



# IT3010

## Network Design and Management

### Lecture 01

#### Module Introduction

## Shashika Lokuliyan

Faculty of Computing  
Department of CSE



# SLIIT

*Discover Your Future*

# IT3010

# NETWORK DESIGN MANAGEMENT

Lecture 1  
Introduction to Network Design & Management

# Syllabus Overview

Week	Lecture
1	Introduction to Network Design & Management
2	Network Server and Monitoring
3	Simple Network Management Protocol (SNMP)
4	Lightweight Directory Access Protocol (LDAP)
5	Introductions to Sockets and TCP/IP simple client-server example
6	Multiplexing I/O sockets for applications
7	Socket options
8	Web sockets

# Method of delivery

Lecture	2 hours/week
Tutorial	1 hour/week
Lab Practical	2 hours/week

## **Important**

**Do not miss any lecture, tutorial or lab session as both the mid-term and semester end examination will be based on all 3 components of delivery.**

# Assessment Criteria

- Assessments • 50%
  - Assignment: 25%
  - MID Term Examination: 25%
  - Semester end Final Examination 50%



# Personal

## Lecturer-in-charge

Ms. Shashika Lokuliyana

Senior Lecturer

Department of Computer  
Systems Engineering

E: *shashika.l@sliit.lk*

P: 011754-3921

## Co - Lecturers

Ms. Pipuni Wijesiri

Lecturer

Department of Computer  
Systems Engineering

E: *pipuni.w@sliit.lk*

**IMPORTANT : Students who needs to meet the lecturer/s or the instructor/s out of the normal allocated hours are required to send an email to the above mentioned mail addresses and get a date and time. Please make a note that the lecturers will only consider mails sent from their SLIIT mail addresses.**

## Learning Resources

### Principal References

- Lienwand, A. and Fang, K. C., Network Management: A Practical Perspective (2<sup>nd</sup> Edition), ISBN: 0-201-60999-1, Addison Wesley, 1996 – **Available in library.**
- Howes, T., Smith, M. C., Good, G. S., Understanding and Deploying LDAP Directory Services, ISBN: 1-56870-070-1, 1999 – **Available in library.**
- Stalling W., SNMP SNMP2 and RMON Practical Network Management (2<sup>nd</sup> Edition), ISBN: 0-201-63479-1, Addison Wesley, 1996

# Academic Integrity Policy

- Are you aware that following are not accepted in SLIIT???
- Plagiarism - using work and ideas of other individuals intentionally or unintentionally
- Collusion - preparing individual assignments together and submitting similar work for assessment.
- Cheating - obtaining or giving assistance during the course of an examination or assessment without approval
- Falsification – providing fabricated information or making use of such materials
- From year 2018 the committing above offenses come with serious consequences !
- See General support section of Courseweb for full information.

Thank you



# IT3010

## Network Design and Management

### Lecture 02

### ISO Network Management Framework

**Shashika Lokuliyana**

Faculty of Computing  
Department of CSE



**SLIIT**

*Discover Your Future*

# IT3010

# NETWORK DESIGN MANAGEMENT

Lecture 1  
Introduction to Network Design & Management

# Today's lecture overview

- Design Methodology and Considerations
- Rationale for Network Management
- Network Management Process
- Network Management Systems

# Introduction

**Definition of a Data Network:** A collection of devices and circuits for transferring data from one computer to another (or device, e.g. printer).

*Purpose:*

It enables users at different locations to share the resources of a computer stationed elsewhere.

E.g. : Automated Teller Machine (ATM)

# Goals

*Why bother about the network design..?*

Primary goal of network design is to meet the organizations communication needs.

Productivity 

Budget 

# Considerations

*Achieving the goal..*

Need to develop a **comprehensive plan**. Must take into account the following:

- Suitability
- Reliability
- Scalability
- Durability

# Network Engineer/Administrator

## Role of a Network Engineer/Administrator

Network engineers have the responsibility for installing, maintaining, troubleshooting, optimizing and expanding the network.

- As a network expands, so too the size and number of potential problems.
- The overall goal of network management is to help network engineers deal with the complexity of data networks.
- Design based on Network Management principles.

# The Network Management Process

**Network Management** is the process of controlling complex data networks to maximize its efficiency and productivity.

## ISO Framework for Network Management

- Configuration management
- Security management
- Performance management
- Accounting management
- Fault management

# Configuration Management

**Configuration management** is the process of

- **Gathering** information about the current network environment.
- Using that data to **modify** the configuration of network devices.
- **Storing** the data, maintaining an up-to-date inventory of all network components and producing various reports.

Bridge Configuration Management Information	
Name	Software Version
Payroll Mainframe Subnet	A
Terminal Server Subnet	B
Engineering Computer subnet	A

# Data Collection, Modification and Storing

## Data Collection

*Two methods..*

- Manual collection
  - Tedium, error prone, time consuming
- Auto-discovery/Auto-mapping
  - ICMP (ping, traceroute)
  - Network Management Protocol

# Data Collection, Modification and Storing

## Data Modifications

- Once configuration management information has been obtained, it will usually need to be updated.
- Network devices usually contain many pieces of modifiable parameters.
  - E.g. – routers (routing tables, network interfaces), servers (application services, operating systems).

# Data Collection, Modification and Storing

## Storing Information

*Methods of storage:*

- **Unstructured** (e.g. ASCII files)

- Advantages:

- Easily read.
    - Easily accessed from remote locations.
    - Easy to administer.

- Disadvantages:

- Inefficient storage.
    - Slow to search.
    - **Unable to provide complex data relationships.**

# Data Collection, Modification and Storing

## Storing Information

*Methods of storage:*

- **Structured** (e.g. DBMS)

- Advantages:

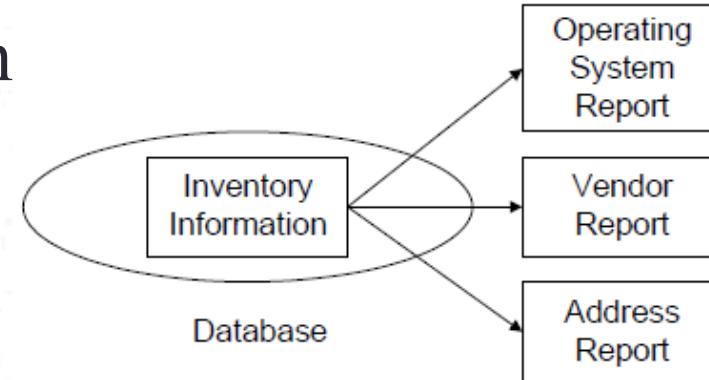
- Stores data efficiently.
    - Enables users to relate various types of information to one another.
    - Versioning.

- Disadvantages:

- Need to learn query language to access data (e.g. SQL).
    - Administrative overheads.

# Configuration Management - Benefits

- Automatically gather and update data on network devices.
  - Allows devices to be configured remotely.
  - Provides central storage location for configuration data.
  - Facilitates the production of network inventory and other reports.



# Security Management

- Security management involves protecting sensitive information on devices attached to a data network by controlling access points to that information.
- Security management consists of the following aspects:
  - Identifying the sensitive information to be protected.
  - Finding the access points (vulnerabilities).
  - Securing the access points.
  - Maintaining the secure access points.

# Security Management - Benefits

- Increases confidence in utilizing the network.
  - A lack of security may force drastic measures, such as eliminating network access of sensitive information altogether.
  - Properly set up and maintained security management can offer more practical alternatives.
- Some examples:
  - 1988: Internet worm
  - More recently: ICMP and TCP-based DoS attacks

# Performance Management

**Performance management** involves ensuring that networks remain accessible and free from congestion:

- Monitoring network devices and their associated links to determine utilization and error rates.
- Helping the network provide consistent quality of service (QoS) by ensuring that the capacity of devices and links is not over taxed to the extent of adversely impacting performance.
  - Context-specific

# Performance Management

Performance management entails the following steps:

1. Collecting data on current utilization of network devices and links.
2. Analyzing relevant data.
  - Statistical analysis
  - Workload modeling
3. Setting utilization thresholds.
4. Using simulation to determine how the network can be altered to maximize performance.

# Performance Terms

- Availability
- Bandwidth/Throughput
- Propagation
- Congestion
- Latency
- Threshold
- Utilization

# Performance Management - Benefits

- Reduces probability of network congestion and inaccessibility so as to provide a consistent level of service to users.
  - E.g. – Knowing the network's utilization workload can help one schedule large data transfers for non-peak times.
- Assist in examining network trends:
  - Content Creators
  - Content-hosting companies
  - Network operators
  - Networking researchers

# Accounting Management

**Accounting management** is the process of gathering network statistics to help the network engineer make decisions about the allocation of network resources.

Accounting management involves the following tasks:

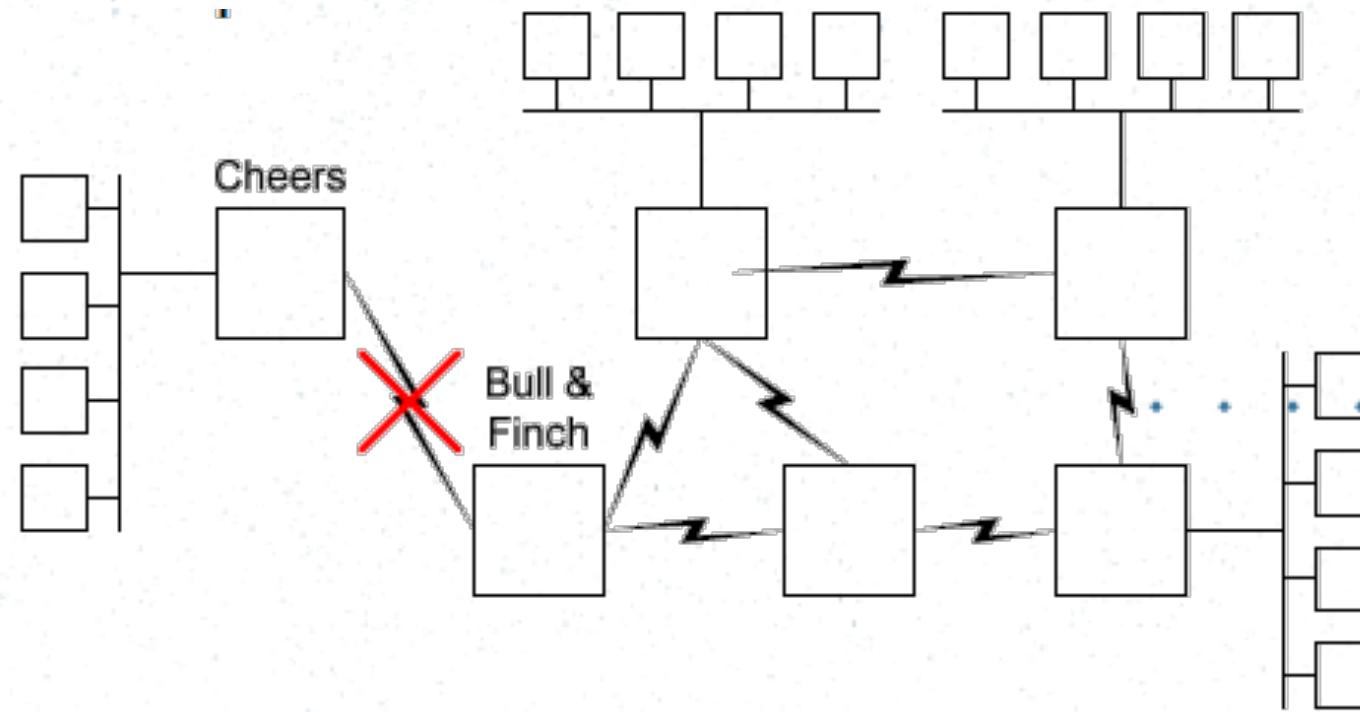
- ❑ Gathering data about the utilization of network resources.
  - Need to establish metrics -RFC 1272: “Internet Accounting Background”
    - E.g. Application layer – per-transaction, network layer – number of packets.
- ❑ Setting usage quotas using metrics.
- ❑ Billing users for their use of the network.

# Accounting Management - Benefits

- Allows effective measurement and reporting of accounting information.
- Increases the engineer's understanding of user utilization.
  - Helps the network engineer make informed decisions about the allocation of network resources.
  - Ensure that users have a fair share of the network.

# Fault Management

**Fault management** is the process of locating problems or faults, on the data network.



# Fault Management

- The fault management process involves:
  - Detecting symptoms that may lead to a problem/fault
    - in the network.
    - Isolating the cause of the symptoms.
      - Find correlations between symptoms and potential problems.
        - Alarms do not usually include explicit information regarding the exact location of the fault.
    - Correcting the problem if possible.

# Gathering Information for Fault Management

*Two methods..*

- Interrupt driven

- Critical events (e.g. link failure).
  - Solely relying on such events may not facilitate effective fault management.

- Polling

- Finds faults in a **timely manner**.
  - Higher bandwidth consumption.
  - Polling can be implemented using ICMP messages (ping).

# Fault Management - Benefits

- Enhances network reliability by providing tools to aid/facilitate rapid fault detection, isolation and recovery.
  - Maintains the illusion of complete and continuous connectivity between the users and the network.

# Network Management Systems

A Network Management System (NMS) comprises:

- An underlying **architecture** (aka platform).
  - A software package that provides **generic/basic functionality** of network management for managing a variety of network devices.
- A set of **applications** built on top of the platform.

# Network Management Systems

## Example NMSs

- Commercial implementations
  - HP Openview
  - SunConnect SunNet Manager
  - IBM Netview
- Freeware implementations
  - Net-SNMP (formally UCD-SNMP, CMU-SNMP)
  - OpenNMS

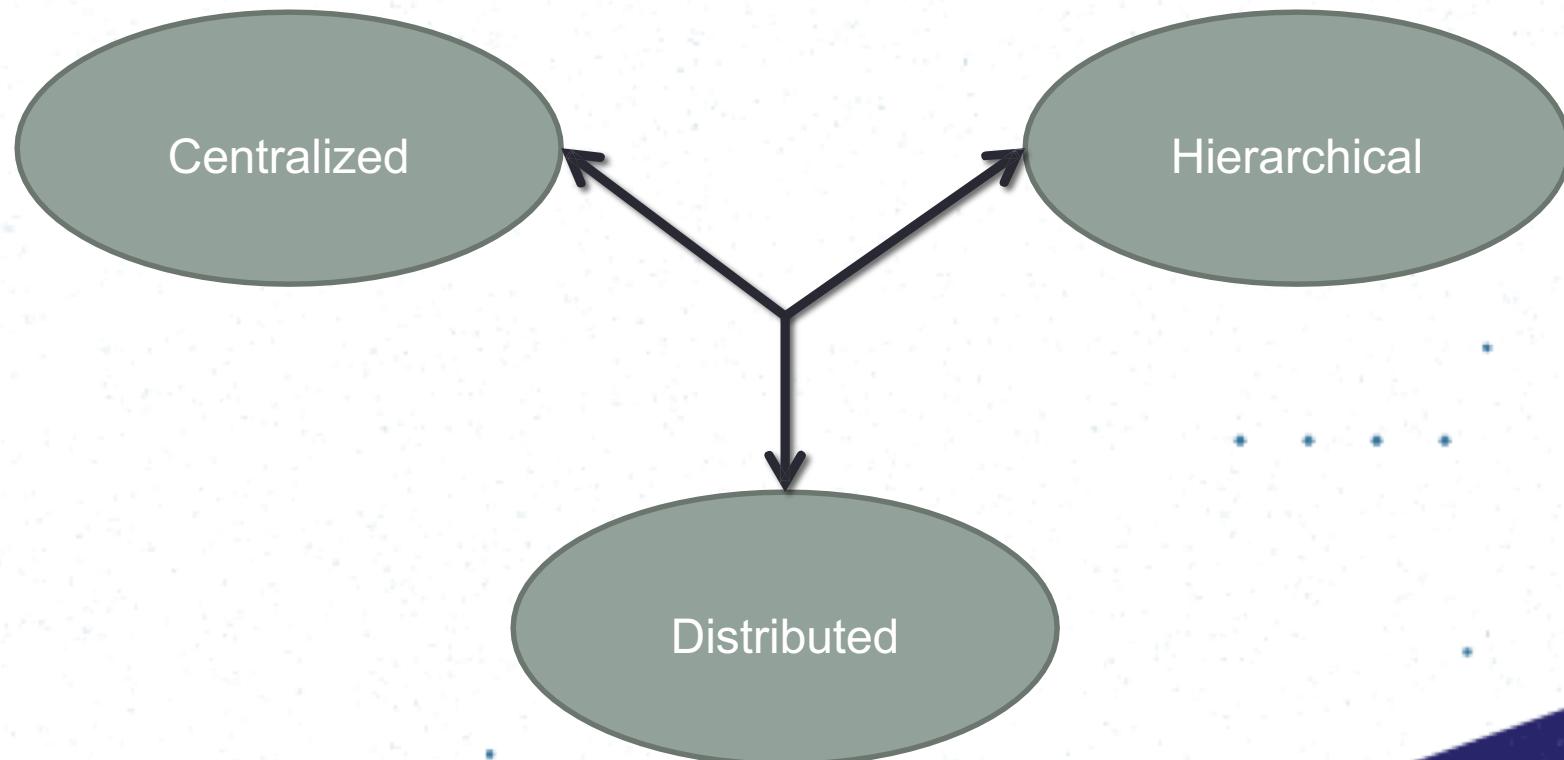
# Network Management Systems

The **platform** should include the following functionality:

- A user interface
- A network map
- A **Database Management System (DBMS) / Management Information Base (MIB)**
- A query language
- A customizable menu system
- An event log

# Network Management Architecture

An NMS platform can use **three architectures** to provide functionality.



# Next Lecture...!!!

## Network Mapping and Baselining

# Thank You



# IT3010

## Network Design and Management

### Lecture 03

#### Network Servers

## Shashika Lokuliyan

Faculty of Computing  
Department of CSE

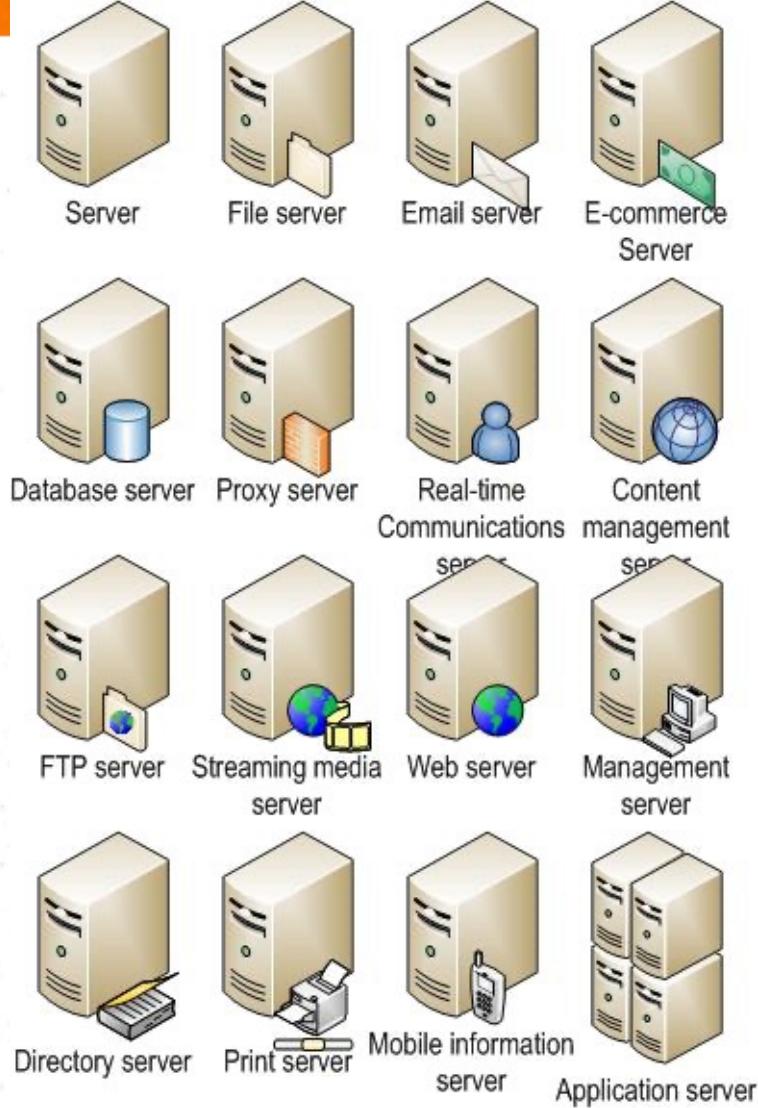


# SLIIT

*Discover Your Future*

# To be Covered...!!!

- ❖ Client – Server Architecture
  - ❖ Domain Name System (DNS)
  - ❖ Dynamic Host Configuration Protocol (DHCP)
  - ❖ Audit
  - ❖ Network Mapping
  - ❖ Baseling



# Networked applications cont..



- ❖ Read write data over network
- ❖ Dominant model : bidirectional, reliable byte stream connection
  - On-side reads what the other writes
  - Operates in both directions
  - Reliable (unless connection breaks)

# Server & Client



- **Server** is a piece of software that mange's a shareable resource.
- Usually the resource resides at one location in the network and the server is run on the computer at which the resource resides.
- The server offers acceptable level of service to the users.
- The mechanism of accessing this server are hidden from the network user by interface software which resides at the separate stations, usually referred to as the **client**.

# Client-server model

- Standard model for developing network applications
- Notion of client and server:
  - A server is a process that is offering some service.
  - A client is a process that is requesting the service
  - Server or client may be running in different machines.
  - Server waits for requests from client(s).
- Roles of the client and the server processes are asymmetric.

# Domain name system (DNS)



# *Domain Name System*



*“The Domain Name System (**DNS**) is a hierarchical distributed naming system for computers, services, or any resource connected to the Internet or a private network.*

Wikipedia

# Name server

- A name server is a computer hardware or software server that implements a network service for providing responses to queries against a directory service.
- It translates an often humanly-meaningful, text-based identifier to a system-internal, often numeric identification or addressing component.
- **What is Naming?**
  - A naming scheme must provide the facility to identify uniquely entities across the entire network.
  - Naming is associated with an addressing mechanism since it does not only provide a unique identifier but also the location of existence.

# What is dns?

- The Internet maintains two principal namespaces, the domain name hierarchy and the Internet Protocol (IP) address spaces (RFC 781)
  - A certain kinds of partial ordered sets
- The Domain Name System maintains the domain name hierarchy and provides translation services between it and the address spaces.

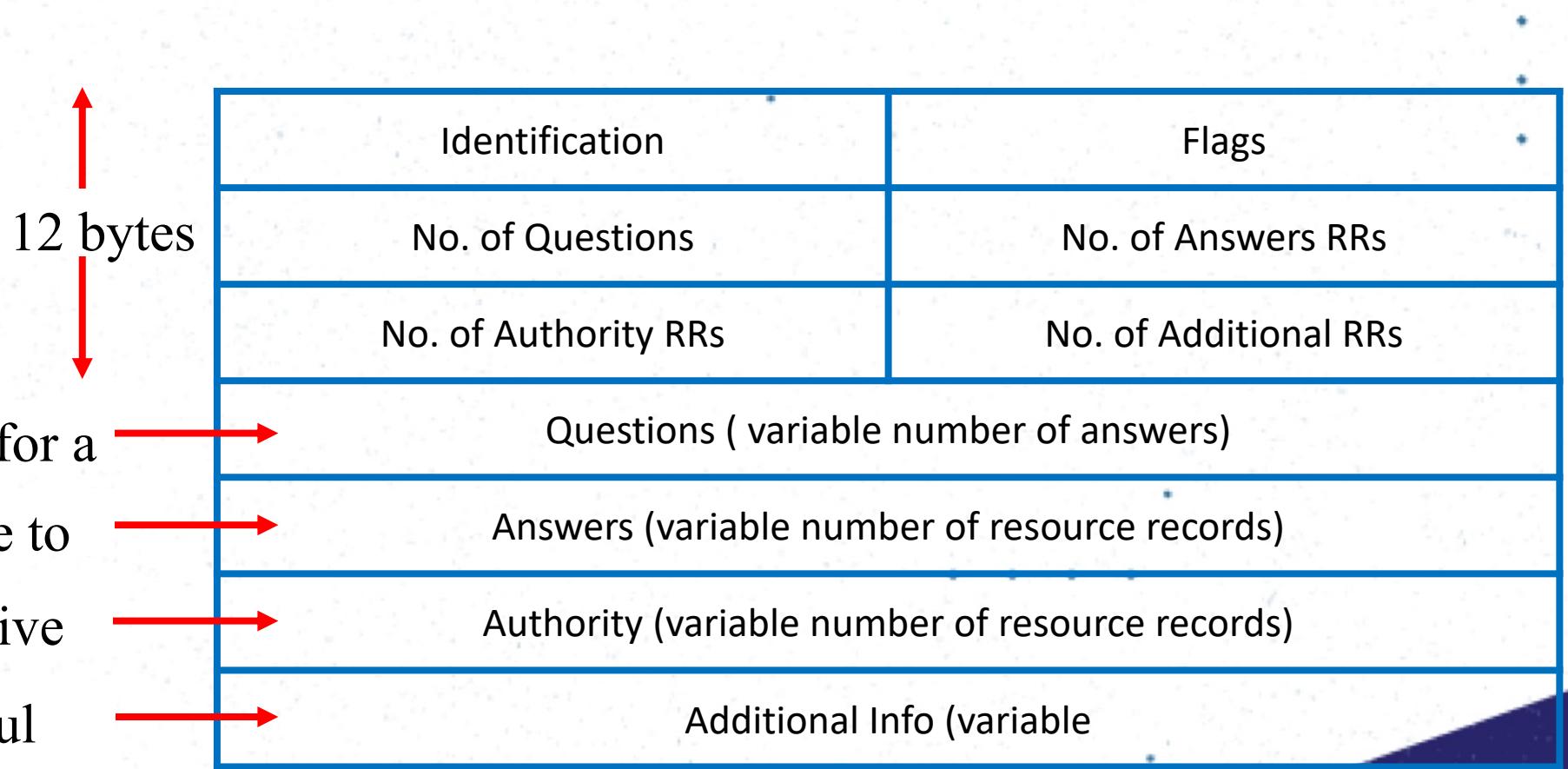
A container for a set of identifiers (aka symbols, names)

A ranked system

A unique name that identifies a website  
(RFC-1034) - URL

**Why use a domain name (URL) and not the IP ???**

# DNS message format



# DNS Header Fields

- ❖ Identification

- Used to match up request/response

- ❖ Flags 1-bit each to mark

- Query or response
  - Authoritative or not
  - Recursive resolution
  - To indicate support for recursive resolution

# DNS Record

RR format : *(class, name, value, type, ttl)*

- DB contains tuples called resource records (RRs)
- Classes = Internet (IN), Chaosnet (CH), etc.
- Each class defines value associated with type

# DNS Record cont.....

## For “IN” class:

- **Type = A**
  - **name** is hostname
  - **value** is IP address

- **Type = CNAME**
  - **name** is an alias name for some “canonical” name
  - **value** is canonical name

**Type = NS**

- **name** is domain (e.g. foo.com)
- **value** is name of authoritative name server for this domain

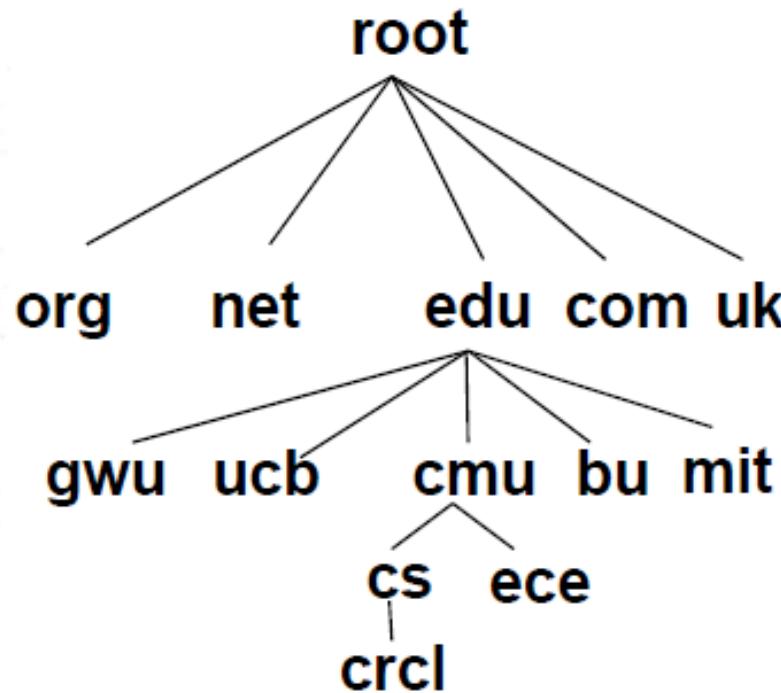
**Type = MX**

- **value** is hostname of mailserver associated with **name**

# Properties of DNS Host Entries

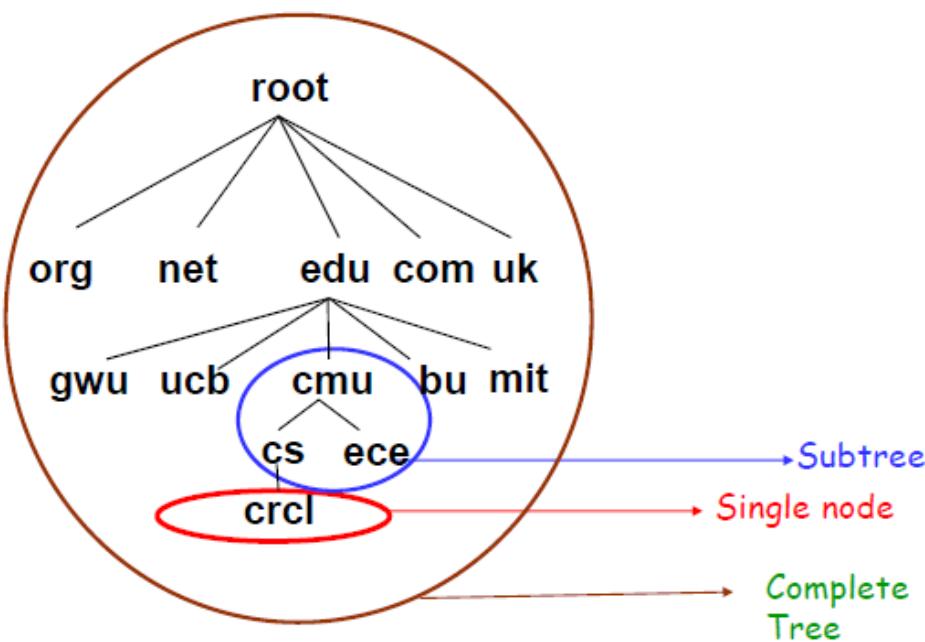
- Different kinds of mappings are possible:
  - ❖ 1-1 mapping between domain name and IP addr:
    - provolone.crcl.cs.cmu.edu maps to 128.2.218.81
  - ❖ Multiple domain names maps to the same IP addr:
    - www.scs.cmu.edu and www.cs.cmu.edu both map to 128.2.203.164
  - ❖ Single domain name maps to multiple IP addresses:
    - www.google.com map to multiple IP addrs.
  - ❖ Some valid domain names don't map to any IP addr:
    - crcl.cs.cmu.edu doesn't have a host

# DNS Design: Hierarchy Definitions



- Each node in hierarchy stores a list of names that end with same suffix
- Suffix = path up tree
- E.g., given this tree, where would following be stored:
  - Amal.com
  - Amal.edu
  - Amal.cmu.edu
  - Amal.crcl.cs.cmu.edu
  - Amal.cs.mit.edu

# DNS Design: Zone Definitions



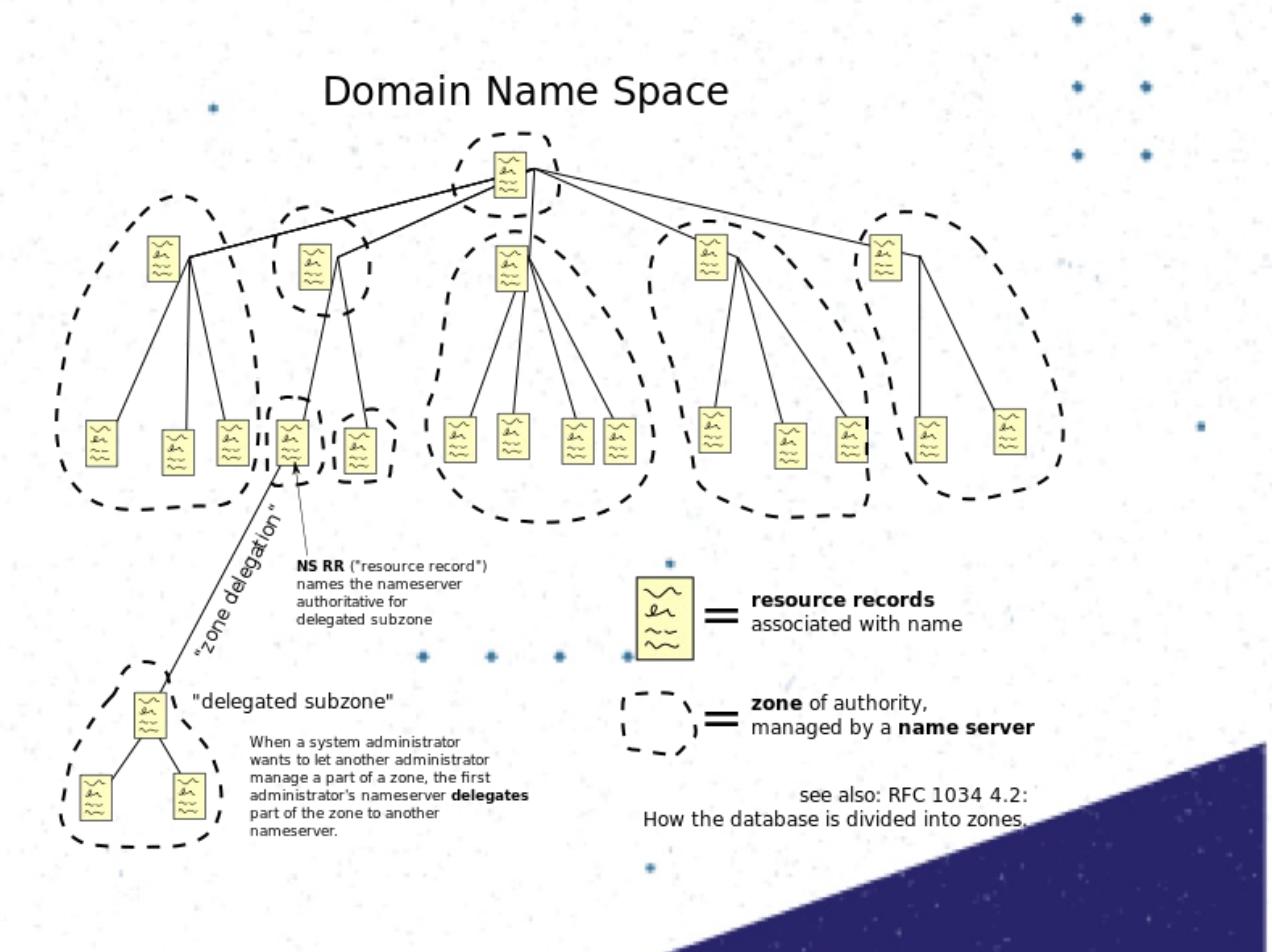
- Zone = contiguous section of name space
- E.g., Complete tree, single node or subtree
- A zone has an associated set of name servers
- Must store list of names and tree links

# DNS Design: cont...

- ❖ Zones are created by convincing owner node to create/delegate a subzone
  - Records within zone stored in multiple redundant name servers
  - Primary/master name server updated manually
  - Secondary/redundant servers updated by zone transfer of name space
    - Zone transfer is a bulk transfer of the “configuration” of a DNS server – uses TCP to ensure reliability
- ❖ Example:
  - CS.CMU.EDU created by CMU.EDU admins
  - Who creates CMU.EDU or .EDU?

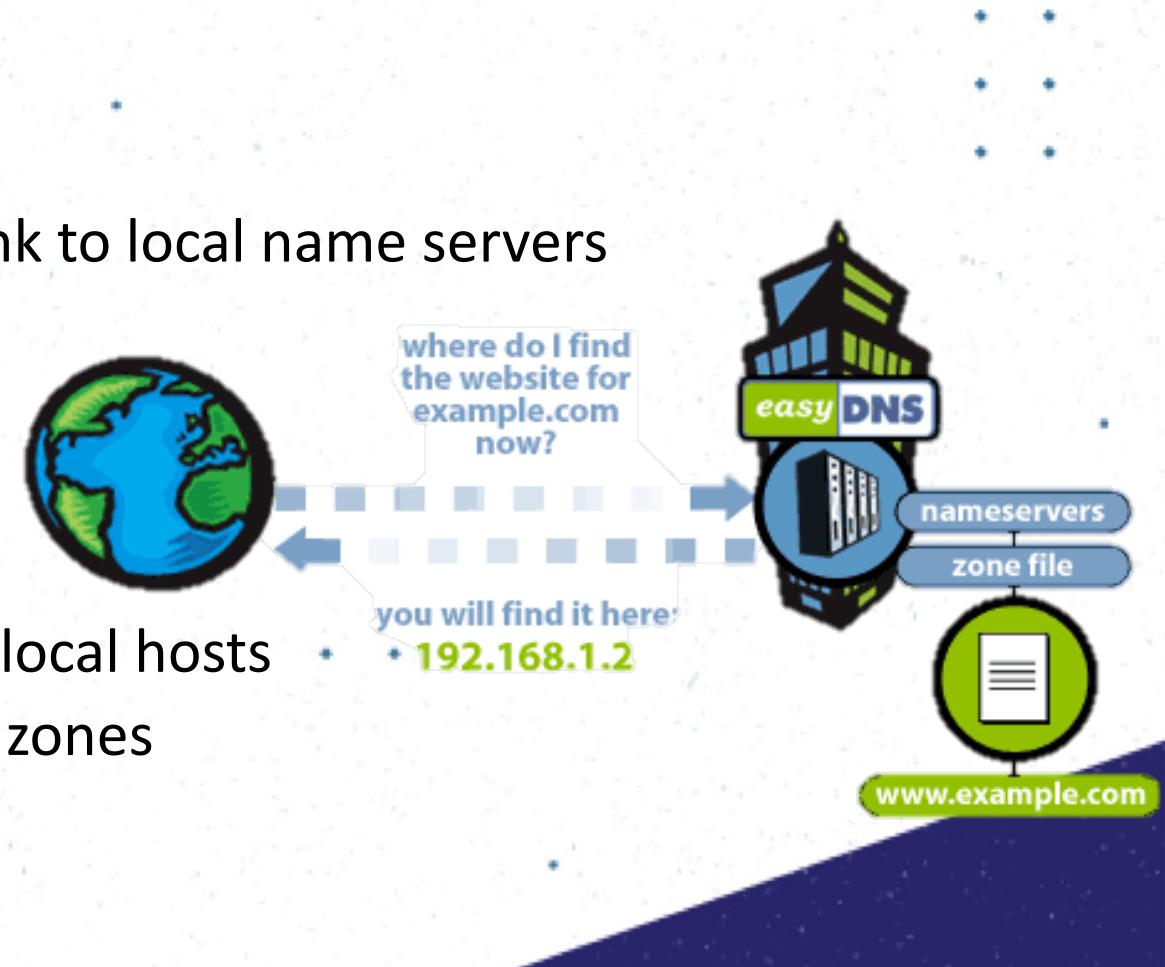
# DNS: Root Name Servers

- ❖ Responsible for “**root**” zone
- ❖ 13 root name servers
  - ❖ Currently {a-m}.root-servers.net
- ❖ Local name servers contact root servers when they cannot resolve a name



# Servers/Resolvers

- ❖ Each host has a resolver
  - Typically a library that application can link to local name servers (i.e. /etc/resolv.conf)
- ❖ Name server
  - Either responsible for some zone or
  - Local servers
    - Do lookup of distant host names for local hosts
    - Typically answer queries about local zones



# Lookup Methods

- **Recursive query:**

- Server goes out and searches for more information
- Only returns the final answer or “not found”

- **Iterative query:**

- Server responds with as much as it knows.
- “I don’t know this name but ask this server”

Workload impact on choice?

- ❖ Root/distant server does
- ❖ Local server typically does

# Workload and Caching

- DNS responses are cached
  - ❖ Quick response for repeated translations
  - ❖ Other queries may reuse some parts of lookup
    - E.g., NS records for domains
- DNS negative queries are cached
  - ❖ Don't have to repeat past mistakes
  - ❖ E.g., misspellings, search strings in resolv.conf

- Cached data periodically times out
  - ❖ Lifetime (TTL) of data controlled by owner of data
  - ❖ TTL passed with every record

# Reliability

- ❖ DNS servers are replicated
  - Name service available if  $\geq$  one replica is up
  - Queries can be load balanced between replicas
- ❖ UDP used for queries
  - Why not just use TCP?
- ❖ Try alternate servers on timeout
  - Exponential backoff when retrying same server
- ❖ Same identifier for all queries
  - Don't care which server responds

# Dynamic Host Configuration Protocol (DHCP)

192.168.1.18

192.168.1.19

192.168.1.20

# *Dynamic Host Configuration Protocol*



*The Dynamic Host Configuration Protocol (**DHCP**) is a standardized networking protocol used on Internet Protocol (**IP**) networks for dynamically distributing network configuration parameters, such as IP addresses for interfaces and services. With **DHCP**, computers request IP addresses and networking parameters automatically from a **DHCP** server, reducing the need for a network administrator or a user to configure these*

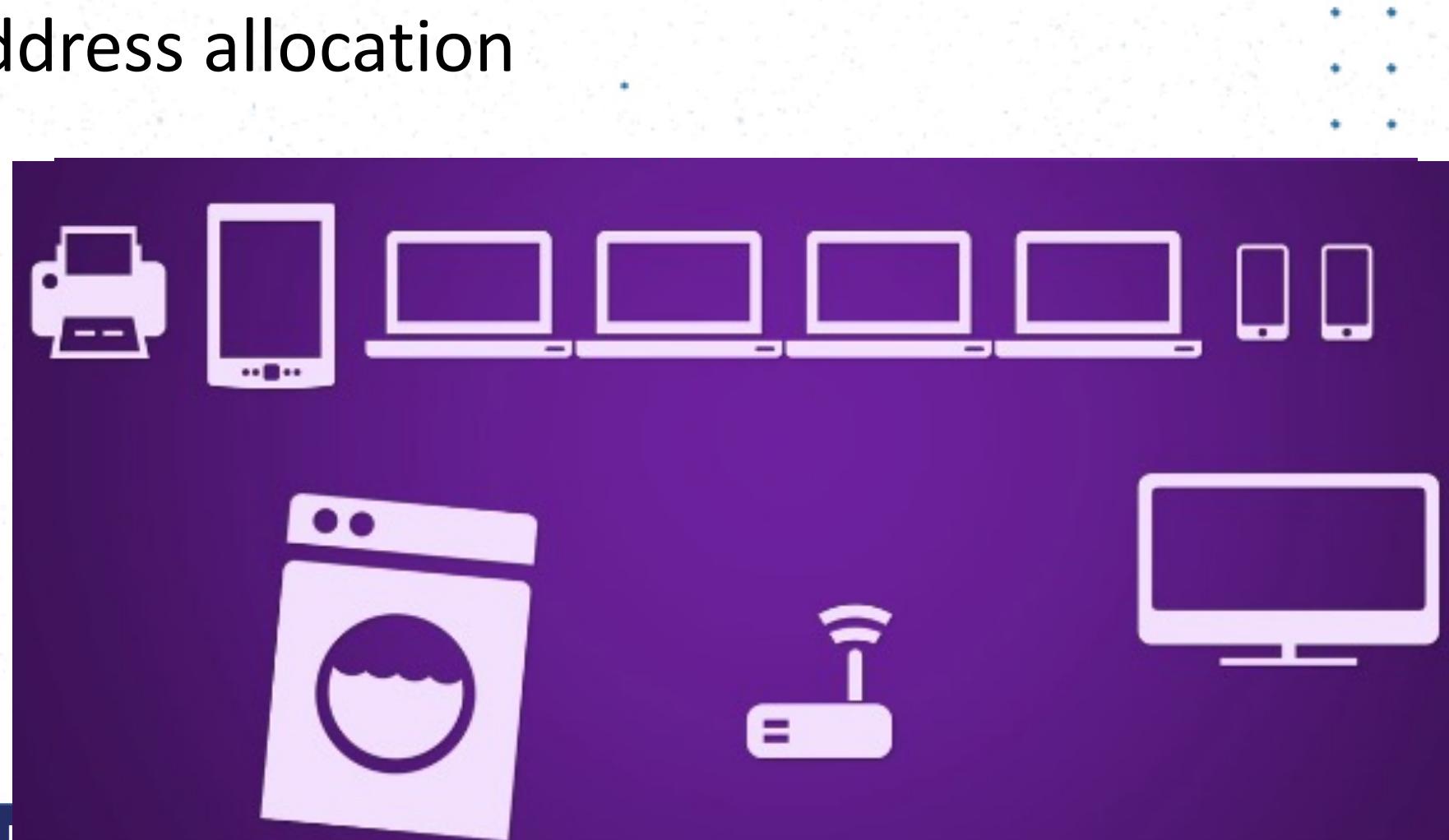
# What a Device needs



- Most computers today need four pieces of information:
  - 1) IP address => to uniquely define itself in a network
  - 2) Subnet mask => to define which network (or sub network) the device belongs to
  - 3) Address of a default router => to be able to communicate with other networks
  - 4) Address of a name server => to be able to use names instead of addresses.....

