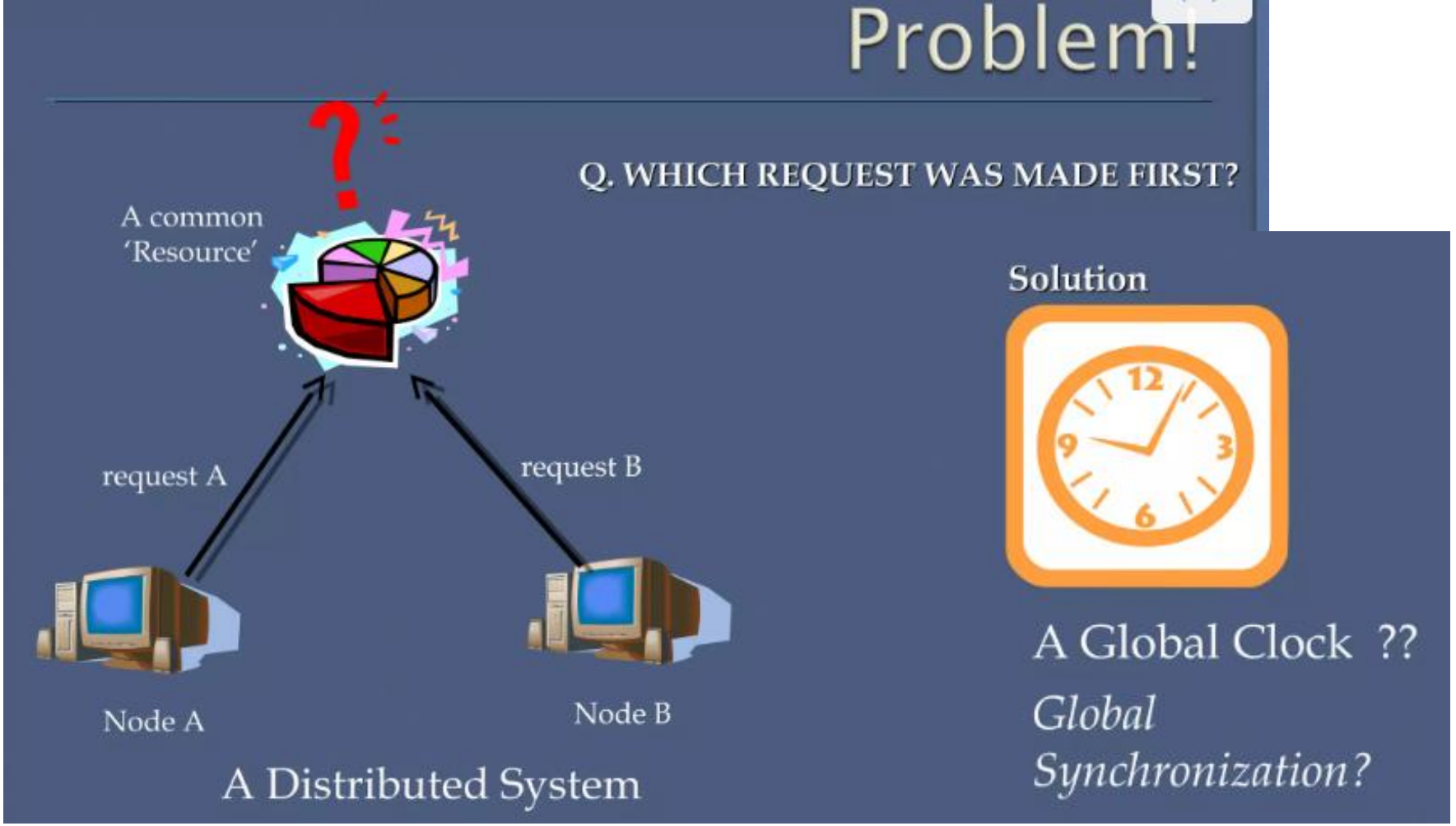
If the distributed clients do not synchronize their write operations to the distributed file, then the data in the file can be corrupted

A Broad Taxonomy of Synchronization

Reason for synchronization and cooperation	Entities have to agree on ordering of events	Entities have to share common resources
Examples	e.g., Vehicle tracking in a Camera Sensor Network, Financial transactions in Distributed eCommerce Systems	e.g., Reading and writing in a Distributed File System
Requirement for entities	Entities should have a common understanding of time across different computers	Entities should coordinate and agree on when and how to access resources
Topics we will study	Time Synchronization	Mutual Exclusion

Clock Synchronization in distributed System

- Clock synchronization is the mechanism to synchronize the time of all the computers in the distributed environments or system.
- Assume that there are three systems present in a distributed environment. To maintain the data i.e. to send, receive and manage the data between the systems with the same time in synchronized manner you need a clock that has to be synchronized. This process to synchronize data is known as Clock Synchronization.
- Synchronization in distributed system is more complicated than in centralized system because of the use of distributed algorithms.
- Properties of Distributed algorithms to maintain Clock synchronization:
  - Relevant and correct information will be scattered among multiple machines.
  - The processes make the decision only on local information.
  - Failure of the single point in the system must be avoided.
  - No common clock or the other precise global time exists.
  - In the distributed systems, the time is ambiguous.
- As the distributed systems has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in distributed environment.



Individual Clocks?

Are individual clocks accurate, precise?

One clock might run faster/slower?

Types of Clock Synchronization

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion synchronization

Physical clock synchronization

- In Physical clock synchronization, All the computers will have their own clocks.
- The physical clocks are needed to adjust the time of nodes. All the nodes in the system can share their local time with all other nodes in the system.
- The time will be set based on UTC (Universal Coordinate Timer).
- The time difference between the two computers is known as “Time drift”. Clock drifts over the time is known as “Skew”. Synchronization is necessary here.
- **Physical clocks**: In physical synchronization, physical clocks are used to time stamp an event on that computer.
- If two events, E1 and E2, having different time stamps t1 and t2, the order of the event occurring will be considered and not on the exact time or the day at which they are occur.
- Several methods are used to attempt the synchronization of the physical clocks in Distributed synchronization:
    1. UTC (Universal coordinate timer)
    2. Christian’s algorithm
    3. Berkely’s algorithm

