Distributed Systems Cheatsheet

Edoardo Riggio

November 2, 2022

Distributed Systems - S.A. 2022
Software and Data Engineering
Università della Svizzera Italiana, Lugano

Contents

1 Introduction

With the advent in the mid 1980's of 16-, 32-, and 64-bit CPUs – as well as the invention of high-speed computer networks, made it possible for the creation of large numbers of geographically-dispersed networks of computers. These are known as **distributed systems**.

1.1 Definition

A distributed system is a collection of independent computers that appear to the user as a single coherent system.

Each computing element of these massive systems are able to behave independently from one another. A computing element is often referred to as a **node**. This node can either be a hardware device or a software process.

1.2 Consequences

Some of the main negative consequences that arise when dealing with distributed systems are the following:

- **Concurrency**
  This happens when several processes try to read and write on a shared storage service.
- **Absence of a global clock**
  Each node will have its own notion of time. This means that there is no common reference of time between the nodes.
- **Failure independency** The failure of a node can make another node unusable.

1.3 Challenges

Some of the challenges that arise when dealing with distributed systems are outlined in the following sections.

1.3.1 Openness

The services are offered according to standard rules. These rules describe both the syntax and semantics of such services. The standard rules of distributed systems are called **COBRA**(Common Object Request Broker Architecture).

1.3.2 Scalability

Scalability can be with respect to the **system size** – which would mean adding more users to the system; to the **geography** – which would deal with users lying far apart from one another; and to **administration** – which would deal with the complexity to manage an increasing system.

Some scalability techniques are the following:

- **Hiding communication latencies**
  This is important for **geographical scalability**. Asynchronous communications can be used to reduce the waiting time of the users. This can be achieved with the use of batch processing and parallel applications.

- **Distribution**
  Components are split into parts and spread across the system. An example of this would be the DNS, where we have a tree of domains divided into non-overlapping zones. Furthermore, the name in a zone is handled by a single name service.

- **Replication**
  This can be use to increase both **availability** and **performance**, as well as reduce **latency**. Moreover, caching can be used as a form of replication, but is typically done on-demand.

1.3.3 Transparency

Transparency is the ability of a system to hide some of its characteristics or errors to the user. There exist several different forms of transparency:

- **Access transparency**
  Hide the differences in data representation and machine architecture.

- **Location transparency**
  Users cannot tell where a resource is physically located.

- **Relocation transparency**
  Even if the entire service was moved from one data center to the other, the user wouldn't be able to tell.

- **Migration transparency**
  Moving processes and resources initiated by users, without affecting any ongoing communication and operation.

- **Replication transparency**
  Hide the existence of multiple replicas of one resource.

- **Concurrency transparency**
  Each user is not going to notice if another user is making use of the same resource.

- **Failure transparency**
  The user or application does not notice that some piece of the system fails to work properly. The system is then able to automatically recover from the failure.

1.4 Types of Distributed Systems

There are three main types of distributed systems: **distributed computing systems**, **distributed information systems**, and **distributed pervasive systems**.

1.4.1 Distributed Computing Systems

These systems aim at high-performance computing tasks. These systems can be part of two subtypes:

- **Cluster Computing**
  All the resourced are located in a local-area network, and are using the same OS. In addition, they also have a common administrative domain.
- **Grid Computing**
  This is a "federation" of computer systems. Such systems may have different administrative domains, different hardware, software... Such systems are used to make collaboration between organisations feasible.

1.4.2 Distributed Information Systems

These systems are based on transactions. These transactions are used to make systems communicate between themselves. Transactions follow the **ACID** properties:

- **Availability**
  To the user, the transaction happens indivisibly.
- **Consistency**
  The transaction does not violate system invariants.
- **Isolation**
  Concurrent transactions do not interfere with each other.
- **Durability**
  Once a transaction commits, the changes are permanent.

The application components of each node communicate directly with each other.

1.4.3 Distributed Pervasive Systems

These systems are composed by mobile and embedded computing devices. This means that pervasive systems need to have the following characteristics:

- Embrace contextual changes
- Encourage ad-hoc composition
- Recognise sharing as the default

Some examples of such architectures are home systems, electronic health care systems, and sensor networks.

2 Architectures

In a distributed system there are two types of architectures:

- **Software Architectures**
  How software components are organised and interact between each other.
- **System Architectures**
  How software components are instantiated on real machines.