# A world without DHCP

- Manual IP address allocation



**How much time it would consume?  
What if it was wrongly configured and had to reconfigure again?**

# DHCP – Dynamic Host Configuration Protocol

- ❖ Issues or leases dynamic IP addresses to clients in a network
- ❖ The lease can be subject to various conditions
  - Duration
  - Computer ID etc.





# IP Address Assignment

- ❖ The DHCP server assigns or leases a client an IP address for a predetermined period of time
- ❖ In most cases, the IP address is automatically renewed when a client logs into a network
- ❖ The IP address assigned is taken from a pool of IP addresses defined as the scope of IP addresses available for assignment .

If a windows user:

A user can manually release and renew an IP address by typing the commands??

If Linux user??

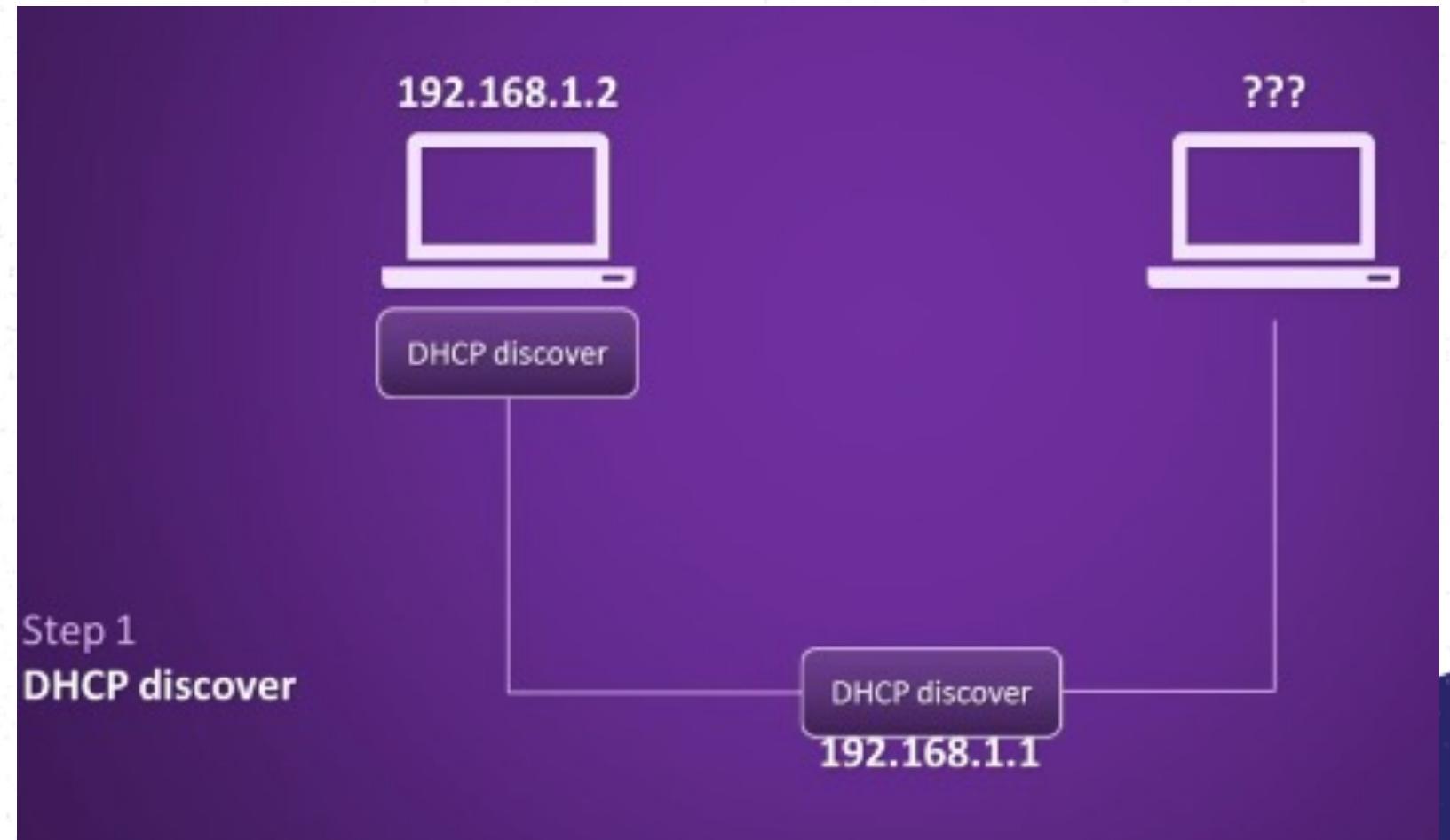


# Assignment Conditions

- ❖ IP addresses can be reserved for clients based on MAC addresses and host names
- ❖ For security, the lease of IP addresses can be restricted to clients with known MAC addresses
- ❖ Some IP addresses may be excluded so that they could be reserved for assignment to servers as static addresses
  - Servers, in general, requires the assignment of static addresses
  - The router address is also normally excluded
- ❖ Specific or a range of IP addresses may be excluded in this manner

# How DHCP server work

- Host searches for any available DHCP servers to get an address from.



DHCP Client UDP  
port – 68



# Multiple DHCP servers

- ❖ Multiple servers can respond with an address offer
  - ❖ New host chooses one offer
  - ❖ Servers see which offer the client picked.



## DHCP Server



## DHCP Client

# DHCP Message types

- ***DHCPDISCOVER***
- ***DHCPOFFER***
- ***DCHPREQUEST***
- ***DHCPACK***
- ***DHCPNAK***
- ***DHCPDECLINE***
- ***DHCPIINFORM***
- ***DCHPRELEASE***

# Audit

Network management should **start with an audit**,

- Document/Map the entire network.
- Evaluate and baseline the physical and data link layer infrastructure.
- Evaluate and baseline network traffic and protocols.
- Evaluate and baseline platforms, operating systems and applications.
- Evaluate security

# Network Mapping Definition

**Network mapping** in general is getting to know your network inside-out.

- Detailed description of everything
  - Complex networks are difficult to visualize
  - Big rewards
  - Time consuming, boring!
- . . . . .

# Network Mapping OSI

- Physical Layer
- Data Link Layer
- Network Layer
- Transport Layer
- Session Layer
- Presentation Layer
- Application Layer

# OSI model: Open Systems Interconnection model

- The OSI model defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.
- • •
- Forouzan, *TCP/IP Protocol Suite*, Section 2.2 provides a concise description about OSI model. Following subsections are a summary of this reference. You are required to read this section of the book.
- • •

# Physical Layer

- Coordinates the functions required to **carry bit streams over the physical medium**.
  - Deals with the mechanical and electrical specifications of the interface and transmission media.
  - Defines the procedures and functions that physical devices and interfaces have to perform for transmission to occur.

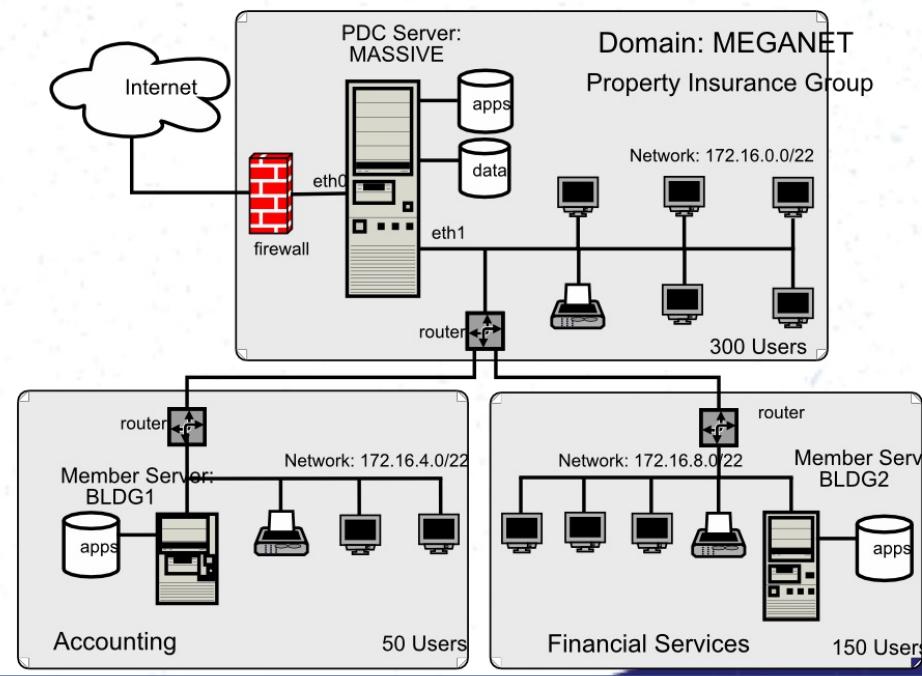
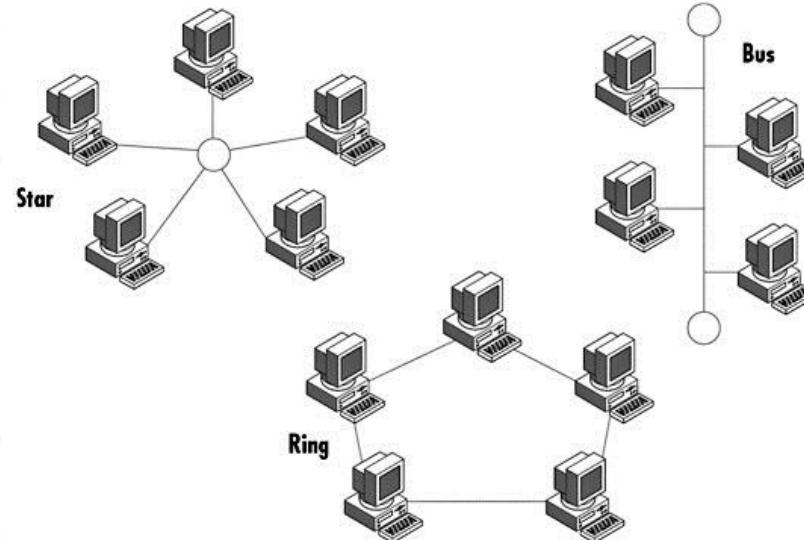
## Key points:

Physical characteristics of interfaces and media,  
Representation of bits, Data rate, Synchronization of bits,  
Line configuration, Physical topology, Transmission mode.

# Mapping the Physical Layer

- The Biggest Job
  - Every Device
  - Cabling Patch Panels
- Topology and Topography

# Topology Vs. Topography



# Data Link Layer

- This layer transforms the physical layer (a raw transmission facility), **to a reliable link**.
- It makes the physical layer appear **error free** to the upper layer (network layer).
- The data link layer is divided into two sub layers:
  - **The Media Access Control (MAC) layer**  
The MAC sub layer controls how a computer on the network gains access to the data and permission to transmit it.
  - **Logical Link Control (LLC) layer**  
The LLC layer controls frame synchronization, flow control and error checking.

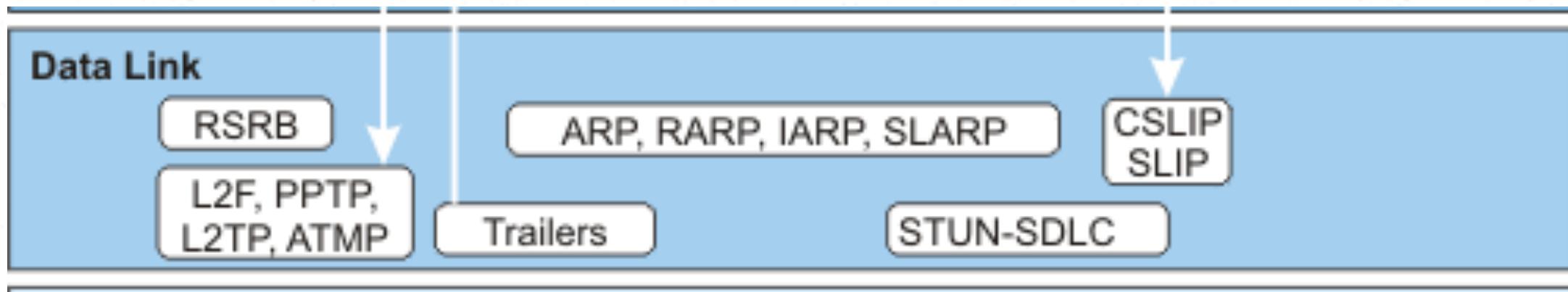
## Key points:

Framing, Physical addressing, Flow control, Error control,  
Access control to the link.

# Mapping the Data Link Layer

## ▪ NIC (Network Interface Card)

A network interface card (NIC) is a hardware component without which a computer cannot be connected over a network. It is a circuit board installed in a computer that provides a dedicated network connection to the computer. It is also called **network interface controller**, **network adapter** or **LAN adapter**.



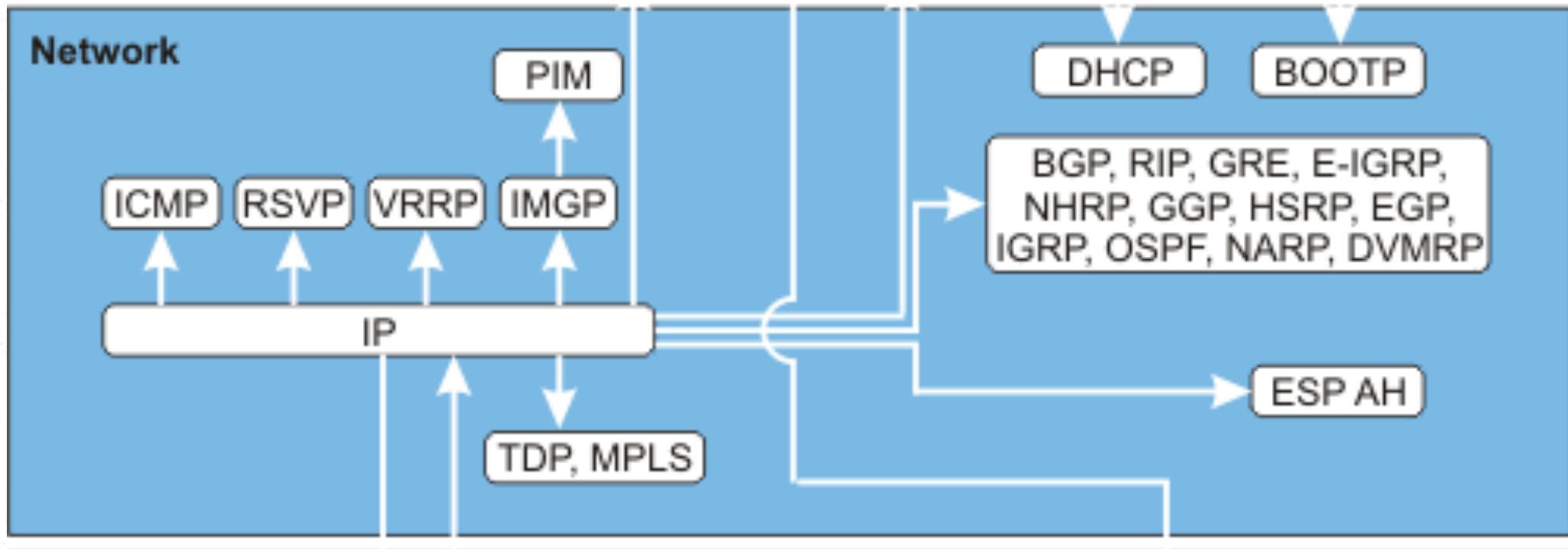
# Network Layer

- Network layer is responsible for the **source-to-destination delivery** of a packet, whereas, the data link layer oversees the hop-to-hop delivery.
- Ensures that each packet gets from its point of origin to its final destination.

**Key points:**

Logical addressing, Routing.

# Mapping the Network Layer



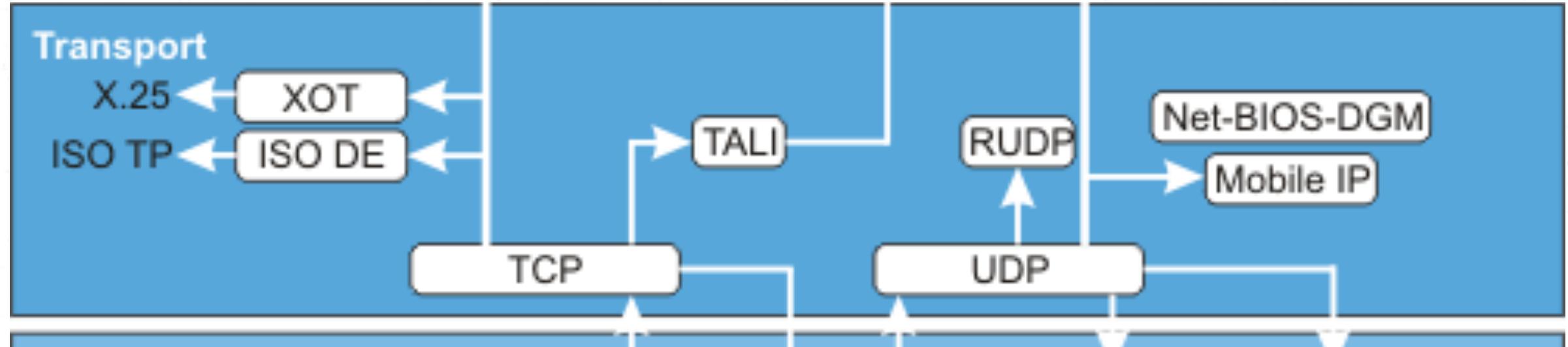
# Transport Layer

- Transport layer is responsible for process-to-process delivery of the entire message.
  - Ensures that the whole message arrives intact and in-order.

## Key points:

Service-point addressing (aka port addressing),  
Segmentation and reassembly, Connection control, Flow  
control, Error control.

# Mapping the Transport Layer



# Session Layer

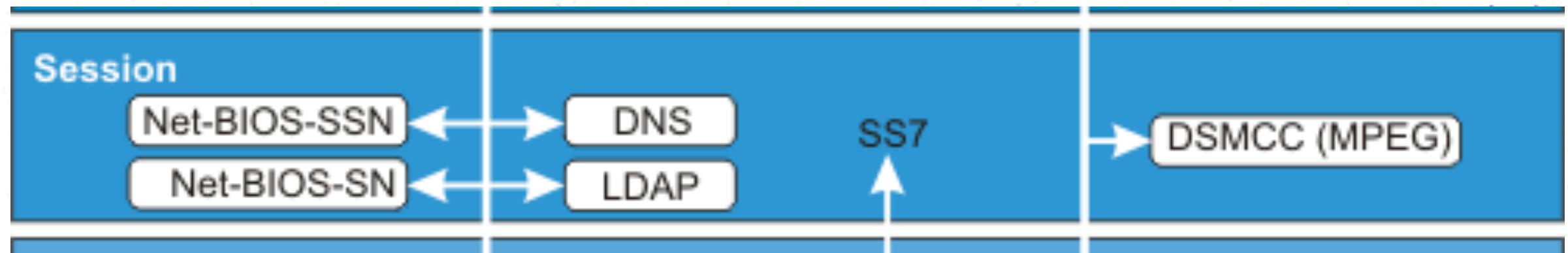
- This layer is considered as the network **dialog controller**.
  - It establishes, maintains, and synchronizes the interaction between communicating systems.

## Key points:

# Dialog control, Synchronization.



# Mapping the Session Layer



# Presentation Layer

- Presentation layer is concerned with the **syntax and semantics** of the information exchanged between two systems.

⋮ ⋮ ⋮

## Key points:

Translation, Encryption, Compression.

## Mapping the Presentation Layer

- Type of encryption used.
- Type of compression used.

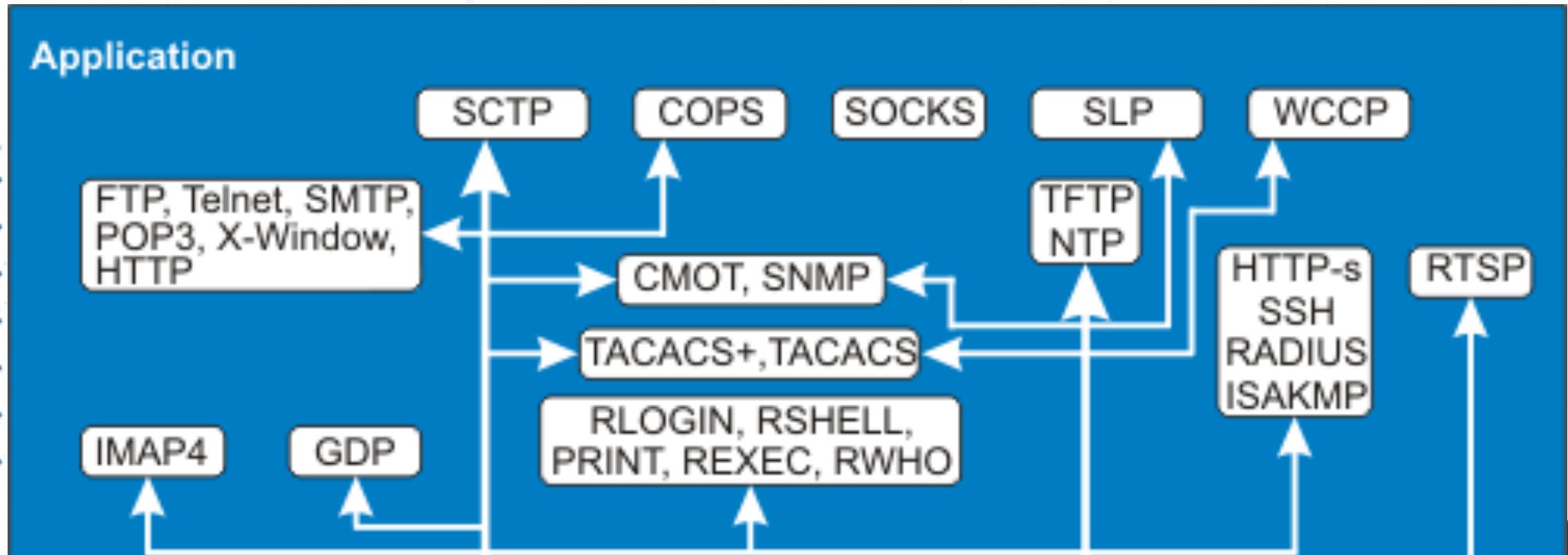
# Application Layer

- Enables the users, human or software, to **access** the network.
- Provides user interfaces and support for services such as electronic mail, remote file access and transfer, shared database management, and other types of distributed information services.

## Key points:

Network virtual terminal (aka remote terminal), File transfer, access and management (FTAM), Mail services, Directory services.

# Mapping the Application Layer



# Non Network Information

Non network information refers to the information that are directly not corresponding to you networking principles, BUT, is vital to the day-to-day management of the network related tasks. For example,

- Network purpose statement.
- Network overview documentation.
- Physical locations.
- Vendors.
- Signatories.
- Etc..

# Non Network Information

## Physical Locations

- Floor Plans
  - Pictures
  - Fire exits
- Addresses
  - Visitor entrances
  - Deliveries
  - Ship to
  - Driving directions
- Managers (with contact name and phone numbers)
  - IT Infrastructure
  - Non IT Infrastructure (e.g., HVAC - *heating, ventilation, and air conditioning*, hallways, offices, etc.)
  - Others

# Non Network Information

## Resources

- Account management
- Usernames and passwords for web resources you utilize
- ▪ ▪ ▪

## Signatories

- Who makes decisions?
- Who can authorize purchases?
- ▪ ▪ ▪

## Suppliers/Vendors

- List of all contractors who work on your network
- List of all vendors you purchase equipment from
- List of all service contracts (and/or warranty fulfillment)

# Network Mapping Tools

**Network mapping tools** can make your life easy by assisting you with many network mapping related tasks.

## Open source

- Nagios
- OpenNMS
- knetmap

## Commercial

- SmartDraw™
- Visio
- netViz™
- Neon LANsurveyor

# Baselining

- Why?
  - What?
  - When?
  - How?



# Baselining

- Optimize quality of service.
  - Gather performance data.
  - Analyze the data.
  - Determine appropriate performance thresholds.
- The act of measuring and rating the performance of a network in real-time situations.
  - [http://www.webopedia.com/TERM/n/network\\_baselining.html](http://www.webopedia.com/TERM/n/network_baselining.html)
- Comparing current performance to a historical metric, or “baseline”.
  - <http://en.wikipedia.org/wiki/Baselining>

# Network Baselining

[http://www.webopedia.com/TERM/n/network\\_baselining.html](http://www.webopedia.com/TERM/n/network_baselining.html)

- Network baselining is the act of measuring and rating the performance of a network in real-time situations.
- Providing a network baseline requires testing and reporting of the physical connectivity, normal network utilization, protocol usage, peak network utilization, and average throughput of the network usage.
- Such in-depth network analysis is required to identify problems with speed and accessibility, and to find vulnerabilities and other problems within the network.
- Once a network baseline has been established, this information is then used by companies and organizations to determine both present and future network upgrade needs as well as assist in making changes to ensure their current network is optimized for peak performance.

# Why Baseline

- To determine normal operating conditions
  - To identify and forecast problems
  - Troubleshooting
  - Predict network operation
  - Predict the ability to handle new tasks (scaling)
  - Optimization
- . . . . .

# When to Baseline

- Begin immediately
- Long term
- Use the baseline to determine the baseline schedule
- Special attention areas
- All levels of system activity

# How to Baseline

- Determine what we have (inventory)
- Determine what needs to be measured
- Determine when it needs to be measured
- Use the long term baseline to determine how often items need to be measured
- Repeat the measurements regularly
- Implement a way of obtaining alerts
- Implement a way of detecting trends
- Create a data repository

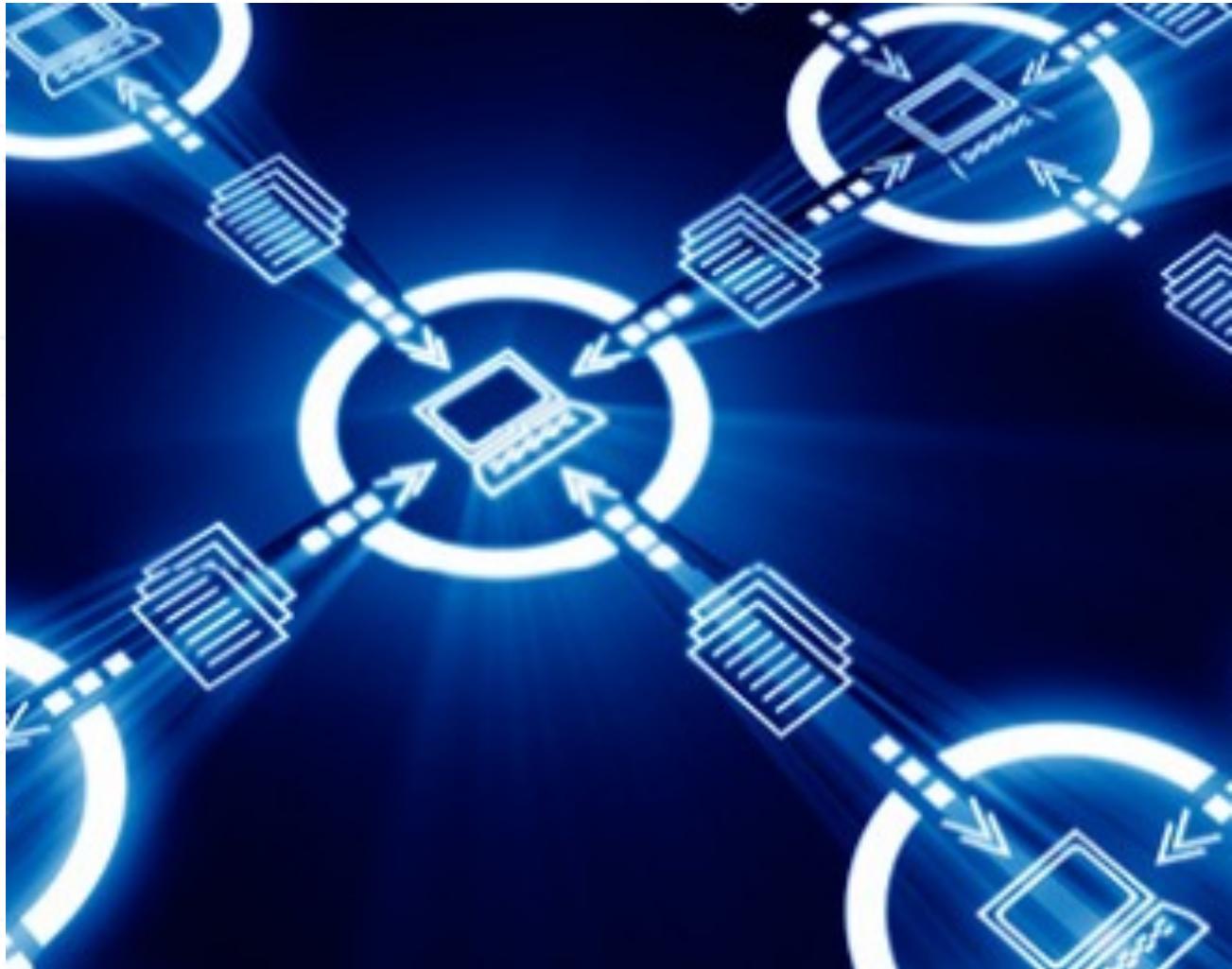
# Care When Baselining

- The very act of recording data for a baseline can skew the results
  - Known as **measurement degradation**, it may occur
    - Because the act of measuring an object's performance may increase its workload
    - If too short a time interval is used when measuring
    - When measuring multiple objects on a single system
- A base line is not an analysis, it is a tool that can be used to do analysis
  - Do not start the analysis until you map the network

# Analysis

- What are normal operating conditions?
- What are the peak operating conditions?
- Why are the conditions the way they are?
- How will problems be assessed?
- How will modifications be assessed?

Questions???



# Self Review

# Glossary (aka *Vocabulary*)

## ROI – Return On Investment

ROI is an accounting formula used to obtain an actual or perceived future value of an expense or investment.

## SPOF – Single Point Of Failure

A generic phrase for any component of a system that upon failure will cause a malfunction in the entire system. A SPOF can be a hardware or electrical component or a software component. Each time a system expands (e.g., adding a workstation to a network or adding a new application to a network of workstations) the number of places where an SPOF can occur also expands.

# Glossary (aka Vocabulary)

- MTBF – Mean Time Between Failures
- The average time a device will function before failing. MTBF ratings are measured in hours and indicate the sturdiness of hard disk drives and printers.
- Typical disk drives for personal computers have MTBF ratings of about 500,000 hours. This means that of all the drives tested, one failure occurred every 500,000 hours of testing. Disk drives are typically tested only a few hours, and it would be unlikely for a failure to occur during this short testing period. Because of this, MTBF ratings are also predicted based on product experience or by analyzing known factors such as raw data supplied by the manufacturer.

# Glossary (aka *Vocabulary*)

## MTTR – Mean Time To Repair

In data storage, MTTR is the average time before an electronic component can be expected to require repair.

## AFR – Annualized Failure Rate

Is the relation between the MTBF and the hours that a number of devices are run per year, expressed in percent. AFR does not specifically apply to a single component, but rather to a population of like components.

# Glossary (aka *Vocabulary*)

## Uptime

Amount of time the utility is available to users.

⋮ ⋮  
⋮ ⋮  
⋮ ⋮

## Downtime

Amount of time the utility is unavailable to users.

## Availability

Percentage of time the utility is available to the user.

# Proxy Server



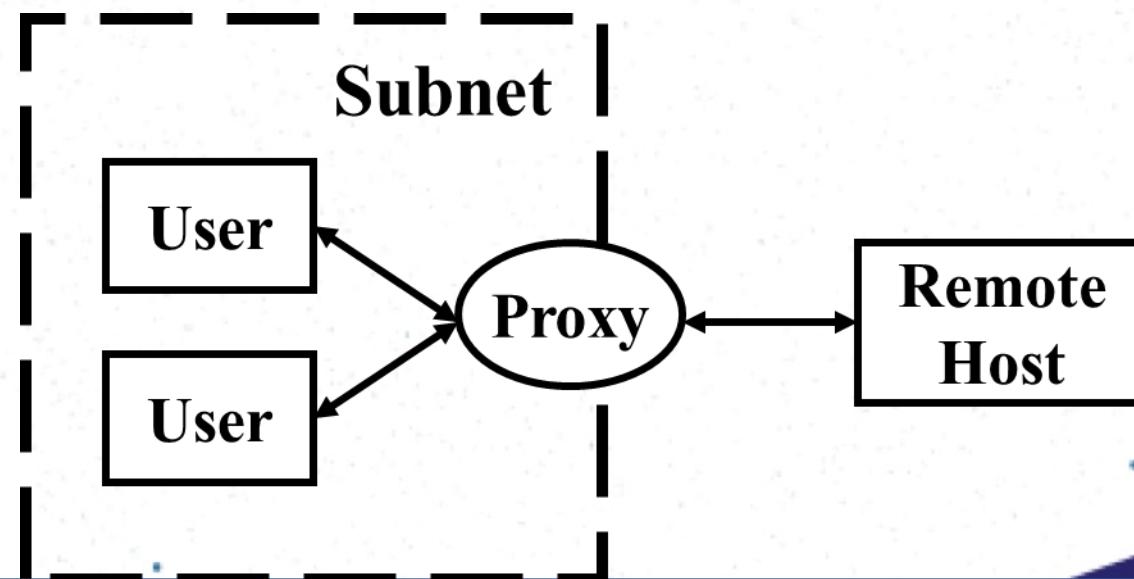
# Proxy server



*In computer networks, a **proxy server** is a server (a computer system or an application) that acts as an intermediary for requests from clients seeking resources from other servers. Proxies were invented to add structure and encapsulation to distributed systems. Today, most proxies are web proxies, facilitating access to content on the World Wide Web and providing anonymity.*

# Basic concept

- A proxy server is usually associated with or part of a gateway server that separates the subnet from the outside network and a firewall server that protects the subnet from outside intrusion



# Proxy Servers

- ❖ Part of an overall Firewall strategy
- ❖ Sits between the local network and the external network
  - Originally used primarily as a caching strategy to minimize outgoing URL requests and increase perceived browser performance
  - Primary mission is now to insure anonymity of internal users
    - Still used for caching of frequently requested files
    - Also used for content filtering
- ❖ Acts as a go-between, submitting your requests to the external network
  - Requests are translated from your IP address to the Proxy's IP address
  - E-mail addresses of internal users are removed from request headers
  - Cause an actual break in the flow of communications

# Types of proxy

- **Forwarding proxies**
- Forward proxies are proxies in which the client server names the target server to connect to. Forward proxies are able to retrieve from a wide range of sources (in most cases anywhere on the Internet).

# Types of proxy

- **Open proxies**
- An open proxy is a forwarding proxy server that is accessible by any Internet user. According to estimates there are "hundreds of thousands" of open proxies on the Internet. An anonymous open proxy allows users to conceal their IP address while browsing the Web or using other Internet services.

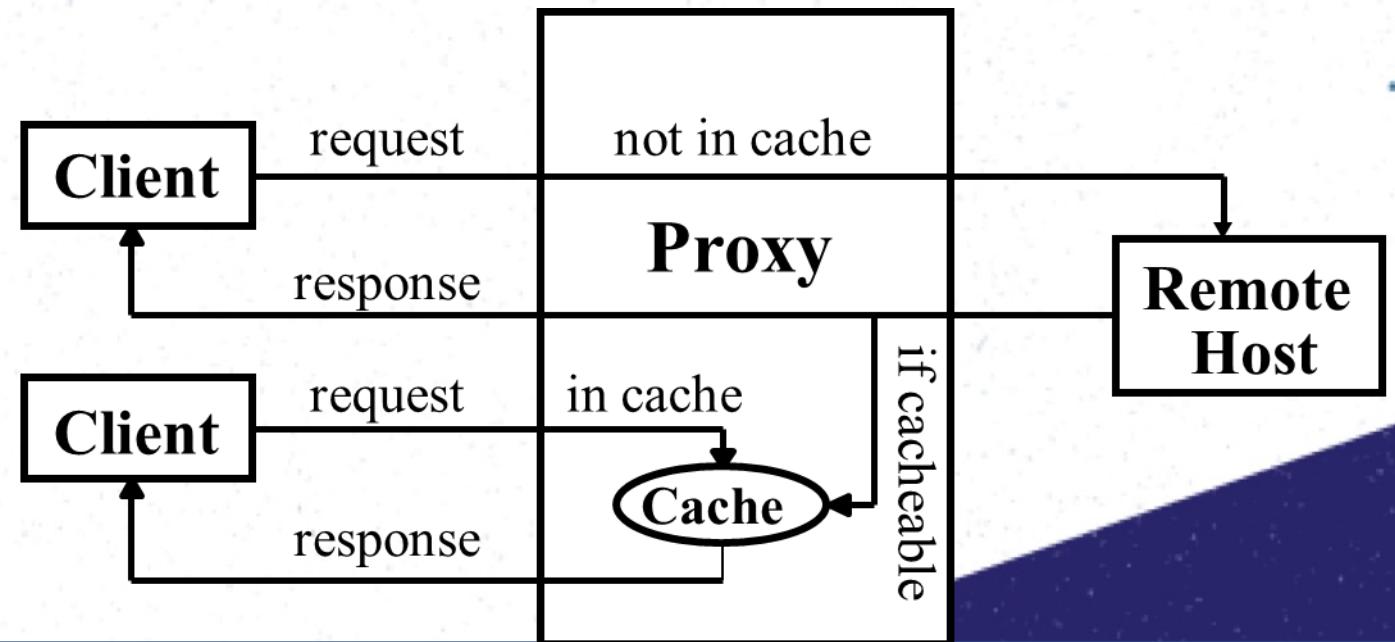
# Types of proxy

- **Reverse proxies**
- A reverse proxy (or surrogate) is a proxy server that appears to clients to be an ordinary server. Requests are forwarded to one or more proxy servers which handle the request. The response from the proxy server is returned as if it came directly from the origin server, leaving the client no knowledge of the origin servers.

# Proxy Cache

- One of the most important uses of Proxy, is as a Cache Server. Cache mechanism allows saving some cacheable requests for later recall by any user and thus reduce both latency and Internet traffic.

- Browser caches
- Proxy caches
- Server cache



# WEB Server



# *web server*

*The term web server, also written as Web server, can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.*

- Wikipedia

# Web server



- ❖ HTTP (Hyper Text Transfer Protocol) is used to transfer web pages from a Web Server to Web Client (Browser)
- ❖ Web Pages are arranged in a directory structure in the Web Server
- ❖ HTTP supports CGI (Common Gateway interface)
- ❖ HTTP supports Virtual Hosting (Hosting multiple sites on the same server)



## Popular Web Servers

- Apache
- Windows IIS
- nginx
- GWS



## Web cache

Squid  
Polipo  
Traffic server

## Web server

Apache  
Cherokee  
Lighttpd  
Nginx

## CGI scripting

Perl  
PHP  
Python

## Database

MariaDB  
MySQL  
Drizzle

## Linux kernel

AppArmor  
SELinux  
Smack  
TOMOYO

Process Scheduler

Netfilter

Linux network stack

Network scheduler

NIC  
device  
driver

kmod-fs-ext4  
kmod-fs-btrfs  
Lustre  
...



## Hardware

CPU  
&  
RAM

Networking  
hardware

Storage

SATA  
SAS  
RAID  
iSCSI  
NAS

## Environment: CCC

### Crackers

Botnets for DDoS-attacks  
cracking attempts

### Competitors

compete for customers

### Internet

Attacks  
stave off  
&  
Requests  
serve

Responses  
low latency

### Customers

want attendance

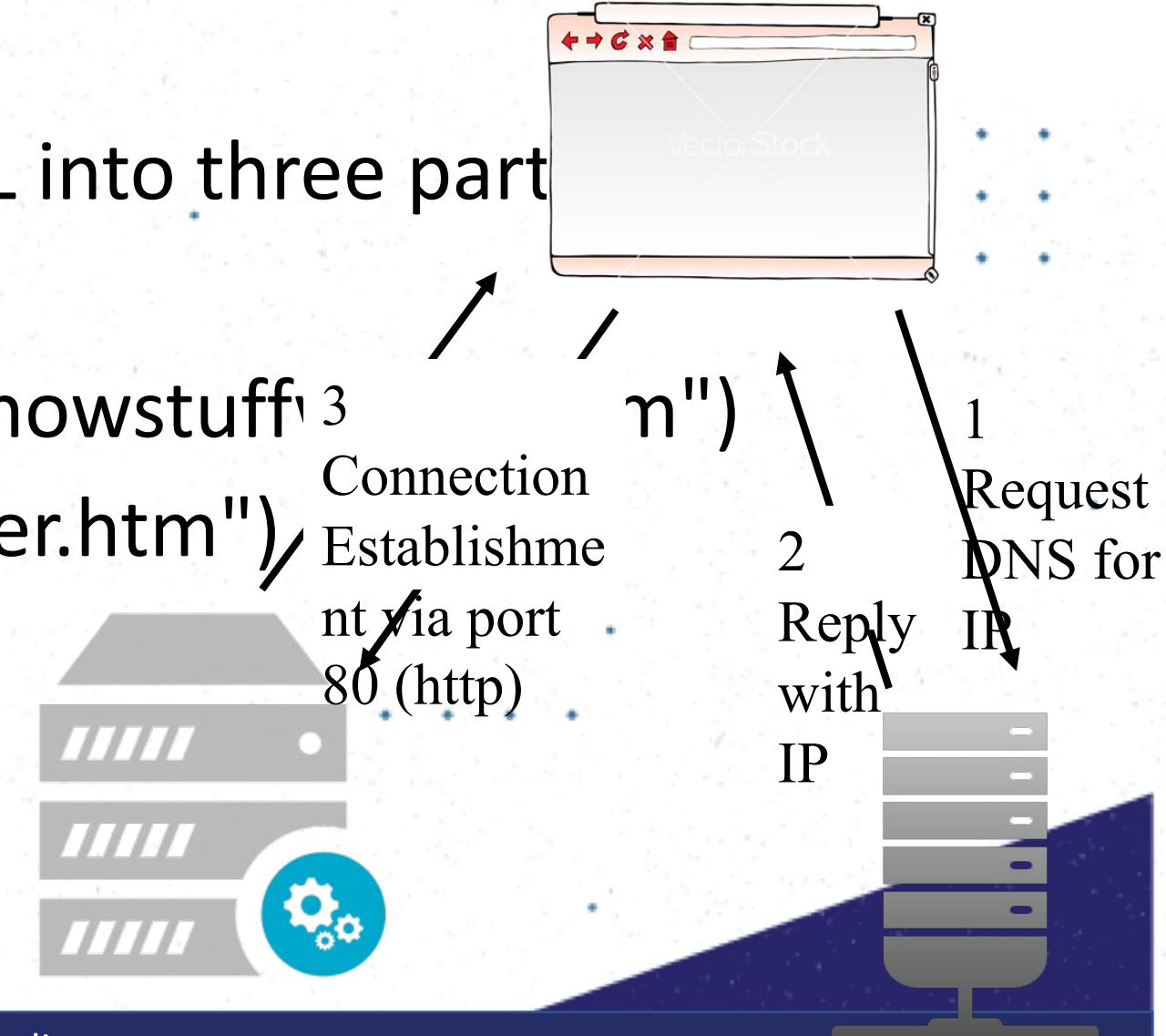
### Botnets

DDoS-Attacks

- LAMP web service solution stacks, which is highly suitable for building dynamic web sites and web applications.

# Behind the Scenes

- The browser broke the URL into three parts
  - 1. The protocol ("http")
  - 2. The server name ("www.howstuff")
  - 3. The file name ("web-server.htm")



# MAIL Server



# *mail server*

*Within Internet message handling services (MHS), a message transfer agent or mail transfer agent (MTA) or mail relay is software that transfers electronic mail messages from one computer. An MTA implements both the client (sending) and server (receiving) portions of the Simple Mail Transfer Protocol.*

# Mail server



- ❖ Simple Mail Transfer Protocol (SMTP) is used to transfer mail between Mail Servers over Internet
- ❖ Post Office Protocol (PoP) and Interactive Mail Access Protocol (IMAP) is used between Client and Mail Server to retrieve mails
- ❖ The mail server of a domain is identified by the MX record of that domain

Popular Mail Servers:

- Sendmail/Postfix
- Microsoft Exchange Server
- IBM Lotus



# OTHER Server types



# Application server



- Also called an appserver, an application server is a program that handles all application operations between users and an organization's backend business applications or databases.
- An application server is typically used for complex transaction-based applications. To support high-end needs, an application server has to have built-in redundancy, monitor for high-availability, high-performance distributed application services and support for complex database access.