1. Universal Coordinate Time (UTC)

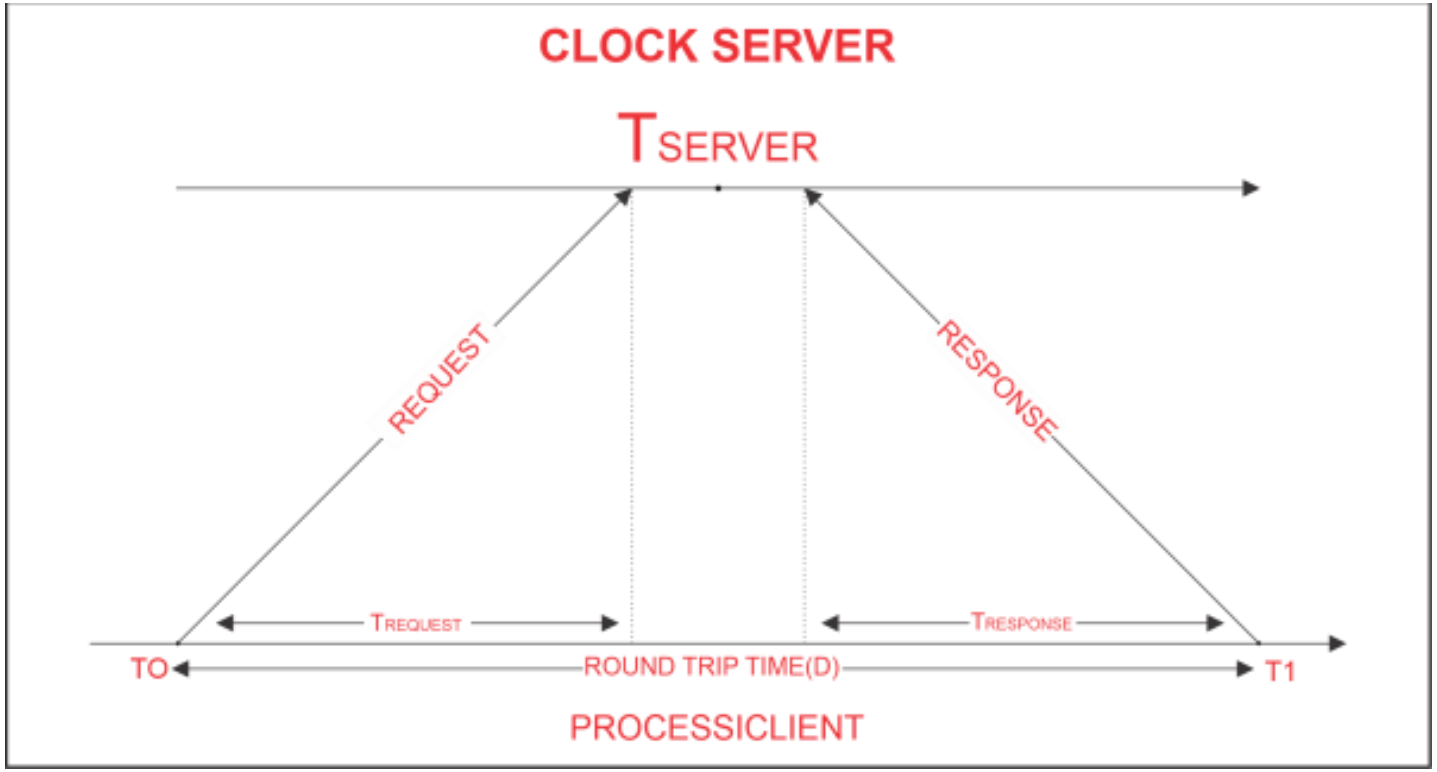
- All the computers are generally synchronized to a standard time called Universal Coordinate Time (UTC).
- UTC is the primary time standard by which the time and the clock are regulated in the world. It is available via radio signals, telephone line and satellites (GPS).
- UTC is broadcasted via the satellites.
- Computer servers and online services with the UTC resources can be synchronized by the satellite broadcast.
- Kept within 0.9 seconds of UTI.

- **UTO -** Mean solar time on Greenwich meridian, Obtained from astronomical observation.
- **UT1 -** UTO corrected for polar motion.
- **UT2 -** UT1 corrected for seasonal variations in earth's rotation.
- **UTC -** Civil time will be measured on an atomic scale.

2. Christian’s Algorithm:

- The simplest algorithm for setting time, it issues a remote procedure call (RPC) to the time sever and obtains the time.
- The machine which send requests to the time server is “d/z” seconds, where d is the maximum difference between the clock and the UTC.
- The time server sends the reply with current UTC when receives the request from the receiver.

Cristian's Algorithm is used for synchronizing the time on a client machine with a time server, assuming that the server's time is accurate. The algorithm accounts for network delay to estimate the correct time.



Suppose the following times are recorded:

- $T_0 = 10:00:00.000$ (Client sends request)
- $T_1 = 10:00:01.500$ (Server receives request and responds with this time T_s)
- $T_2 = 10:00:02.000$ (Client receives the server's response)

Now, let's apply Cristian's Algorithm:

1. **Calculate RTT**:

$$RTT = T_2 - T_0 = 10:00:02.000 - 10:00:00.000 = 2 \text{ seconds}$$

2. **Estimate One-Way Delay**:

$$\delta = \frac{RTT}{2} = \frac{2 \text{ seconds}}{2} = 1 \text{ second}$$

3. **Adjust the Client's Clock**:

$$T_c = T_s + \delta = 10:00:01.500 + 1 \text{ second} = 10:00:02.500$$

Thus, the client's clock is set to 10:00:02.500, which is the server's time adjusted for the estimated network delay. This provides a more accurate synchronization of the client's clock with the server's clock.

3. Berkeley's Algorithm

- Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

Algorithm

1) An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.

2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.

3) Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.