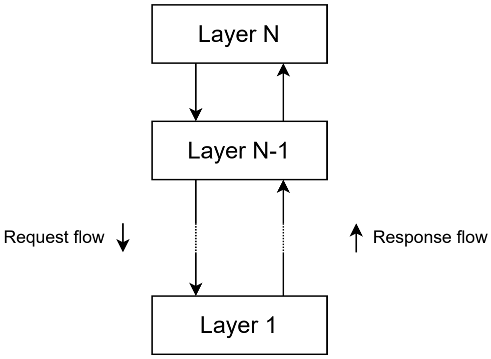
2.1 Software Architectures

Software architectures are based on **components**, which are modular units with well-defined interfaces. Software architectures can be of several different types:

- **Layered architectures**
- **Object-based architectures**
- **Data-centred architectures**
- **Event-based architectures**

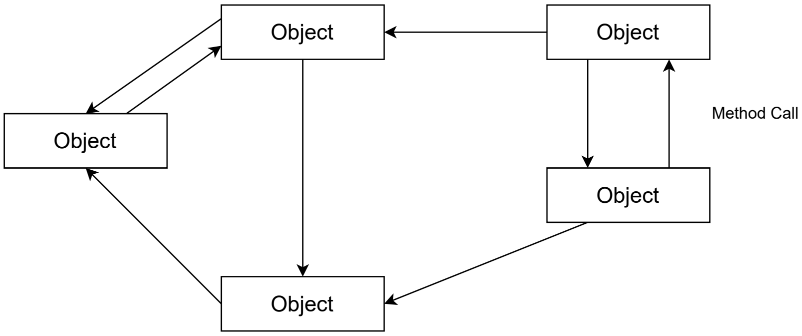
2.1.1 Layered Architectures

In this kind of architecture, the components at layer L_i can call components in layer L_{i-1}, but not components in layer L_{i+1}.



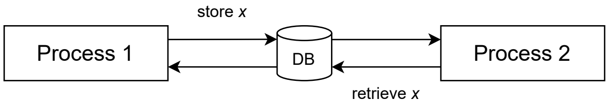
2.1.2 Object-Based Architectures

Each object is a component connected through a remote procedure call mechanism.



2.1.3 Data-Centred Architectures

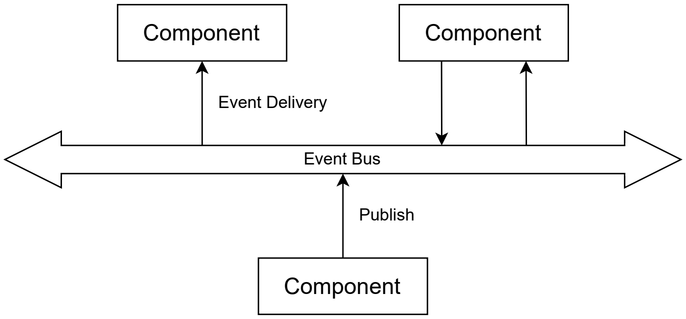
The processes communicate through a common repository. This repository can be, for example, a shared distributed file system or a database.



2.1.4 Event-Based Architectures

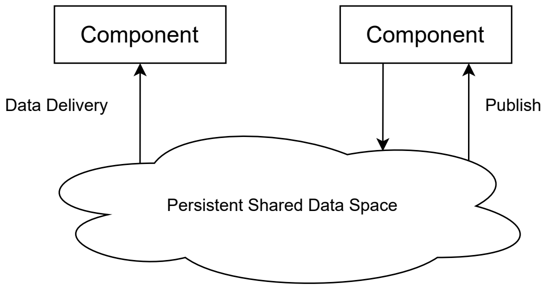
The processes communicate through the propagation of events (**publish-subscribe system**). The middleware ensures that the process which subscribes to that events will receive them.

Processes are loosely coupled, this means that they do not to refer to each other.



2.1.5 Shared Data-Space Architecture

These architectures are similar to data-centred and event-based architectures.



2.2 System Architectures

System Architectures describe what software components are used, where to place each component, and how the components interact with each other.

There are two types of system architectures:

- **Centralised architectures**
- **Decentralised architectures**

2.3 Centralised Architecture

Centralised architectures follow the Client-Server model. The **server** implements some services, while the **client** requests services and waits for replies.