# File server



- In the client/server model, a file server is a computer **responsible for the central storage and management of data files so that other computers on the same network can access the files.**
- A file server allows users to share information over a network without having to physically transfer files by floppy diskette or some other external storage device.

Ex. ://public drive at sliit

# File server



- A file server may be :
  - An ordinary PC that handles requests for files and sends them to the network.
  - A dedicated network-attached storage (NAS) device that also serves as a remote hard disk drive for other computers, allowing anyone on the network to store files on it as if it were their own hard drive.
  - ▪ ▪ ▪

# Print server



- A print server, or printer server, is a device that connects printers to client computers over a network. It accepts print jobs from the server computers and sends the jobs to the appropriate printers, queuing the jobs locally to accommodate the fact that work may arrive more quickly than the printer can actually handle it.
- Print servers may support a variety of industry-standard or proprietary printing protocols including Internet Printing Protocol, Line Printer Daemon protocol, NetWare, NetBIOS/NetBEUI, or JetDirect.



# IT3010

## Network Design and Management

### Lecture 04

#### Introduction to SNMP

# Shashika Lokuliyan

Faculty of Computing

Department of CSE



# SLIIT

*Discover Your Future*

# **SNMP**

## Simple Network Management Protocol

# Network management

- In computer networks, network management refers to the **activities, methods, procedures, and tools** that pertain to the operation, administration, maintenance, and provisioning of **networked systems**. Network management is essential to **command** and **control** practices and is generally carried out of a network operations center.



# network management

1. *Operation* deals with keeping the network (and the services that the network provides) up and running smoothly.
2. *Administration* deals with keeping track of resources in the network and how they are assigned.
3. *Maintenance* is concerned with performing repairs and upgrades.
4. *Provisioning* is concerned with configuring resources in the network to support a given service.

# network management

- Data for network management is collected through several mechanisms.
  - *Agents* installed on infrastructure
  - Synthetic monitoring that simulates transactions
  - Logs of activity
  - Sniffers
  - Real user monitoring

In the past network management mainly consisted of monitoring whether devices were up or down; today performance management has become a crucial part of the IT team's role which brings about a host of challenges—especially for global organizations.

# Need For Network Management Tools

- In the early days of the Arpanet, the predecessor of the Internet, the name service was accomplished by maintaining and distributing one file with all the IP addresses of the network. But no more ... DNS etc
- As networks increase in size
  1. The network becomes more indispensable to the organization.
  2. More things can go wrong, disabling or degrading the performance of portions of the network.
- Today a large network cannot be managed with software assistance.

# SNMP & Network Management History

- **1983** - TCP/IP replaces ARPANET at U.S. Dept. of Defense, effective birth of Internet
- First model for net management - **HEMS** - High-Level Entity Management System (*RFCs 1021, 1022, 1024, 1076*)
- **1987** - ISO OSI proposes **CMIP** - Common Management Information Protocol, and **CMOT** (CMIP over TCP) for the actual network management protocol for use on the internet
- **Nov. 1987** - **SGMP** - Simple Gateway Monitoring protocol (*RFC 1028*)
- **1989** - Marshall T. Rose heads up **SNMP** working group to create a common network management framework to be used by both **SGMP** and **CMOT** to allow for transition to **CMOT**

# SNMP & Network Management History

- Aug. 1989 - “Internet-standard Network Management Framework” defined (RFCs 1065, 1066, 1067)
- Apr. 1989 - **SNMP** promoted to **recommended** status as the de facto TCP/IP network management framework (RFC 1098)
- June 1989 - IAB committee decides to let **SNMP** and **CMOT** develop separately
- May 1990 - IAB promotes **SNMP** to a **standard protocol with a recommended status** (RFC 1157)
- Mar. 1991 - format of MIBs and traps defined (RFCs 1212, 1215)
- TCP/IP MIB definition revised to create **SNMPv1** (RFC 1213)

# Technologies

- Main reason for having a standardized protocol was that without one in a heterogeneous environment, the network engineer would be spending too much time developing customized management tools instead of managing the network.
- A small number of accessory methods exist to support network and network device management. Access methods include,
  - SNMP
  - command-line interface
  - custom XML
  - CMIP
  - Windows Management Instrumentation (WMI)
  - Transaction Language 1
  - CORBA
  - NETCONF
  - Java Management Extensions (JMX)

# Use of Standardized Protocol

- Goals:
  - Minimize complexity of management functions.
  - Flexible, extensible.
  - Independent of the architecture and mechanisms of particular hosts and gateways.

# SNMP – Simple Network Management Protocol

- "Internet-standard protocol for managing devices on IP networks"
- SNMP is a tool (protocol) that allows for remote and local management of items on the network including servers, workstations, routers, switches and other managed devices. It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects.

# Key components

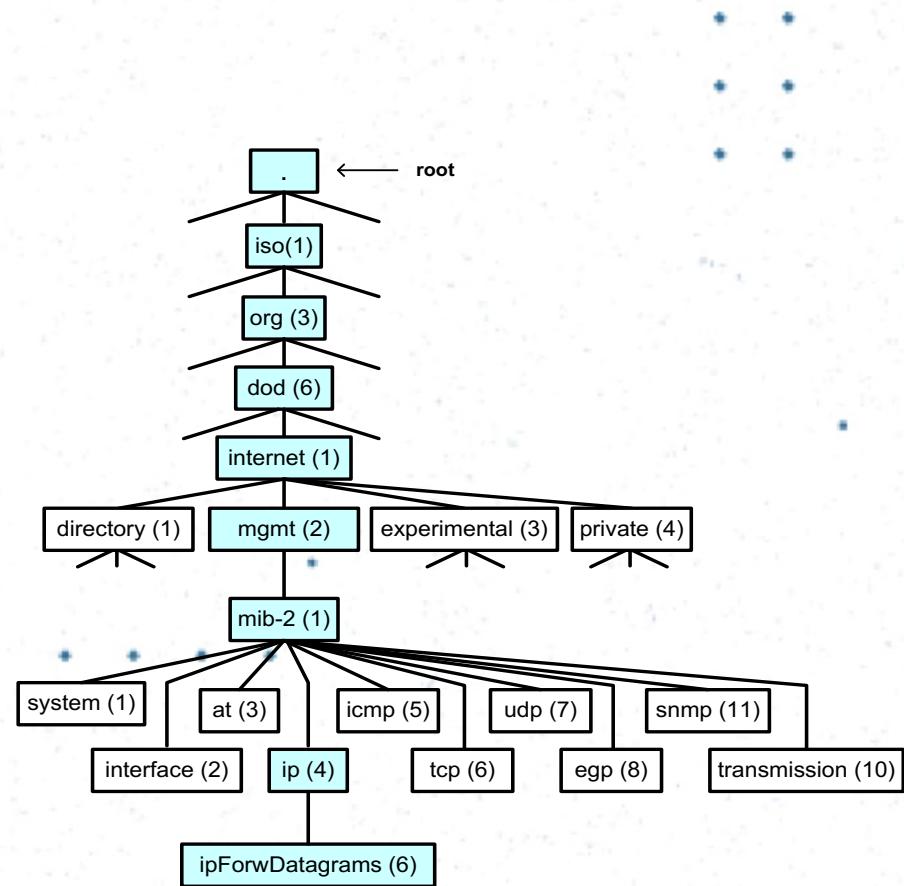
- ❖ **Manager** — an application program that contacts an SNMP agent to query or modify the database at the agent.
- ❖ **Agent** — software that runs on a piece of network equipment (host, router, printer, or others) and that maintains information about its configuration and current state in a database.
- ❖ **Network management station (NMS)** — executes applications that monitor and control managed devices. NMSs provide the bulk of the processing and memory resources required for network management. One or more NMSs may exist on any managed network.
- ❖ **Management Information Bases (MIBs)** — describes the information in the database.
- ❖ **SNMP Protocol** — application layer protocol used by SNMP agents and managers to send and receive data.

# Network management station (NMS)

- The one that executes network management applications (NMAs) that monitor and control network elements (NE) such as hosts, gateways and terminal servers.
- These network elements use a management agent (MA) to perform the network management functions requested by the network management stations.

# Management information base (MIB)

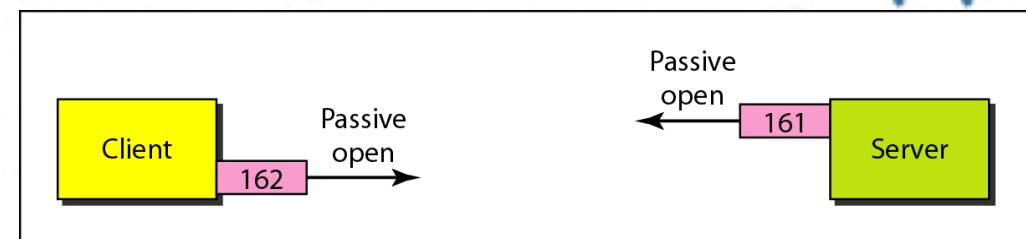
- MIBs describe the structure of the management data of a device subsystem; they use a **hierarchical namespace containing object identifiers (OID)**.
- Each OID identifies a variable that can be read or set via SNMP.
- MIBs use the notation defined by Structure of Management Information Version 2 (SMIv2, RFC 2578), a subset of ASN.1(Abstract Syntax Notation One).



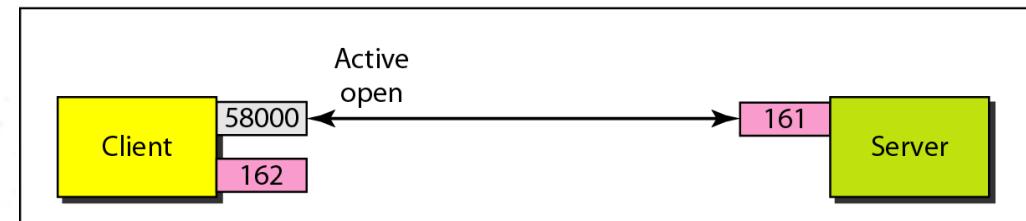
# SNMP Protocol

- **SNMP versions**
  - SNMPv1
  - SNMPv2
  - SNMPv3

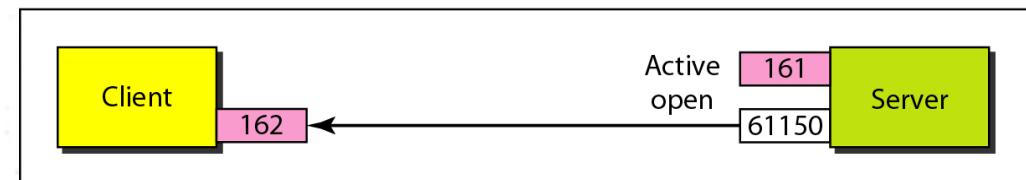
- Operates in the Application Layer.
- The SNMP Manager sends requests on UDP port 161.
- The SNMP Agent sends notifications on UDP port 162.
- SNMPv1 specifies five core protocol data units (PDUs). Two other PDUs were added in SNMPv2 and carried over to SNMPv3.



a. Passive open by both client and server



b. Exchange of request and response messages



c. Server sends trap message

# SNMP PDU Format

# Sending a Request

1. Appropriate PDU is constructed based on the **VarBindList** and the **operation type**.
2. The PDU, security information and agent identity are input to a security mechanism to apply authentication and encryption to the PDU (This is only conceptual in SNMPv1). The result from the security mechanism and the **community string** are used to generate an SNMP message.
3. PDU type –Get Request type
4. The message is then serialized and sent using a transport service (i.e. UDP) to the specified SNMP agent (serialization is done using the BER encoding scheme).

# PDU Type used

Get-Request

Request ID	Error Status	Error Index	Name X	Value X	...
Not used	Not used	Not used			

Get-Response

Request ID	Error Status	Error Index	Name X	Value X	...
Copy value	Set to zero	Set to zero	Copy value	Current Value	

- **GetRequest**

- A manager-to-agent request to retrieve the value of a variable or list of variables. Desired variables are specified in variable bindings (values are not used). Retrieval of the specified variable values is to be done as an atomic operation by the agent. A *Response* with current values is returned.

SNMPv1

# Receiving a Request

1. Incoming message is de-serialized to construct an **ASN.1** message.
2. **Version** number is verified.
3. The **community** name, security information and the data found in the SNMP message are input to a security mechanism.
4. The agent then performs a rudimentary parse of the ASN.1 object returned from the security service to build an ASN.1 object corresponding to an SNMP PDU object.

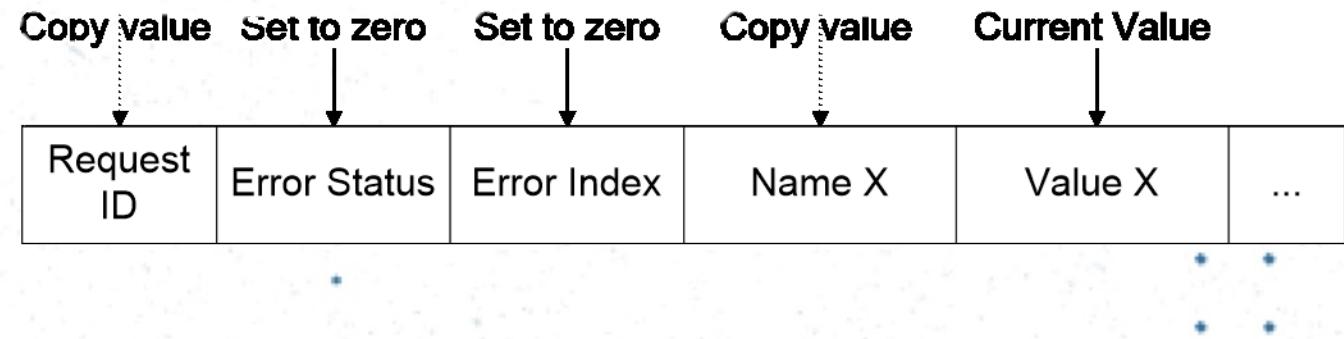
# Sending a Response

1. It first constructs a **Response** PDU using as input, the saved **request-id value** and the values for **error-status**, **error-index** and VarBindList returned from processing the request.
2. The PDU, security information and agent identity are input to a security mechanism. The result from the security mechanism and the **community string** are used to generate an SNMP message.
3. The message is serialized and then sent to the manager address from the request.

# Pdu type used

## Get-Response

- **Response**
- Returns variable bindings and acknowledgement from agent to manager for *GetRequest* or any other request message type. Error reporting is provided by error-status and error-index fields. Although it was used as a response to both gets and sets, this PDU was called *GetResponse* in SNMPv1.



SNMPv1

# Receiving a Response

1. Incoming message is de-serialized.
2. PDU **version** is verified.
3. The **community** name, security information and the data found in the SNMP message are input in the security mechanism.
4. The **ASN.1 object** is parsed.

# GetRequest/GetResponse Errors

- **tooBig** – the size of the response would exceed a local limitation (and the error-index field has no additional information)
- **noSuchName** – a specified instance of the management information is not available to be accessed – due to access control settings or instance does not exist
- **genErr** – specified instance of the management information is not available due to some other reason

Since only one error indication is returned in a get-response message and each identified instance of management information in the request can cause an error, a manager may have to try the GET operation many times before a get-response message is retimed with no error.

# Other Request PDU types

- **GetNextRequest**
- A manager-to-agent request to discover **available variables** and their **values**. Returns a Response with variable binding for the lexicographically **next variable in the MIB**. The entire MIB of an agent can be walked by iterative application of *GetNextRequest* starting at OID 0. Rows of a table can be read by specifying column OIDs in the variable bindings of the request.

SNMPv1

# Other Request PDU types

- **GetBulkRequest**
- Optimized version of *GetNextRequest*. A manager-to-agent request for multiple iterations of *GetNextRequest*. Returns a Response with multiple variable bindings walked from the variable binding or bindings in the request. PDU specific non-repeaters and max-repetitions fields are used to control response behavior. *GetBulkRequest* was introduced in SNMPv2.

**SNMPv2**

# Other Request PDU types

- **SetRequest**
- A manager-to-agent **request to change the value of a variable or list of variables**. Variable bindings are specified in the body of the request. Changes to all specified variables are to be made as an atomic operation by the agent. A Response with (current) new values for the variables is returned.

**SNMPv1**

# SetRequest/GetResponse Errors

- **badValue** – the value provided for the field may not comply with the data types or standards defined
- **noSuchName** – a specified instance of the management information is not available to be accessed – due to access control settings or instance does not exist
- **genErr** – specified instance of the management information is not available due to some other reason

Since only one error indication is returned in a get-response message and each identified instance of management information in the request can cause an error, a manager may have to try the GET operation many times before a get-response message is retimed with no error.

# Other PDU types

- Trap
- Asynchronous notification from agent to manager. SNMP traps enable an agent to notify the management station of significant events by way of an unsolicited SNMP message. Includes current *sysUpTime* value, an *OID* identifying the type of trap and optional variable bindings.
- No response is expected.

SNMPv1

# TRAP types

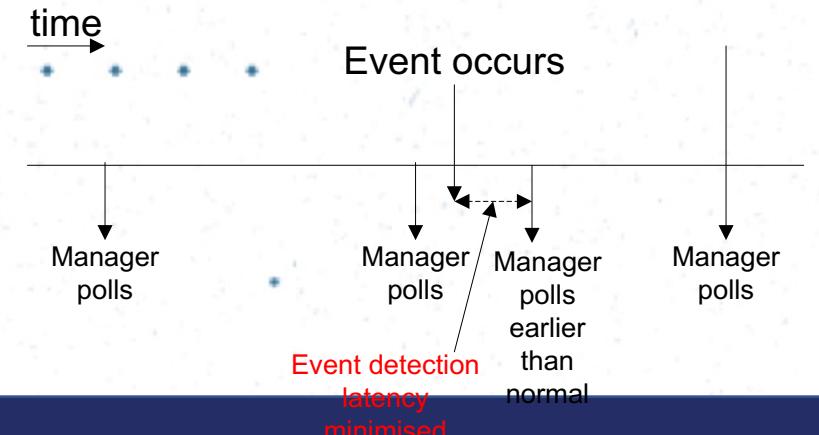
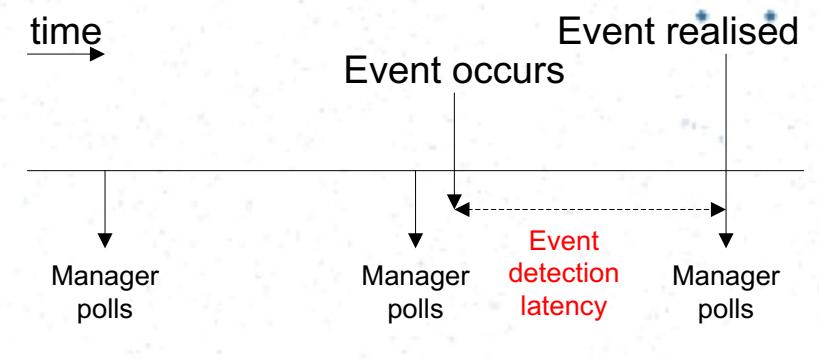
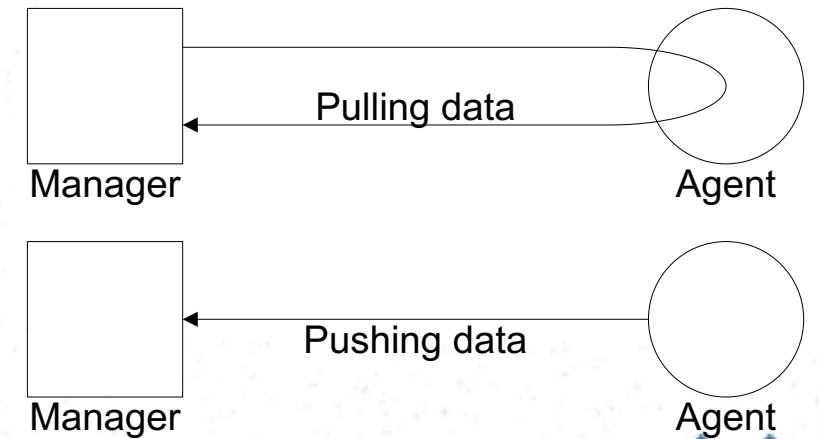
- **coldStart** - the sending protocol entity is reinitializing itself so that the agent's configuration or the protocol entity implementation can be altered.
- **warmStart** - the sending protocol entity is reinitializing itself so that neither the agent configuration nor the protocol entity implementation can be altered.
- **linkDown** - the sending protocol entity recognizes a failure in one of the communication links represented in the agent's configuration.
- **linkUp** - the sending protocol entity recognizes that one of the communication links represented in the agent's configuration has come up.

# TRAP types

- **authenticationFailure** - the sending protocol entity is the addressee of a protocol message that is not properly authenticated.
- **egpNeighborLoss** - an EGP neighbor for whom the sending protocol entity was an EGP peer has been marked down and the peer relationship no longer exists.
- **enterpriseSpecific** - the sending protocol entity recognizes that some enterprise-specific event has occurred.

# Use of Events

- Agents provide the specified data only when requested by managers.
  - This periodic gathering of data is called **polling**.
- Amount of time between an event and a manager realizing that an event has occurred is called the *event detection latency*.
- Another model of management is based on agents in managed systems sending data to configured managers.
  - Traps (**Pushing**).
  - However, during a communication failure, an event report would never reach a manager.



# SNMP versions

1. **SNMPv1** - SNMP version 1 (SNMPv1) is the initial implementation of the SNMP protocol. Version 1 has been criticized for its poor security. Authentication of clients is performed only by a "*community string*".
2. **SNMPv2** - Revises version 1 and includes improvements in the areas of performance, security, confidentiality, and manager-to-manager communications. It introduced *GetBulkRequest*, an alternative to iterative *GetNextRequests* for retrieving large amounts of management data in a single request.
3. **SNMPv3** - SNMPv3 primarily added security and remote configuration enhancements to SNMP.

# SNMP<sub>v</sub>1

# SNMPv1 - Limitations

- Limited data types
  - Reduction in complexity of SNMP has resulted in some increase in complexity of MIB understanding and design.
- Limited performance
  - Inefficient for large retrievals. (e.g. retrieving entire MIB or tables)
    - E.g. 2000 entry table with 4 columns 200ms RTT Using GETNEXT Requests:  $2000 \times 4 \times 2 = 16000$  packets or  $2000 \times 4 \times .2 = 1600$  seconds...!!!
- Limited error codes
- Lack of hierarchies
  - Inherently centralized
- Lack of security
  - Community strings

# SNMP<sub>v</sub>2

# SNMPv2 - features

- ❖ New data types

## *SIMPLE TYPES:*

### SMIv1

INTEGER  
OCTET STRING  
OBJECT IDENTIFIER

### SMIv2

INTEGER  
OCTET STRING  
OBJECT IDENTIFIER

- Integer32

## *APPLICATION-WIDE TYPES:*

- Unsigned32  
Gauge  
Counter  
-  
TimeTicks  
IpAddress  
Opaque  
NetworkAddress

Unsigned32  
Gauge32  
Counter32  
Counter64  
TimeTicks  
IpAddress  
Opaque  
-

## *PSEUDO TYPES:*

- BITS

# SNMPv2 - features

- ❖ Community based (similar to SNMPv1).
- ❖ *GET* and *GETNEXT* operations are the same as SNMPv1.
- ❖ SET Request operation is a conceptual two phase commit;
  - ❖ PHASE 1 : Perform various checks.
  - ❖ PHASE 2 : Perform the actual set operation.
- ❖ SNMPv2 adds and enhances some protocol operations.
  - The *INFORM* Request.
  - The *GETBULK* Request.

# GETBULK Request

Some further insight...

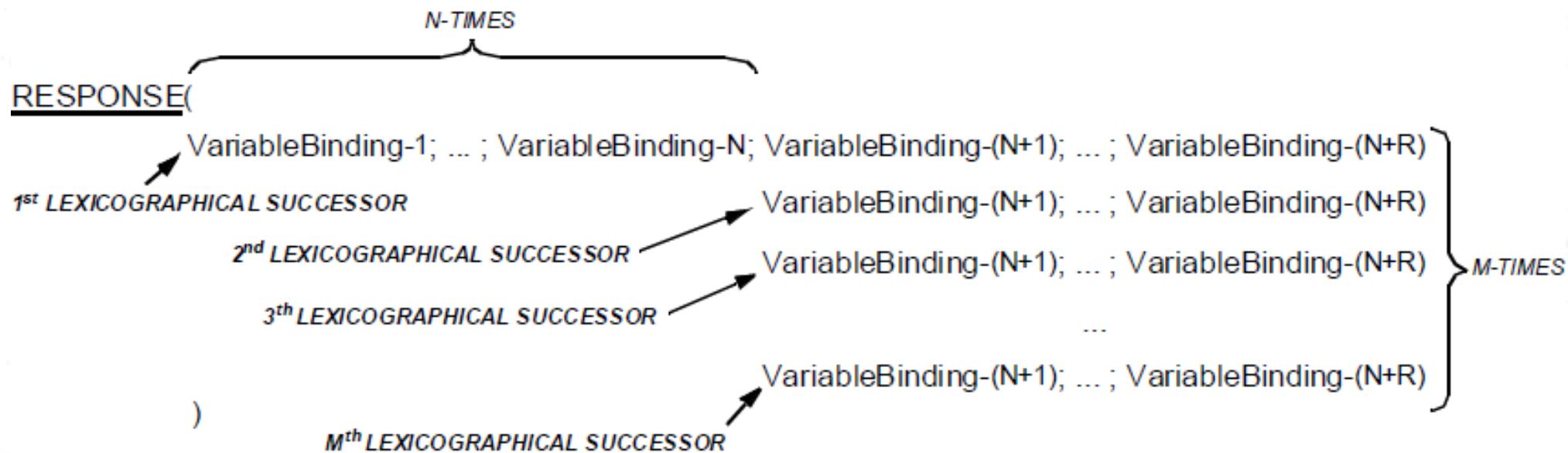
- For efficient retrieval of many VarBinds.
- GETBULK Request has two additional parameters:
  - **non-repeaters**
    - The first N elements (non-repeaters) of the VarBind list are treated as if the operation was a normal GETNEXT Request operation.
  - **max-repetitions**
    - The next elements of the VarBind list are treated as if the operation consisted of a number (max-repetitions) of repeated GETNEXT Request operations.

# GETBULK Request

Some further insight...

```
REQUEST(non-repeaters = N; max-repetitions = M;  
        VariableBinding-1; ... ; VariableBinding-N; VariableBinding-(N+1); ... ; VariableBinding-(N+R)  
        )
```

\* \*  
\* \*  
\* \*



# INFORM Request

Some further insight...

- Remember that for **SNMPv1 trap messages**, there is **no response sent by the manager**. Hence, **no guarantee** that the manager received the trap or not...!!!
- As a solution for this, SNMPv2 INFORM operation was introduced.
- In simple terms this is a confirmed trap.
  - Originally to INFORM a higher level manager
  - Same format as Trap PDU
  - Possible error: tooBig

# SNMPv2 - features

- ❖ Additional error/exceptions
  - GET additional errors:
    - noSuchObject.
    - noSuchInstance.
  - GETNEXT additional errors:
    - endOfMibView.
  - Exceptions are coded within the VarBinds and do not raise Error Status and Index.

# SNMPv2 - Limitations

- ❖ Too complex
- ❖ Overshoot problem - GETBULK does not stop when it reaches the end of a table.
- ❖ Although SNMPv2 was originally intended to overcome the security problem of SNMPv1 because of the high complexity, without the controversial new SNMP v2 security model, using instead the simple community-based security scheme of SNMPv1

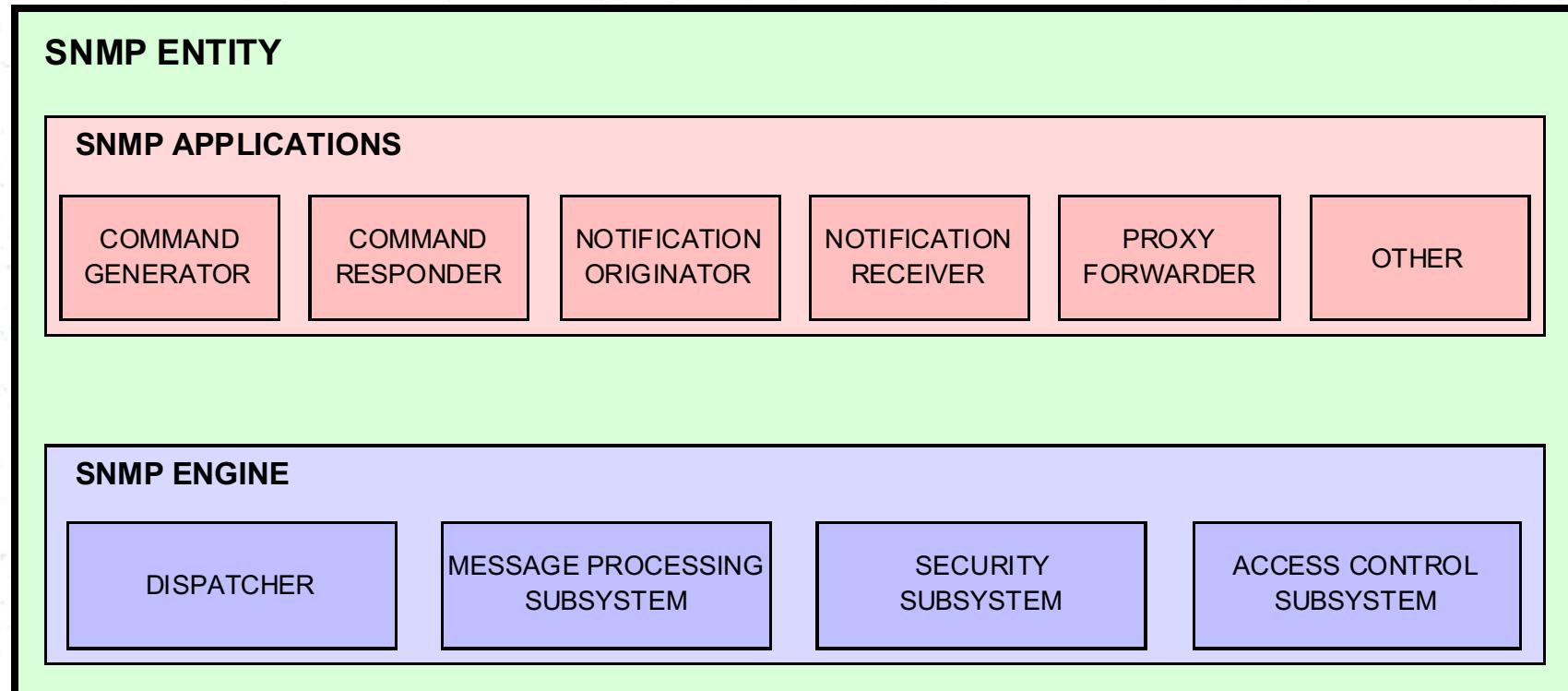
# SNMP<sub>v</sub>3

# SNMPv3 - features

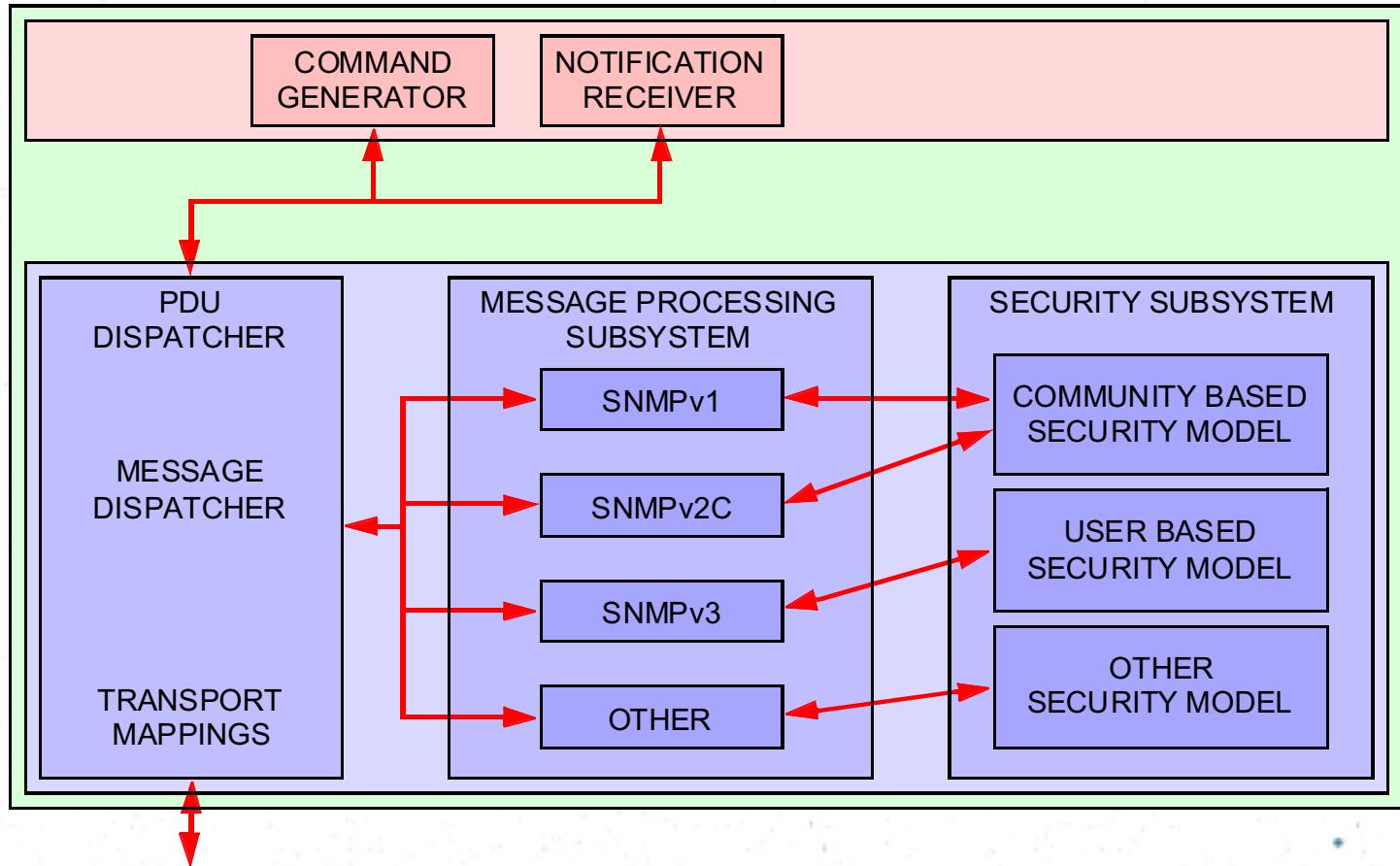
- ❖ Security.
- ❖ Remote configuration capabilities.
- ❖ Ability to control the agents via use of a newly defined specific management MIB.
- ❖ SNMPv3 contains all the functionality of SNMPv1 and SNMPv2.
- ❖ Incorporates the SNMPv2 data types.

# SNMPv3 - Entities

- ❖ SNMP Manager and Agent is created as an SNMP Entity
  - Entity consists of an **SNMP engine** and **SNMP applications**.



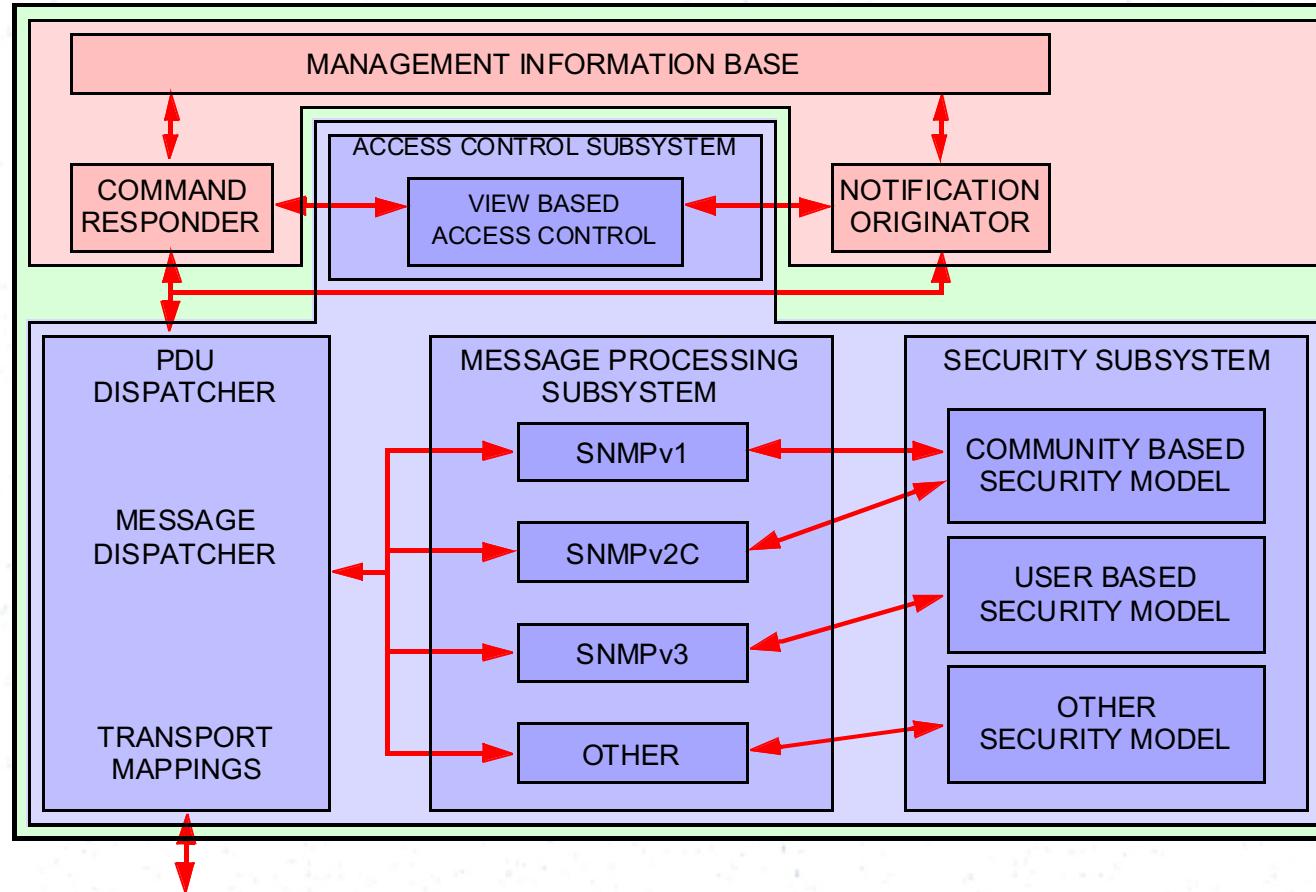
# SNMPv3 – SNMP Manager entity



SNMP  
Application

SNMP  
Engine

# SNMPv3 – SNMP AGENT entity



SNMP  
Application

SNMP  
Engine

# SNMP Engine dispatcher

- ❖ The dispatcher is a **traffic manager** that sends and receives messages.
- ❖ Incoming messages
  - The dispatcher tries to **determine the version number** of the message and then **passes the message to the appropriate message processing model**.
- ❖ Outgoing messages
  - The dispatcher **selects the appropriate transports** for sending messages. Currently this is UDP in the TCP/IP world.
- ❖ The dispatcher is also responsible for dispatching PDUs to applications.

# SNMP Engine

## Message processing subsystem

- Outgoing messages
  - Accepts outgoing PDUs from the dispatcher and prepares them for transmission by wrapping them in a message header.
- Incoming messages
  - Accepts incoming messages from the dispatcher, processes each message header.
  - An implementation of the message processing subsystem may support a single message format corresponding to a single version of SNMP (SNMPv1, SNMPv2c, SNMPv3), or it may contain a number of modules, each supporting a different version of SNMP.

# SNMP Engine

## security subsystem

- Authenticates and encrypts messages.
- Outgoing messages are passed to the security subsystem from the message processing subsystem.
  - Depending on the services required, the security subsystem may encrypt the enclosed PDU and some fields in the message header.
  - The security subsystem may generate an authentication code and insert it into the message header.
- After encryption, the message is returned to the message processing subsystem.

# SNMP Engine

## security subsystem

- Incoming messages are passed to the security subsystem from the message processing subsystem.
  - If required, the security subsystem checks the authentication code and performs decryption.
  - The processed message is returned to the message processing subsystem.
  - An implementation of the security subsystem may support one or more distinct security models.



# SNMP Engine

## security subsystem

Consists of three sub modules,

1. Community Based Security Model
  - For SNMPv1 and SNMPv2
2. User Based Security Model
  - For SNMPv3 covering the CIA concepts
3. Other Security Model

# SNMP Engine

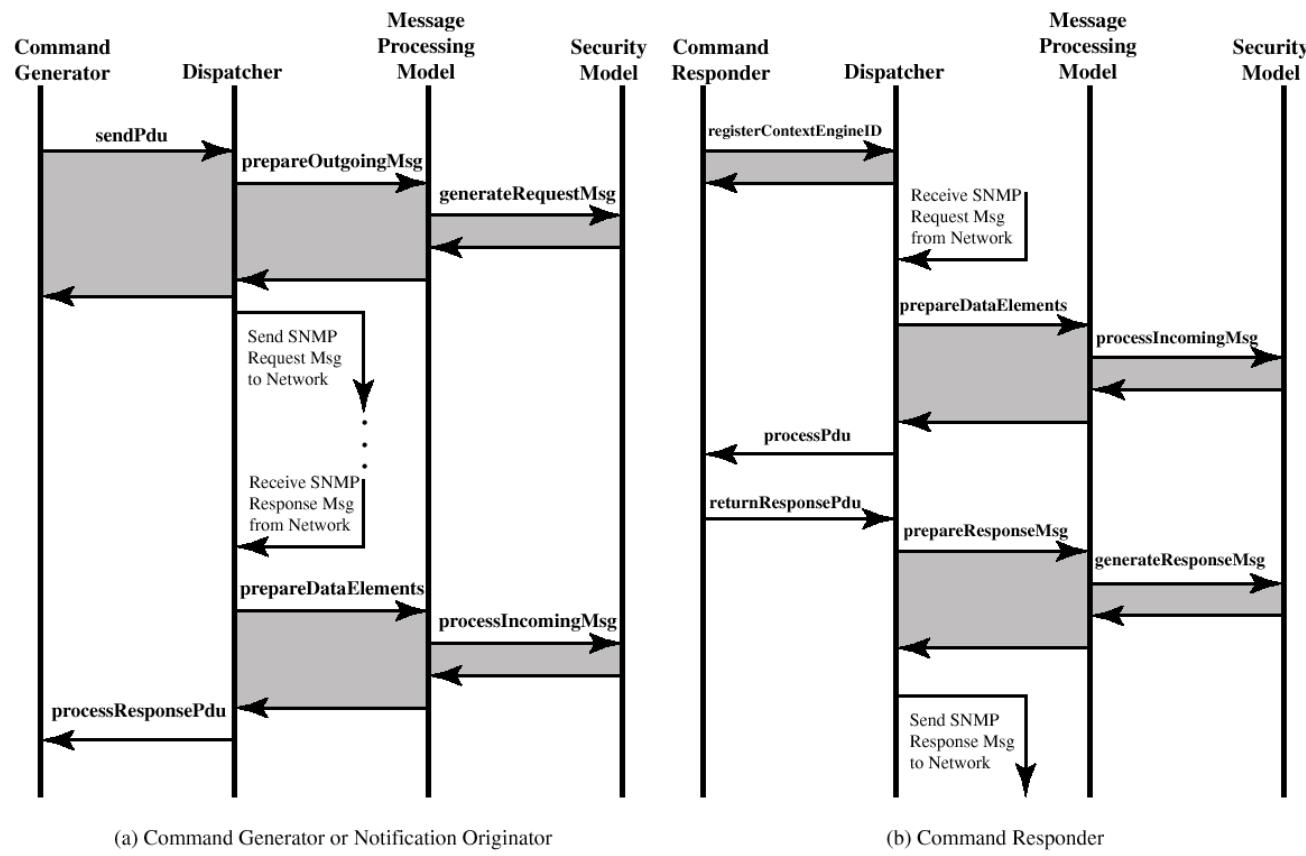
## Access Control Subsystem

- Provides authentication.
  - Determine whether access to a managed object should be allowed.
- Can control which users and which operations can have access to which managed objects.