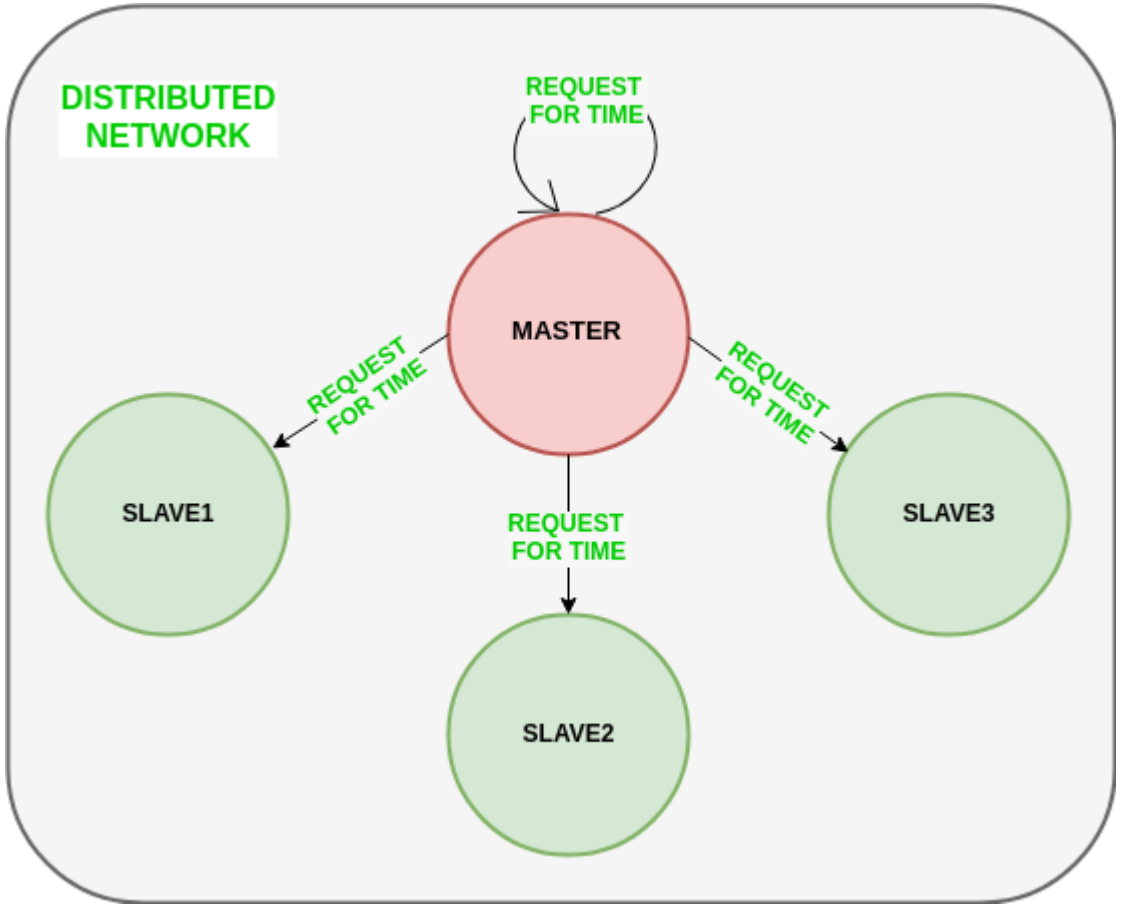


Figure illustrates how the master sends requests to slave nodes

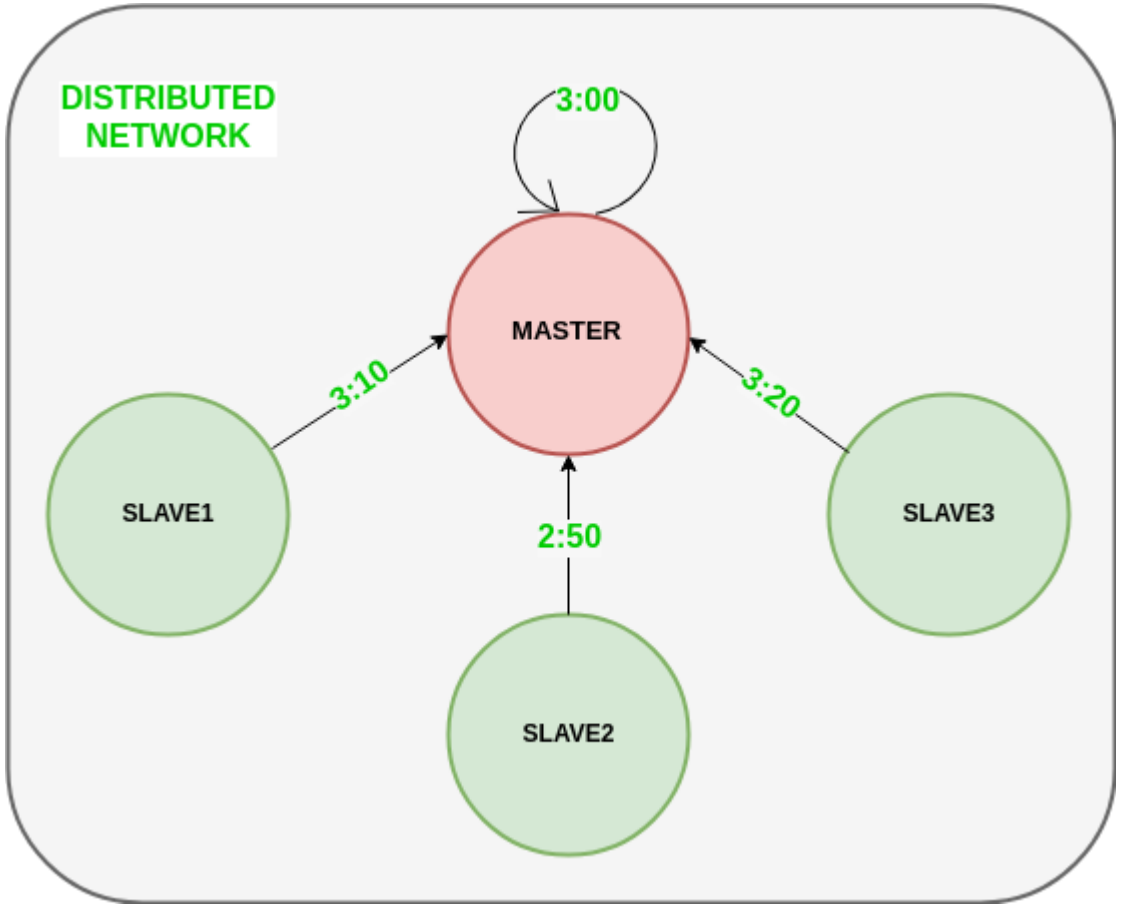


Figure illustrates how slave nodes send back time given by their system clock

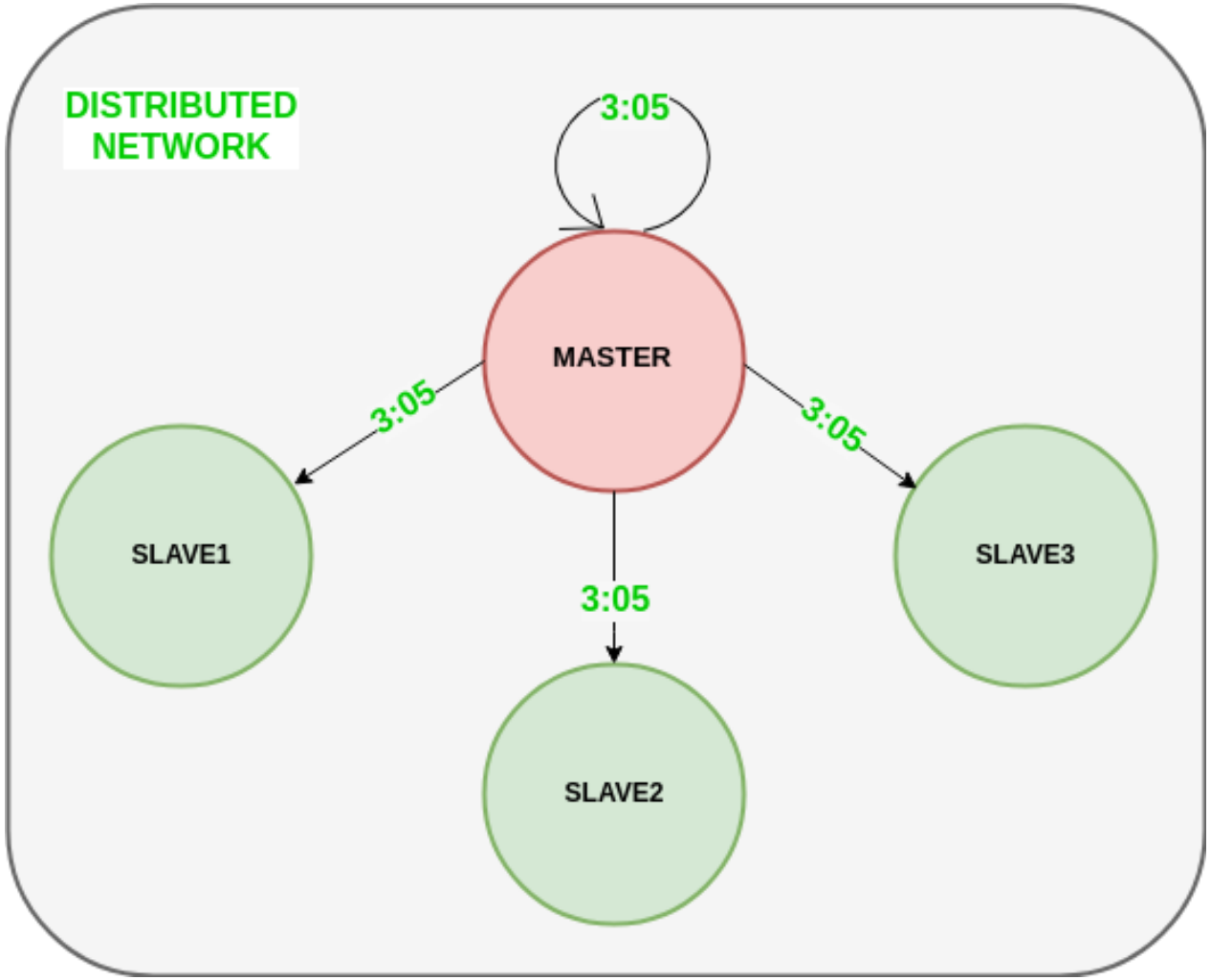


Figure illustrates the last step of Berkeley’s algorithm

Example: Consider a distributed system with four nodes: Node A (coordinator), Node B, Node C, and Node D.

- **Initial Clock Times**
  - Node A (Coordinator): 10:00:00.000
  - Node B: 10:00:05.000
  - Node C: 09:59:55.000
  - Node D: 10:00:10.000

Steps in Berkeley's Algorithm

1. **Coordinator Polls All Nodes**:
   1. Node A (Coordinator) sends requests to Nodes B, C, and D asking for their current times.
2. **Nodes Respond with Their Times**:
   1. Node B responds with 10:00:05.000
   2. Node C responds with 09:59:55.000
   3. Node D responds with 10:00:10.000

- **Coordinator Calculates the Average Time**:
- Node A receives all the times and includes its own time:
  - Node A: 10:00:00.000
  - Node B: 10:00:05.000
  - Node C: 09:59:55.000
  - Node D: 10:00:10.000

The coordinator calculates the average time:

$$\text{Average Time} = \frac{(10:00:00.000 + 10:00:05.000 + 09:59:55.000 + 10:00:10.000)}{4}$$

- Converting times to seconds for easier calculation:
  - Node A: 36000 seconds
  - Node B: 36005 seconds
  - Node C: 35995 seconds
  - Node D: 36010 seconds

$$\text{Average Time (seconds)} = \frac{36000 + 36005 + 35995 + 36010}{4} = \frac{144010}{4} = 36002.5 \text{ seconds}$$