The communication between server and client is **connectionless**. This means that it is highly efficient, but it is more complex to handle transmission failures (UDP).

Server and client may also use another way to communicate. This protocol is connection-oriented, relatively low performance, but more reliable. This is because the node makes a request and receives a reply back in the same connection.

Centralised architectures can be either **single-** or **multi-layered**. Multi-layered architectures enable to divide concerns, meaning that clients can contain only the program implementing the user-interface level (or only part of it). Moreover, architectures can also be **multi-tiered**.

2.4 Decentralised Architectures

Decentralised architectures can be either vertically or horizontally distributed.

- **Vertical Distribution**
  It can be achieved by placing logically different components on different machines. An example of this is a three-tiered application.

- **Horizontal Distribution**
  Both clients and servers are physically split up into logically equivalent parts. Each part operates on its own share of the dataset. An example of this is a peer-to-peer system.

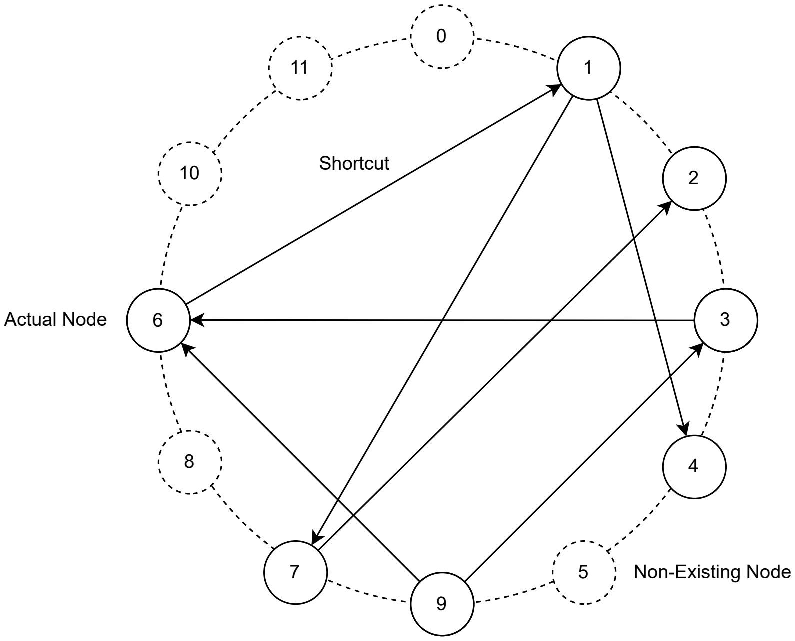
2.4.1 Structured Peer-to-Peer Architectures

Structured peer-to-peer architectures are deterministic procedures used to build an **overlay network**. Here nodes are processes, and links are possible communication channels.

In the case of a structured peer-to-peer architecture, we use **Distributed Hash Tables** (DHT). Each data item that is maintained by the system, is uniquely associated with a key, and this key is subsequently used as an index. Moreover, to each node of the system is assigned an identifier. Each node is made responsible for storing data associated with a specific subset of keys.

Any node can be asked to look up a given key, which boils down to efficiently routing that lookup request to the node responsible for storing the data associated with the given key.

An example of a structural peer-to-peer architecture is a **Chord**, which is depicted below.

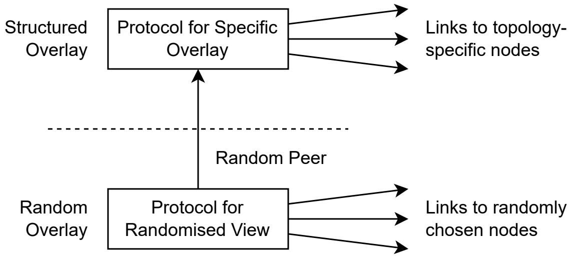


2.4.2 Unstructured Peer-to-Peer Architectures

In this type of architecture, the overlay network is built using a randomised procedure. Each node in the network has a list of neighbours, constructed in a random way. This results in a random graph.

In these kinds of systems, when a node joins the network, it often contacts a well-known node to obtain a starting list of other peers in the system. Moreover, the nodes generally change their local list almost continuously.