# SNMP Application

- There are five types of applications.
  - **Command generators** - Generate SNMP commands to collect or set management data.
  - **Command responders** - Provide access to management data.
    - For example, processing get, get-next, get-bulk and set PDUs.
    - Commands are used in a command responder application.
  - **Notification originators** - Initiate Trap or Inform messages.
  - **Notification receivers** - Receive and process Trap or Inform messages.
  - **Proxy forwarders** - Forward messages between SNMP entities.

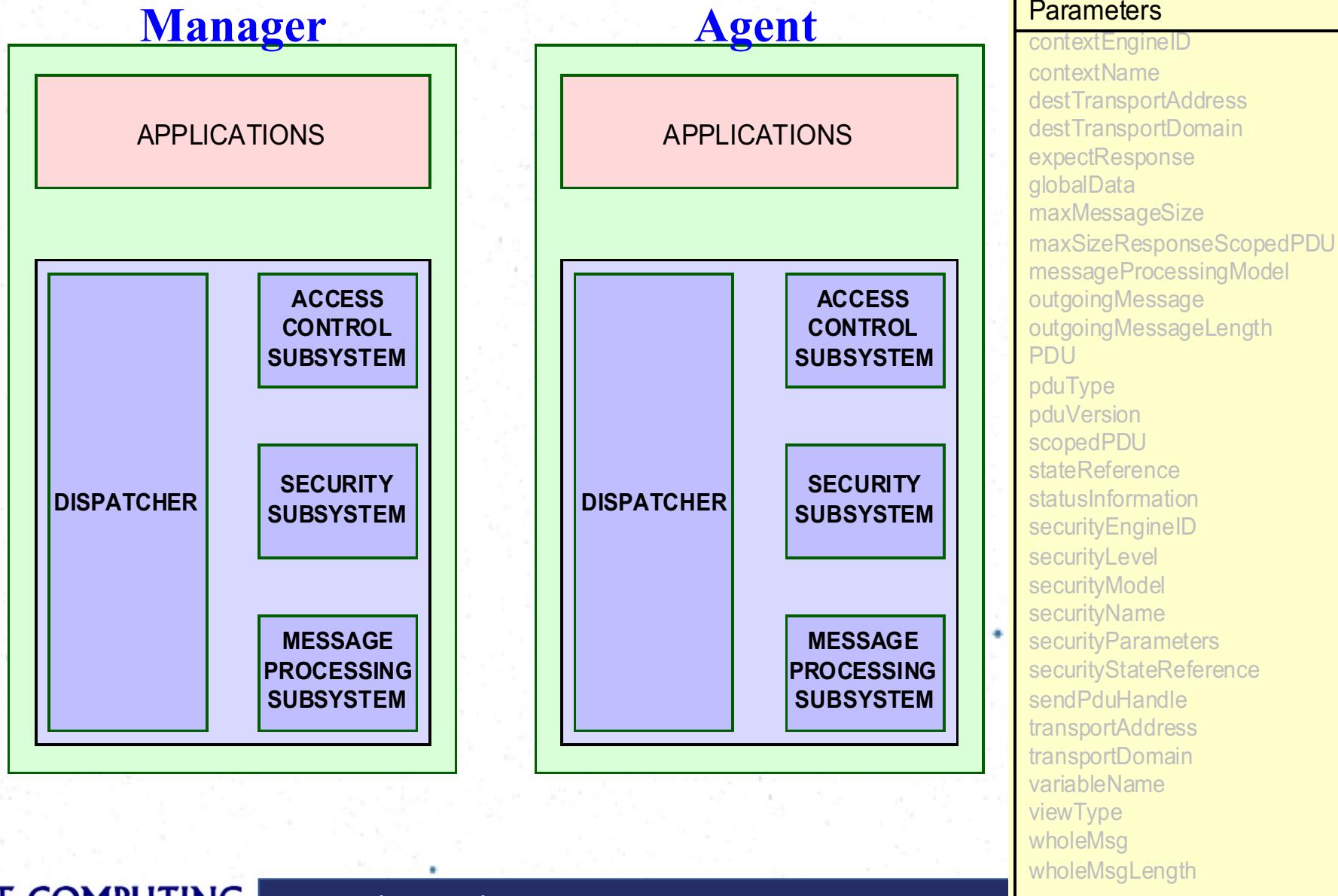
# SNMPv3 flow



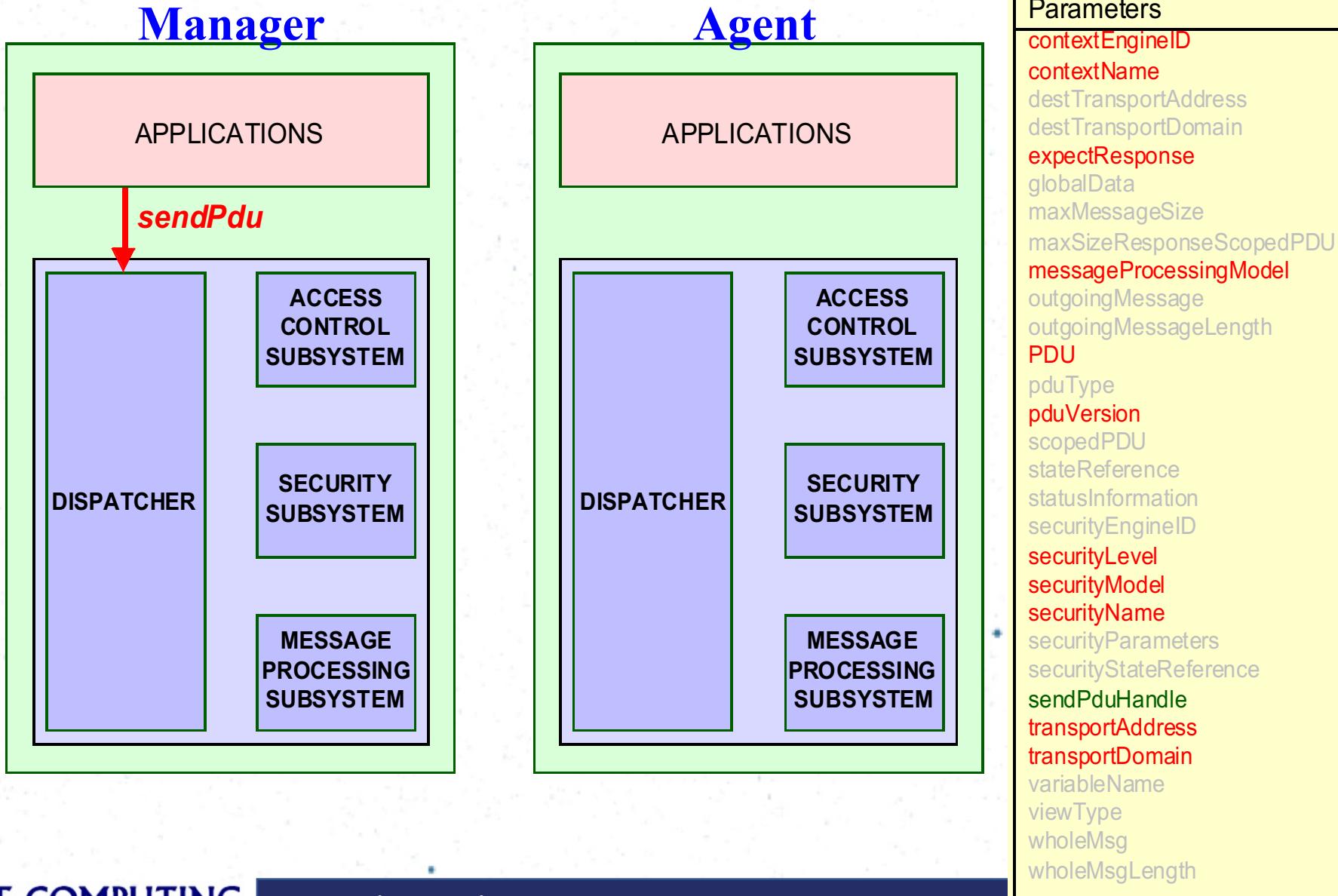
(a) Command Generator or Notification Originator