Converting back to time format:

$$36002.5 \text{ seconds} = 10:00:02.500$$

4. **Coordinator Sends Adjustment Instructions**:

   - Node A calculates the adjustments needed for each node:
     - Node A: $10:00:02.500 - 10:00:00.000 = +2.500$ seconds
     - Node B: $10:00:02.500 - 10:00:05.000 = -2.500$ seconds
     - Node C: $10:00:02.500 - 09:59:55.000 = +7.500$ seconds
     - Node D: $10:00:02.500 - 10:00:10.000 = -7.500$ seconds

5. **Nodes Adjust Their Clocks**:

   - Node A sets its clock to 10:00:02.500
   - Node B sets its clock to 10:00:02.500
   - Node C sets its clock to 10:00:02.500
   - Node D sets its clock to 10:00:02.500

Summary

After the synchronization process using Berkeley's Algorithm, all nodes in the distributed system have their clocks set to the average time, 10:00:02.500. This approach ensures that all nodes in the system are synchronized as closely as possible, despite initial differences in their clock times.

Logical clock synchronization

Why Logical Clocks?

- Lamport (in 1978) showed that:
  - Clock synchronization is not necessary in all scenarios
    - If two processes do not interact, it is not necessary that their clocks are synchronized
  - Many times, it is sufficient if processes agree on the order in which the events has occurred in a DS
    - For example, for a distributed *make* utility, it is sufficient to know if an input file was modified *before* or *after* its object file

- Logical clocks are used to define an order of events without measuring the physical time at which the events occurred