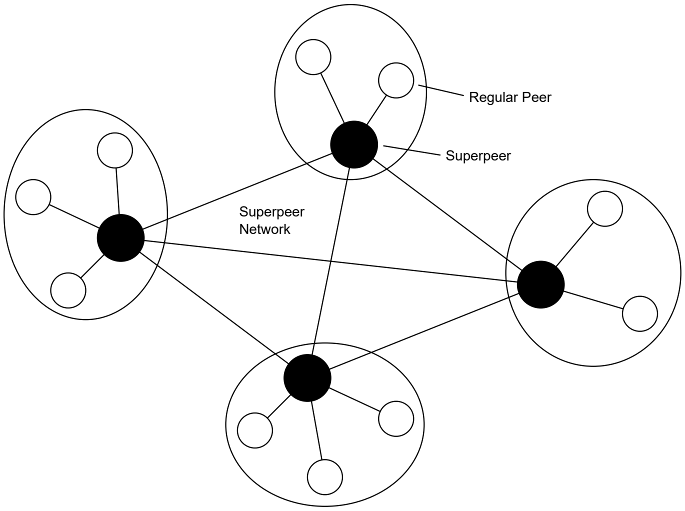
Unlike in structured peer-to-peer architectures, looking up data cannot follow a predetermined route. Instead, in this case we need to resort to searching for data. This can be done in different ways, such as **flooding**, **random walks**, or **policy-based** search methods.


Structured Overlay
Protocol for Specific Overlay
Links to topology-specific nodes
Random Peer
Random Overlay
Protocol for Randomised View
Links to randomly chosen nodes

2.5 Super Peer

Brokers are services that collect data on resource usage and availability for a number of nodes that are in each other's proximity. This allows to quickly select the node with sufficient resources.

Nodes that act as brokers are called **super peers**. Super peers organise themselves into a structured peer-to-peer network, leading to a hierarchical organisation of nodes. The nodes that connect to these super peers are known as **weak peers**.



2.6 Middleware

Middleware provides a degree of distribution transparency. They can either follow an object-based architectural style, or an event-based architectural style.

2.6.1 Interceptors

They offer a means to adapt the middleware. Interceptors break the usual flow of control and allow other application-specific code to be executed. Moreover, they are used to improve software management.

2.6.2 Wrappers and Adapters

Wrappers and adapters provide a similar interface to the original abstraction, and implement additional logic before or after invoking the original interface.

2.6.3 Proxies and Stubs

They provide – as in the case of wrappers and adapters – a similar interface as the original abstraction. However, in this case proxies and stubs usually own components rather than extending previously defined ones.

3 Processes

3.1 Introduction

Virtual processors are created by the OS. A process is a program executed on one of these virtual processors. Each processor has a process table in which several different values relative to the process are stored.

Concurrent processes are protected from each other by making the sharing of CPU and hardware resources transparent and using special hardware.

Every time a new process is created, the OS must create an independent address space as well. This is known as **context switching**, and it's really expensive.

3.2 Threads

A thread is a lightweight process that can run inside of a process. Threads share the same memory space. For this reason, context switching between threads is less expensive than between processes.

Threads are used in distributed systems mainly for making communication calls non blocking.

In the case of servers with **single-threaded models**, one thread receives the request and executes it rapidly. This is known as sequential processing.

On the other hand, in the case of servers with **finite state machine models**, one thread executes the requests and the replies. Blocking system calls are replaced by non-blocking ones, and multiple requests can be handled in parallel.

3.3 Virtualisation

The role of virtualisation – in distributed systems – is to extend or replace existing interfaces and mimic the behaviour of other systems.

The main reasons of virtualisation are the following:

- Allow legacy software to run on new mainframe hardware
- Porting legacy interfaces to new platforms
- Isolating systems running on the same server

3.4 Superservers

A superserver is a kind of server that listens on many ports and fork a process to execute a service.

3.5 Stateless vs Stateful Servers

A server is said to be **stateless** if it does not keep any information about the state of the client.

On the other hand, a server is said to be **stateful** if it maintains persistent information about the clients. This solution is better in terms of performance (it caches data of the users), but can be problematic in the case that the server crashes or experience other memory-related issues.

3.6 Server Clusters

A server cluster is a collection of machines connected through a network. Such clusters have three tiers:

1. **Logical Switch**
   It receives client requests and route them to the servers.

2. **Application Servers**
   Low-end servers are used when the storage is the bottleneck of the system. Otherwise, high-end servers are used when the service deals with computationally expensive computations.

3. **Data-Processing Servers**
   The databases.