(b) Command Responder

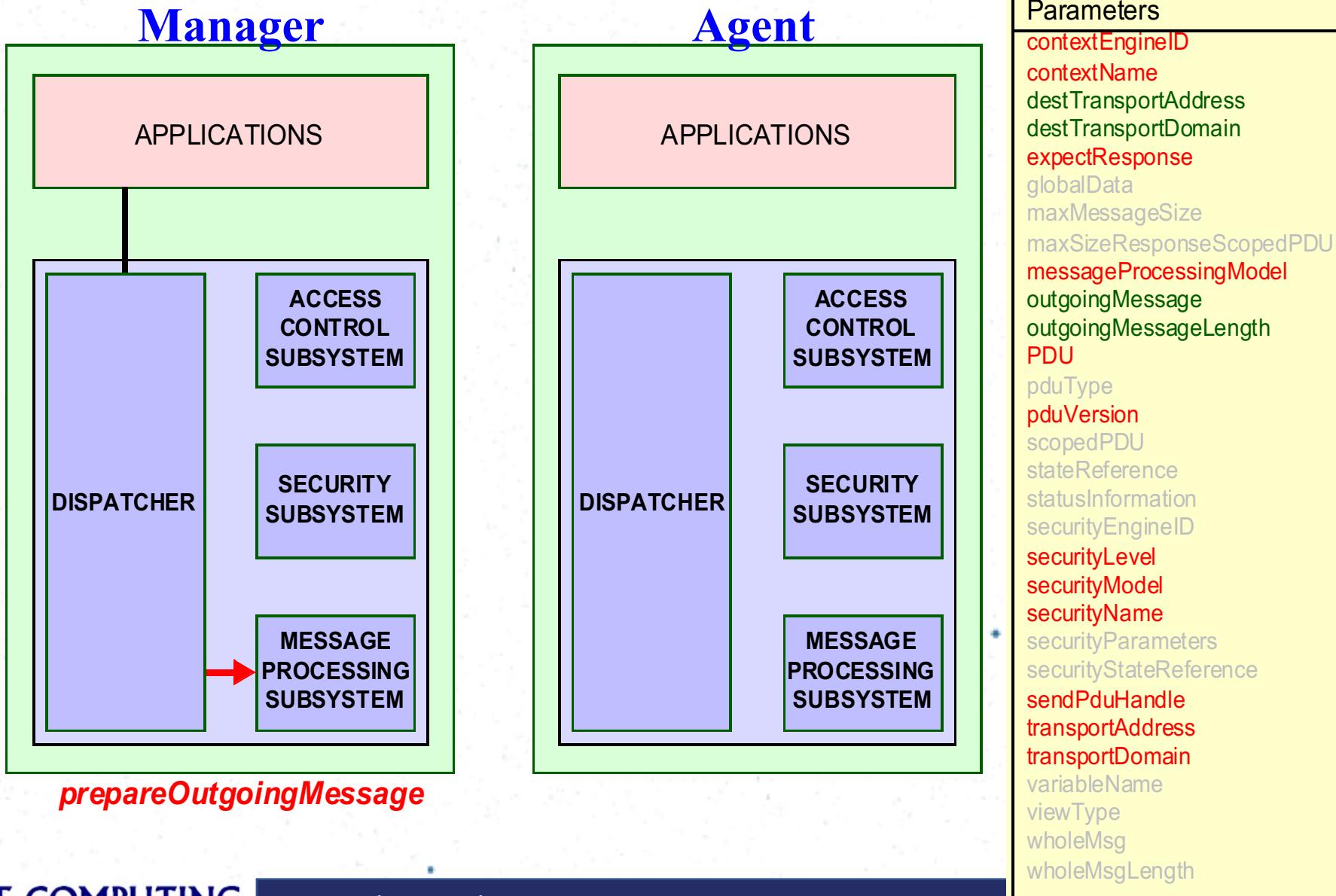
# Primitives Between Modules



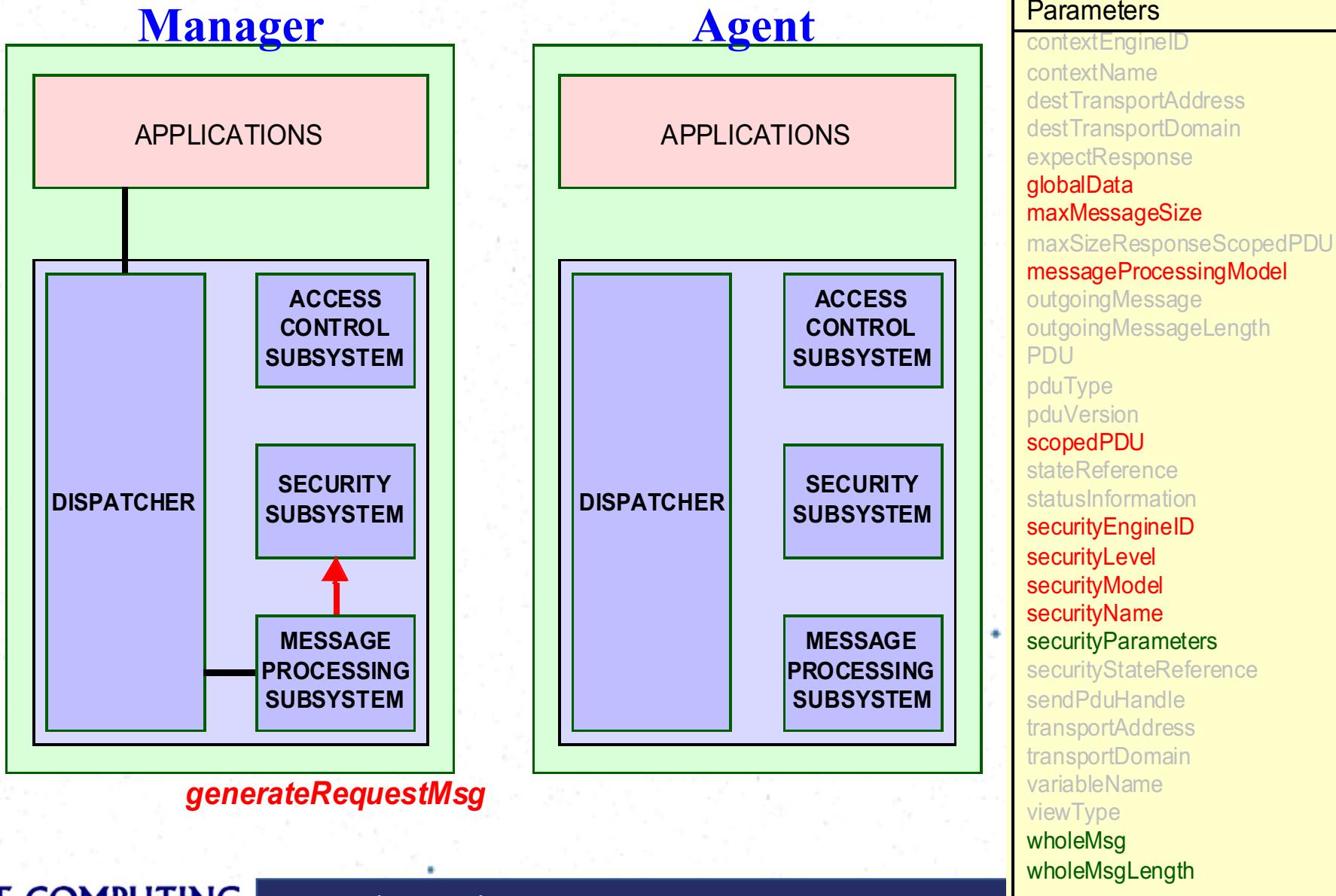
# sendPdu



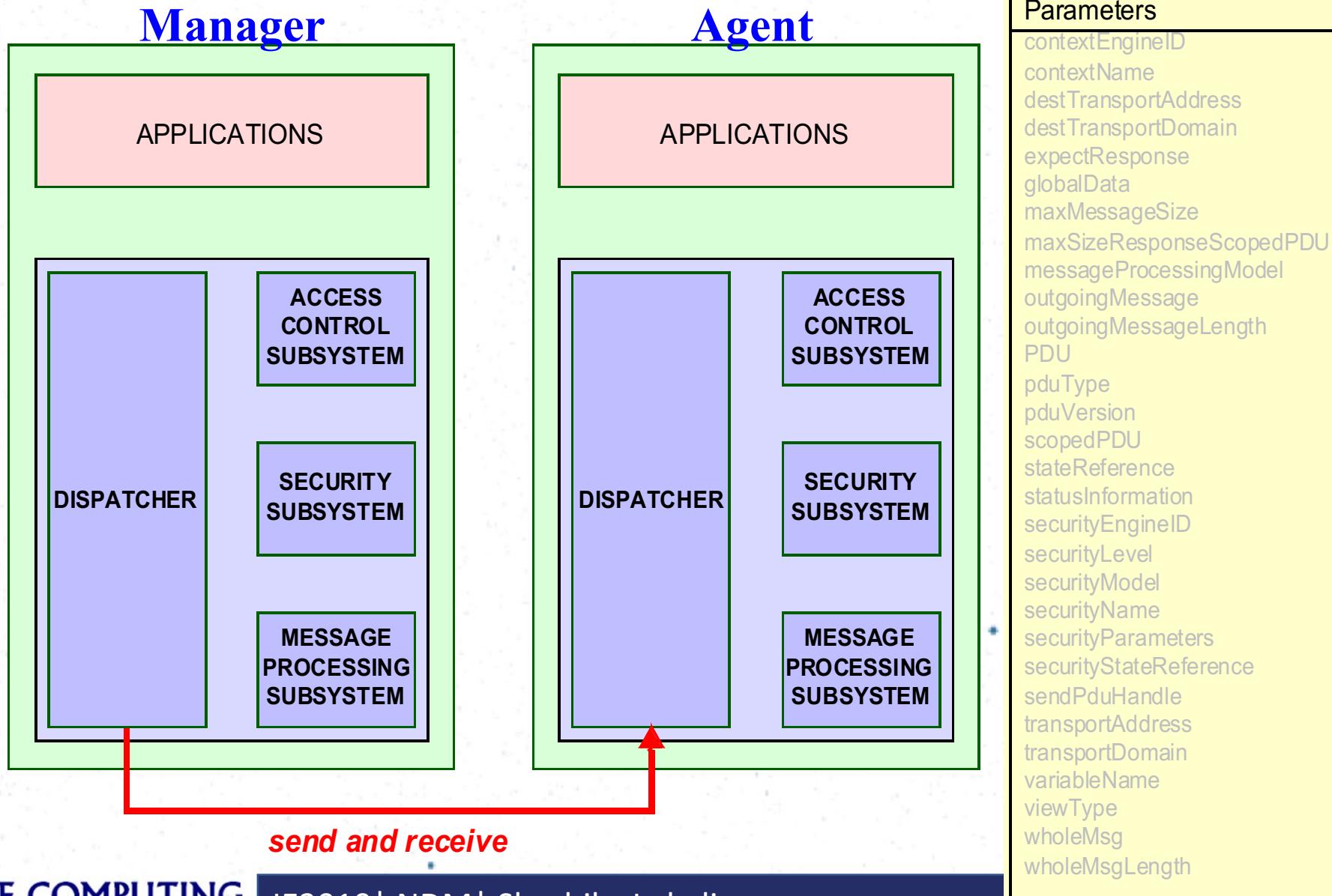
# prepareOutgoingMessage



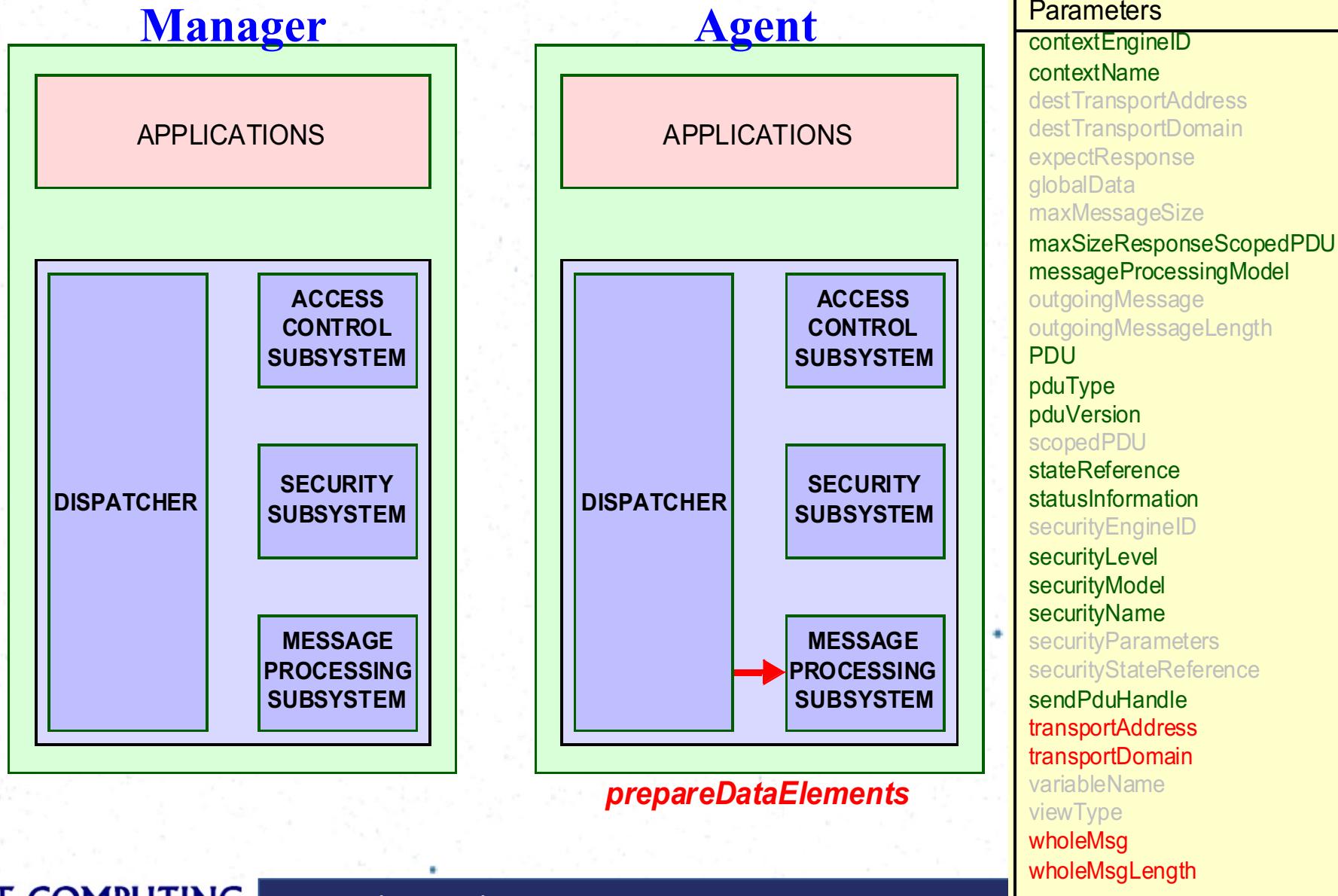
# generateRequestMsg



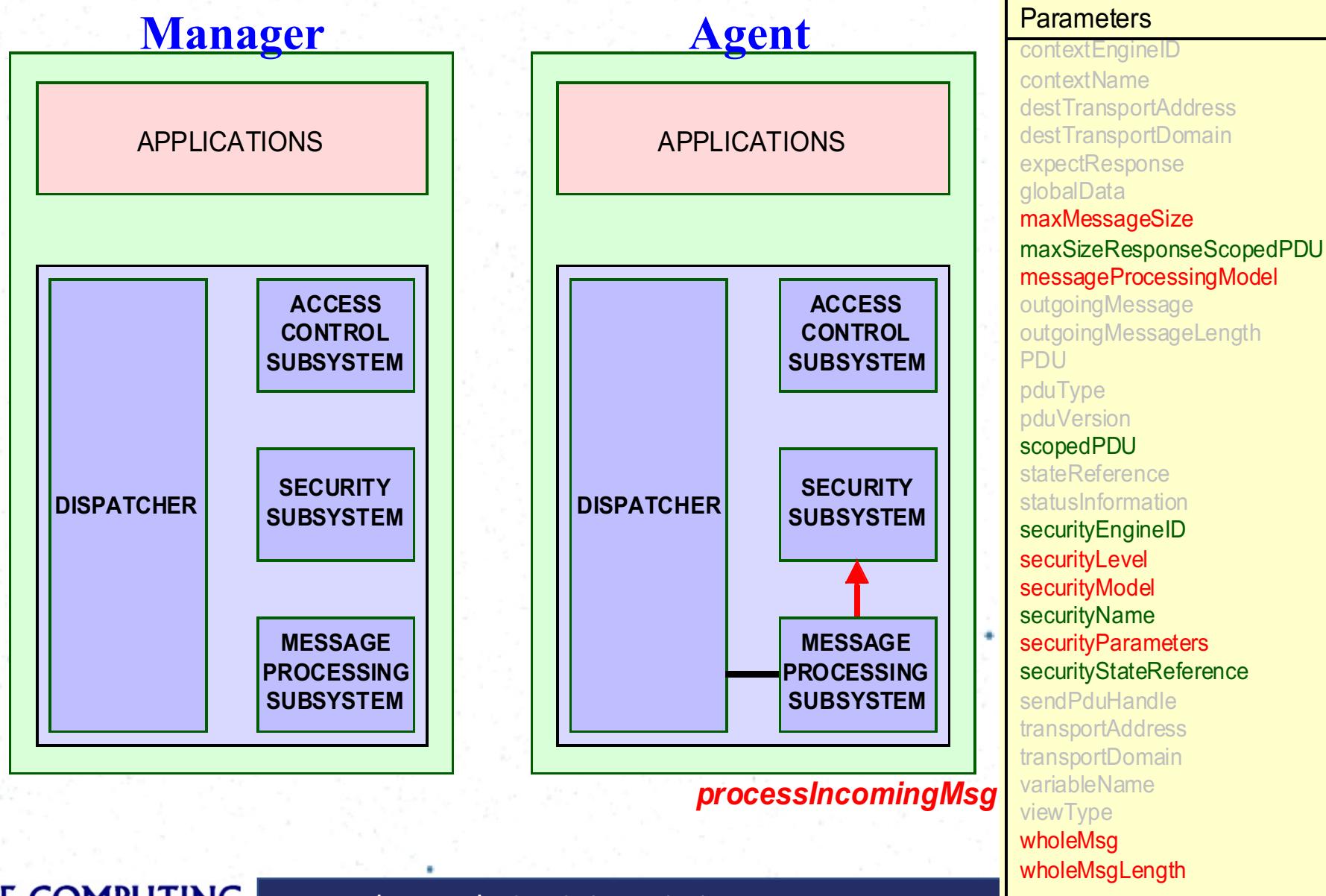
# send / receive



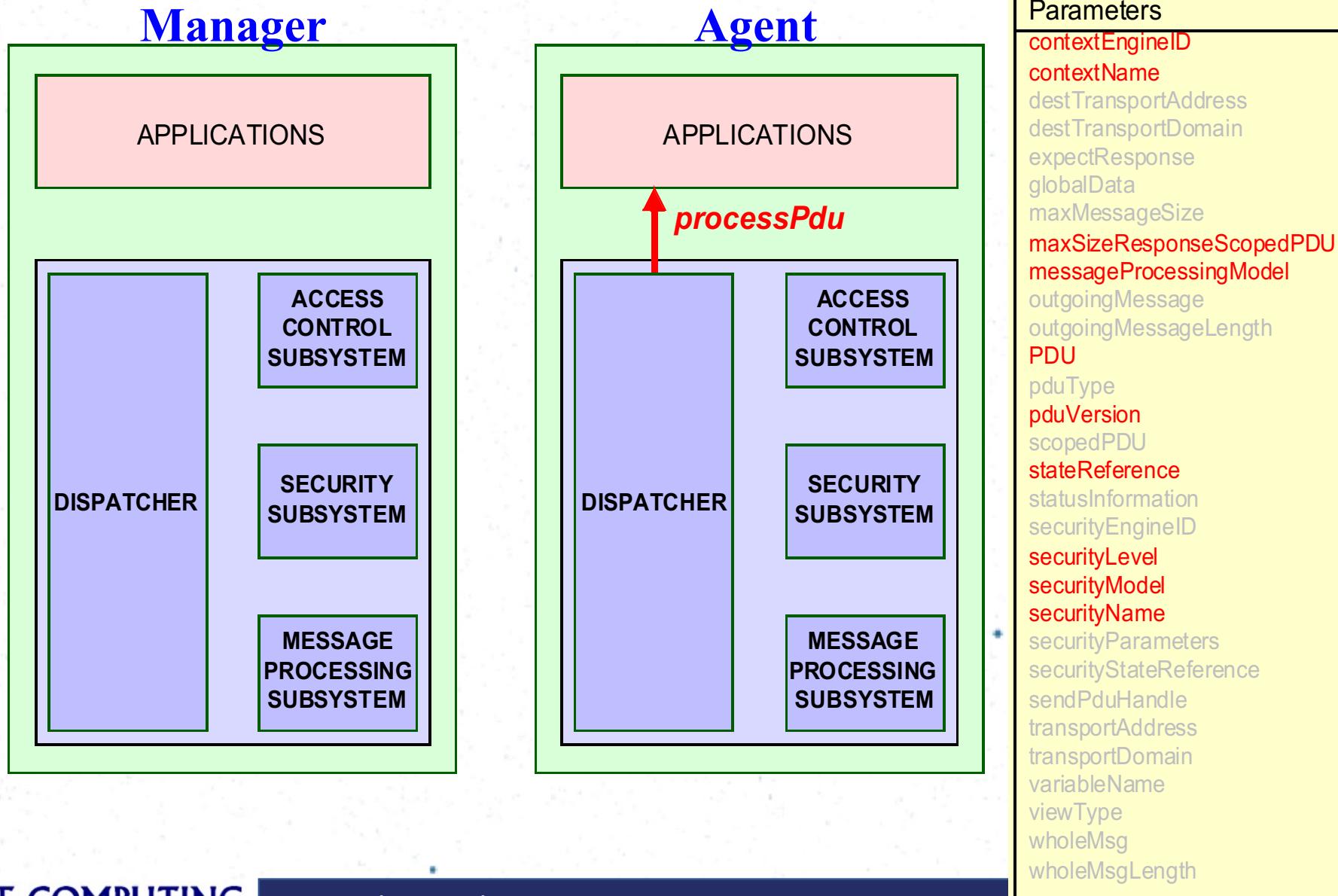
# prepareDataElements



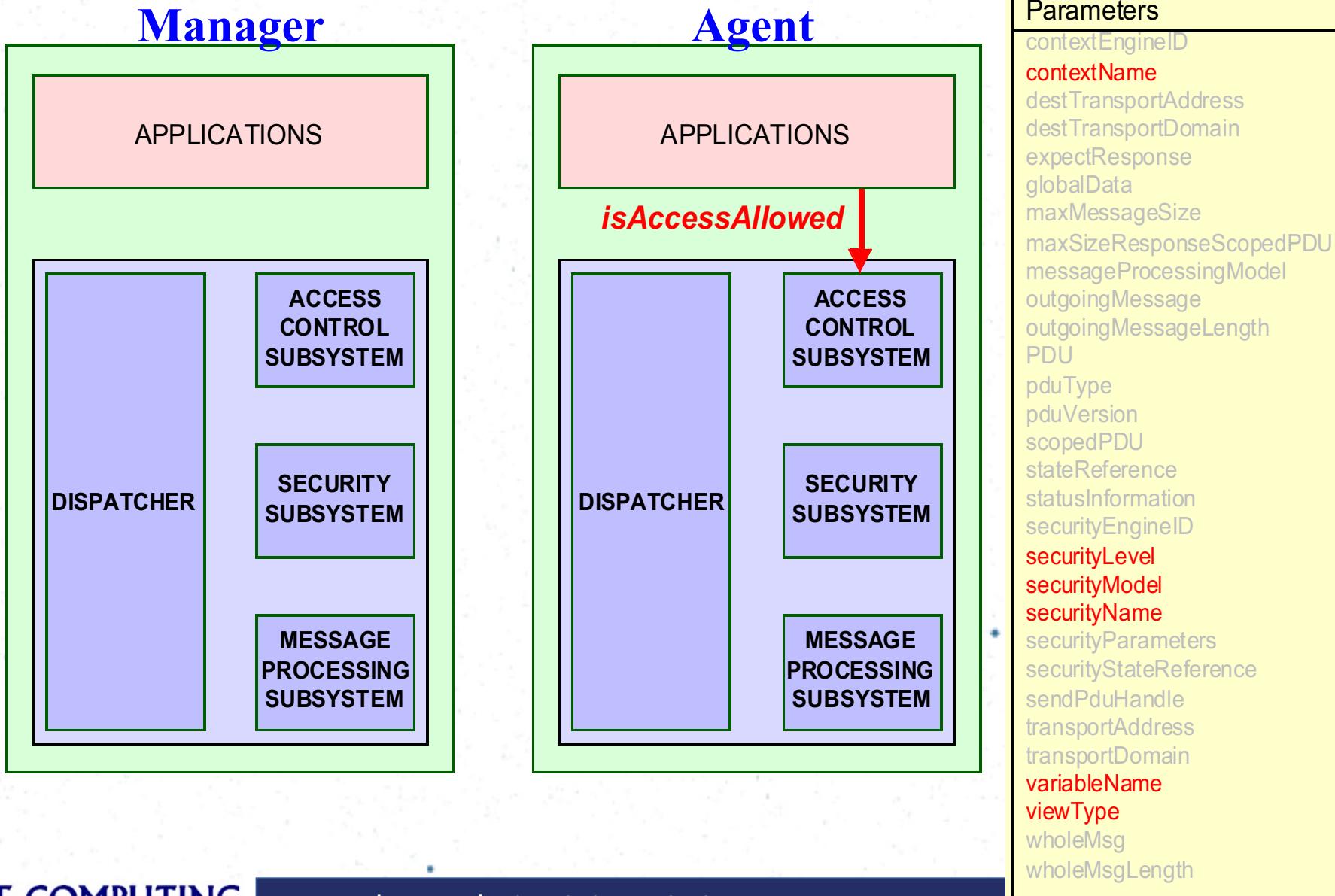
# processIncomingMsg



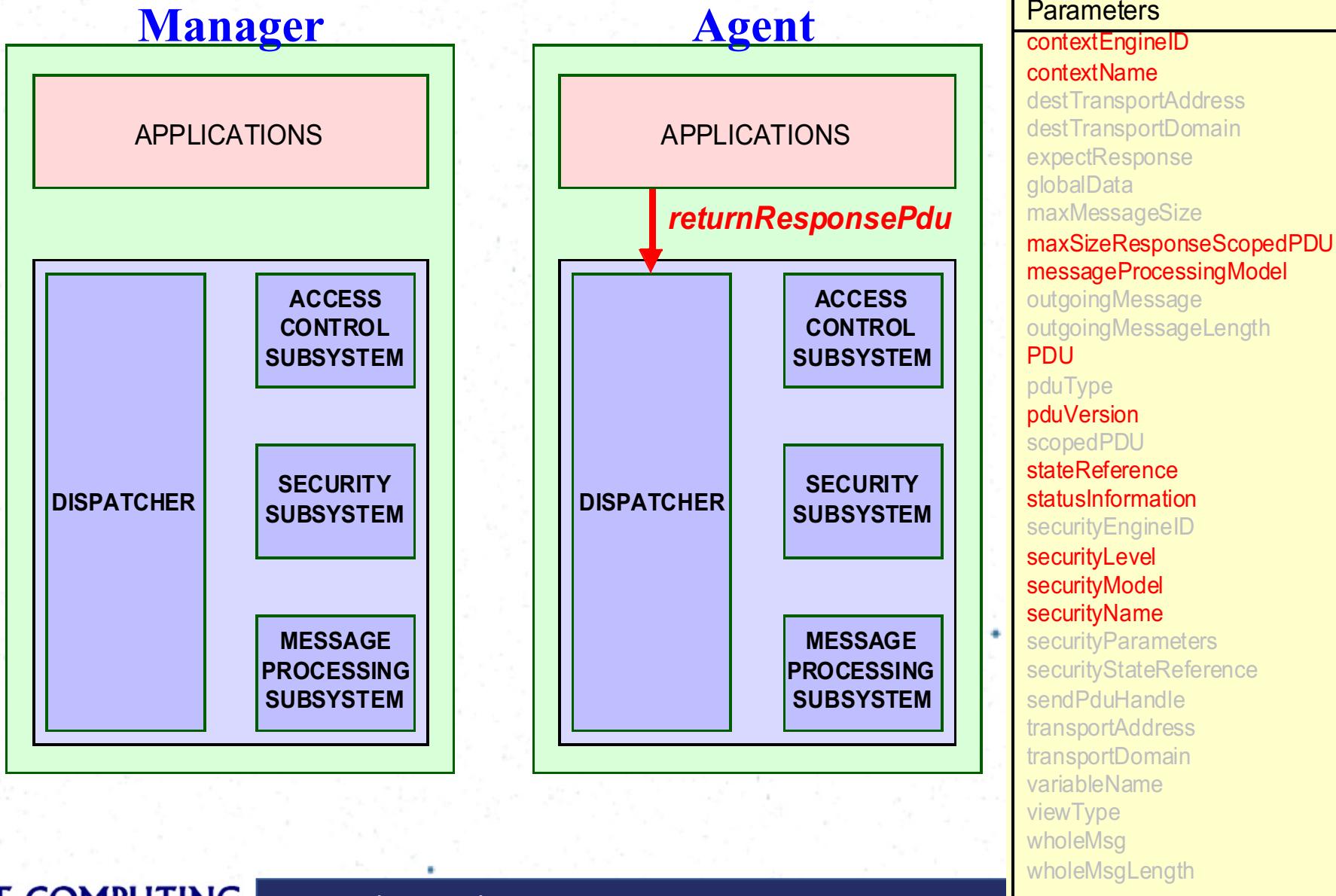
# processPdu



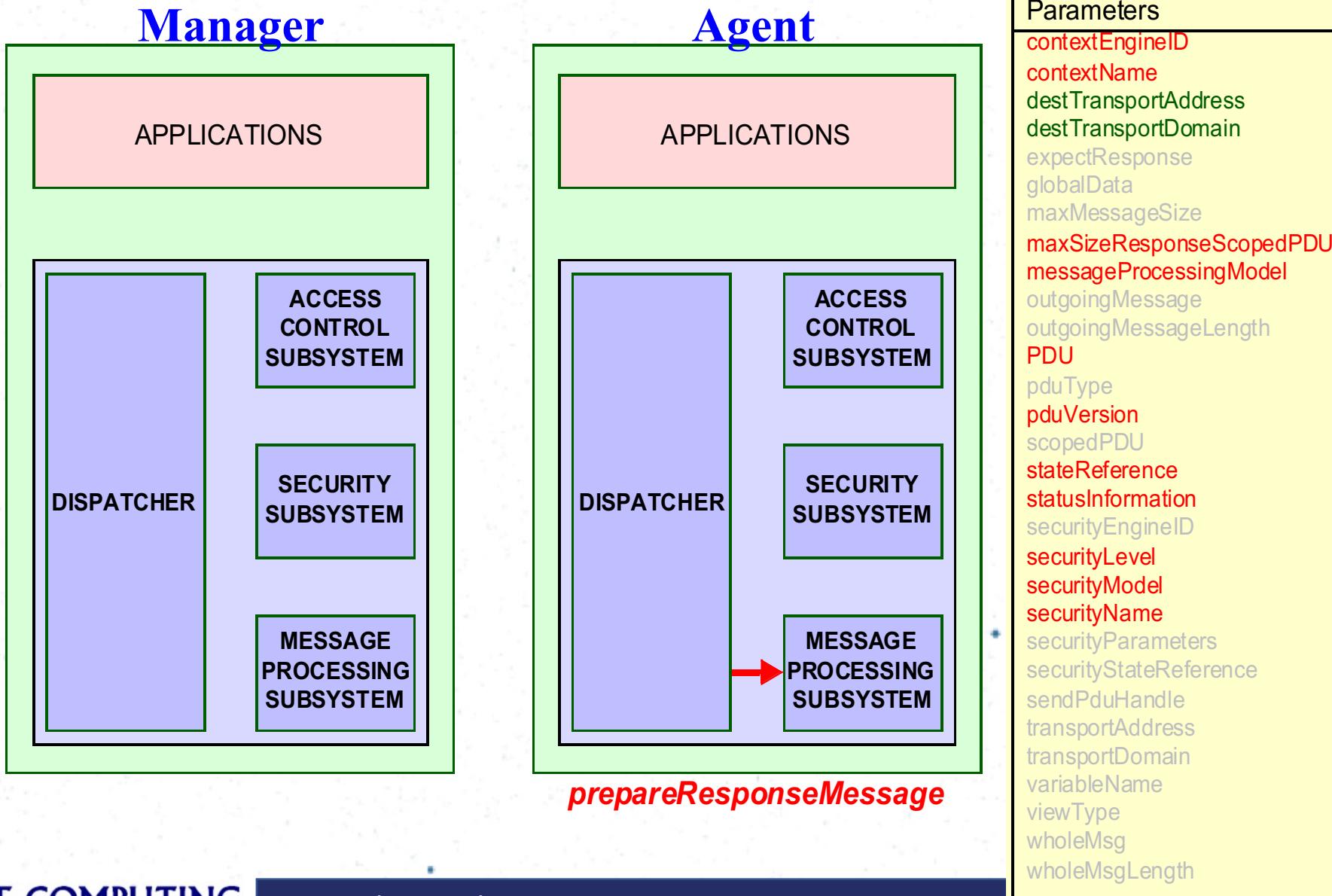
# isAccessAllowed



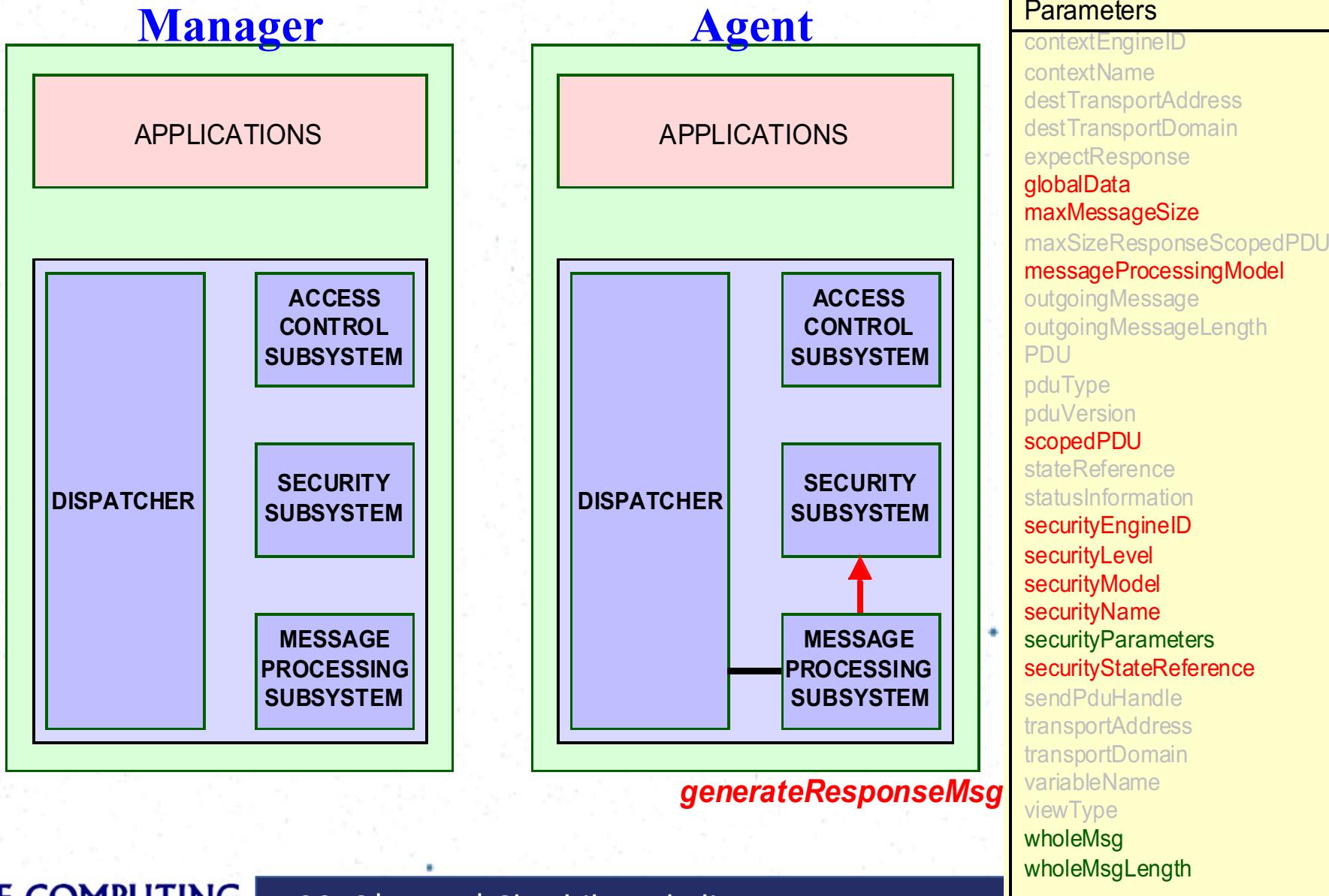
# returnResponsePdu



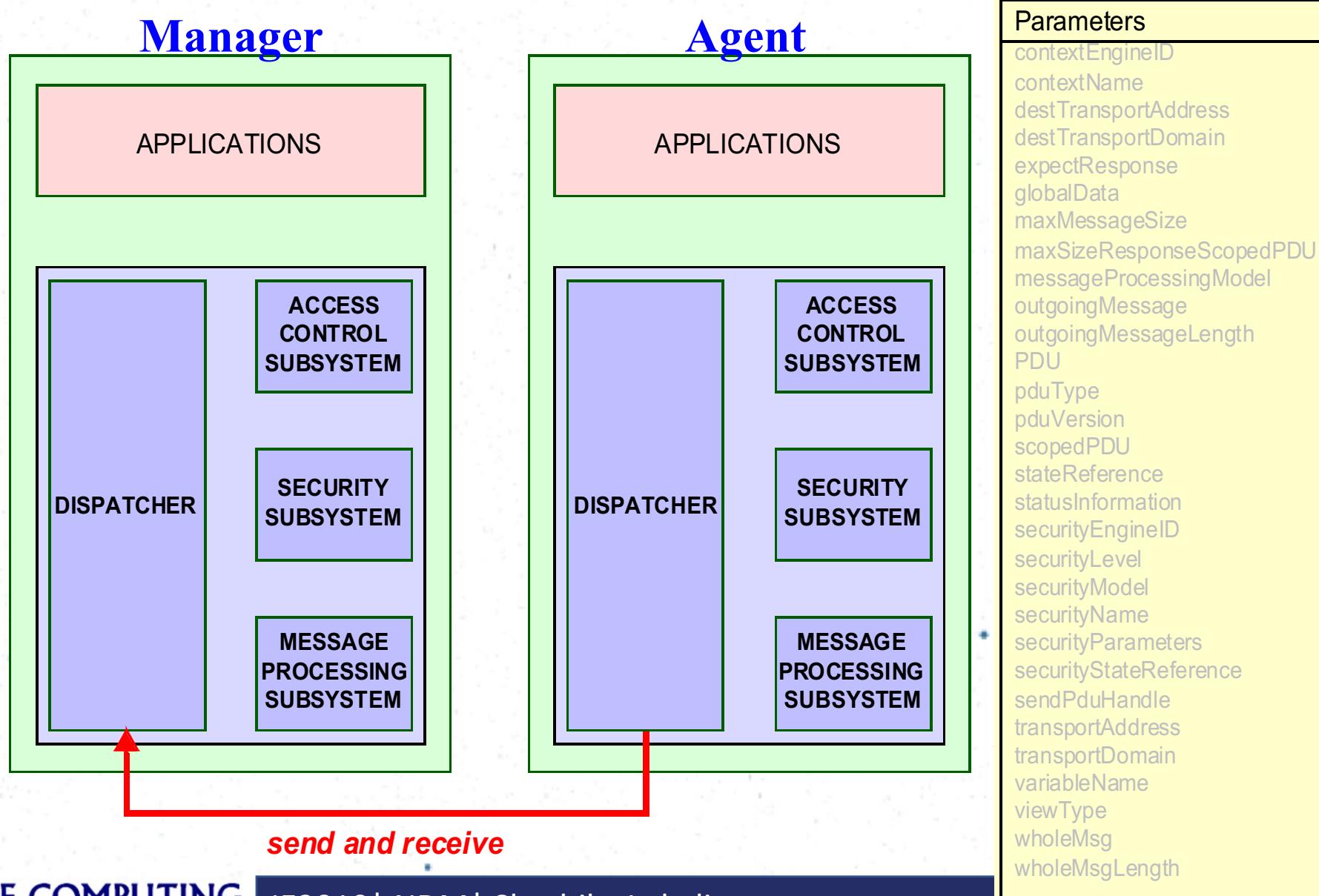
# prepareResponseMessage



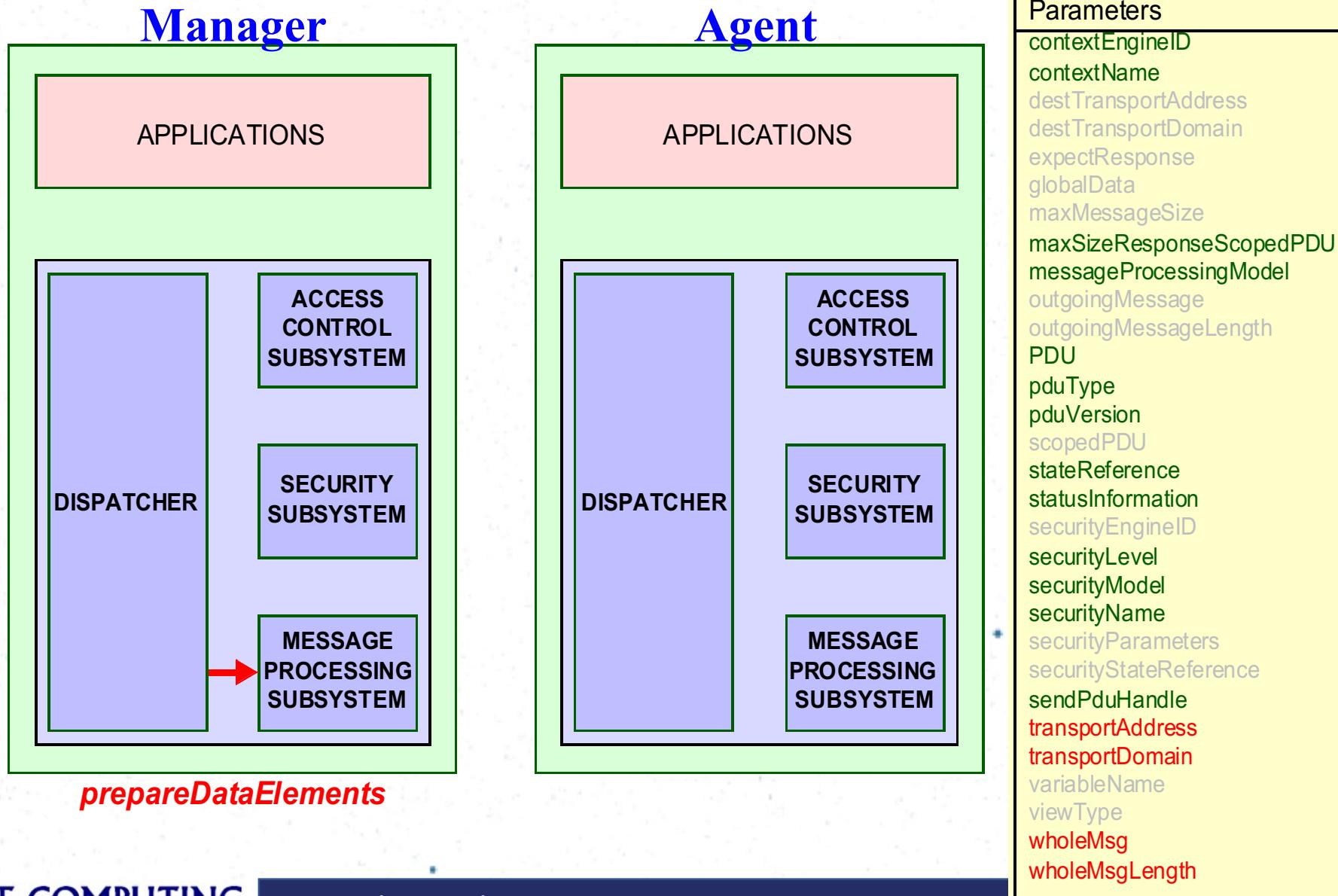
# generateResponseMsg



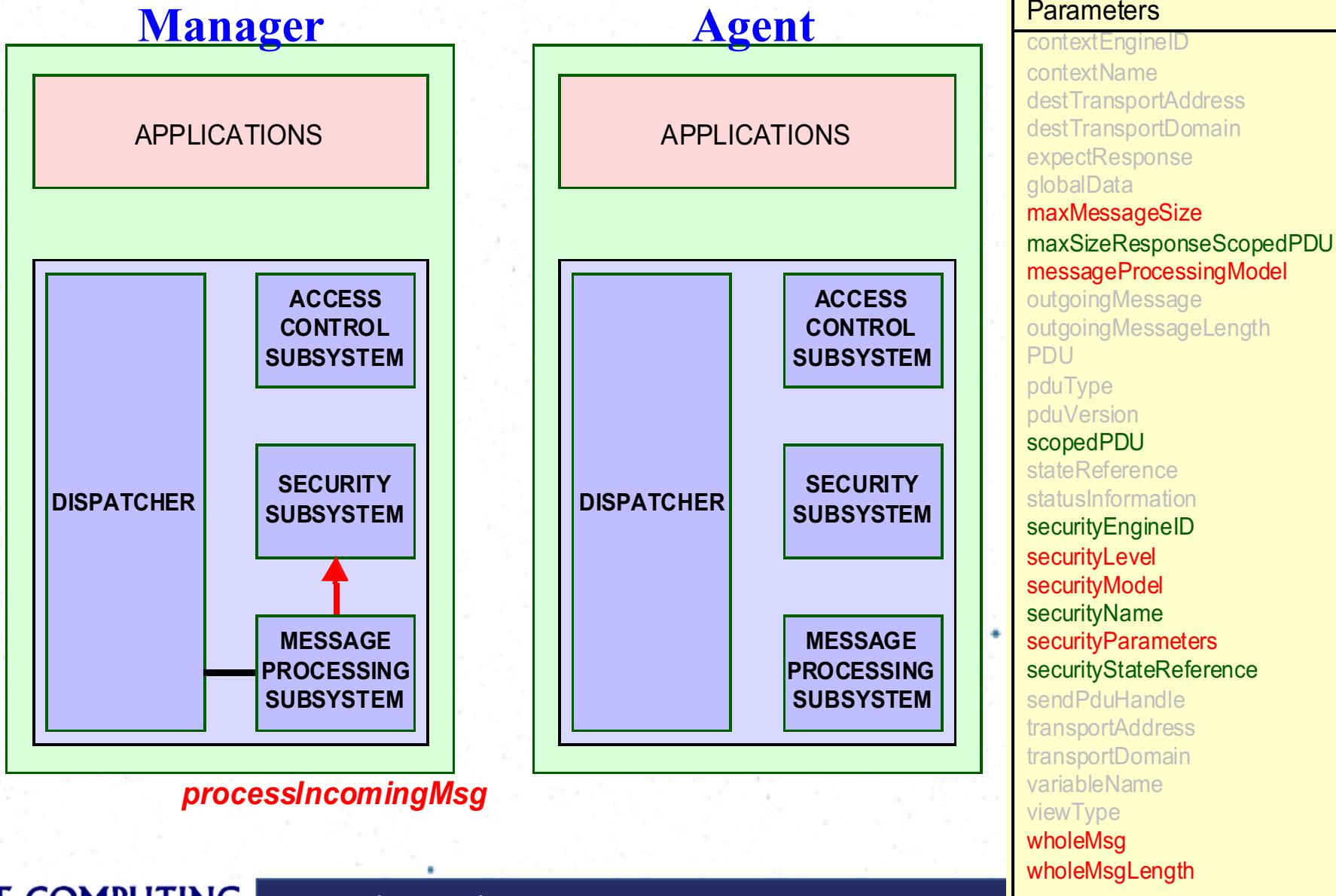
# send / receive



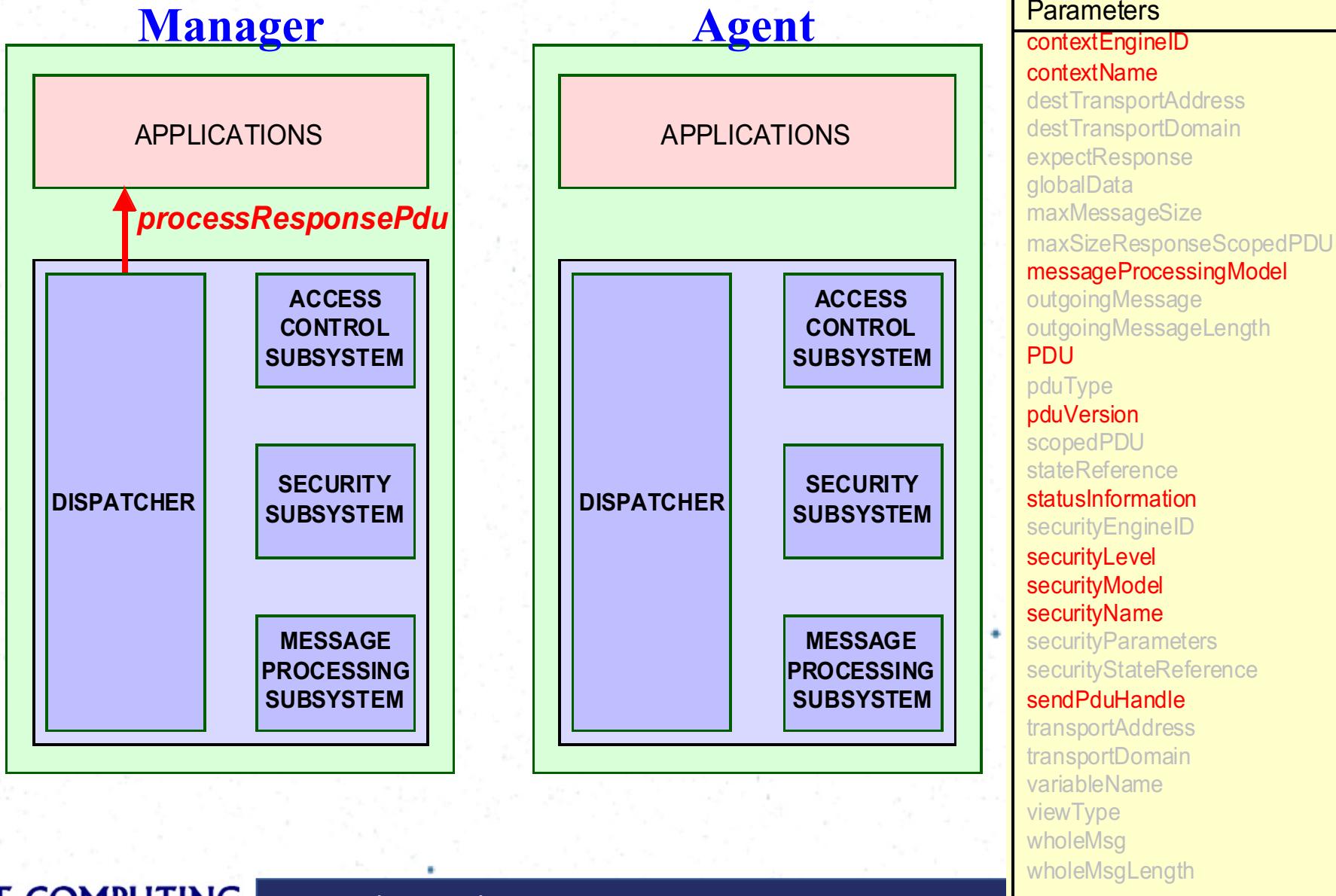
# prepareDataElements



# processIncomingMsg



# processResponsePdu



# SNMP RFC's

RFC	Description	Published	Current Status
1065	SMLv1	Aug-88	Obsoleted by 1155
1066	SNMPv1 MIB	Aug-88	Obsoleted by 1156
1067	SNMPv1	Aug-88	Obsoleted by 1098
1098	SNMPv1	Apr-89	Obsoleted by 1157
1155	SMLv1	May-90	Standard
1156	SNMPv1 MIB	May-90	Historic
1157	SNMPv1	May-90	Standard
1158	SNMPv1 MIB-II	May-90	Obsoleted by 1213
1212	SNMPv1 MIB definitions	Mar-91	Standard
1213	SNMPv1 MIB-II	Mar-91	Standard
1215	SNMPv1 traps	Mar-91	Informational
1351	Secure SNMP administrative model	Jul-92	Proposed Standard
1352	Secure SNMP managed objects	Jul-92	Proposed Standard
1353	Secure SNMP security protocols	Jul-92	Proposed Standard
1441	Introduction to SNMPv2	Apr-93	Proposed Standard
1442	SMLv2	Apr-93	Obsoleted by 1902
1443	Textual conventions for SNMPv2	Apr-93	Obsoleted by 1903
1444	Conformance statements for SNMPv2	Apr-93	Obsoleted by 1904
1445	SNMPv2 administrative model	Apr-93	Historic

RFC	Description	Published	Current Status
1448	SNMPv2 protocol operations	Apr-93	Obsoleted by 1905
1449	SNMPv2 transport mapping	Apr-93	Obsoleted by 1906
1450	SNMPv2 MIB	Apr-93	Obsoleted by 1907
1451	Manger-to-manger MIB	Apr-93	Historic
1452	Coexistence of SNMPv1 and SNMPv2	Apr-93	Obsoleted by 1908
1901	Community-Based SNMPv2	Jan-96	Experimental
1902	SMLv2	Jan-96	Draft Standard
1903	Textual conventions for SNMPv2	Jan-96	Draft Standard
1904	Conformance statements for SNMPv2	Jan-96	Draft Standard
1905	Protocol operations for SNMPv2	Jan-96	Draft Standard
1906	Transport mapping for SNMPv2	Jan-96	Draft Standard
1907	SNMPv2 MIB	Jan-96	Draft Standard
1908	Coexistence of SNMPv1 and SNMPv2	Jan-96	Draft Standard
1909	Administrative infrastructure for SNMPv2	Feb-96	Experimental
1910	User-based security for SNMPv2	Feb-96	Experimental

# SNMP Security

- SNMPv1 uses plain text community strings for authentication as plain text without encryption
- SNMPv2 was supposed to fix security problems, but effort derailed (The “c” in SNMPv2c stands for “community”).
- SNMPv3 has numerous security features:
  - Ensure that a packet has not been tampered with (**integrity**),
  - Ensures that a message is from a valid source (**authentication**)
  - Ensures that a message cannot be read by unauthorized (**privacy**).



# IT3010

## Network Design and Management

### Lecture 03

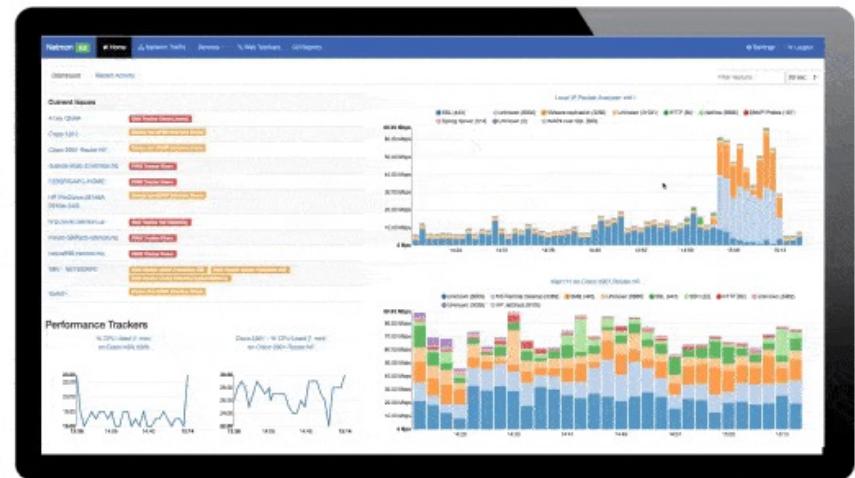
#### Network Monitoring

# Shashika Lokuliyan

Faculty of Computing  
Department of CSE

# Network Design and Management

Network Monitoring  
Lecture 5



# Today's lecture overview

- Definition of Network Monitoring
- Active vs. Passive monitoring
- Categories for monitoring
  - Network specifications: *Ethernet*
  - Network traffic and protocols
  - Platforms and operating systems (next week lecture)

# A definition for Network Monitoring

## WIKIPEDIA

The term network monitoring describes the use of a system that constantly monitors a computer network for slow or failing components and that notifies the network administrator (via email, SMS or other alarms) in case of outages. It is a subset of the functions involved in network management.



Monitoring an active communications network in order to diagnose problems and gather statistics for administration and fine tuning.



# What is Network Monitoring??

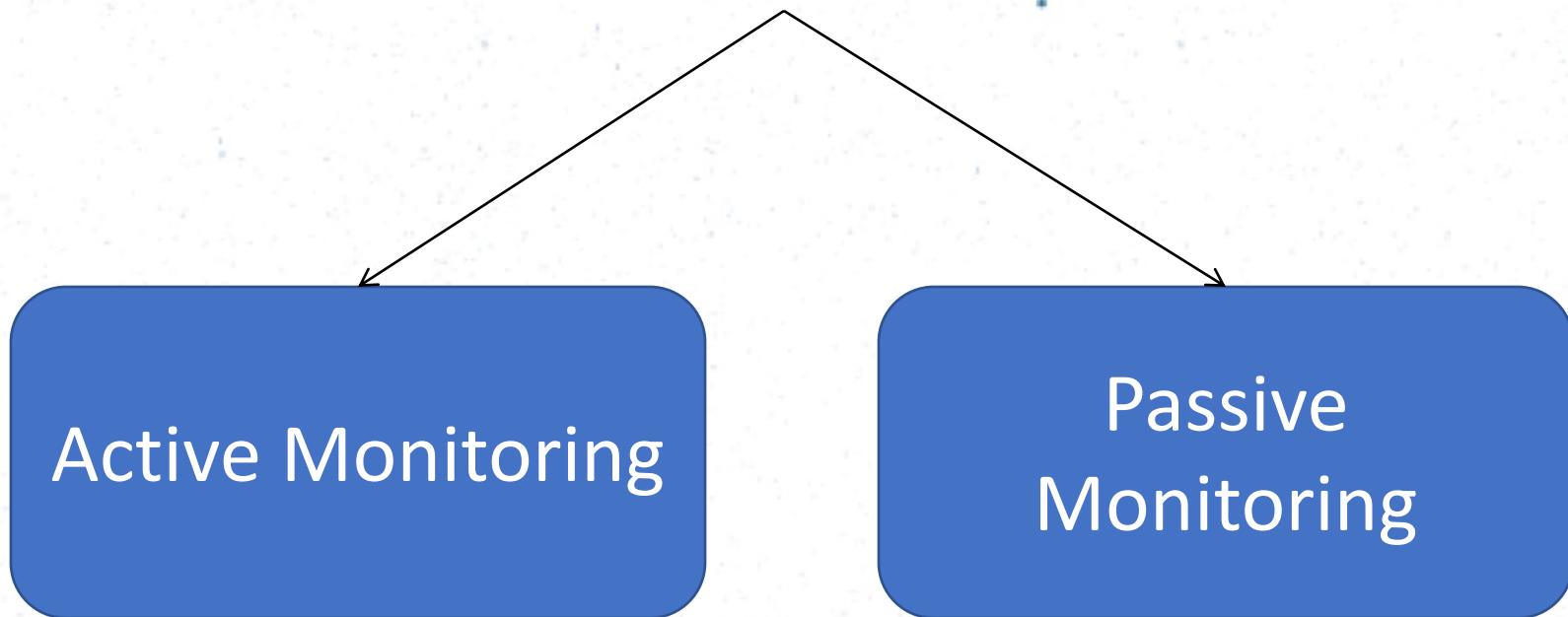
- Write it in your terms.



Kevin had a funny feeling that his  
boss was monitoring his emails

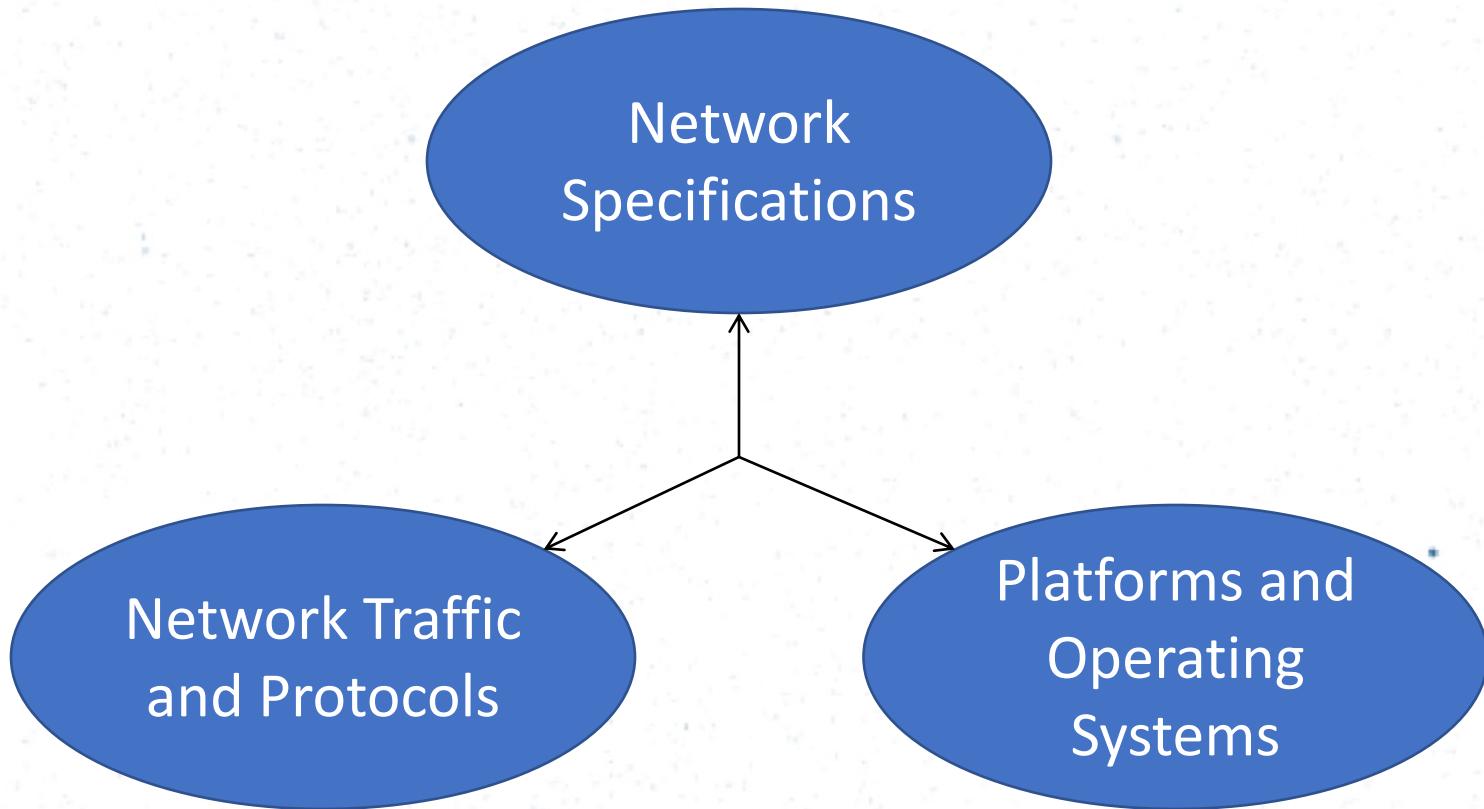
# Types of Network Monitoring

Two types of Network Monitoring



# Monitoring Categories

*Things we will need to monitor...*



# Monitoring and analysis of Network Specifications

*Ethernet*

# Establishing an Ethernet Baseline

Things to monitor with respect to Ethernet..

- Network utilization
- ...
- ...
- Collision rate
- Errors

# Where it all starts..

```
Router# show interfaces ethernet 0
Ethernet 0 is up, line protocol is up
    Hardware is MCI Ethernet, address is aa00.0400.0134 (via 0000.0c00.4369)
        Internet address is 131.108.1.1, subnet mask is 255.255.255.0
        MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
        Encapsulation ARPA, loopback not set, keepalive set (10 sec)
            ARP type: ARPA, PROBE, ARP Timeout 4:00:00
            Last input 0:00:00, output 0:00:00, output hang never
            Output queue 0/40, 0 drops; input queue 0/75, 2 drops
            Five minute input rate 61000 bits/sec, 4 packets/sec
            Five minute output rate 1000 bits/sec, 2 packets/sec
            2295197 packets input, 305539992 bytes, 0 no buffer
            Received 1925500 broadcasts, 0 runts, 0 giants
            3 input errors, 3 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
                0 input packets with dribble condition detected
            3594664 packets output, 436549843 bytes, 0 underruns
            8 output errors, 1790 collisions, 10 interface resets, 0 restarts
```

# Ethernet Utilization

- **Utilization** is a network performance measure that specifies the amount of time a LAN spends successfully transmitting data.
- Many performance monitoring tools will provide a user with **average and peak utilization times**, which are **reported as a percentage**.

*Fact*                   => Delays occur 40% to 50%

*Reason*   => Due to increased collisions

*Solution*  => .....

*Expectation* => Should achieve 15% to 25%

# Peak Utilization

Peak utilization means that, .....

....., a certain percentage of the LAN's capacity was utilized.

- Need to look at
  - Protocols
  - Devices
  - Users
- Determine when peaks occur

# Average Utilization

Average utilization means that .....

.....(e.g. 10 hours), on average, a certain percentage of the LAN's capacity is used for successfully transmitting data. In simple terms this is the calculated level over longer time.

## ■ What are we averaging?

```
Last input 0:00:00, output 0:00:00, output hang never  
Output queue 0/40, 0 drops; input queue 0/75, 2 drops  
Five minute input rate 61000 bits/sec, 4 packets/sec  
Five minute output rate 1000 bits/sec, 2 packets/sec
```

```
2295197 packets input, 305539992 bytes, 0 no buffer
```

```
0 dropped, 1025500 overruns, 0 underruns, 0 collisions
```

## □ What is bits-per-second?

# Current Utilization

Current utilization is the moving average calculated over a small time period (e.g. 5 minutes).

$$\text{new average} = ((\text{average} - \text{interval}) * \exp(-t/C)) + \text{interval}$$

Where:

- t is five seconds, and C is five minutes.  $\exp(-5/(60*5)) == .983$ . This value is known as the “weighting factor” or “decay factor”.
- newaverage = the value we are trying to compute.
- average = the “newaverage” value calculated from the previous sample.
- interval = the value of the current sample.

# Additional Resources for Utilization Monitoring

Please **make sure to read** the following PDF documents uploaded to the course web.

1. Extracted\_from\_Networking\_Explained\_Part\_1.pdf (*2 pages*)
2. Extracted\_from\_Networking\_Explained\_Part\_2.pdf (*2 pages*)
3. Understanding\_the\_bits\_per\_second.pdf (*3 pages*)

## Note

Please note that the first two documents in the above list are two parts of the same document. You should refer starting from Ques #26 in part 1 and continue up to and including Ques #32 in part 2.

# Broadcasts

```
2295197 packets input, 305539  
Received 1925500 broadcasts,  
3 input errors 3 CRC 0 fram
```

**Broadcast**

+

**Multicast**

- Excessive amounts of broadcast or multicast traffic,
- Broadcasts Rate should not exceed 5-10%

# Multicasts

- Communication between **small groups of devices**.
- Same rules as broadcast.

# Examining Ethernet Errors

- Collisions
- Short frames
- Bad FCS
- Long frames
- Ghosts

# Collisions

```
sets output, 436549843 bytes  
ors, 1790 collisions, 10 i
```

If two frames are transmitted **simultaneously** by two stations, they overlap in time and the resulting signal is garbled. This event is known as a collision.

- Collisions are normal
- CSMA/CD
- Jam signal

# Additional Resources for Collisions

- Please **make sure to read** chapter, “4.2.2 Carrier Sense Multiple Access Protocols (from Pg 255 to Pg 258)” of Tanenbaum’s book.
- Please **make sure to read** the following PDF documents uploaded to the course web.
  1. Causes\_for\_collisions.pdf (*1 page*)
  2. Troubleshooting\_collisions.pdf (*6 pages*)

# Short Frames

```
305539992 bytes, 0 :  
asts, 0 runts, 0 gi  
0 frame. 0 overrun.
```

- A short frame is a frame **smaller than the minimum legal size of 64 bytes**, with a good frame check sequence.
- Caused by,

# Bad FCS (Frame Check Sequence)

500 broadcast  
s, 3 CRC, 0  
ackets with

- A received frame that has a bad Frame Check Sequence, also referred to as a checksum or CRC error, **differs from the original transmission by at least one bit.**
- In an FCS error frame, **the header information is probably correct and the frame may also have a valid size**, but the checksum calculated by the receiving station does not match the checksum appended to the end of the frame by the sending station. The frame is then discarded.

# Long Frames

- A long frame is a frame **larger than the maximum legal size of 1518 bytes.**
- It does not consider whether or not the frame had a valid FCS checksum.
- Causes

```
es, 0 no buffer
, 0 giants
errun. 0 ignored
```

# Ghosts

- Ghosts are classified as energy (noise) detected on the cable that appears to be a frame, but is lacking a valid SFD.
- To qualify as a ghost, the frame must be at least 72 bytes long, including the preamble.
- Slows network, not increased utilization.
- Causes,

# Documentation

Ethernet Baseline Statistics			
Network-Based		Node-Based	
% Utilization - Peak		% Utilization - Peak	
% Utilization - Average		% Utilization - Average	
Frames/Second - Peak		Frames/Second - Peak	
Frames/Second - Average		Frames/Second - Average	
Frame size - Peak		Frame size - Peak	
Frame size - Average		Frame size - Average	
Total Frame Count		Total Frame Count	
Total Byte Count		Total Byte Count	
Node count - Total		Node/Node Interaction - Total	
Top 10 Nodes		Node/Node Int. - Predominant	
Protocol count - Total		Protocol count - Total	
Protocol count - Top 3		Protocol count - Top 3	
Network Errors		Station Errors	
Collisions - Total		Collisions - Total	
Collisions/Second		Collisions/Second	
Runts/Fragments - Total		Runts/Fragments - Total	
Jabbers - Total		Jabbers - Total	
# of CRC/FCS Errors - Total		# of CRC/FCS Errors - Total	

# Additional Resources for Monitoring the Ethernet

Please **make sure to read** the following PDF documents uploaded to the course web.

1. Ethernet\_errors.pdf (*5 pages*)
2. Troubleshooting\_ethernet.pdf (*12 pages*)

# Monitoring and analysis of the Network Traffic

# Network Traffic

What & how should we measure..?

- Measure amount and type
  - Need hardware tools

What are possible types to monitor..?

- Number of Nodes/Users
- Protocols
- Broadcast/Multicast/Unicast
- Conversations
- Errors

# Number of Nodes/Users

- Workstations
- Servers
- Peripherals
- Routers and switches
- Who is on the network
- Physical access

# Protocols

- Device dependent
- Segment dependent

# How much of your traffic is overhead protocols

## ARP – Address Resolution Protocol

To find the physical address for a given logical address.

## DNS – Domain Name Service

To find the IP address for a given domain name.

## ICMP – Internet Control Message Protocol

One of the core protocols of the Internet Protocol Suite used primarily for the purpose of sending error messages.

# How much of your traffic is overhead protocols

LDAP – Lightweight Directory Access Protocol

For the purpose of accessing and maintaining distributed directory information services.

RIP, EIGRP, OSPF etc.

For the purpose of managing network devices.

# Connections

- Who is talking to who?
  - How much?
    - Routers
    - Servers
- Applications
  - What applications are on the network
  - What protocols are they using
  - Which users access them

# Where do errors occur?

- 65% to 75% of network errors occur in the first three layers

- Causes

- Duplicate addresses
- Host/Station/Network unreachable
- Time-To-Live (TTL) exceeded

# Monitoring and analysis of Platforms and Operating Systems

# Determining Server Workload Characterization

What is **workload characterization..?**

- Within the confines of a network, **workload** is the **amount of work assigned to, or done by**, a client, workgroup, **server**, or internetwork in a given time period.
- Therefore, **workload characterization** is the science that observes, identifies and explains the phenomena of work in a manner that simplifies your understanding of how the client, workgroup, **server**, or internetwork **is being used**.

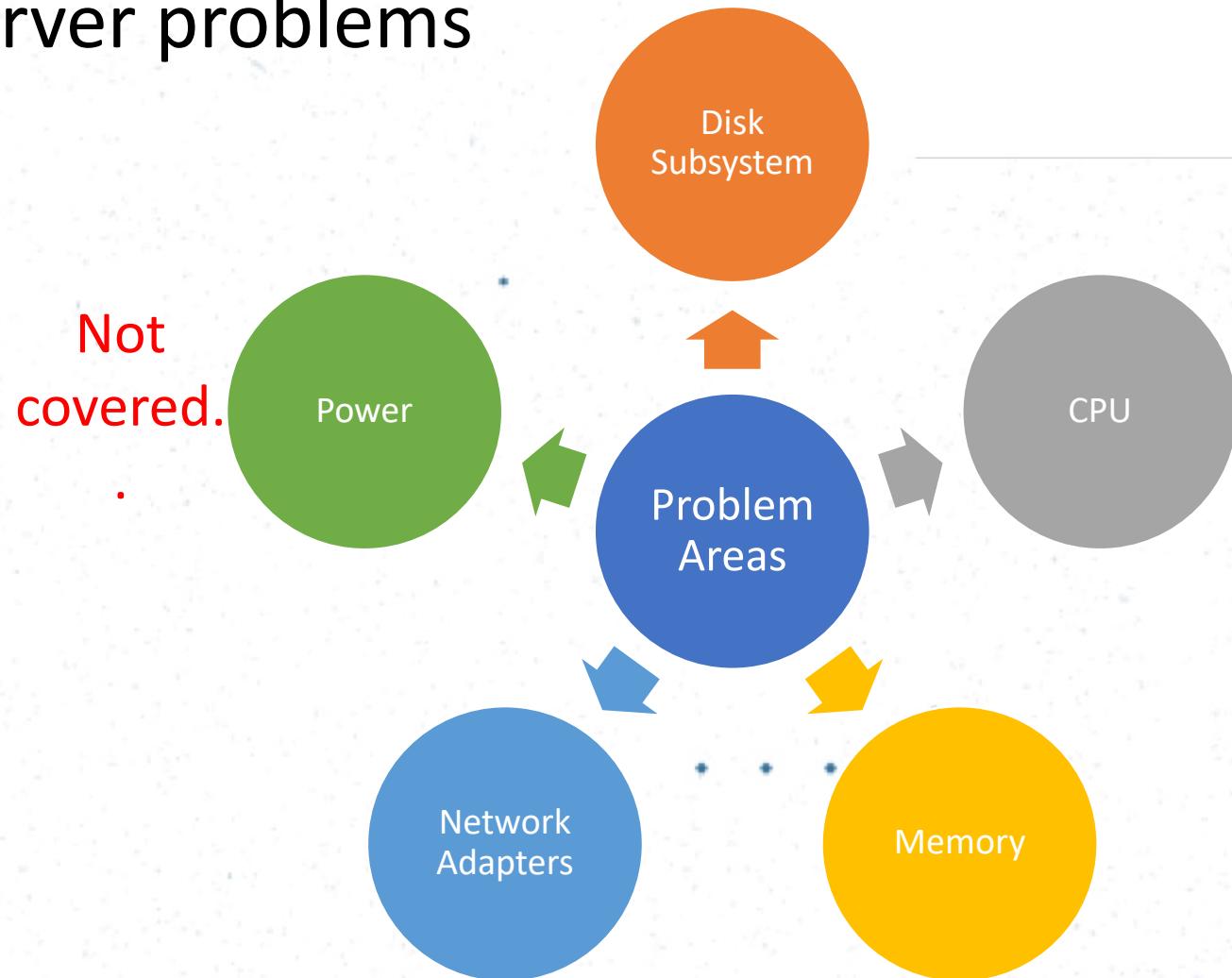
# Determining Server Workload Characterization

*Things that should be considered..*

- Server type
- Workload characterization
- Isolate components that restrict data flow
- Set expectations

# What are common server problems

Problems can occur in..



# What are common server problems

## Disk Subsystem

- The disk subsystem is more than the disk itself.
- It will include,
- Problems can occur with any of the above components..!!

### Note

In NT based windows server environments, the disk subsystem is divided into two part for ease of monitoring and troubleshooting.

- Physical disk - used for the analysis of the overall disk, despite the partitions that may be on the disk.
- Logical disk - analyzes information for a single partition.

# What are common server problems

## CPU

- Most server machines today, support 1-4, 1-8 or even 1-16 processors.
- And each processor can have up to 18 CPU cores.
- **That is A LOT to monitor and troubleshoot..!!**

*Leads to common problems..*

- Overheating due to not been correctly thermally bonded with heat sink during installation & replacement.
- Mismatches between CPU and memory speeds.
- Different CPUs populated with different number and size of memory modules.
- Etc.. Etc..

# What are common server problems

## Memory

- In server machines each processor can be populated with one or more memory modules.
- Some modern server stations even support up to 96 memory modules.

*Leads to common problems..*

- The number and size of modules not same for all CPUs in a server.
- Memory module not seated properly in the slot.
- Using modules having different speeds.
- Memory module not supported by the particular server model.
- Etc.. Etc..

# What are common server problems

## Network Adapters

- Some modern server stations can support a large number of NIC ports, even up to 16 ports.
- Larger the number of ports become, so does the complexity of troubleshooting..!!

*Leads to common problems..*

- Not loading the appropriate firmware version for the adapters.
- If/when using dual adapters, not following the restrictions on the supported combinations.
- Etc.. Etc..

# File and Print Servers

- File and printer servers manage the storage of data and the various printers on the network.  
E.g. Windows Server 2008, Mac OS X Server, Red Hat Linux Server, Ubuntu Server Edition.
- **Key Concern:** Disk I/O or the number of user's attempting access to the server is the most critical concern.
- Focus on the number of users accessing server concurrently (also how they are accessing the server) and amount of resources demanded.

# Web Servers

- Web servers allow Internet users to attach to your server **to view and maintain web pages.**
- Ordering of problem areas to focus,  
Memory > Network >
- Must fulfill requests from **cache** to achieve maximum performance.

# Application Server

- Application server is a server that handles all application operations **between users and an organization's backend business applications or databases**. Aka *appserver*.
- **Features** include, built-in redundancy, monitor for high-availability, high-performance distributed application services and support for complex database access.
- Ordering of problem areas to focus,  
Memory >
- Application server **usually has smaller**, more frequent requests to it than File and Print Server environment.

# Logon Server/System Services

- As the name implies, logon server is used for the purpose of **authenticating users to the domain.**
- Logon servers can provide convenient authentication features like **Single Sign On (SSO)**, which enables the users to **access multiple applications/services using the same username and password.**
- Ordering of problem areas to focus,  
Processor > Disk
- Things to keep an eye on,
  - Activity generated between Servers.
  - Users - Peak activity more of a concern.

Why..?

# Factors affecting performance

- Performance degradation is proportional to the problems.
- Hence, areas that problems can occur are the same areas that will affect performance.
  - Disk Subsystem
  - Memory
  - CPU
  - Network

# Common Hard Disk Measurements

- Current Disk Queue Length
- % Disk Time
- Avg. Disk Queue Length
- Disk Reads/sec
- Disk Reads Bytes/sec
- Avg. Disk Bytes/Transfer
- Avg. Disk sec/Transfer

# Paging and Swapping

## Paging

- Move individual pages of process to the disk to reclaim memory.
- The paging algorithm keeps track of when each page was last used and tries to keep pages that were used recently in memory.

## Swapping

- Move an entire process to disk to reclaim memory.
- Next time the system runs the process, it has to copy it from the disk swap space back into memory.

*Revisit OS lecture slides..*

# Common Memory Measurements

- Page Faults/sec
- Pages Input/sec
- Pages Output/sec
- Pages/sec
- Page Reads/sec
- Page Writes/sec
- Available Memory
- Nonpageable memory pool bytes
- Pageable memory pool bytes
- Committed Bytes
- Pool Paged Bytes
- Pool NonPaged Bytes
- Working Set
- Paging File, %pagefile in use

# Common Processor (CPU) Measurements

- % Processor Time
- Interrupts/sec
- % Interrupt Time
- % User Time
- % Privilege Time
- % DPC Time
- % Processor Time
- **Processor Queue Length**
- System Calls/sec
- % Total Processor Time
- % Total User Time
- % Total Privilege Time
- % Total Interrupt Time

# Common Network Card Measurements

- Bytes Sent/sec
- Bytes Received/sec
- **Bytes Total/sec**
- % DPC Time
- DPCs queued/sec
- % Broadcasts
- % Multicasts
- Segments Sent/sec
- Segments Received/sec
- Segments/sec
- Segments Retransmitted/sec
- Connection Failures
- Connections Reset
- Connections Established
- **Server Sessions**
- **Output Queue Length**

## Further reading..

If you are **interested in knowing** some further information about performance counters you can refer the following PDF uploaded to moodle,

[\*\*Performance\\_Counters.pdf\*\*](#)

Don't overdo it...!!!