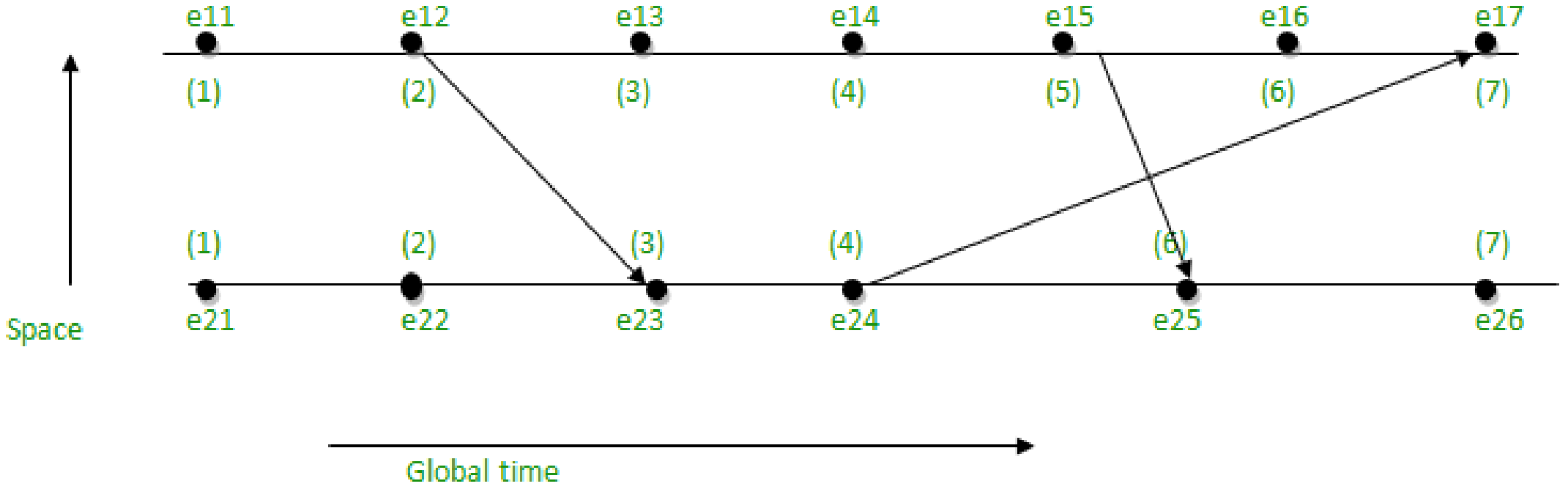
- We will study two types of logical clocks
  1. Lamport's Logical Clock (or simply, Lamport's Clock)
  2. Vector Clock

1. Lamport's Logical Clock

- **Lamport's Logical Clock** was created by Leslie Lamport. It is a procedure to determine the order of events occurring. It provides a basis for the more advanced Vector Clock Algorithm. Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

Algorithm:

- **Happened before relation(->):** a -> b, means 'a' happened before 'b'.
- **Logical Clock:** The criteria for the logical clocks are:
  - [C1]: C_i (a) < C_i(b), [C_i -> Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]
  - [C2]: C_i(a) < C_j(b), [Clock value of C_i(a) is less than C_j(b)]
- **Implementation Rules[IR]:**
- [IR1]: If a -> b ['a' happened before 'b' within the same process] then, Ci(b) =Ci(a) + d
- [IR2]: Cj = max(Cj, tm + d) [If there's more number of processes, then tm = value of Ci(a), Cj = max value between Cj and tm + d]



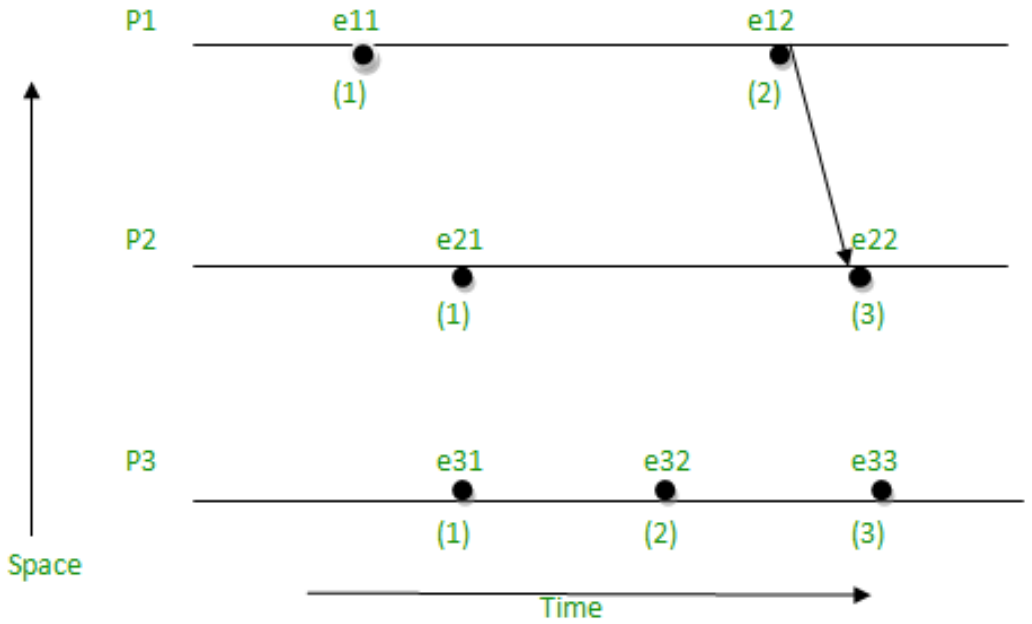
•Take the starting value as 1, since it is the 1st event and there is no incoming value at the starting point:

- e11 = 1
- e21 = 1

•The value of the next point will go on increasing by d (d = 1), if there is no incoming value i.e., to follow [IR1].

- e12 = e11 + d = 1 + 1 = 2
- e13 = e12 + d = 2 + 1 = 3
- e14 = e13 + d = 3 + 1 = 4
- e15 = e14 + d = 4 + 1 = 5
- e16 = e15 + d = 5 + 1 = 6
- e22 = e21 + d = 1 + 1 = 2
- e24 = e23 + d = 3 + 1 = 4
- e26 = e25 + d = 6 + 1 = 7

- When there will be incoming value, then follow **[IR2]** i.e., take the maximum value between $\mathbf{C_j}$ and $\mathbf{T_m + d}$.
  - e17 = max(7, 5) = 7, [e16 + d = 6 + 1 = 7, e24 + d = 4 + 1 = 5, maximum among 7 and 5 is 7]
  - e23 = max(3, 3) = 3, [e22 + d = 2 + 1 = 3, e12 + d = 2 + 1 = 3, maximum among 3 and 3 is 3]
  - e25 = max(5, 6) = 6, [e24 + 1 = 4 + 1 = 5, e15 + d = 5 + 1 = 6, maximum among 5 and 6 is 6]
- **Limitation**:
- In case of [IR1], if a -> b, then C(a) < C(b) -> true.
- In case of [IR2], if a -> b, then C(a) < C(b) -> May be true or may not be true.