4 Communication

In layered protocols, process A con communicate with process B by building a message in its address space. After doing so, a system call is used to send the message from A to B.

4.1 OSI Model

The **Open System Interconnection** (OSI) **Reference Model** is an ISO standard. This type of model is never widely used or implemented.

The OSI model makes use of connection-oriented protocols. Both the sender and the receiver explicitly establish a connection, they communicate, and finally explicitly terminate the connection.

The OSI model is divided into the following parts:

- Application layer
- Presentation layer
- Session layer
- Transport layer
- Network layer
- Data link layer

- Physical layer

In the following sections, we will have a bottom-up look at the different layers.

4.1.1 Physical Layer

This layer deals with how to send bits from one end to the other. It standardises electrical, mechanical, and signalling interfaces. finally, it does not handle errors.

4.1.2 Data Link Layer

The data link layer groups the bits into frames and ensures their correct transmission. It uses checksums to verify the integrity of the data frames.

4.1.3 Network Layer

It routes the messages from source to destination. To do so, it needs a routing protocol. The most common ones are:

- **TCP**
  Transmission Control Protocol (TCP) is both reliable and connection-oriented.
- **UDP**
  Universal Datagram Protocol (UDP) is both unreliable and connectionless.

4.1.4 Session Layer

The session layer is an enhanced version of the transport layer – but rarely supported.

4.1.5 Presentation Layer

This layer simplifies the communication between machines with different internal data representation.

4.2 Middleware Protocols

This type of protocols usually resides at the application layer. The middleware protocols can have several different roles, such as:

- Authentication
- Commit protocols
- Distributed locking protocols

4.3 Types of Communication

There are several different ways in which a client and a server can communicate:

- **Persistent Communication**
  The message is stored by the communication middleware until it can be delivered to the receiver.
- **Transient Communication**
  The message is delivered only if both sender and receiver are executing.
- **Asynchronous Communication**
  The sender continues immediately after submitting the message. This message is temporarily stored inside of the middleware.
- **Synchronous Communication**
  the server is blocked until the message has been received by the receiver.

4.4 Remote Procedure Call

A **Remote Procedure Call** (RPC) is a simple communication mechanism, and is more natural than the send/receive primitives.

4.4.1 Basic RPC Operation

The RPC first makes itself look like a local procedure call. When a read operation is performed, a client stub is called. The client stub pack all the parameters into a message and sends it to the server.

Both the calling and the receiving procedures are not aware of the distribution of the system.

4.5 Message-Oriented Communication

Message-oriented transient communication uses **Berkeley sockets**. These sockets are communication endpoints used by applications to read are write. The process of the socket is the following:

Server:

1. Create the socket
2. Bind the socket to a port
3. Listen on the opened port
4. Accept upon communication request
5. Communicate with the client

6. Close the socket

Client:

1. Create the socket
2. Request connection to the server
3. Communicate with the server
4. Close the socket

4.5.1 Message-Queuing Models

The applications communicate between one another by inserting messages into queues. Each application has its own private queue, and both the sender and receiver do not have to necessarily be active at the same time.

There is guarantee of delivery at destination, but the time at which it reaches its destination cannot be known.

Queue managers interact with both relays and applications. The relays forward the messages to other queue managers. The message is delivered as follows:

1. Queue managers interact with both application and relays
2. The relays forward the messages to other queue managers
3. The overlay network composes the sender, the destination, and the relays

4.5.2 Message Brokers

A message broker acts as an application-level gateway. They do so by integrating existing and new applications into a single coherent system. Moreover, they convert the incoming messages ti formats that can be understood by the destination application.

4.6 Multicast Communication

Application-level multicasting has the following characteristics:

- Support for sending data to multiple receivers
- Nodes reorganise themselves into an overlay network at the application level

There are two main types of overlays:

- **Tree-Based Overlay**
  There is a single path between every pair of nodes.

- **Mesh-Based Overlay**
  There are multiple paths between every two pair of nodes. Although it is harder to find the best path between a pair of nodes, these overlay networks are generally more robust.

5 Naming

It is impossible to memorise IPv4 and IPv6 addresses to reach websites. For this reason. we use **naming**. Naming can be of three types:

- Flat
- Structured
- Attribute-Based

5.1 Flat Naming

Flat names are composed of names, identifiers, and addresses. They can be described as follows:

- **Name**
  It is "human-friendly" and it is not necessarily unique or used for identifying.
- **Identifier**
  It uniquely identifies an entity.
- **Address**
  It is not "human-friendly" and is used for low-level naming of access points.