Excessive network monitoring (active) can and will slow your network...!!!

~ THE END ~



# IT3010

## Network Design and Management

### Special Topics in SNMP

### Shashika Lokuliyan

Faculty of Computing  
Department of CSE



# SLIIT

*Discover Your Future*

# Structure of management information (SMI)

3

# Goal of SNMP

Or at least the intention of the inventors...

- A world where a person's audio system, video system, HVAC system and toaster are all connected to the same network.
- To that end, computer scientists developed a protocol capable of managing **any network device**.
- The result was Simple Network Management Protocol (**SNMP**).

# Problem 1

- Different software languages have slightly different sets of data types (integers, strings, bytes, characters etc.).
- But an SNMP manager sending a message full of Java data types may not be understood by an SNMP agent written in C.
- The solution for this problem is to use ASN.1 defined data types.
- Since ASN.1 is independent of any particular programming language, the SNMP agent/manager can be written in any programming language.

# Problem 2

- When sending a particular data type over the wire, how should it be encoded?
- Should strings be null terminated as in the programming language C, or not? Should Boolean values be 8 bits as in C++ or 16 bits as in VB6?
- To address this problem ASN.1 includes Basic Encoding Rules (BER).
- To send a properly formatted message, the programmer must understand ASN.1 and BER encoding.

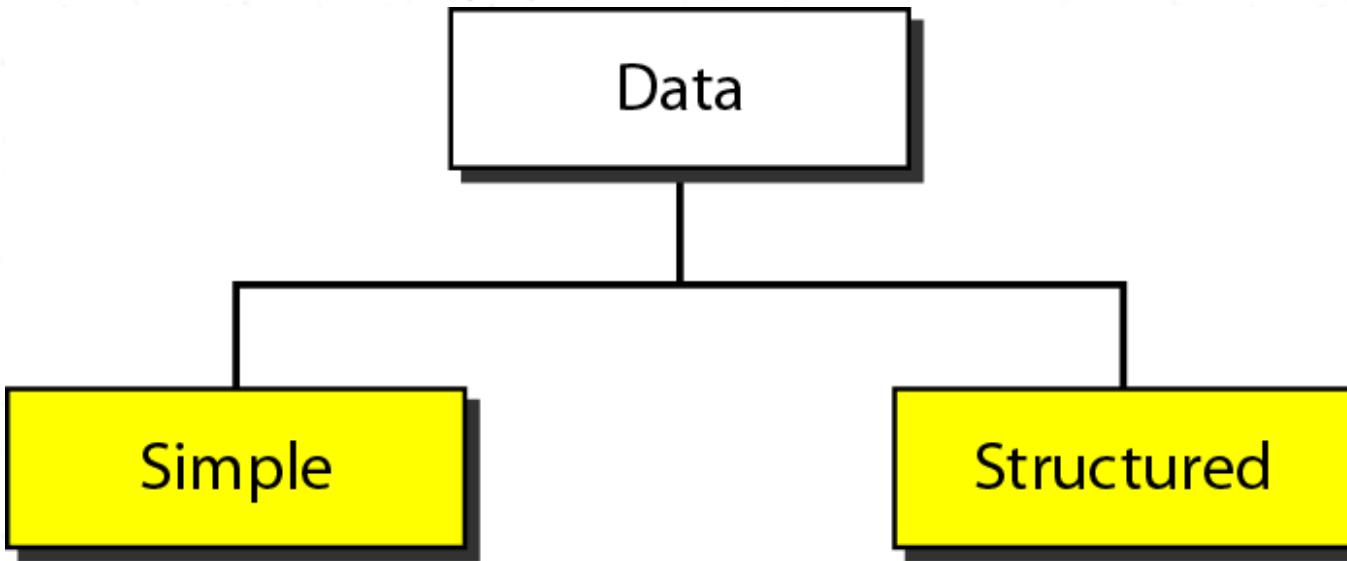
# Structure of Management Information (SMI)

- Adapted subset of **ASN.1**
- Structure of Management Information (SMI) is used to define the objects in MIBs.
  - Module definitions
    - MODULE-IDENTITY
  - Object definitions
    - OBJECT-TYPE
  - Notification definitions
    - NOTIFICATION-TYPE

# ASN.1

- Abstract syntax notation one
- Formal notation **for describing data structures and message formats**
- Type definitions, value definitions, combined
- Predefined basic types
  - BOOLEAN, INTEGER, OCTET STRING, BIT STRING, REAL, ENUMERATED, CHARACTER STRING, OBJECT IDENTIFIER
- Constructed types
  - SEQUENCE, SEQUENCE OF, CHOICE
  - Arbitrary nesting of types and sub-types
- Encoding

# data type



# Simple type

Type	Size	Description
INTEGER	4 bytes	An integer with a value between $-2^{31}$ and $2^{31} - 1$
Integer32	4 bytes	Same as INTEGER
Unsigned32	4 bytes	Unsigned with a value between 0 and $2^{32} - 1$
OCTET STRING	Variable	Byte string up to 65,535 bytes long
OBJECT IDENTIFIER	Variable	An object identifier
IPAddress	4 bytes	An IP address made of four integers
Counter32	4 bytes	An integer whose value can be incremented from 0 to $2^{32}$ ; when it reaches its maximum value, it wraps back to 0.
Counter64	8 bytes	64-bit counter
Gauge32	4 bytes	Same as Counter32, but when it reaches its maximum value, it does not wrap; it remains there until it is reset
TimeTicks	4 bytes	A counting value that records time in $\frac{1}{100}$ s
BITS		A string of bits
Opaque	Variable	Uninterpreted string

# Structured Types

Structured Types	Typical Use
SEQUENCE	Models an ordered collection of variables of different type
SEQUENCE OF	Models an ordered collection of variables of the same type
SET	Model an unordered collection of variables of different types
SET OF	Model an unordered collection of variables of the same type
CHOICE	Specify a collection of distinct types from which to choose one type
SELECTION	Select a component type from a specified CHOICE type
ANY	Enable an application to specify the type  <b>Note:</b> ANY is a deprecated ASN.1 Structured Type. It has been replaced with X.680 Open Type.

# ASN.1 types and values

- **Type definitions**

- NumberofStudents ::= INTEGER
- PassorFail ::= BOOLEAN
- GradeType ::= ENUMERATED {A, B, C}
- PointsScored ::= REAL
- Image ::= BIT STRING
- Data ::= OCTET STRING

- **Value definitions**

- studentsMonaySession  
NumberofStudents ::= 9
- NDMCourse PassorFail ::= TRUE
- NumberofStudents ::= 10

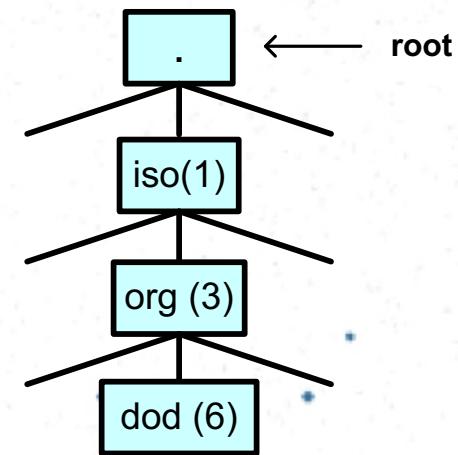
- **Combine type value definitions**

- StudentType ::= INTEGER {  
    ugrad (0)  
    ms (1)  
    phd (2)  
}
- NumberofStudents ::= 10

# ASN.1 structured types and values

# ASN.1 OBJECT IDENTIFIER (MIB)

- Define an information object that is managed at the international level
- internet OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) }



# ASN.1 MACRO (MIB)

provide the capability of defining types and values that are not included in the standard repertoire.

```
OBJECT-TYPE MACRO ::=  
  BEGIN  
    TYPE NOTATION ::= "SYNTAX" type (TYPE  
ObjectSyntax)  
      "ACCESS" Access  
      "STATUS" Status  
    VALUE NOTATION ::= value (VALUE ObjectName)
```

```
Access ::= "read-only"  
        | "read-write"  
        | "write-only"  
        | "not-  
accessible"  
Status ::= "mandatory"  
        | "current"  
        | "optional"  
        | "obsolete"  
END
```

# ASN.1 Encoding

- ASN.1 defines syntax and not how to encode them
- ASN.1 encoding rules
  - Basic encoding rules (BER) (will be discuss separately)
  - DER encoding rules (DER)
  - Canonical encoding rules (CER)
  - XML encoding rules (XER)
  - Packet encoding rules (PER)
  - Generic string encoding rules (GSER)

# Management Information Base (MIB)

# MIB

- A MIB specifies the managed objects
- MIB is a text file that describes managed objects using the syntax of ASN.1
- What is a managed object?
  - interface, TCP stack (RTO, congestion control alg.), ARP etc.
- In Linux, MIB files are in the directory */usr/share/snmp/mibs*
  - Multiple MIB files
  - MIB-II (defined in RFC 1213) defines the managed objects of TCP/IP networks

# Managed Objects

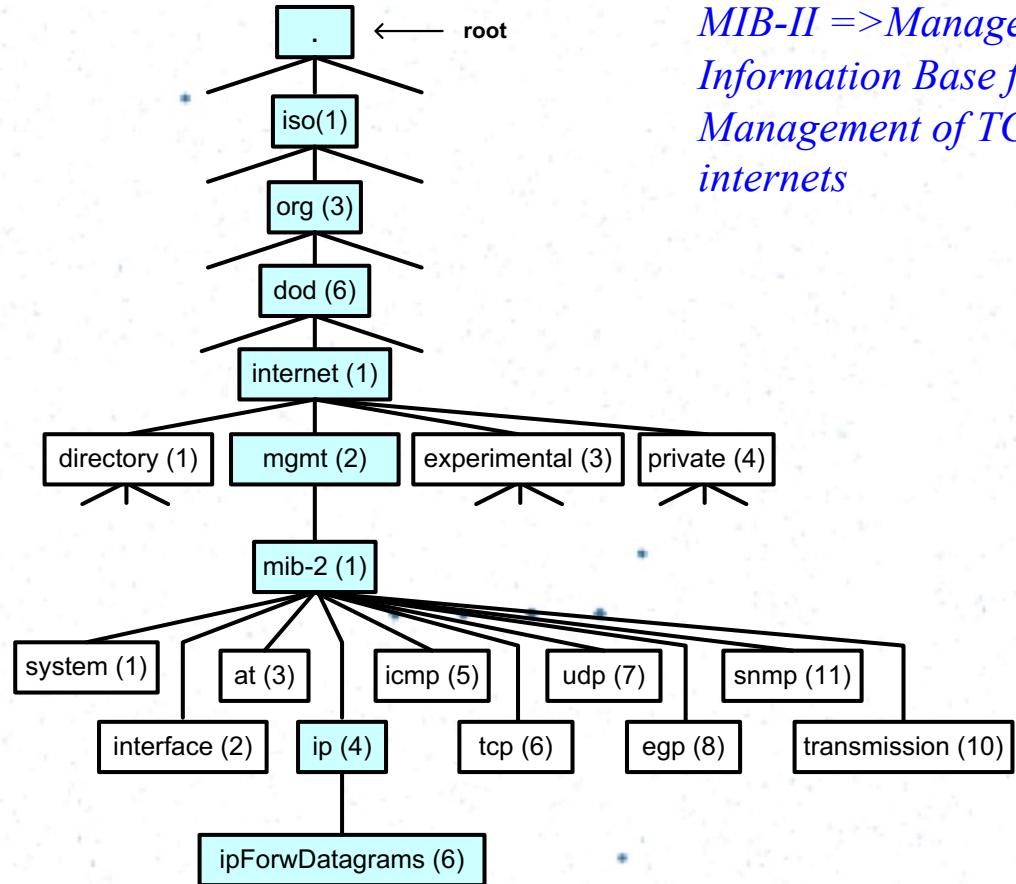
- Each managed object is assigned an *object identifier (OID)*
- The OID is specified in a MIB file.
- An OID can be represented as a sequence of integers separated by decimal points or by a text string:

*Example:*

- *1.3.6.1.2.1.4.6.*
- *iso.org.dod.internet.mgmt.mib-2.ip.ipForwDatagrams*
- When an SNMP manager requests an object, it sends the OID to the SNMP agent.

# Organization of managed objects

- Organized in a **tree-like hierarchy**
- **OIDs reflect the structure of the hierarchy.**
- Each OID **represents a node** in the tree.
- The OID **1.3.6.1.2.1** (*iso.org.dod.internet.mgmt.mib-2*) is at the top of the hierarchy for all managed objects of the **MIB-II**.
- Manufacturers of networking equipment can add product specific objects to the hierarchy.

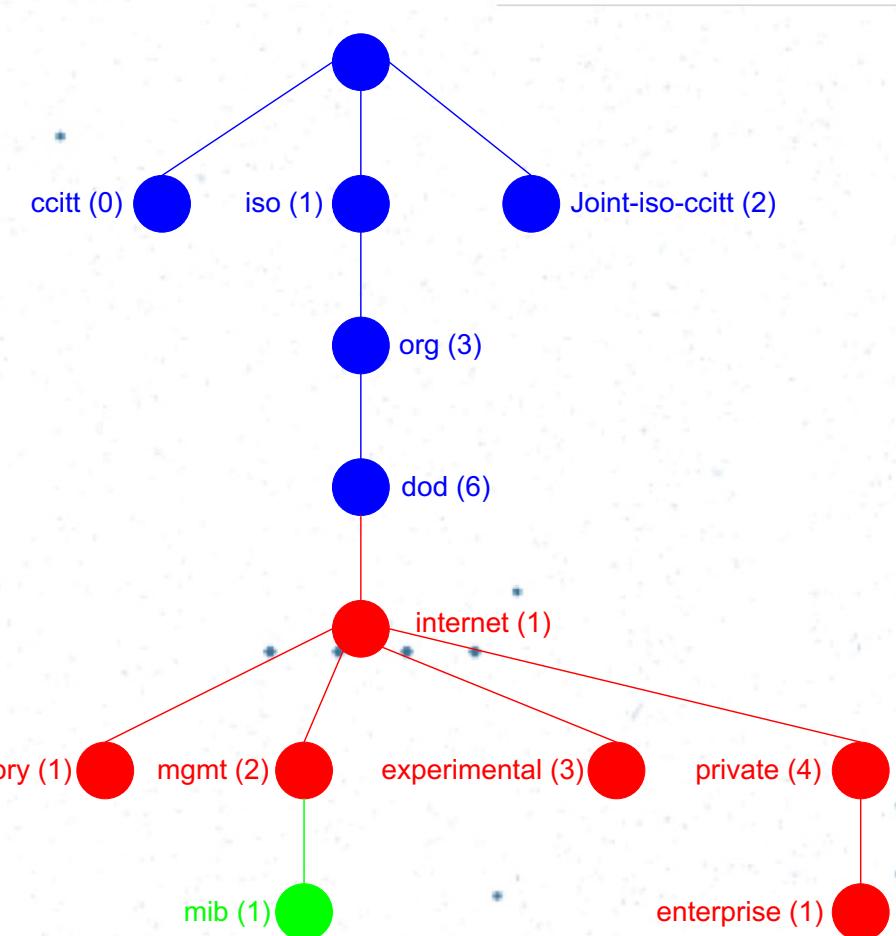


*MIB-II => Management Information Base for Network Management of TCP/IP-based internets*

# Organization of managed objects

RFC 1155 defines **top of the administrative domain managed by the IETF**:

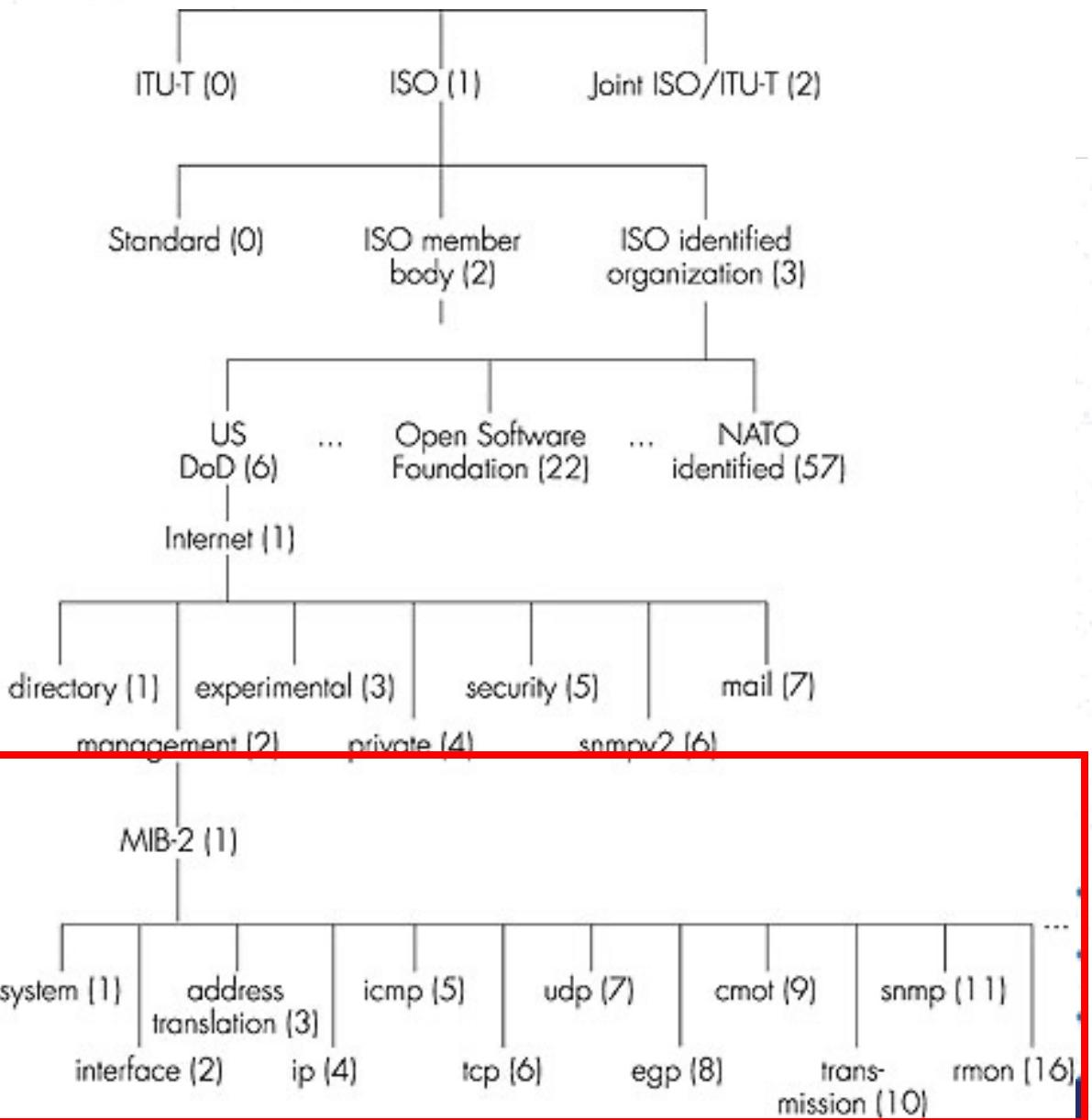
- 1.3.6.1 - Internet
  - ❑ **Directory** – reserved for use with a future memo that discusses how the OSI directory may be used in the Internet.
  - ❑ **Mgmt** – area for items defined in standards track documents.
  - ❑ **Experimental** – area for IETF experimental items.
  - ❑ **Private** – area for delegation of subtrees to **enterprises**, that is, anyone who asks for an enterprise number.



# MIB-II (RFC1213)

MIB-II defines **11 separate groups** :

- system (1)
- interfaces (2)
- address translation (3)
- ip (4)
- icmp (5)
- tcp (6)
- udp (7)
- egp (8)
- cmot (9) CMIS over TCP (for historic reasons only)
- transmission (10)
- snmp (11)



# System Group

- Contains **data pertaining to the system** where the agent is residing in.
- **Fault management objects:**
  - ❑ **SysObjectID** – System manufacturer.
  - ❑ **sysServices** – Protocol layers that device services, using formula  $2^{(L-1)}$ .
    - e.g. host that runs transport + application layer services.
    - $2^{(4-1)} + 2^{(7-1)} = 72$ .
  - ❑ **sysUptime** – Amount of time system has been operational.
- **Configuration management objects:**
  - ❑ **sysDescr** – Description of the system.
  - ❑ **sysLocation** – System's physical location.
  - ❑ **sysContact** – System's name.

# Interfaces Group

- The interfaces group provides information pertaining to each specific network interface (**ifTable**).
- Useful for configuration, performance, fault and accounting management.
- **ifNumber** - number of interfaces.
- **ifTable** example:

ifIndex	ifDescr	ifOperStatus	ifInUPackets	ifSpeed
0	DEC Ethernet 1	1	8169	8000000
1	SUN Ethernet 1	2	16184	100000

# Interfaces Group cont.

## Example – Determining Utilization

Total bytes =  $(\text{ifInOctects}_y - \text{ifInOctects}_x) + (\text{ifOutOctects}_y - \text{ifOutOctects}_x)$

Total bytes per sec = Total bytes / (y-x)

Utilization = (Total bytes per sec \* 8) / ifSpeed

# IP Group

- Provides information about the IP layer in a systems network protocol stack.
  - Information pertaining to errors and types of packets seen.
  - Routing table (i.e. `ipRouteTable`).
- Configuration/Fault management objects:
  - `ipForwarding` – If device is set up to route IP packets.
  - `ipAddrTable` – Addresses on the device.
  - `ipRouteTable` – Routing table.
- Performance management objects:
  - `iplnDiscards` – Rate of input datagrams discarded.
  - `iplnHdrErrors` – Rate of input header errors.
  - `iplnAddrErrors` – Rate of input address errors.
- Accounting management objects:
  - `ipOutRequests` – Number of IP datagrams sent.
  - `iplnDelivers` – Number of IP datagrams received.

# ICMP Group

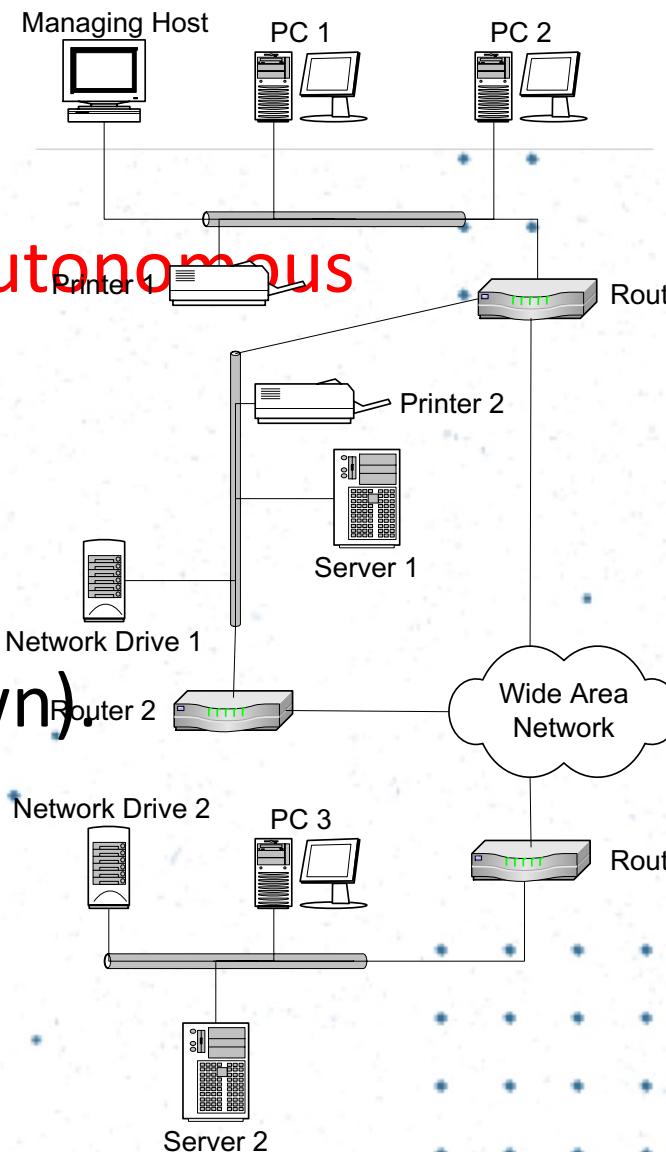
- Provides information pertaining to systems ICMP entity.
  - ❑ `icmpInRedirects` – rate of redirect messages received.
  - ❑ `icmpOutDestUnreachs` – rate of destination unreachable errors sent.
  - ❑ `icmpInSrcQuenches` – input rate of source quench messages (source quench msgs – requests that the sender decrease the rate of messages sent to a router or host).
  - ❑ `icmpOutEchos` – rate of output echo messages.
- Mainly useful for performance management.

# TCP Group

- TCP – The Transmission Control Protocol (TCP) provides reliable transport services between applications (UDP is the unreliable transport service).
- Configuration management objects:
  - `tcpRtoAlgorithm` – Retransmission algorithm.
  - `tcpConnTable` – Connection table (i.e. `netstat`).
- Performance management objects:
  - `tcpAttemptFails` – Number of failed attempts to make a connection.
  - `tcpEstabResets` – Number of resets in established connections.
- Accounting management objects:
  - `tcpActiveOpens` – Number of times this system has opened a connection.
  - `tcpInSegs` – Number of TCP segments received.
- Security management objects:
  - `tcpConnTable` – Connection table (i.e. `netstat`).
- The User Datagram Protocol (UDP) group provides similar information. (e.g. `udpTable`)

# EGP Group

- EGP (RFC 904) is a protocol that tests for the reachability of IP networks.
  - An IP network can be divided into networks of **autonomous systems**.
- **egpNeighTable** – information about this entity's EGP neighbors.
- **Fault management** objects:
  - **egpNeighState** – state of EGP neighbour (up,down).
- **Configuration management** objects:
  - **egpIntervalHello** – hello message interval.
  - **egpAs** – local EGP autonomous system.



# Transmission Group

- Reserved for **information pertaining to specific media** underlying the interfaces of a system.
- Various RFCs:
  - RFC 1512 FDDI.
  - RFC 1493 Bridge.
  - RFC 1743 Token Ring.

# SNMP Group

- Management protocols also need to be managed...!!!
  - Useful to all 5 areas of network management.
- Fault management objects:
  - **snmpInASNParseErrors** – Number of malformed SNMP messages.
  - **snmpInNoSuchNames** – Number of requests to invalid objects.
- Configuration management objects:
  - **EnableAuthenTraps** – Enables entity to send traps when authentication errors occur.
- Performance/Accounting management objects:
  - **snmpInPkts** – Rate of SNMP packets input.
  - **snmpInTraps** – Rate of traps input.
- Security management objects:
  - **snmpInBadCommunityNames** – Number of authentication failures.
  - **snmpInBadCommunityUses** – Number of requests without sufficient privileges.

# Definition of managed objects in a MIB

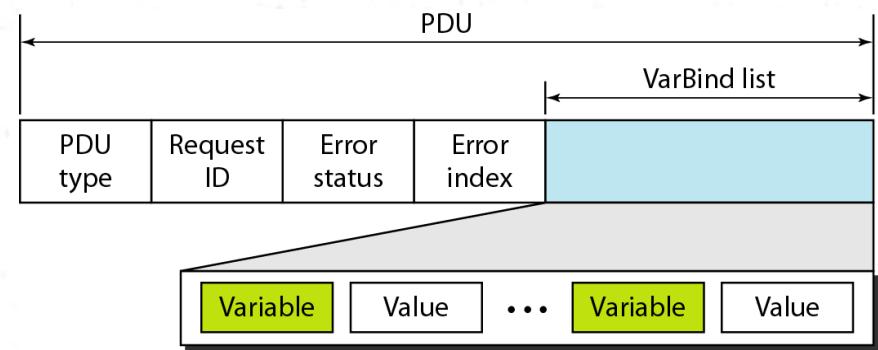
- Specification of **ipForwDatagrams** in MIB-II.

```
ipForwDatagrams OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS   current
    DESCRIPTION
        "The number of input datagrams for which this
        entity was not their final IP destination, as a
        result of which an attempt was made to find a
        route to forward them to that final destination.
        In entities which do not act as IP Gateways, this
        counter will include only those packets which were
        Source-Routed via this entity, and the Source-
        Route option processing was successful."
    ::= { ip 6 }
```

# BER encoding

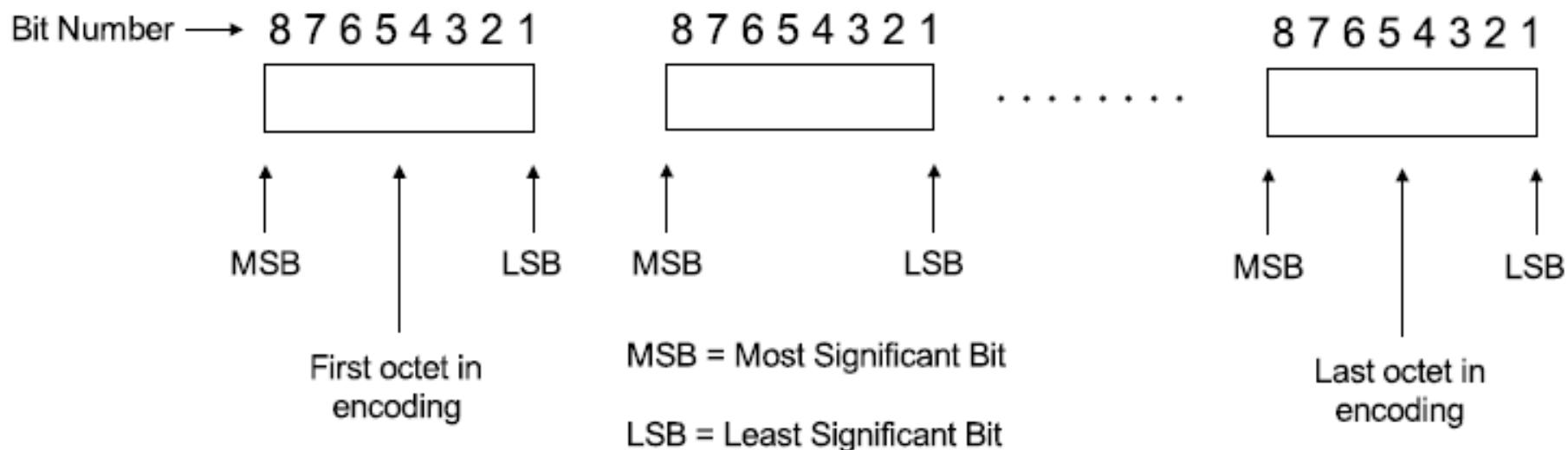
# Format of a BER encoded message

- In BER encoding, the most fundamental rule states that each field is encoded in three parts:
  - Type (or Tag)
  - Length
  - Value
- Hence, this is also known as TLV encoding.



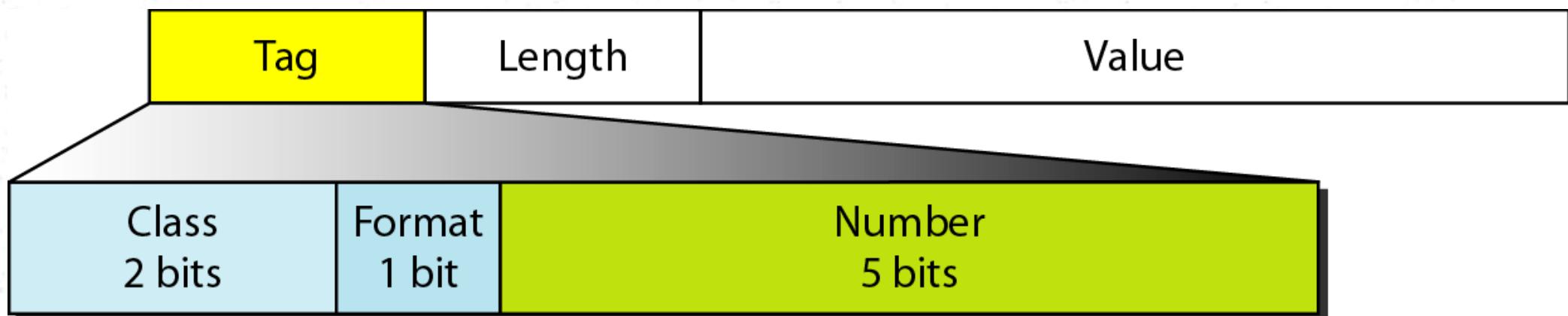
# Format of a BER encoded message cont..

- Like in most protocols, in BER also the Most Significant Bit (MSB)/Octet is encoded on the left.



# Type/Tag format

- Every BER encoded ASN.1 data type has a type field.
- This type can be one of four classes which is indicated by the first two bits of the type octet.



# Data Type : class

<b>Class</b>	<b>Bit 8</b>	<b>Bit 7</b>	
Universal	0	0	Primitive/ Constructed types
Application	0	1	Primitive SNMP application types
Context-specific	1	0	...
Private	1	1	SNMP PDU types

Class  
2 bitsFormat  
1 bitNumber  
5 bits

# ASN.1 Primitive Types

Data Type	Class	Format	Number	Type/Tag (Binary)	Type/Tag (Hex)
BOOLEAN	00	0	00001	00000001	01
INTEGER	00	0	00010	00000010	02
BIT STRING	00	0	00011	00000011	03
OCTET STRING	00	0	00100	00000100	04
NULL	00	0	00101	00000101	05
OBJECT IDENTIFIER	00	0	00110	00000110	06

Class  
2 bitsFormat  
1 bitNumber  
5 bits

# ASN.1 Constructed Types

Data Type	Class	Format	Number	Type/Tag (Binary)	Type/Tag (Hex)
SEQUENCE and SEQUENCE OF	00	1	10000	00110000	30



# Primitive SNMP Application Types

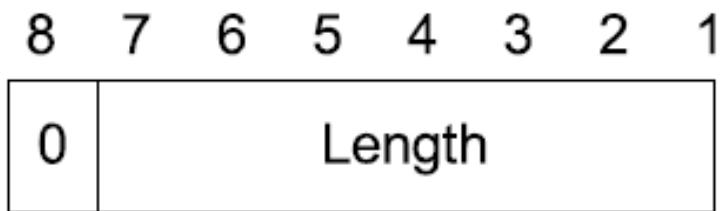
Data Type	Class	Format	Number	Type/Tag (Binary)	Type/Tag (Hex)
IpAddress	01	0	00000	01000000	40
Counter (Counter32)	01	0	00001	01000001	41
Gauge (Gauge32)	01	0	00010	01000010	42
TimerTicks	01	0	00011	01000011	43
Opaque	01	0	00100	01000100	44
NsapAddress	01	0	00101	01000101	45
Counter64	01	0	00110	01000110	46
UInteger32	01	0	00111	01000111	47



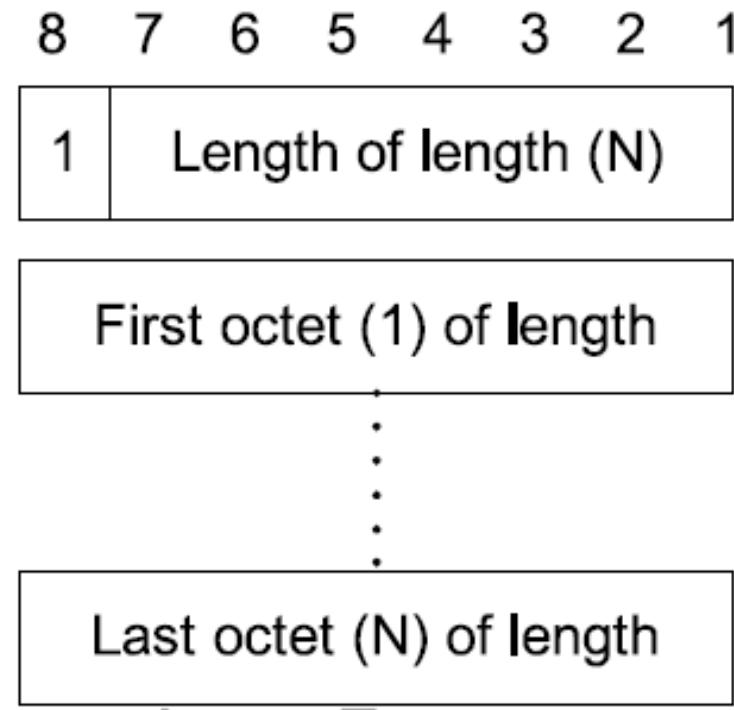
# Context Specific SNMP PDU Types

Data Type	Class	Format	Number	Type/Tag (Binary)	Type/Tag (Hex)
GetRequest	10	1	00000	10100000	A0
GetNextRequest	10	1	00001	10100001	A1
Get/Response	10	1	00010	10100010	A2
SetRequest	10	1	00011	10100011	A3
Trap	10	1	00100	10100100	A4
GetBulkRequest	10	1	00101	10100101	A5
InformRequest	10	1	00110	10100110	A6
SNMPv2 Trap	10	1	00111	10100111	A7

# Length format

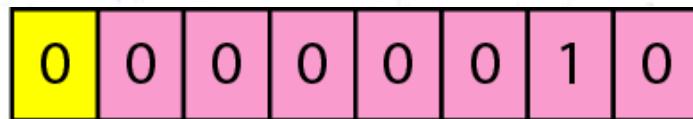


Short Form

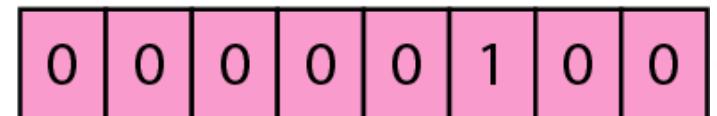
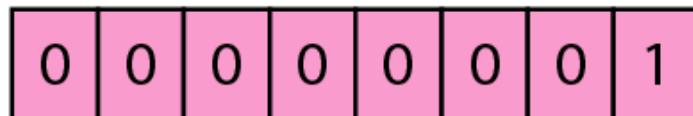
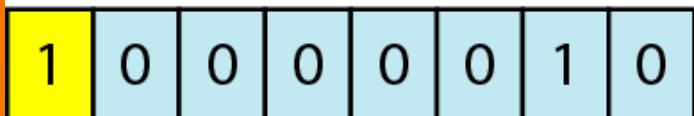


Long Form

# Length format



a. The colored part defines the length (2).



b. The shaded part defines the length of the length (2 bytes);  
the colored bytes define the length (260 bytes).

# Value Format

- How to define an integer value;
- *integer INTEGER ::= 14*

02	04	00	00	00	0E
00000010	00000100	00000000	00000000	00000000	00001110
Tag (integer)	Length (4 bytes)	Value (14)			

# Value Format

- How to define a string value;
- *Octetstring OCTET STRING ::= 'HI'*

04	02	48	49
00000100	00000010	01001000	01001001
Tag (String)	Length (2 bytes)	Value (H)	Value (I)

# Value Format

- How to define a null value;

ASN.1

null NULL ::= NULL

BER

null

T  
05

L  
00

V  
Empty

# Value Format

- *How to define ObjectIdentifier 1.3.6.1 (iso.org.dod.internet).*

06	04	01	03	06	01
00000110	00000100	00000001	00000011	00000110	00000001
Tag (Objectld)	Length (4 bytes)	Value (1)	Value (3)	Value (6)	Value (1)

← 1.3.6.1 (iso.org.dod.internet) →

# Encoding OBJECT IDENTIFIER

- Two rules apply when encoding OIDs using BER.
- The first rule states that, the first two numbers ‘x.y’ of the OID are encoded as a single value using the formula  $(40*x)+y$ .
- The first two numbers of any SNMP related OID is always 1.3. Therefore the first two numbers of an SNMP related OID is always encoded as 43 or 0x2B, because  $(40*1)+3 = 43$ .

# Encoding OBJECT IDENTIFIER

- Second rule applies when encoding large numbers in OIDs that cannot be represented using one octet (i.e. one byte or 8 bits).
- For example, the OID 1.3.6.1.4.1.2680.1.2.7.3.2 contains 2680 which cannot be encoded using a single octet (since 8 bits can only represent 0-255).
- The rule indicates that, when encoding large numbers in OIDs, only the lower 7 bits of the octet are used for holding the actual value (0-127). The highest order bit is used as a flag to indicate that this number spans more than one byte.

# Encoding OBJECT IDENTIFIER

- According to the rule discussed for large numbers in OIDs, value of 2680 will be encoded as 0x8A 0x78.
- And the fully BER encoded value for OID 1.3.6.1.4.1.2680.1.2.7.3.2 will be,

T	L	V
06	0C	2B 06 01 04 01 8A 78 01 02 07 03 02

# Value Format

- How to define IPAddress 131.21.14.8

40	04	83	15	0E	08
01000000	00000100	10000011	00010101	00001110	00001000
Tag (IPAddress)	Length (4 bytes)	Value (131)	Value (21)	Value (14)	Value (8)

← 131.21.14.8 →

# Value Format

- How to define SEQUENCE or SEQUENCE OF

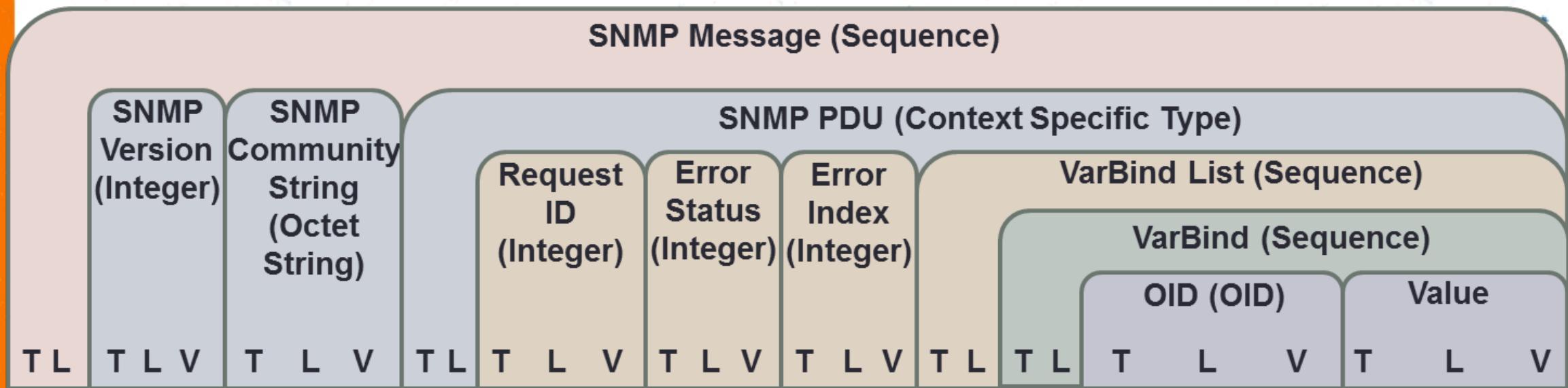
## ASN.1

Temperature-each-day SEQUENCE (3) OF INTEGER  
::= { 21, 15, -2}

## BER

Temperature-each-day:	T	L	V				
	30	9		T	L	V	
				02	01	15	
				02	01	0F	
				02	01	FE	
							⋮ ⋮ ⋮ ⋮

# BER encoded format of a SNMP message



You must remember this format...!!!

# How to organize the decoded values

Tag (SNMP Message), Length,  
Tag, Length, Version  
Tag, Length, Community  
Tag (PDU Type), Length  
Tag, Length, Request ID  
Tag, Length, Error Status  
Tag, Length, Error Index  
Tag (VarBind List), Length  
Tag (VarBind), Length  
Tag, Length, OID  
Tag, Length, Value

# Let's try an example

30 2B 02 01 00 04 08 53 65 63 75 72 69 74 79 A1 1C 02 04 3B 0B 16 36 02 01 00 02  
01 00 30 0E 30 0C 06 08 2B 06 01 02 01 01 02 00 05 00

30 2B	=> SEQUENCE
02 01 00	=> INTEGER 0
04 08 53 65 63 75 72 69 74 79	=> OCTET STRING Security
A1 1C	=> Context Specific 1 Constructor
02 04 3B 0B 16 36	=> INTEGER 3B0B1636
02 01 00	=> INTEGER 0
02 01 00	=> INTEGER 0
30 0E	=> SEQUENCE
30 0C	=> SEQUENCE
06 08 2B 06 01 02 01 01 02 00	=> OBJECT IDENTIFIER
05 00	=> NULL

# Example cont..

SEQUENCE => SNMP Message

INTEGER 0 => Version 0 (SNMPV1)

OCTET STRING Security => Community String

Context Specific 1 Constructor => GetNextRequest

INTEGER 3B0B1636 => Request ID (aka Sequence #)

INTEGER 0 => Error Status

INTEGER 0 => Error Index

SEQUENCE => VarBind List

SEQUENCE => VarBind

OBJECT IDENTIFIER => 1.3.6.1.2.1.1.2.0

NULL => Empty value

# Try this by yourself...

- 30 2F 02 01 00 04 08 53 65 63 75 72 69 74 79 A2 20  
02 04 3B 0B 16 36 02 01 00 02 01 00 30 12 30 10 06  
08 2B 06 01 02 01 01 03 00 43 04 1B E1 55 80

~ THE END ~

# SNMP Operations

## SET Request

Initializes or changes the value of a network element.