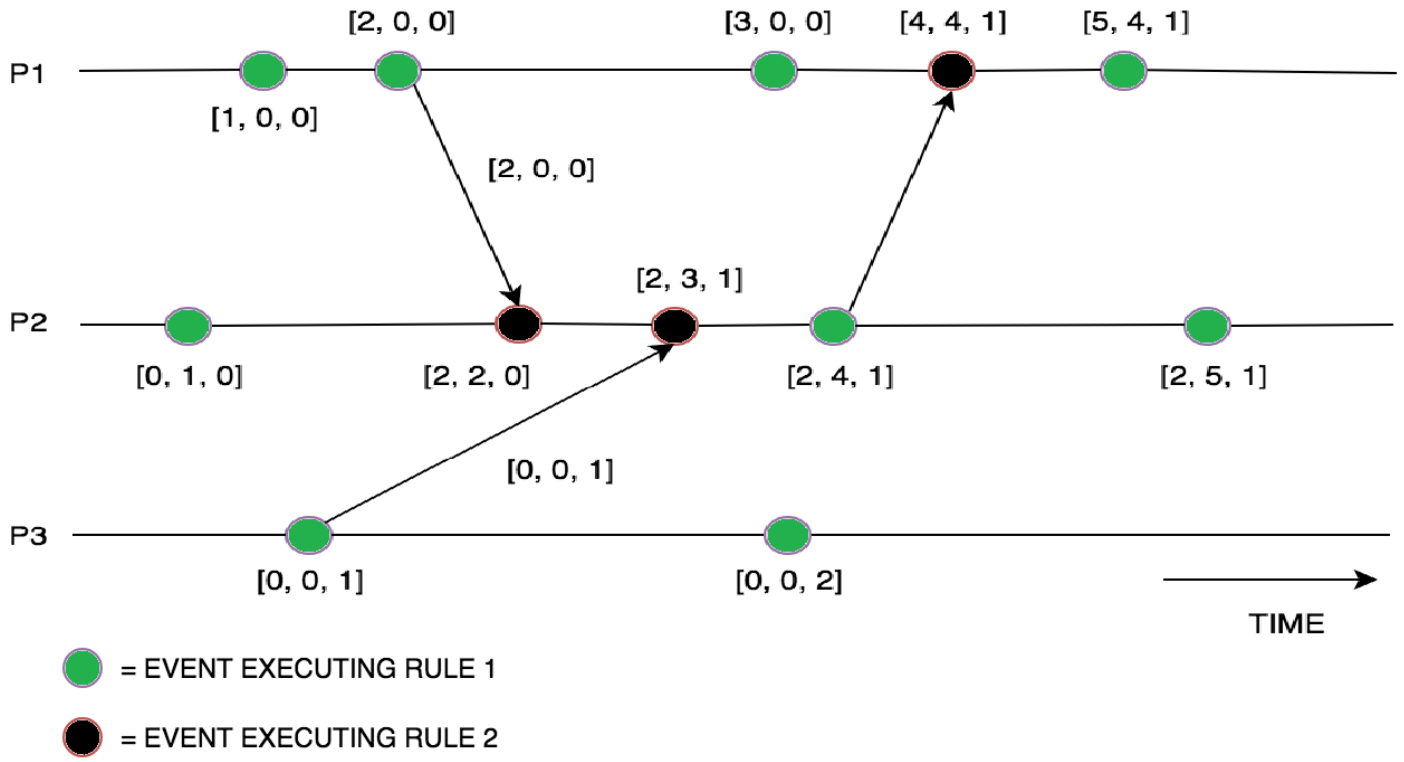
2. Vector clock

- **Vector Clock** is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector(a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/ array of size N.
- **How does the vector clock algorithm work :**
- Initially, all the clocks are set to zero.
- Every time, an Internal event occurs in a process, the value of the processes' logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the processes' logical clock in the vector is incremented by 1.
- Every time, a process receives a message, the value of the processes' logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example : Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:

- The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

- To sum up, Vector clocks algorithms are used in distributed systems to provide a **causally consistent** ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.



Mutual Exclusion Synchronization

- **Mutual Exclusion** is a property of process synchronization that states that “no two processes can exist in the critical section at any given point of time”. The term was first coined by Dijkstra.
- Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.
- Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections •
- The requirement of mutual exclusion is that when process P1 is accessing a shared resource R1, another process should not be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers, and shared data structures.
- *Race conditions can be avoided by implementing mutual exclusion techniques, such as locks, semaphores, or monitors. These techniques ensure that only one process or thread can access a shared resource at a time, preventing simultaneous conflicting accesses and maintaining data integrity.*

Occupied

Boy A

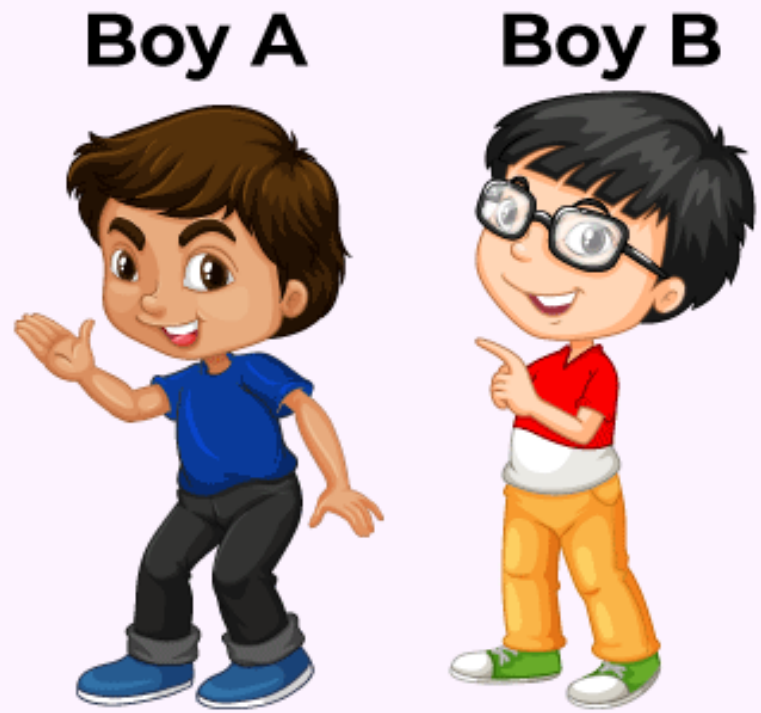
Since Boy A is Inside the changing room, the sign on it prevent the other from entering the room.

Boy B

Boy B has to wait outside the changing room till boy A comes out

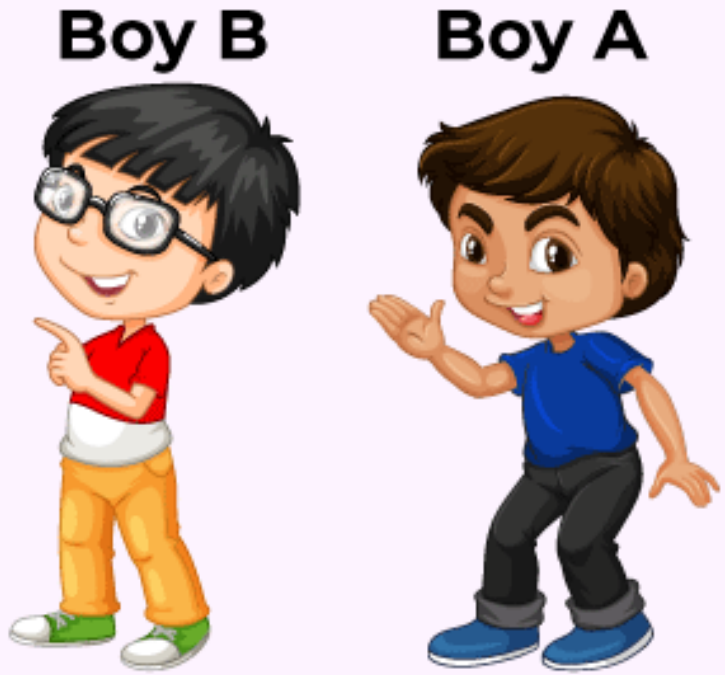
VACANT

Changing room initially lies vacant, allowing people to enter the room and use it