5.1.1 Forwarding Pointers

When an entity is moved from point A to point B, it leaves a reference in point A pointing to B. The requests to A must be forwarded to B in a transparent manner.

There are some drawback to forwarding:

- Fast moving entities leave a long chain of entities
- Each intermediate location needs to allocate resources for the forwarding process
- Any missing step in the chain makes the whole chain unusable

5.1.2 Home-Based Approaches

In this case, a home location is used to keep track of the current location. By doing so, IPv6 addresses become identifiers. The following is the procedure of home-based approaches:

1. Mobile devices requests a local temporary address
2. The address is registered with the home agent
3. The home agent forwards the packets to the mobile device
4. The home agent tells the sender the new temporary location

There are some drawbacks that come with home-based approaches, such as:

- Increased latency
- Home agent must always be available
- Permanent relocations must be handled

A possible solution to all of these problems comes with registering home locations using a **Domain Name Server** (DNS).

5.1.3 Distributed Hash Tables

We assign an m-bit identifier space to each node of the system. To resolve the successor node k, we use a **finger table**.

$$FT_p[i] = succ(p + 2^{i-1})$$

Where p is the node, and i is the number of nodes succeeding p. The complexity of the successor computation is:

$$O(\log(nodes))$$

A drawback of this method is that nodes joining and leaving force the finger table to continuously update. This issue can be solved by making the nodes run the function `succ(k)` in the background.

5.1.4 Hierarchical Approaches

Each **root node** keeps track only of the location of the directory nodes of the next lower-level subdomains. On the other hand, each **leaf node** contains the location of the entities.

A lookup request is performed bottom-up. This means that the request is forwarded amongst the children of the directory node. Afterwards, the request goes up the hierarchy.

5.2 Structured Naming

5.2.1 Name Spaces

Name spaces can be represented as labeled, directed graphs with directory and leaf nodes.

A **pathname** is a sequence of labels corresponding to the edges in that path.

5.2.2 Linking and Mounting

A **symbolic link** is a link between a node and an absolute path. This is particular since normally nodes are linked to files – in the case of a OS's file system.

Mounting a foreign name space in a distributed system is a type of symbolic link. For this symbolic link to work it requires:

- Name of an access protocol
- Name of a server
- Name of the mounting point in the foreign name space

5.2.3 Name Space Distribution

Generally, large-scale name services are organised into **organisational layers**. The layers are the following:

- **Global Layer**
  It is composed by the root name and its children (very stable)
- **Administrational Layer**
  The nodes are managed by a single organisation (somewhat stable)
- **Managerial Layer**
  Nodes are managed locally (may change frequently)

5.2.4 Name Resolution

All clients have access to a name resolver. This name resolution can happen in two ways:

- **Iteratively**
  The name resolver performs each step of the name resolution.
- **Recursively**
  The name resolver delegates the name resolution to the root server.

Each one of the previously described methods has its own drawbacks. In the case of the **iterative** method, it has high communication costs and poor caching capabilities. In the case of the **recursive** method, it has an increased load on each server of the chain, as well as increased latency for non-cached requests.

5.2.5 DNS Name Space

The DNS is a hierarchically organised name space. Some of its characteristics are:

- DNS root has no name
- Each subtree is a domain
- The path to the subtree root node is the domain name
- All nodes contain records
- One server is responsible for one zone

5.3 Attribute-Based Naming

Attribute-based naming systems are also known as directory services. In this case, an entity is described in terms of an `(attribute, value)` pair. A set of these attributes is used for searching.

5.3.1 LDAP

Lightweight Directory Access Protocol (LDAP) has the following characteristics:

- Each service contains a number of directory entries
- Each entry is composed of a number of `(attribute, value)` pairs
- Each attribute has an associated type
- Attributes can be single- or multi-valued
- The collection of all entries for a service is called a **Directory Information Base** (DIB)

In a DIB, each entry has a **Relative Distinguished Name** (RDN). This name is globally unique.

RDNs can be used a globally unique names – which is the same thing that happens in DNS. Moreover, the RDNs establish a hierarchy of entries – which is known as the **Directory Information Tree** (DIT).