## GET Request

Sent by manager requesting data from agent.

## GETNEXT Request

Sent by manager requesting data on the next managed object to the one specified.

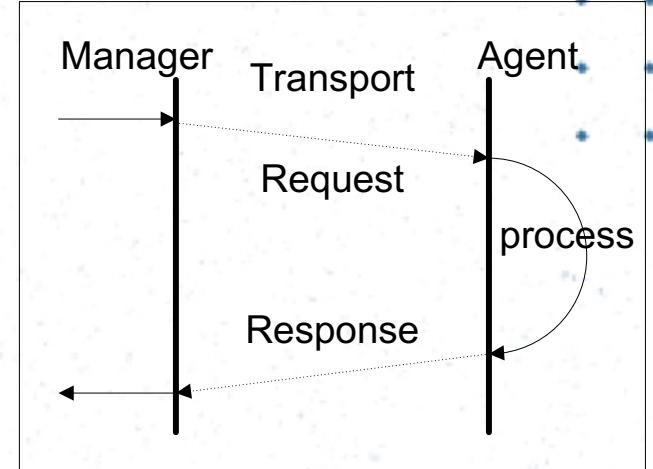
## GET Response

Used by the agent to respond with data to get and set requests from the manager.

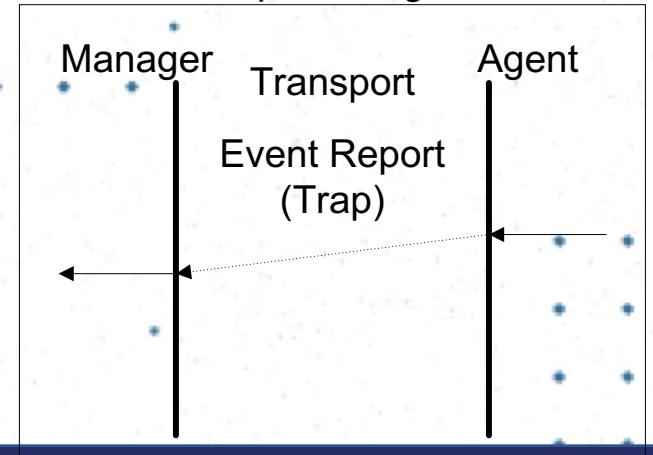
## Trap

Alarm generated by agent.

### Request/Response Messages



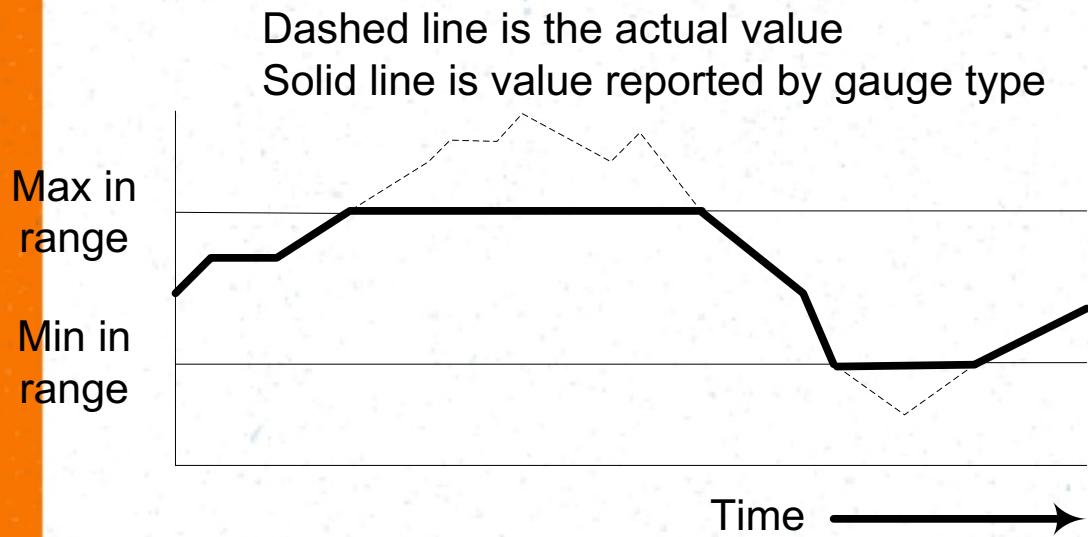
### Trap Messages



# SMI (RFC1155) Defined Data Types

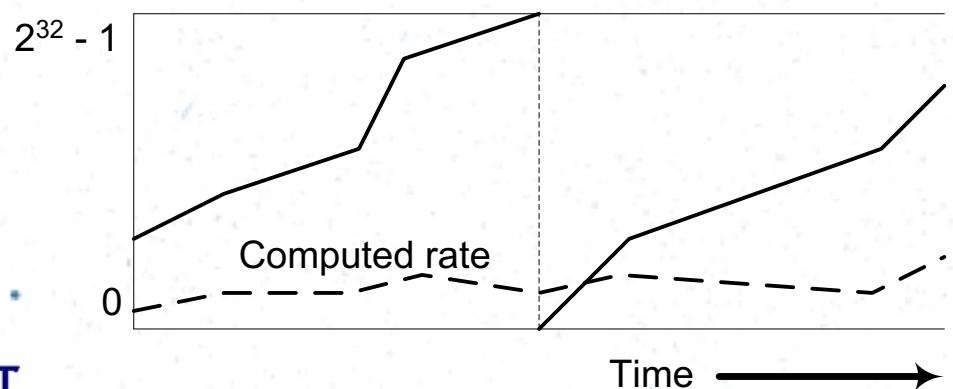
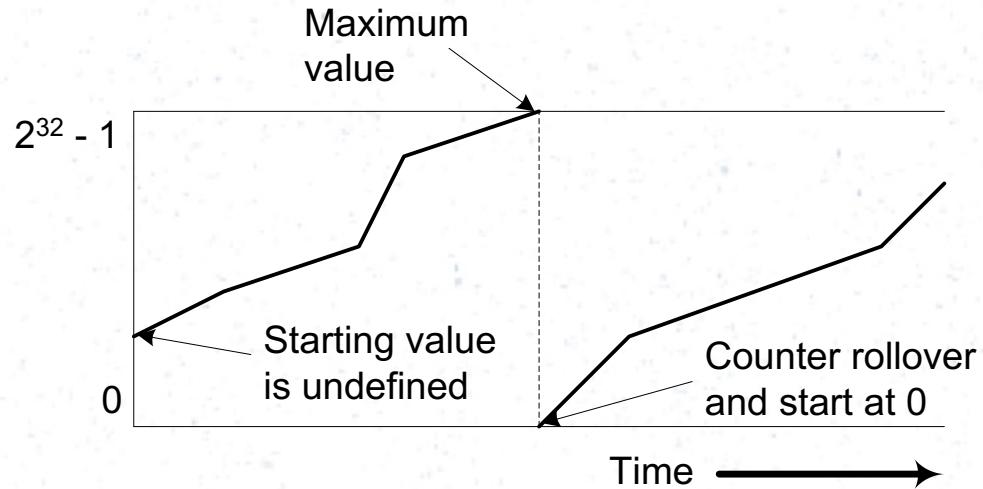
- Integer – Signed 32-bit integer.
  - Enumerated Integer.
- Octet String – String of bytes.
- Object Identifier.
- [NetworkAddress](#) – An address from one of possibly several protocol families.
- [IpAddress](#) – 32 bit IP address.
- [Gauge](#) – Non-negative integer from 0 to  $2^{32} - 1$ , which may increase or decrease.
- [Counter](#) – Non-negative monotonically increasing integer from 0 to  $2^{32} - 1$ .
- [Timeticks](#) – Non-negative integer which counts time in hundredths of a second.
- [Opaque](#) – Arbitrary syntax.

# Gauge



- Used to specify a value whose range includes only **non-negative 32 bit integers**.
- RFC 1155 – *this application-wide type represents a non-negative integer, which **may increase or decrease**, but which **latches at a maximum value**.*

# Counter



- Use to specify a non-negative value whose range includes only **positive 32-bit integers**.
- Values reported by counters are **not absolute**, since the **count is not required to start at 0 and the count may roll over**.
- Counters are used by obtaining a value  $v_0$  at  $t_0$  and then later obtaining a value  $v_1$  at  $t_1$ .

□ **Difference between  $v_0$  and  $v_1$  is the count over the time period.**

□ **Counter rollover can be detected iff  $v_0 > v_1$ .**

# Other Types

- **Timeticks** — used to specify a non-negative value whose range includes only non-negative integers.
  - Units are in hundredths of seconds.
  - Length of time between rollovers is 497 days.
- **Network Address** – used to specify a string of 4 octets.
  - Currently used to store IPv4 addresses.
  - Was designed to allow a network address of any type to be specified.
  - **Obsolete – use IpAddress.**
- **Opaque** – used to specify octets of binary information.
  - Generic type.

# ASN.1 Examples (MIB)

- RFC 1155
- internet OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
  - 1.3.6.1.
- Counter ::= [APPLICATION 1] IMPLICIT INTEGER (0..4294967295)
- TimeTicks ::= [APPLICATION 3] IMPLICIT INTEGER (0..4294967295)
- IpAddress ::= [APPLICATION 0] IMPLICIT OCTET STRING (SIZE (4))
- NetworkAddress ::= CHOICE { internet IpAddress }



# IT3010

## Network Design and Management

Lecture 07  
LDAP

**Shashika Lokuliyan**

Faculty of Computing  
Department of CSE



**SLIIT**

*Discover Your Future*

# Today's lecture overview

- What is a directory?
- Invention of LDAP.
- Understanding LDAP.
  - The LDAP Models.
    - The LDAP Information Model.
    - The LDAP Naming Model.
    - The LDAP Functional Model.
    - The LDAP Security Model.
  - LDAP Data Interchange Format (LDIF).
  - Schema Design.
  - Namespace Design
  - Topology Design
  - Replication Design

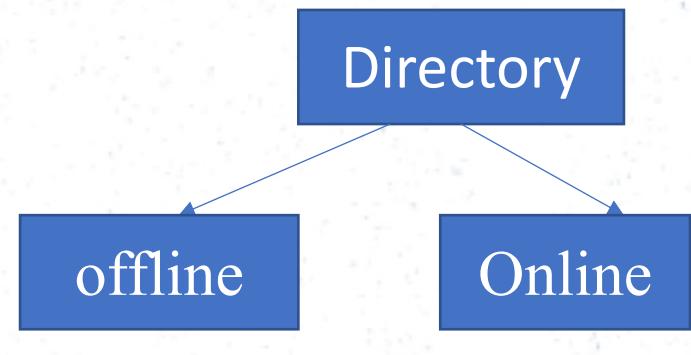
⋮  
⋮  
⋮

# Reference Book

- Understanding and Deploying LDAP Directory Service (2nd Edition) by Timothy A. Howes, Mark C. Smith and Gordon S. Good.

# What is a directory?

- In brief, a directory is a specialized database.
- Most people are familiar with various kinds of directories, whether they realize or not.
- Everyday examples of directories:
  - Phone books (white and yellow pages)
  - TV guide
  - Shopping catalogs
  - Library card catalogs etc.
- These everyday directories are known as offline directories.
  - They are static: Doesn't update/ change often. Maybe yearly.



# What is a directory? Contd.

- But in computing/ networking world we have online directories.
- Online directories are dynamic.
  - Offline directories are relatively static. They do not change often. By contrast online directories can be kept much more up-to-date.
- Online directories are flexible.
  - Offline directories are static in terms of their content. Cost of adding new information to printed (offline) directories is huge (cost of re-designing, re-printing and re-distributing). By contrast online directories can be easily extended with new types of information.
- Online directories can be made secure.
  - Offline directories offer little, if any, security. By contrast online directories centralize information allowing access to that information to be controlled.

# What is a directory? Contd.

- Online directories can be personalized.
  - TV guide and phone book are personalized in a regional basis.
  - Everyone access the same card catalog at the library.
  - But online directories can easily provide more personalized views of the directory for each user.
  - By identifying the users who access the directory and storing profile information about them.

# Directories

- Size of the information.
  - Good for storing small pieces of information, not multi MB files.
  - Good at storing pointers to large things, than the large things themselves.
- Character of the information.
  - Good at storing attribute based information where you can break your information into name-value pairs.
- Read-to-write ratio.
  - Best for information that are read far more than it is written.
  - If more writes, think about using a DB instead.
- Search capability.
  - Directories are designed to make searching fast.
  - If your application requires fast searching, directory maybe a good choice.
- Standards-based access.
  - If you need standards-based access to your information (for example, access by a different vendors software which is using the same standard access protocol).

# What is LDAP?

- Stands for Lightweight Directory Access Protocol.
- A standard, extensible Internet Protocol used to access directory services.
- Lightweight alternative to DAP.
- Uses TCP/IP instead of OSI stack



# The origins and history of LDAP

- X.500
  - World's first standard directory service and the specification.
  - Jointly developed by International Telecommunication Union (ITU-T) and International Organization for Standardization (ISO).
  - First published in early 1990s.
  - Designers expectation was: One day, eventually there would be one fully interconnected global directory service, the Directory.

# X.500 Contd.

- Advantages:
  - Extensibility
  - Rich set of search operations
  - A highly distributed system
  - Open standard: Not controlled by any vendor or tied to any OS
- Disadvantages:
  - Highly complex specification and difficult to implement (as a result, many early versions of X.500 software were buggy)
  - Rely on a large suite of protocols (DAP, DSP, DOP and DISP)
  - Based on OSI network model.

# X.500 Contd.

- Since the designers of the X.500 believed that eventually the OSI model will replace the TCP/IP model which never happened, unfortunately.
- One of the best known implementations of X.500 specification was *Quipu*, developed by UCL which was free and open source.

# The origins and history of LDAP Contd.

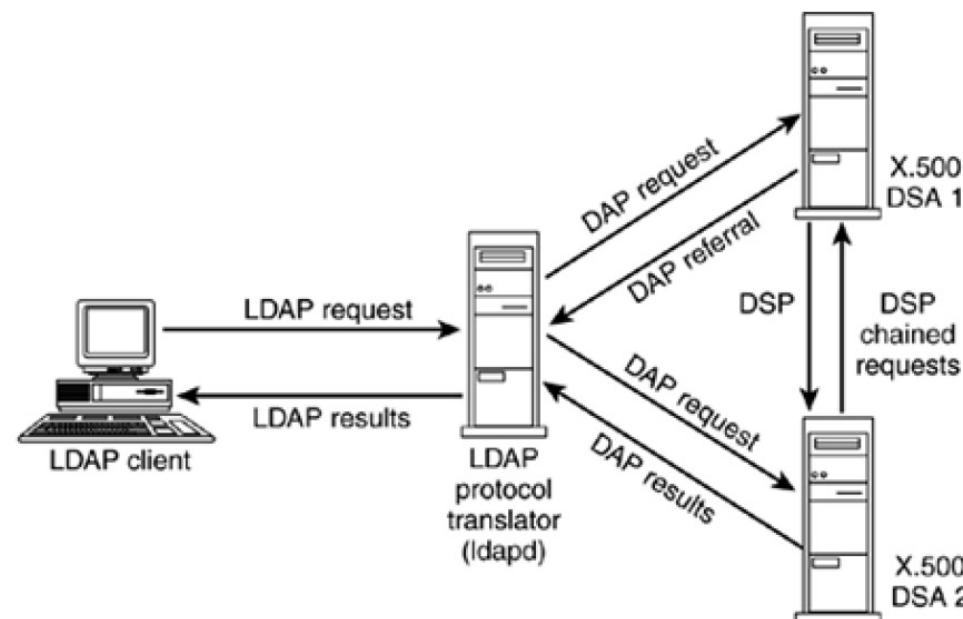
- DAP (Directory Access Protocol): X.500's directory client access protocol (i.e. the protocol that client's used to communicate with the X.500 server)
- Had some issues
  - Large.
  - Complex.
  - Difficult to implement.
  - Most implementations performed poorly.
- Two alternatives were proposed,
  - DAS (Directory Assistance Service) – RFC 1202.
  - DIXIE (Directory Interface to X.500 Implemented Efficiently) – RFC 1249.
- Both proved to be better than DAP.
- But, both had one shortcoming: Both were closely tied with *Quipu*.

# The birth of LDAP

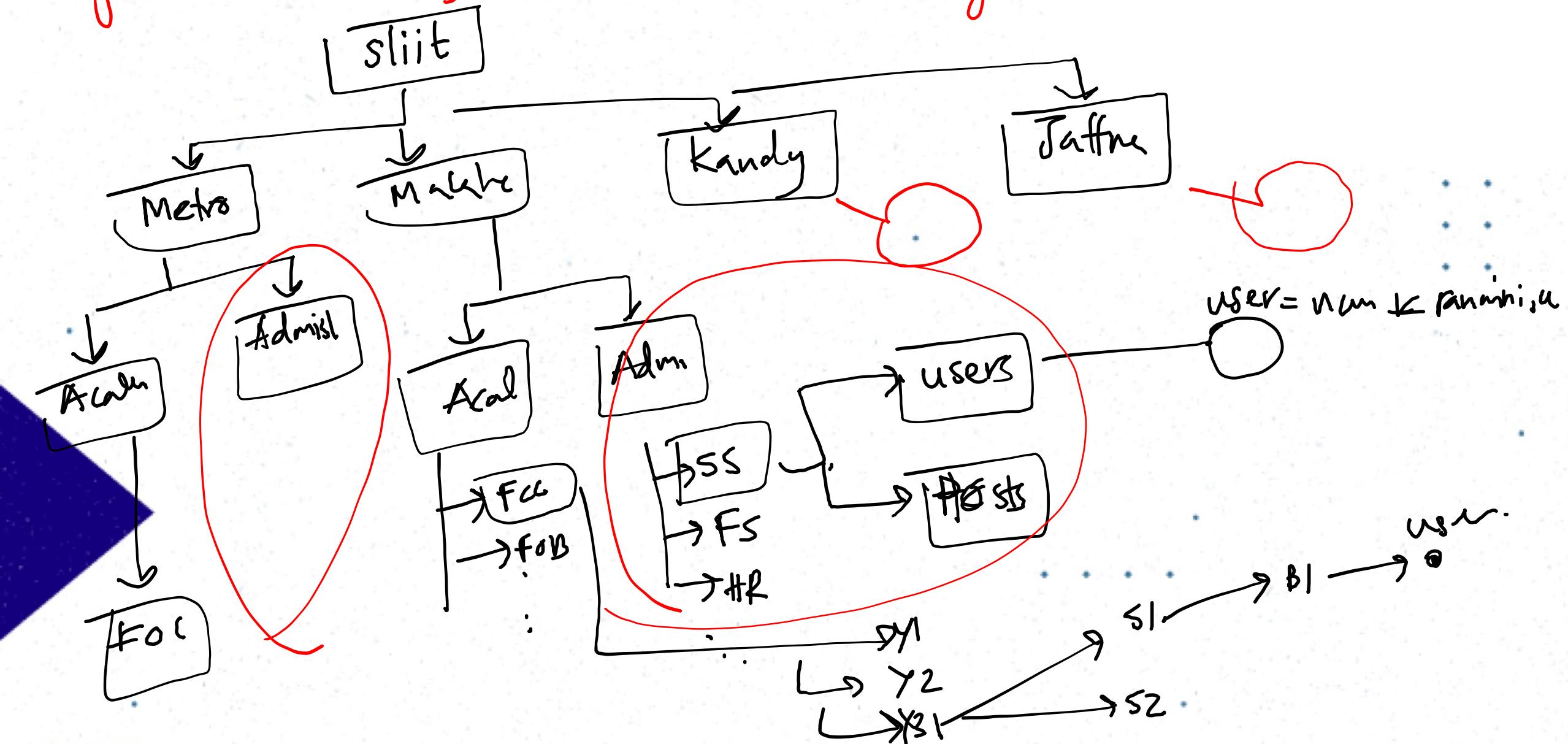
- *With the initiation taken by DAS and DIXIE towards a lightweight access protocol to X.500 directory service, IETF decided to produce a standard, full-featured, lightweight directory access protocol.*
- *As a result Lightweight Directory Access Protocol (LDAP) was born.*
- One key advantage: LDAP was fully based on TCP/IP stack rather than the OSI stack.
- During the initial stages LDAP was exclusively used to provide a front end to X.500 directory services.
- Initial implementation of LDAP was done at University of Michigan and it came to be known as the U-M LDAP implementation.

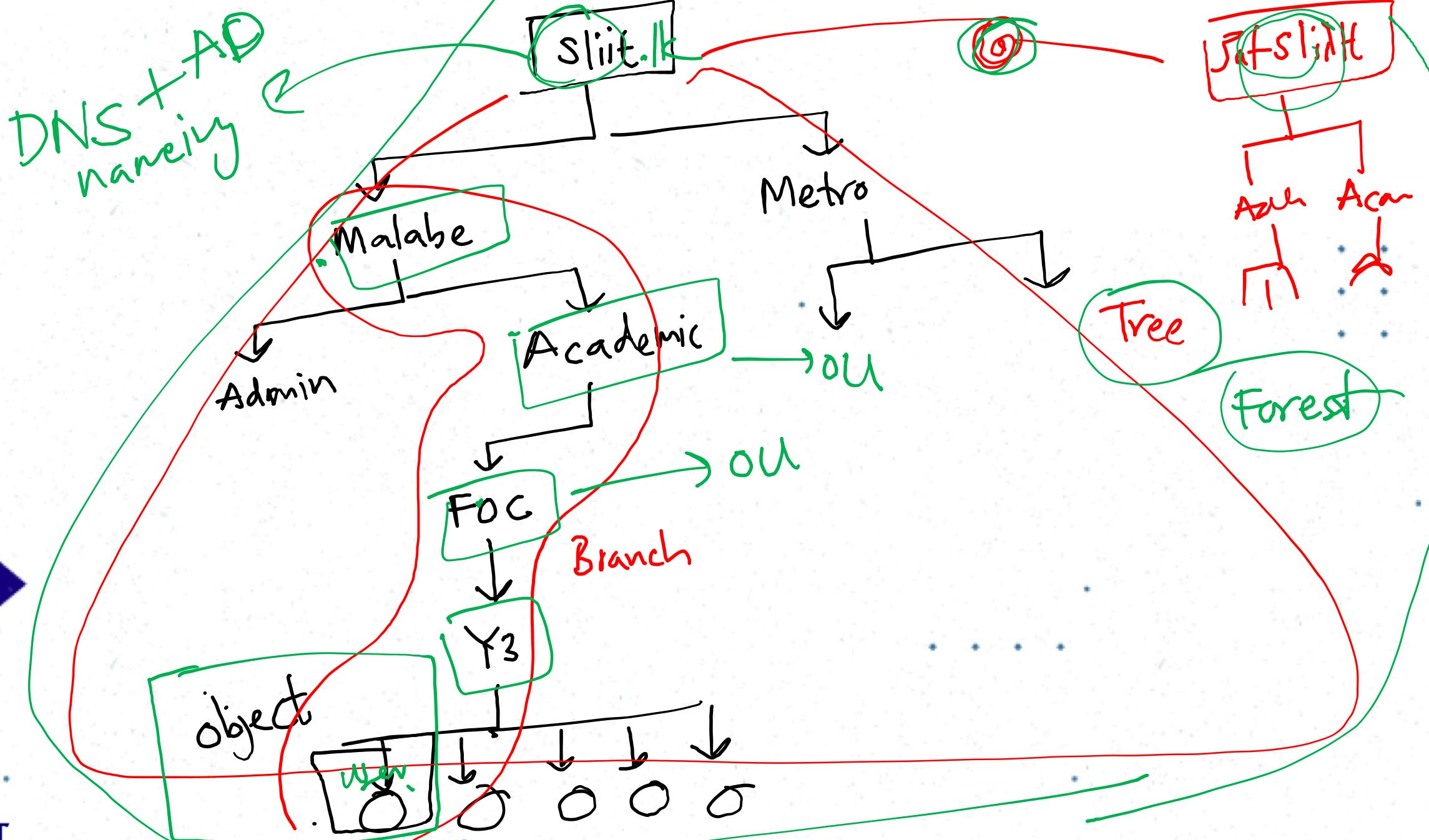
# U-M LDAP

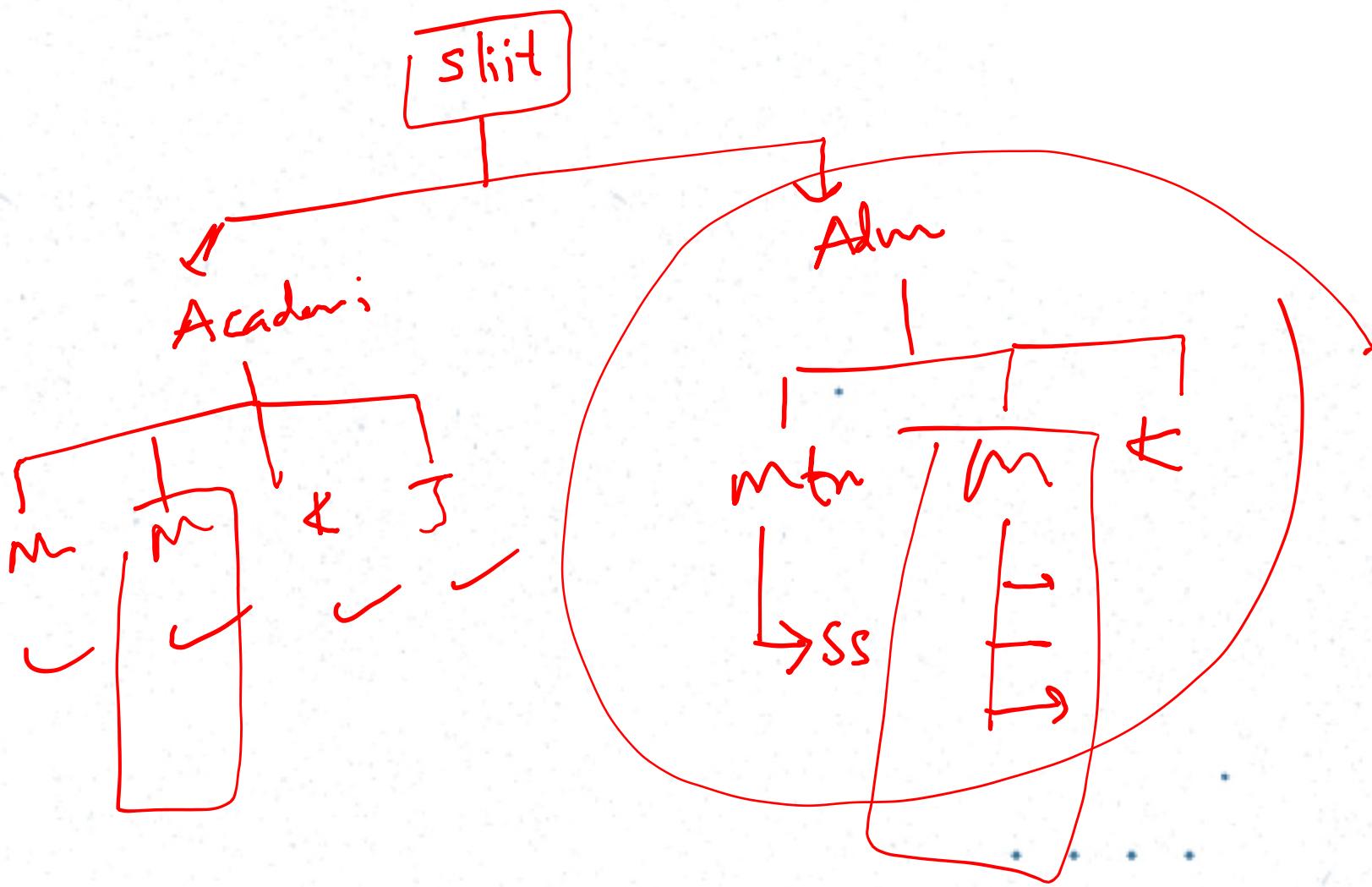
- The key server side component is *ldapd*, which is a LDAP-to-X.500 DAP protocol translator.
- Although LDAP solved part of the problem, still, the server implementation of the X.500 directory was large, complex and difficult to deploy.

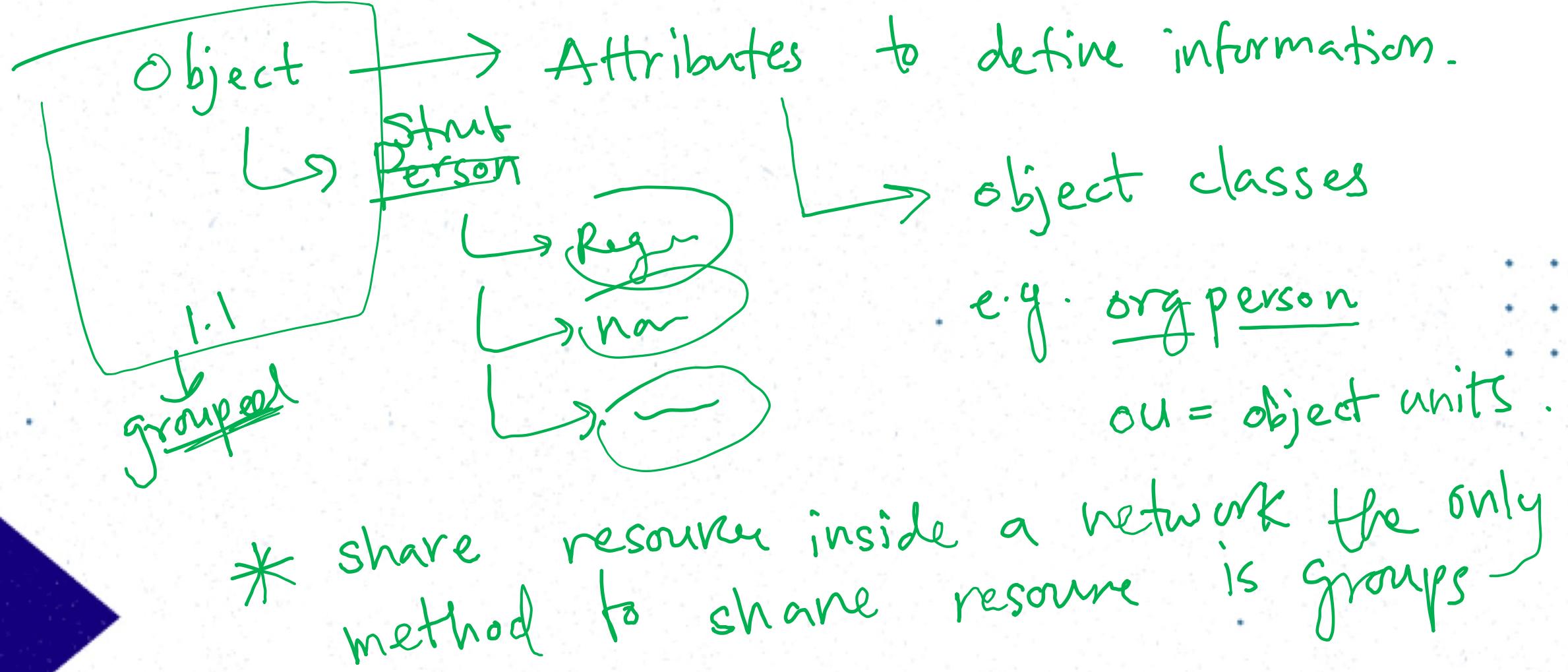


organization directory structure → organization struct.







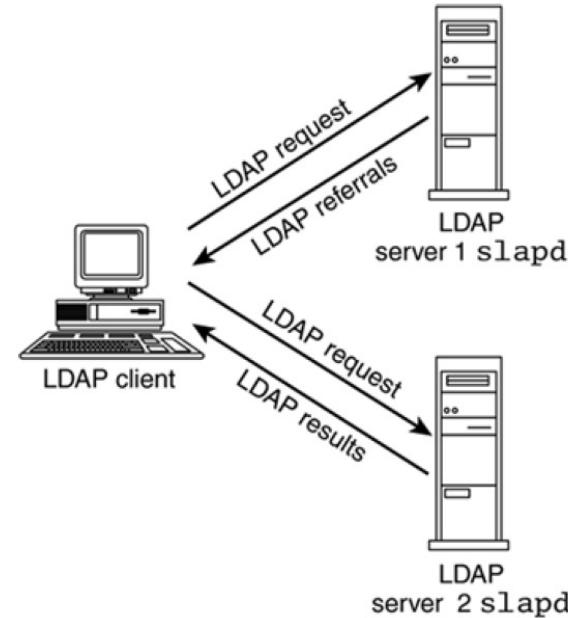






# Standalone LDAP (slapd)

- As a solution designers thought of keeping the best features of X.500 and developing LDAP as a standalone directory service specification.
- *As a result today's standalone LDAP service came to the role.*



# General purpose LDAP directory service implementations

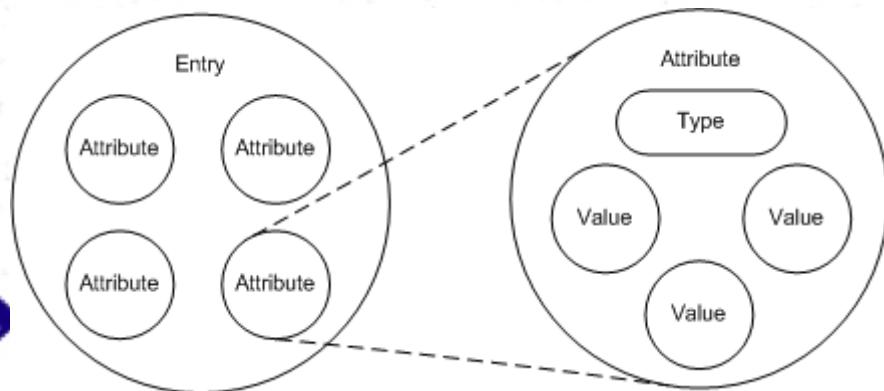
- Microsoft Active Directory
- IBM Directory Server
- Oracle Internet Directory
- OpenLDAP slapd server



# LDAP MODELS

- • Information
  - Structure of information stored in an LDAP directory.
- • Naming
  - How information is organized and identified.
- • Functional operations
  - Describes what operations can be performed on the information stored in an LDAP directory.
- • Security
  - Describes how the information can be protected from unauthorized access.

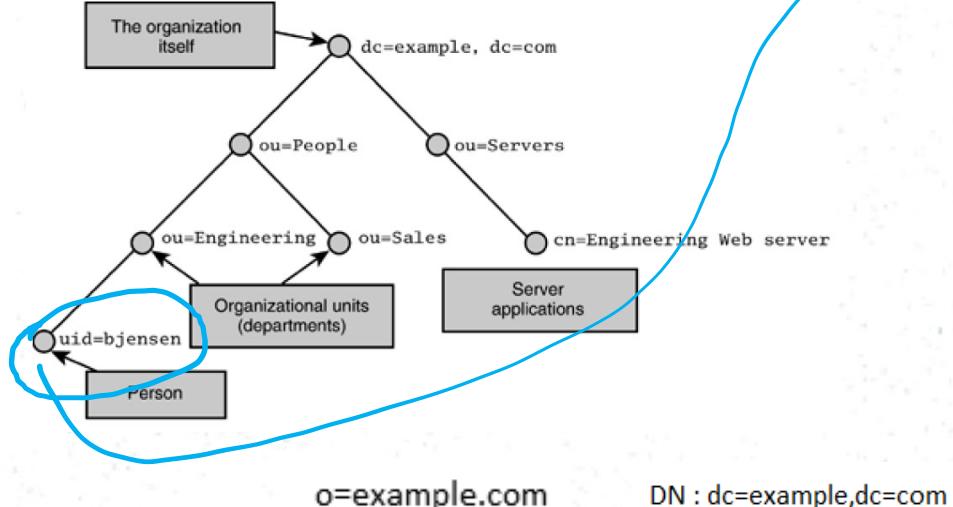
# LDAP Information Model



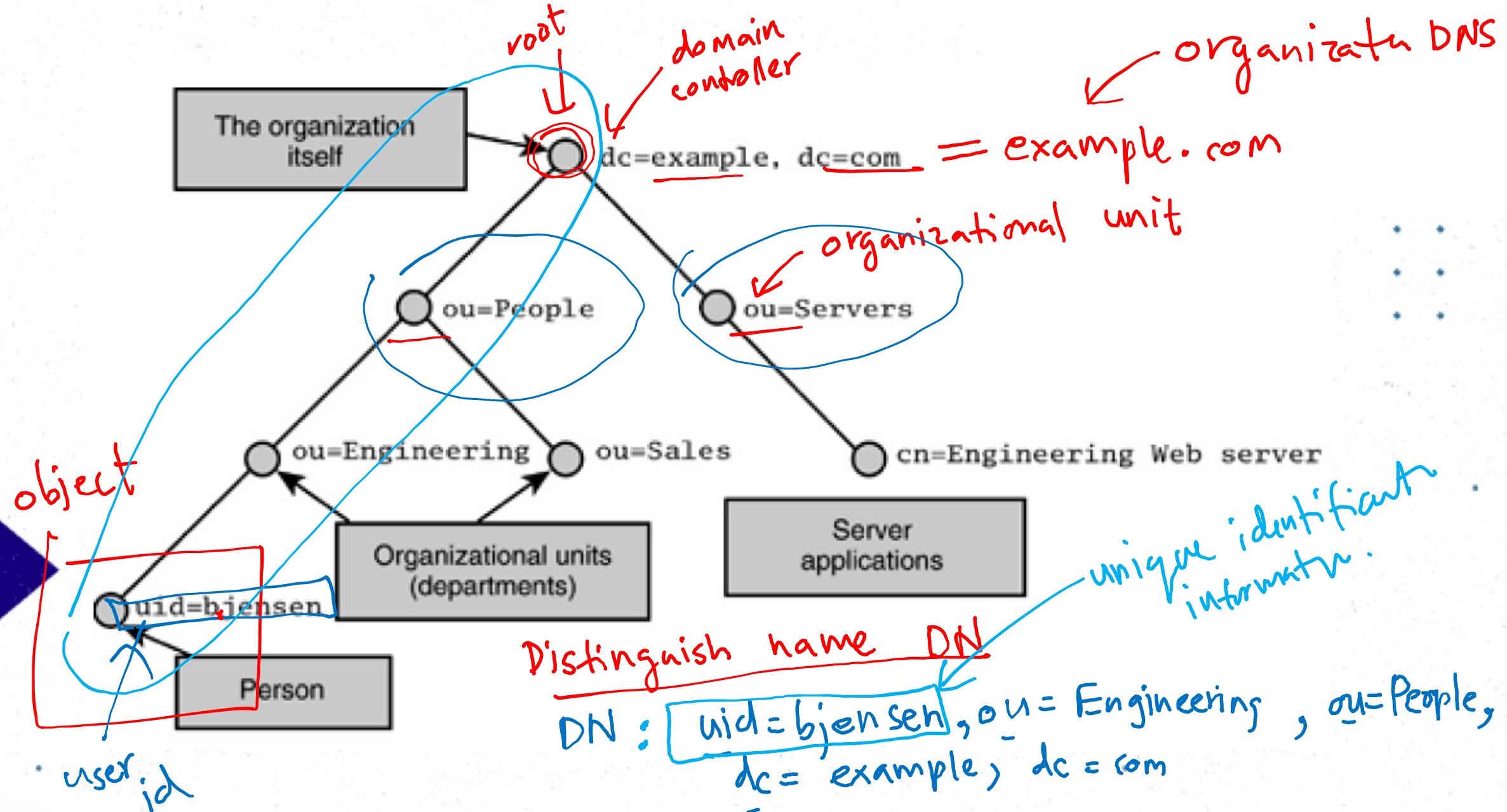
- The LDAP information model defines the types of data and basic units of information that you can store in your directory.
- The basic unit of information in the directory is the entry.
  - A collection of information about an object (for example, real world objects like people, departments, servers, printers etc.).
  - An entry is composed of a set of attributes.
  - Each of which describes one particular characteristic of the object.
  - Each attribute has a type and one or more values.

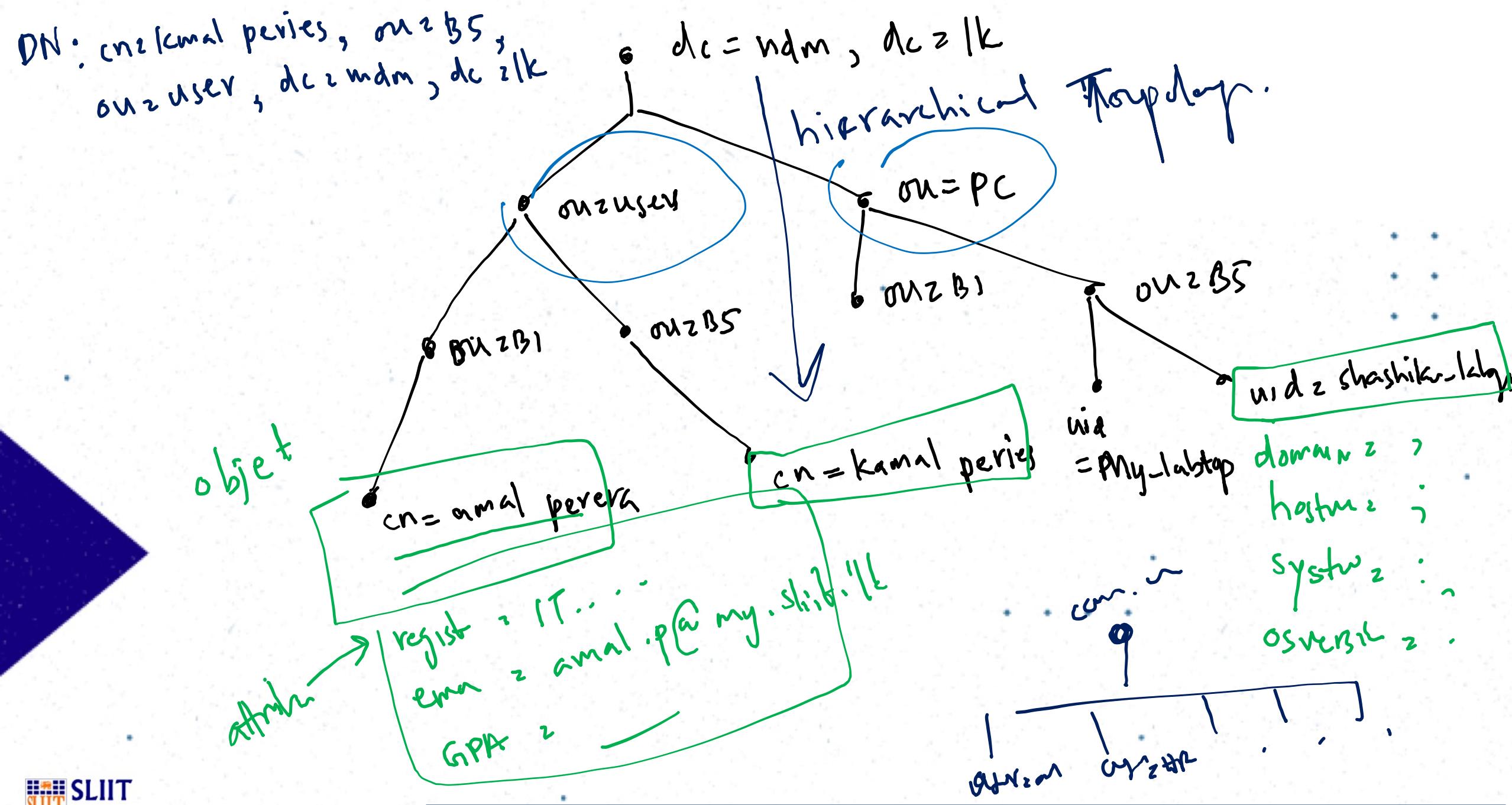
# LDAP Information Model contd.