VACANT

Changing room initially lies vacant, allowing people to enter the room and use it

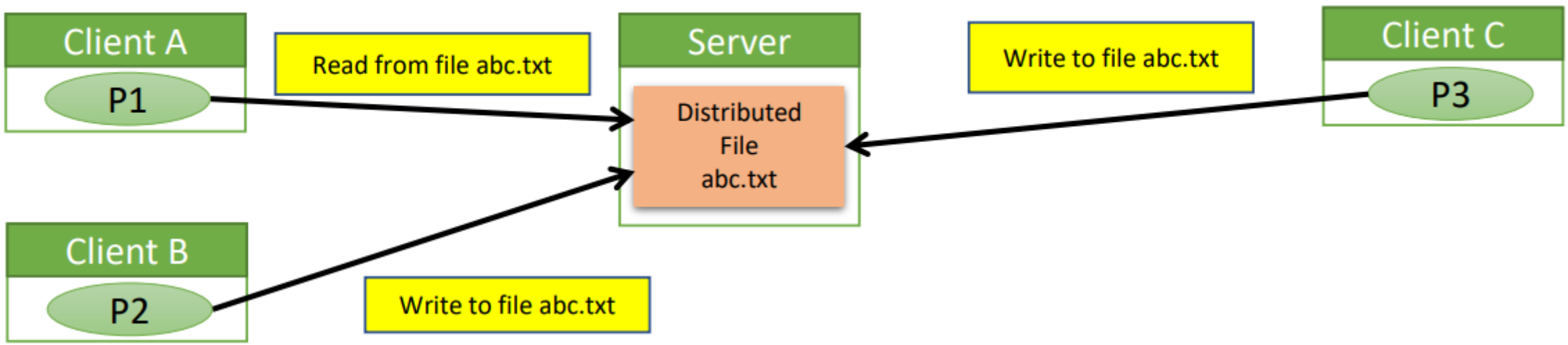


Since Boy A is done using the changing room, he vacates it- making the sign outside the room change.

Since the changing room has been vacted by Boy A, Boy B can enter the hanging room.

Need for Mutual Exclusion

- Distributed processes need to coordinate to access shared resources
- Example: Writing a file in a Distributed File System



In uniprocessor systems, mutual exclusion to a shared resource is provided through shared variables or operating system support.

However, such support is insufficient to enable mutual exclusion of distributed entities

In Distributed System, processes coordinate access to a shared resource by passing messages to enforce *distributed mutual exclusion*

Conditions Required for Mutual Exclusion:

According to the following four criteria, mutual exclusion is applicable:

1. When using shared resources, it is important to ensure mutual exclusion between various processes. There cannot be two processes running simultaneously in either of their critical sections.
2. It is not advisable to make assumptions about the relative speeds of the unstable processes.
3. For access to the critical section, a process that is outside of it must not obstruct another process.
4. Its critical section must be accessible by multiple processes in a finite amount of time; multiple processes should never be kept waiting in an infinite loop.

Requirements of Mutual Exclusion

1. At any time, only one process is allowed to enter its critical section.
2. The solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about the relative speeds of asynchronous concurrent processes.
5. A process cannot prevent any other process from entering into a critical section.
6. A process must not be indefinitely postponed from entering its critical section.

Election Algorithms

- **Election algorithms** are designed to choose a coordinator.
- Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor.
- Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number.
- The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.
- We have two election algorithms for two different configurations of a distributed system.
  - The Bully Algorithm
  - The Ring Algorithm

1. The Bully Algorithm

- The Bully Algorithm is a leader election algorithm used in distributed systems to ensure that one node is designated as the coordinator (leader) among a group of nodes.
- **Example:**
- Consider a distributed system with five nodes: A, B, C, D, and E. Each node has a unique identifier (ID), which can be used to compare nodes. Let's assume the IDs of the nodes are as follows: A=1, B=2, C=3, D=4, E=5. The node with the highest ID will be the leader.
- **Steps:**

1. **Initial State**: Assume all nodes are up and running. No leader has been elected yet.
2. **Election Initiation**:
    1. Node B (ID=2) detects that the current leader (if any) has failed or there is no leader. It initiates an election.
    2. Node B sends an election message to all nodes with higher IDs: C, D, and E.

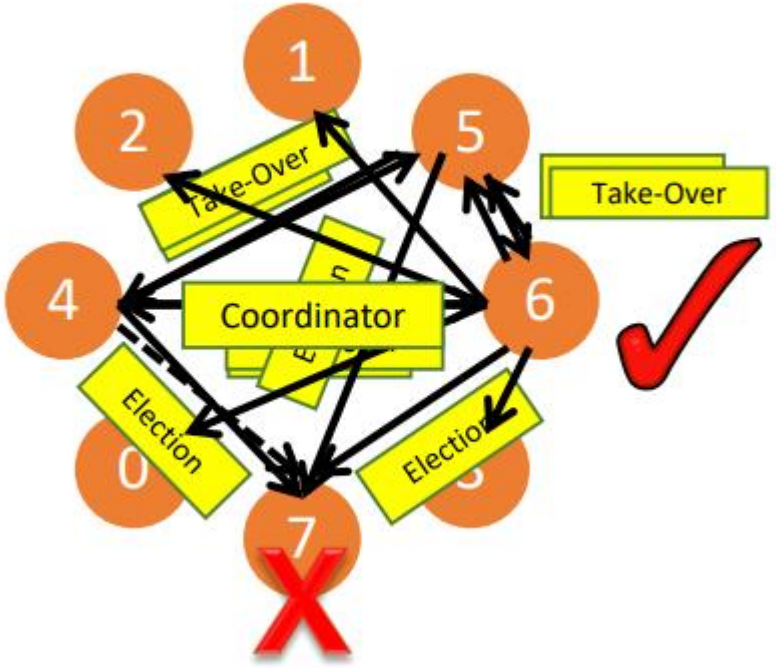
3. **Response to Election Message**:
   - Nodes C (ID=3), D (ID=4), and E (ID=5) receive the election message from B.
   - Since each of these nodes has a higher ID than B, they respond to B with an "OK" message, indicating that they will take over the election process.
   - Node B waits for a response. Since it receives "OK" messages, it knows that there are higher-ID nodes that are alive and will handle the election.

4. **Election by Higher Nodes**:
   - Node C (ID=3) now sends an election message to nodes D and E.
   - Nodes D and E respond with "OK" messages to C.
   - Similarly, node D (ID=4) sends an election message to node E.
   - Node E responds with an "OK" message to D.

5. **Highest-ID Node Declares Victory**:
   - Node E (ID=5) receives no "OK" messages in response to its election message (because there are no higher-ID nodes).
   - Node E declares itself the leader and sends a "Coordinator" message to all other nodes (A, B, C, D).

6. **Acknowledgment**:
   - All nodes receive the "Coordinator" message from node E and acknowledge E as the new leader.

- **Summary:**
- Node B initiates the election process.
- Nodes C, D, and E take over the election process since they have higher IDs.
- Node E, having the highest ID, becomes the leader and informs all other nodes.



Example Message Sequence:

1. B → C, D, E: Election
2. C → B: OK
   D → B: OK
   E → B: OK
3. C → D, E: Election
4. D → C: OK
   E → C: OK
5. D → E: Election
6. E → D: OK
7. E → All: Coordinator (Elected Leader)

In this way, the Bully Algorithm ensures that the node with the highest ID is elected as the leader, and the system can continue to operate with a designated coordinator.

2. The Ring Algorithm

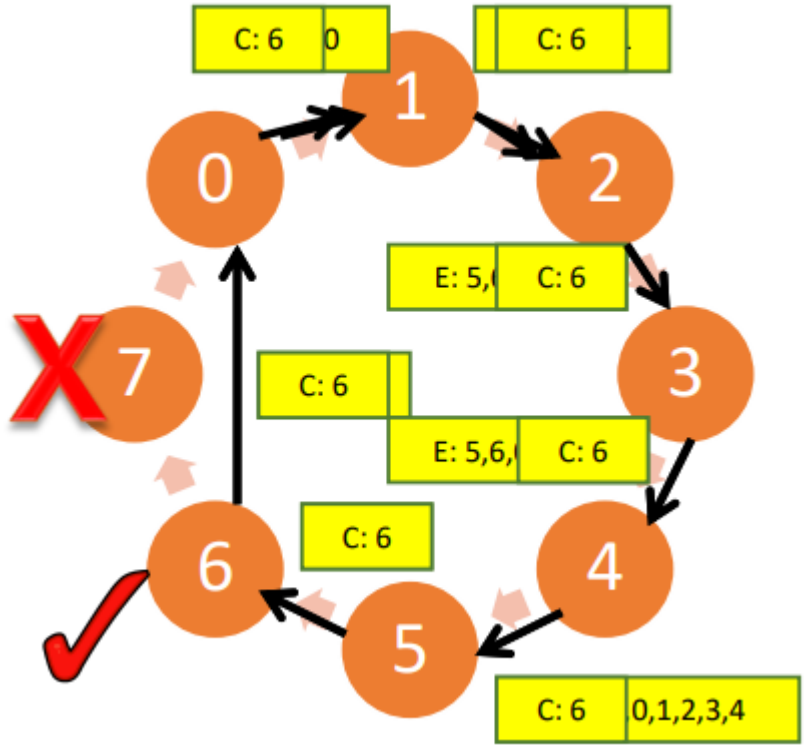
- The Ring Algorithm is another leader election algorithm used in distributed systems. In this algorithm, processes (nodes) are arranged in a logical ring, and messages are passed around the ring to determine the leader. Here's an example to illustrate how the Ring Algorithm works:
- **Scenario:**
- Consider a distributed system with five nodes: A, B, C, D, and E, arranged in a logical ring. Each node has a unique identifier (ID). Let's assume the IDs are as follows: A=1, B=2, C=3, D=4, E=5. The goal is to elect the node with the highest ID as the leader.
- **Steps:**

1. **Initial State**: Assume all nodes are up and running. No leader has been elected yet.
2. **Election Initiation**:
    1. Node B (ID=2) detects that there is no leader or the current leader has failed. It initiates an election.
    2. Node B creates an election message containing its own ID (2) and passes it to its next neighbor in the ring, which is node C.

- **Passing the Election Message**:
  - Node C (ID=3) receives the election message from B.
  - Node C compares its ID with the ID in the message (2). Since 3 > 2, node C updates the election message to contain its own ID (3) and passes it to the next node, D.
  - Node D (ID=4) receives the election message from C.
  - Node D compares its ID with the ID in the message (3). Since 4 > 3, node D updates the election message to contain its own ID (4) and passes it to the next node, E.
  - Node E (ID=5) receives the election message from D.
  - Node E compares its ID with the ID in the message (4). Since 5 > 4, node E updates the election message to contain its own ID (5) and passes it to the next node, A.
  - Node A (ID=1) receives the election message from E.
  - Node A compares its ID with the ID in the message (5). Since 1 < 5, node A does not update the message and passes it back to node B.
- **Leader Announcement**:
  - When the election message, containing the highest ID (5), returns to node B, node B recognizes that the node with ID 5 should be the leader.
  - Node B creates a "Coordinator" message with ID 5 and passes it to the next node, C.
  - Node C receives the "Coordinator" message, recognizes node E (ID=5) as the leader, and passes the message to D.
  - Node D receives the "Coordinator" message, recognizes node E as the leader, and passes the message to E.
  - Node E receives the "Coordinator" message, recognizes itself as the leader, and passes the message to A.
  - Node A receives the "Coordinator" message, recognizes node E as the leader, and passes the message back to B.
- **Acknowledgment**:
  - All nodes have now received the "Coordinator" message and recognize node E (ID=5) as the leader.

Example Message Sequence:

1. B → C: Election(2)
2. C → D: Election(3)
3. D → E: Election(4)
4. E → A: Election(5)
5. A → B: Election(5)
6. B → C: Coordinator(5)
7. C → D: Coordinator(5)
8. D → E: Coordinator(5)
9. E → A: Coordinator(5)
10. A → B: Coordinator(5)



In this way, the Ring Algorithm ensures that the node with the highest ID is elected as the leader by passing election messages around the ring.

Assignment - 3

- What is physical clock synchronization in distributed system? Explain Cristian's algorithm for synchronization
- Explain different entity naming schemes in distributed system.
- Write about Bully's algorithm and explain how it is different than other election algorithm.
- What is Name resolution and distributed name resolution? Explain different types of distributed name resolution algorithm.
- Discuss in detail the design and implementation of name spaces.
- Explain domain name system in brief.
- Why is clock synchronization needed in a Distributed system ? Explain Network Time protocol in brief.
- Describe how Lamport's Logical Clock works?
- How DNS works explain in detail