- Each directory has a Distinguished Name (DN).
- Ex: Organization shown in the following figure has the DN dc=example, dc=com



Attribute type	Attribute values
cn:	Barbara Jensen Babs Jensen
sn:	Jensen
telephoneNumber:	+1 408 555 1212
mail:	babs@example.com





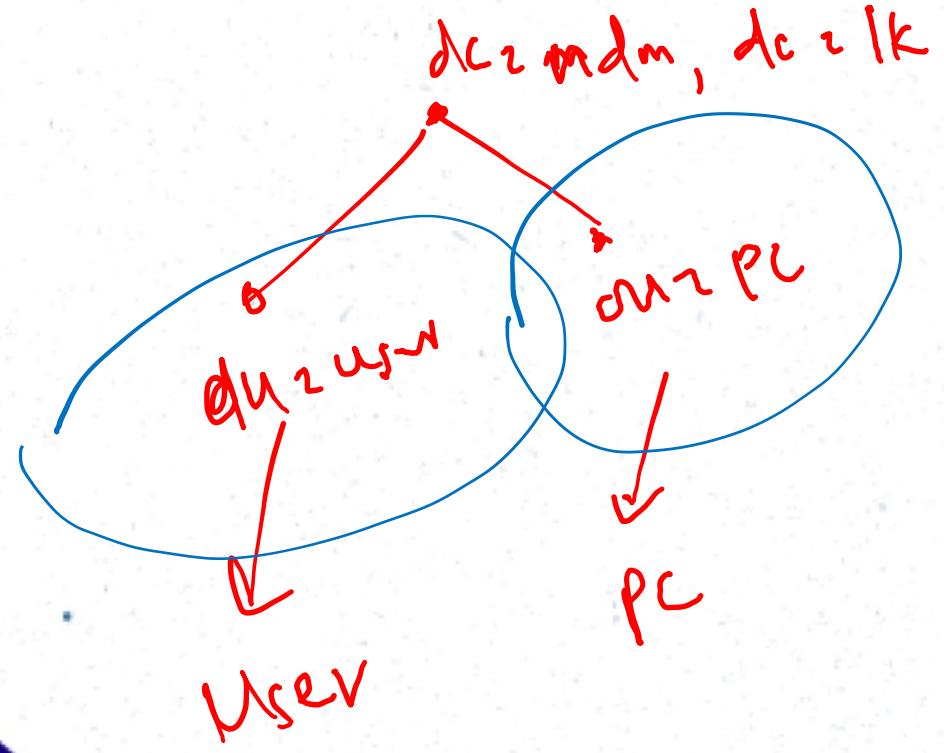
e.g. - slit student  ~~naming even schema for AD / LDAP~~

registration no @ my . slit . lk

staff slit →

firstname , first letter  
of last  
name @ slit . lk

shashika . l @ —



## LDIF file

\* domain

dn : dc = ndm, dc = lk

dc = ndm

dc = lk

\* user object

dn : ou = user, dc = ndm, dc = lk.

o : organizationalUnit

\* PC obje

dn :

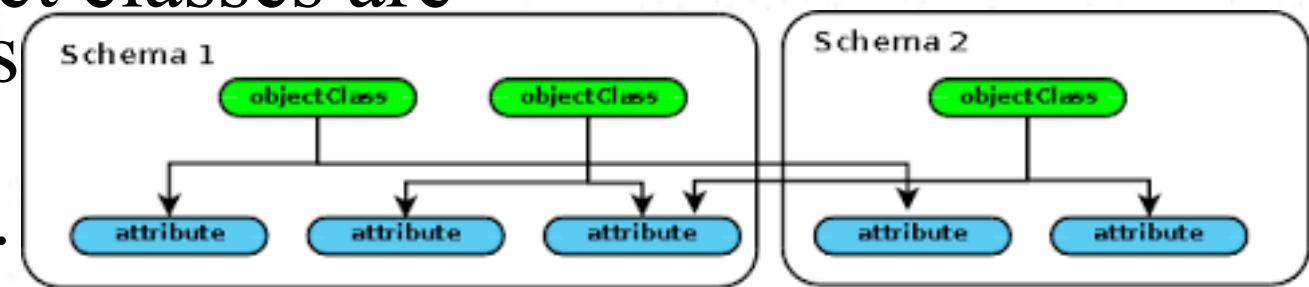
o ?

# LDAP Information Model contd.

- Attributes can be classified as user attributes and operational attributes
  - *User attributes*, the "normal" attributes of an entry, may be modified by the users of the directory (with appropriate permissions).
  - *Operational attributes* are special attributes that either modify the operation of the directory server or reflect the operational status of the directory.
- An example of an operational attribute is the `modifyTimeStamp` attribute, which reflects the time that the entry was last modified and is automatically maintained by the directory.
- When an entry is sent to a client, operational attributes are not included unless the client requests them by name.
- The collections of all information relating to attributes are called the *directory schemas*.

# LDAP Information Model contd.

- LDAP Schemas
  - When you create an entry in a DIT its data contents are contained in attributes, which are grouped into objectclasses, which are packaged into schemas.
  - Schemas define what object classes are allowed.(An objectClass is
    - Where they are stored.
    - What attributes they have.
    - What attributes are optional.
    - Type/ syntax of each attribute.
  - LDAP schema must be readable by the client.



# LDAP Naming Model

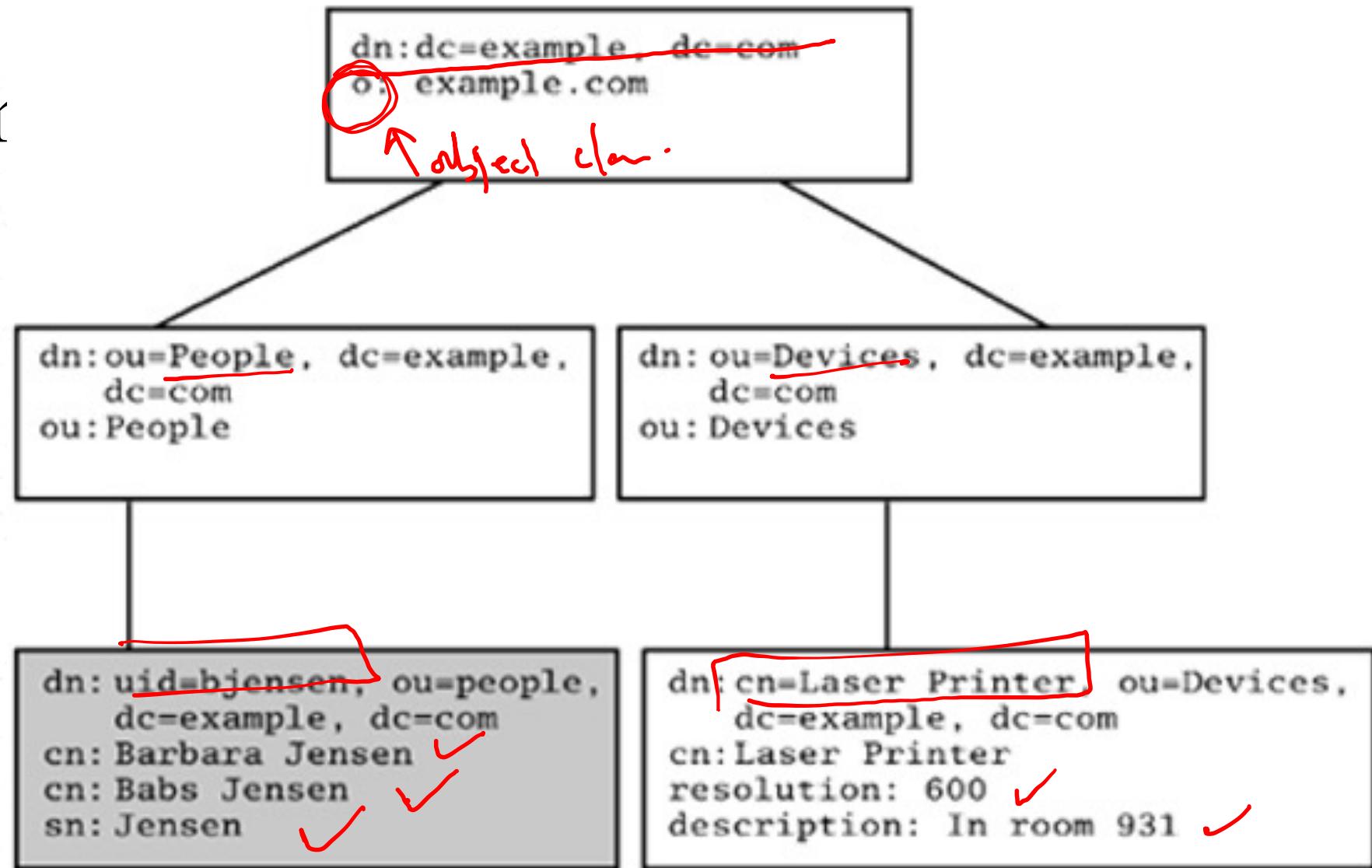
- The LDAP naming model defines how you organize and refer to your data.
- Directory Information Tree (DIT).
  - Usually follows the geographical or organizational scheme.
  - The name of an LDAP entry is formed by connecting, in a series, all the individual names of the parent entries back to the root.
- Distinguished Names (DN).
  - DN is similar to an absolute path of a file within a file system. DN is the unique identifier of an entry within the DIT. This allows any entry of the DIT to be identified unambiguously. DN of an entry should be unique within the entire DIT.
- Relative Distinguished Names (RDN).
  - RDN is similar to the file name of a file itself within a file system. It is the leftmost component of the DN of an entry. RDNs should be unique only if they share a common immediate parent. This is similar to, you not been able to create two files with the same name within the same folder.

# LDAP Naming Example

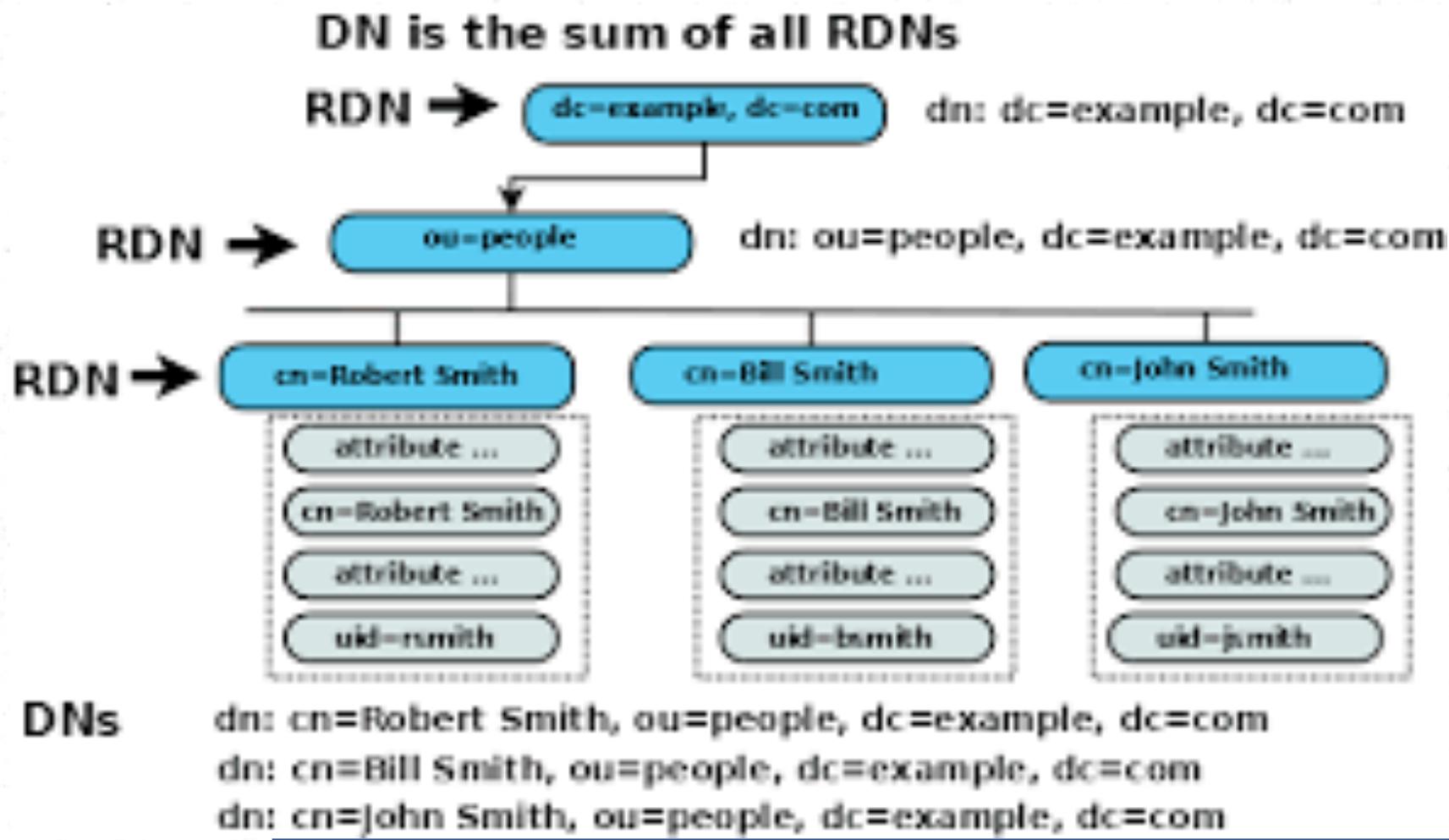
## LDAP Naming Examples

Attribute Type	String
CommonName	CN
LocalityName	L
StateorProvinceName	ST
OrganizationName	O
OrganizationalUnitName	OU
CountryName	C
StreetAddress	STREET
domainComponent	DC
Userid	UID

# LDAP Nano

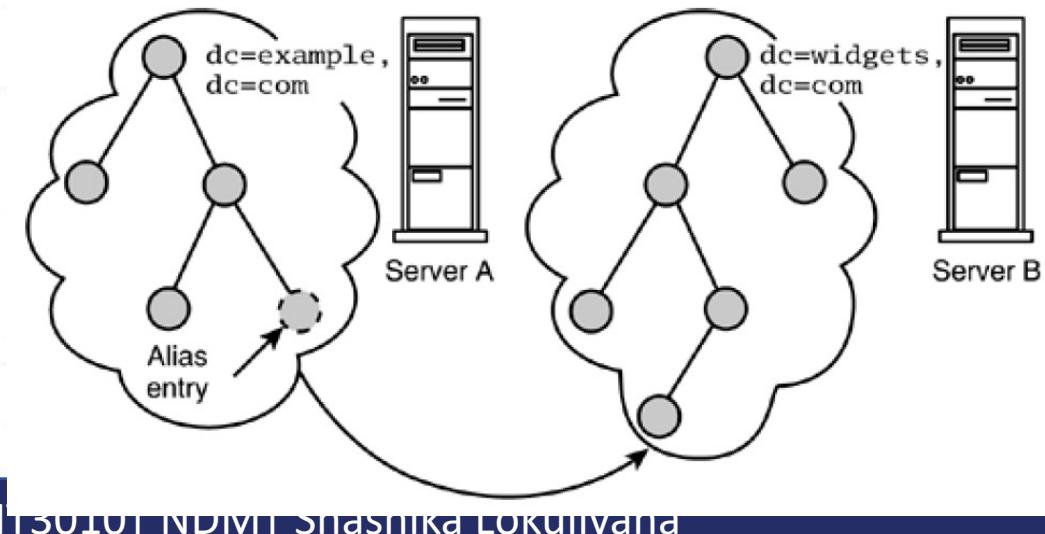


# Example



# LDAP Naming Model

- Alias entries in the LDAP directory allow one entry to point to another one:
- Similar to symbolic links in UNIX or shortcuts in Windows.
- objectClass=alias
- Not all LDAP implementations support aliases.
- Aliases potentially slow down search operations.



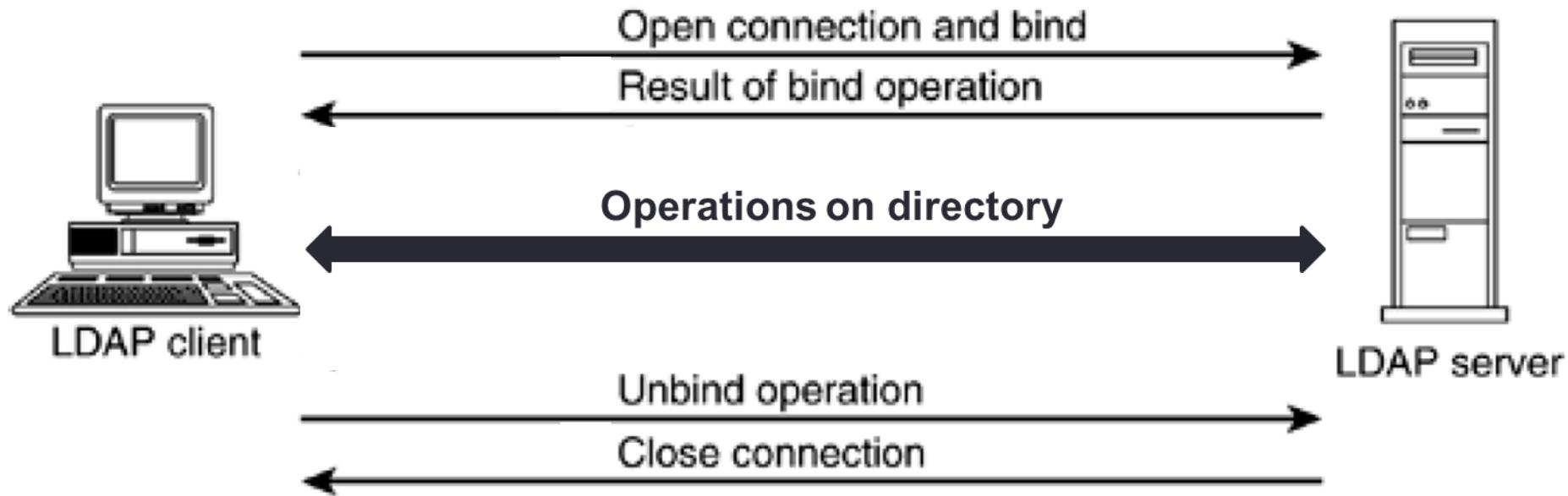
# LDAP Functional Model

- LDAP functional model consists of a set of operations divided into three groups.
- Authentication and Control operations.
  - Allows the clients to identify themselves to the directory and control some aspects of a session.
  - BIND
  - UNBIND
  - ABANDON
- Interrogation operations.
  - Allows to search the directory and retrieve directory data.
  - Search
  - Compare
- Update operations.
  - Allows to add, delete, rename and change directory entries.
  - Add an entry
  - Delete an entry
  - Modify an entry
  - Rename an entry (i.e. Modify DN/ RDN)

# Client/ Server Interaction

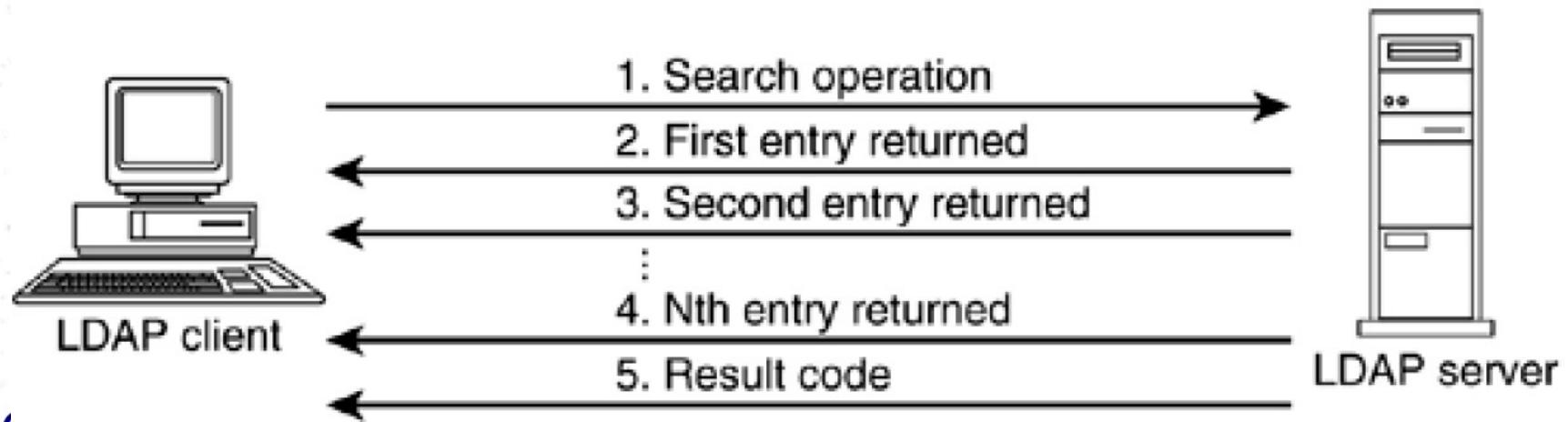
- Client establishes session with server (BIND).
- Client performs operations.
- Client ends the session (UNBIND).

# Client/ Server Interaction contd.



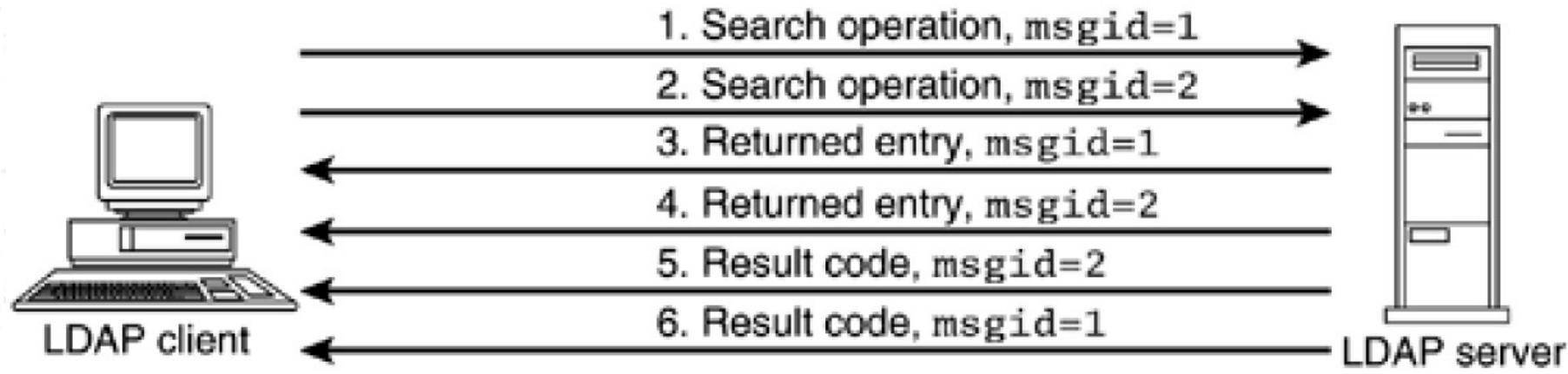
# Client/ Server Interaction contd.

- If the client searches the directory and find multiple entries, those entries are sent to the client in a series of LDAP messages, one for each entry.
- Each entry has a unique name called a *Distinguished Name* (DN) which is carried as a text string in the LDAP message.
- Finally the *Result code* is returned: Contains the overall result of the search operation.



# Client/ Server Interaction contd.

- LDAP protocol is messages based. Therefore it allows the client to issue multiple requests at once.
  - Client generates a unique message ID for each request.
- Returned results for a specific request are tagged with the request's message ID.
- Allows the client to sort multiple replies to different requests arriving out of order at the same time.

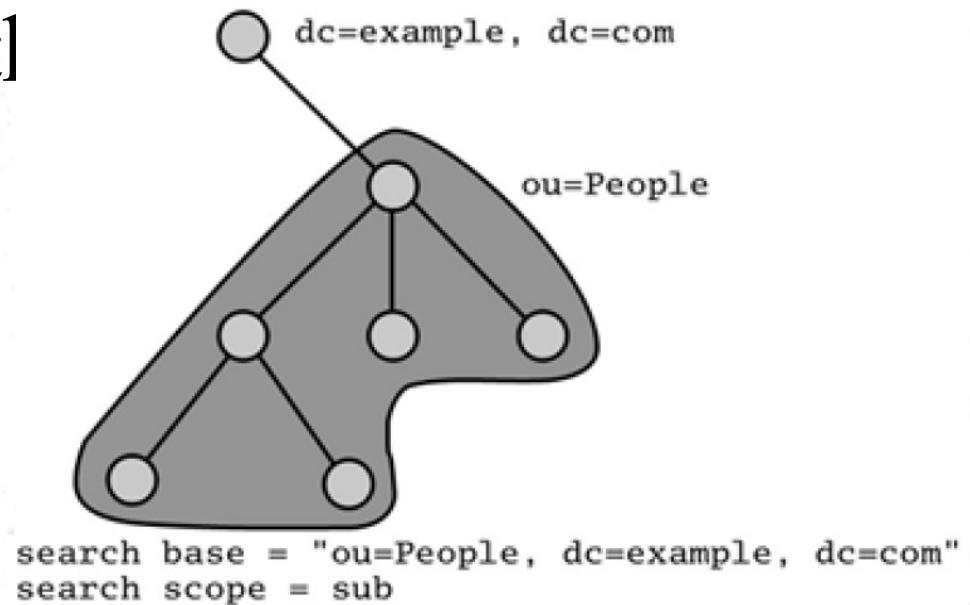


# Search Operation

- LDAP search operation provides many parameters that you can use to customize the search operation based on your need. (Page 70 of the reference book).
- There is no specific read operation defined in LDAP. Therefore, you need to customize the search operation using these parameters to act as a read operation.
- One such important parameter in search operation is the *search scope*.
- There are three *search scope* options.

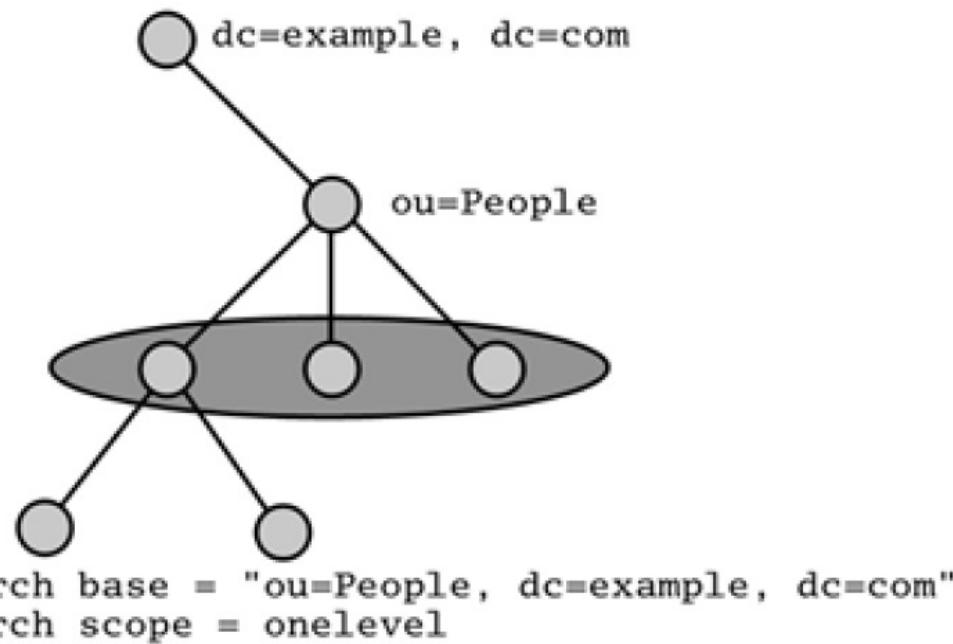
# Search operation contd.

- sub
- Search the entire sub tree form the base object all the way down to t



# Search operation contd.

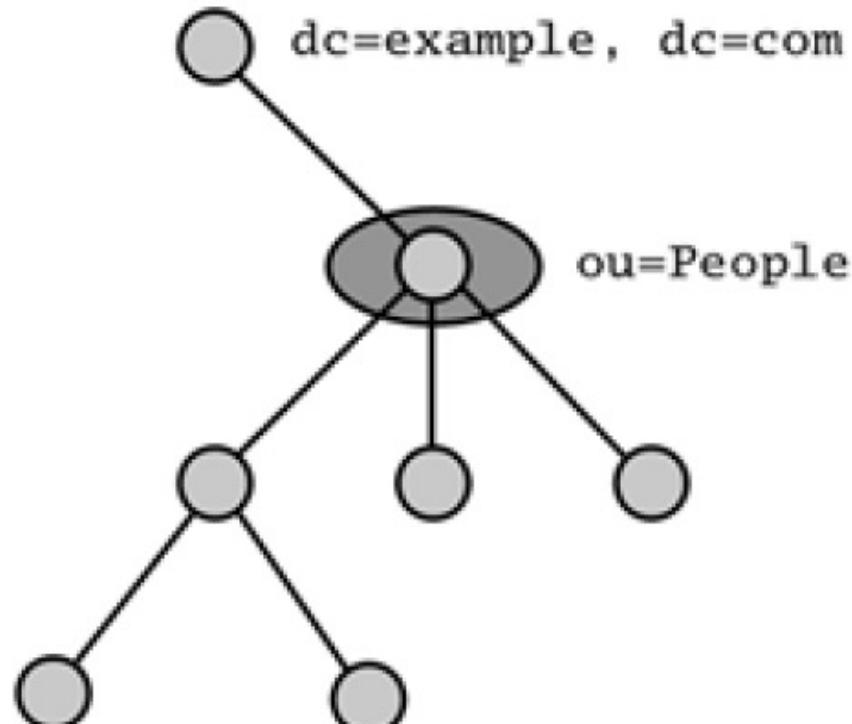
- onelevel
- Search only the immediate children of the entry at the top of the search base



# Search operation contd.

- base
- Limit the search to just the base object

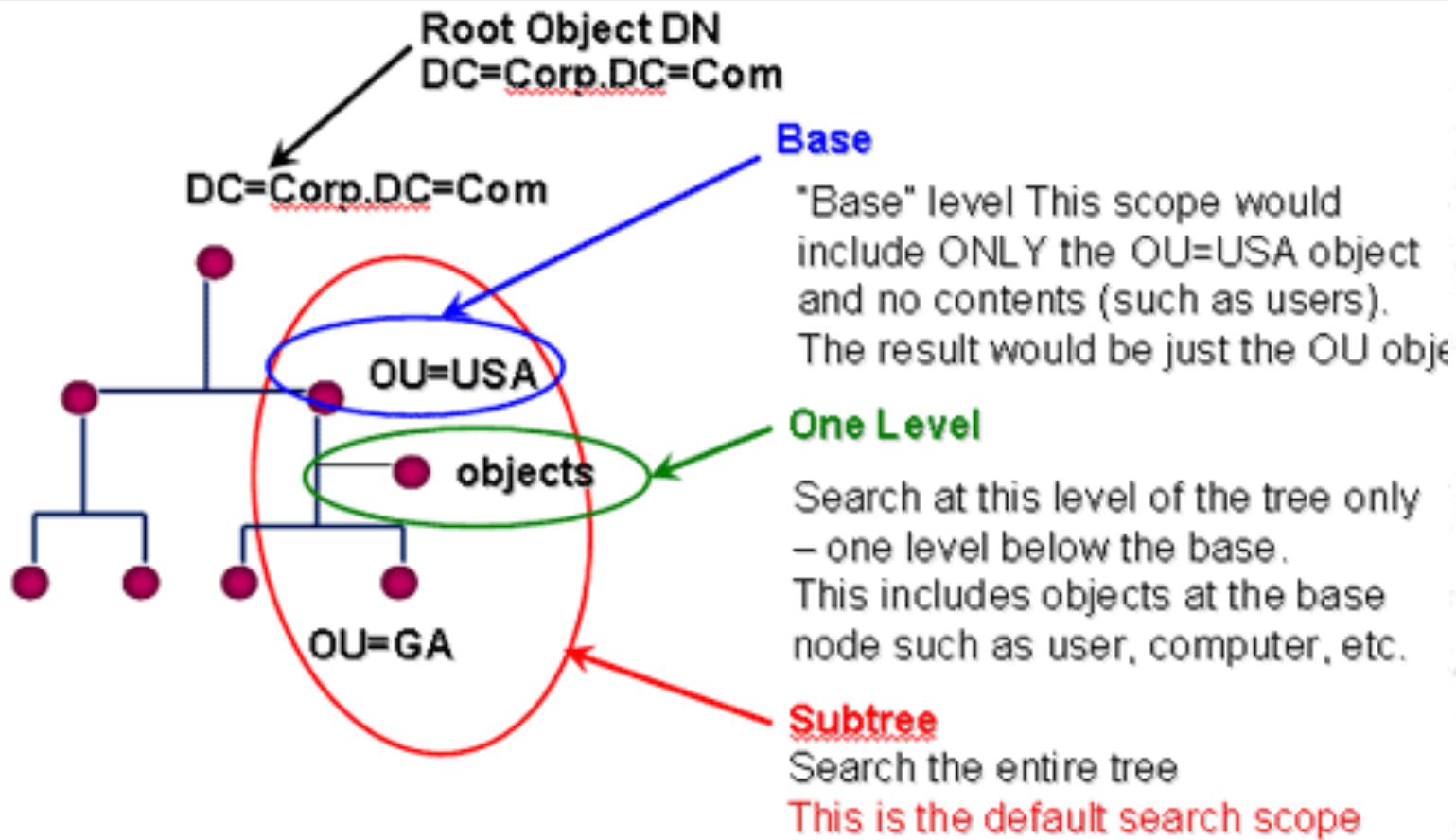
⋮  
⋮



```
search base = "ou=People, dc=example, dc=com"  
search scope = base
```

# Functional Model Interrogation

## Search Scope



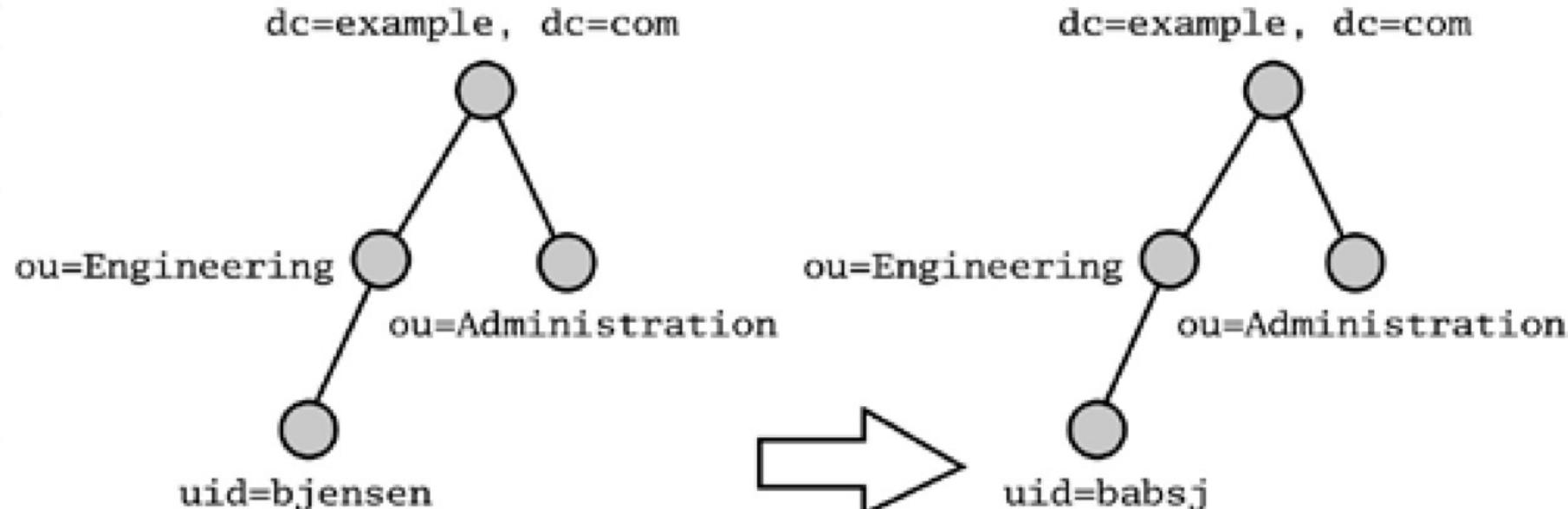
# Update Operations

- There are four LDAP update operations
- Add
- Delete
- Rename (Modify DN)
- Modify

⋮  
⋮  
⋮  
⋮

# Examples: What you can do with LDAP update operations

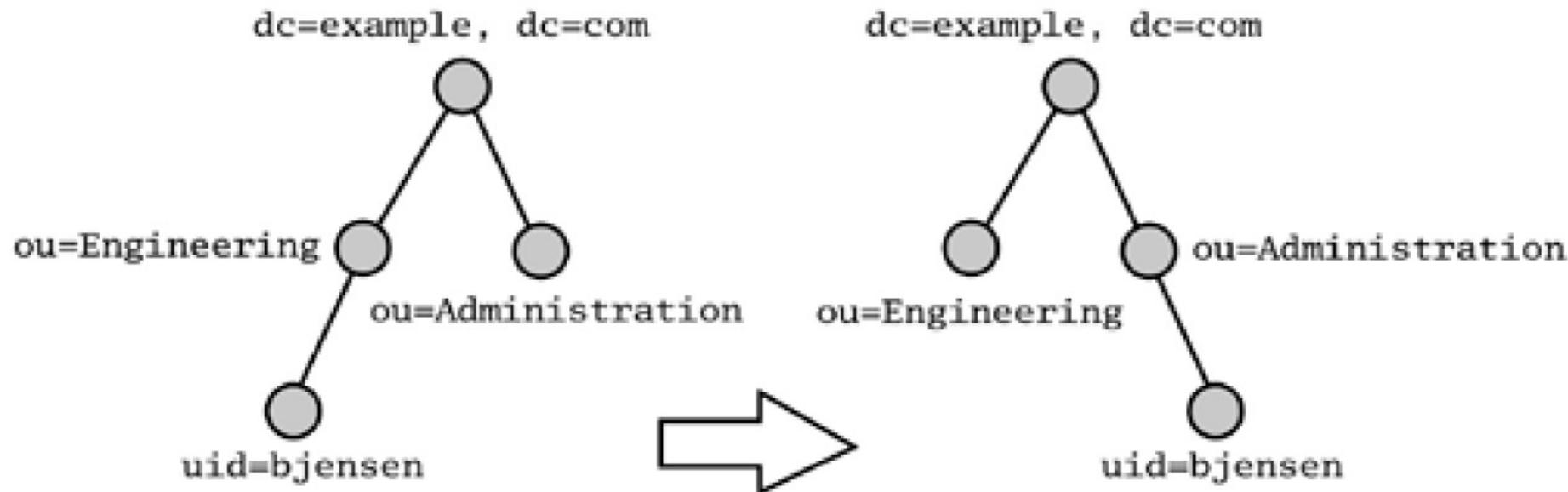
- Renaming an entry without moving it.



Original DN: `uid=bjensen, ou=Engineering, dc=example, dc=com`  
New DN: `uid=babsj, ou=Engineering, dc=example, dc=com`

# Examples: What you can do with LDAP update operations contd.

- Moving an entry without changing its RDN.



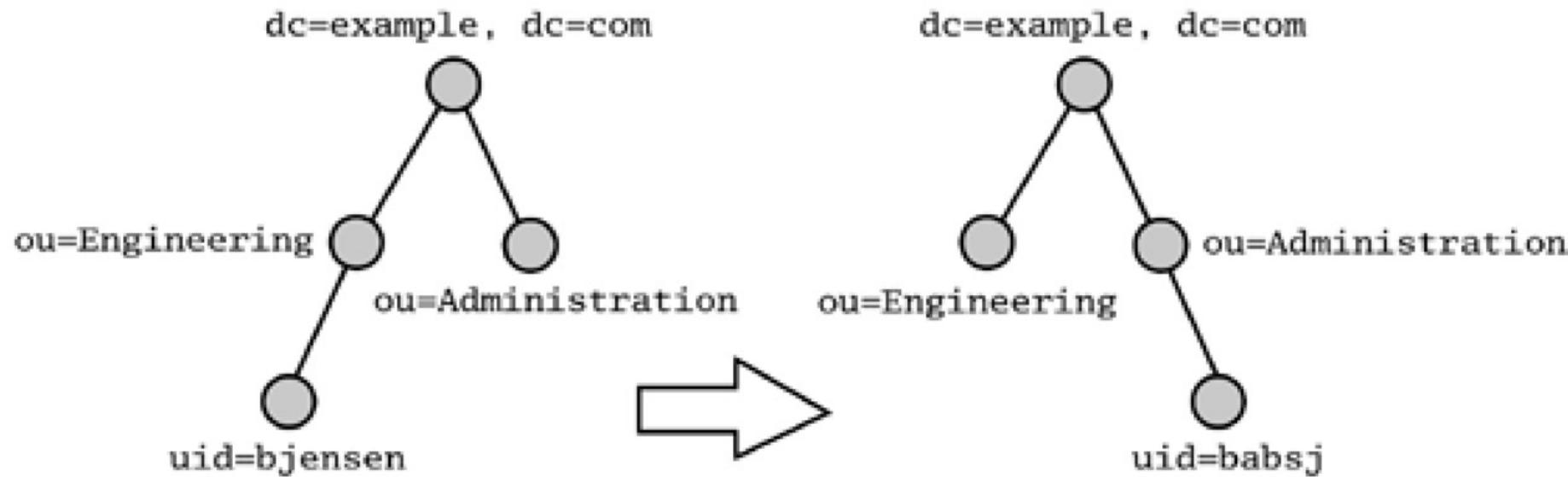
Original DN: uid=bjensen, ou=Engineering, dc=example, dc=com

New DN: uid=bjensen, ou=Administration, dc=example, dc=com

# Examples: What you can do with LDAP update operations contd.

- Moving an entry and changing its RDN

similarly

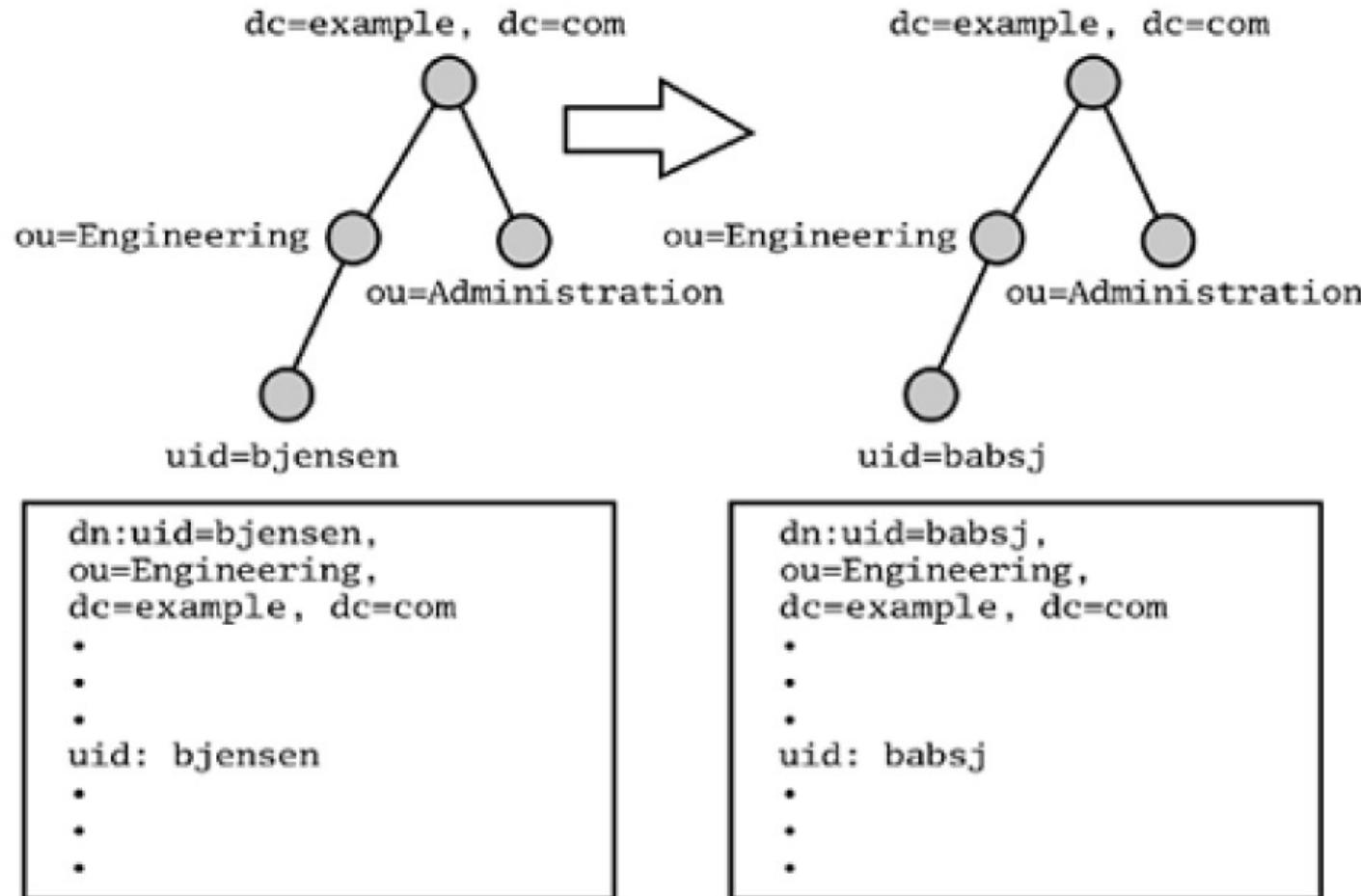


Original DN: uid=bjensen, ou=Engineering, dc=example, dc=com  
New DN: uid=babsj, ou=Administration, dc=example, dc=com

⋮  
⋮  
⋮  
⋮

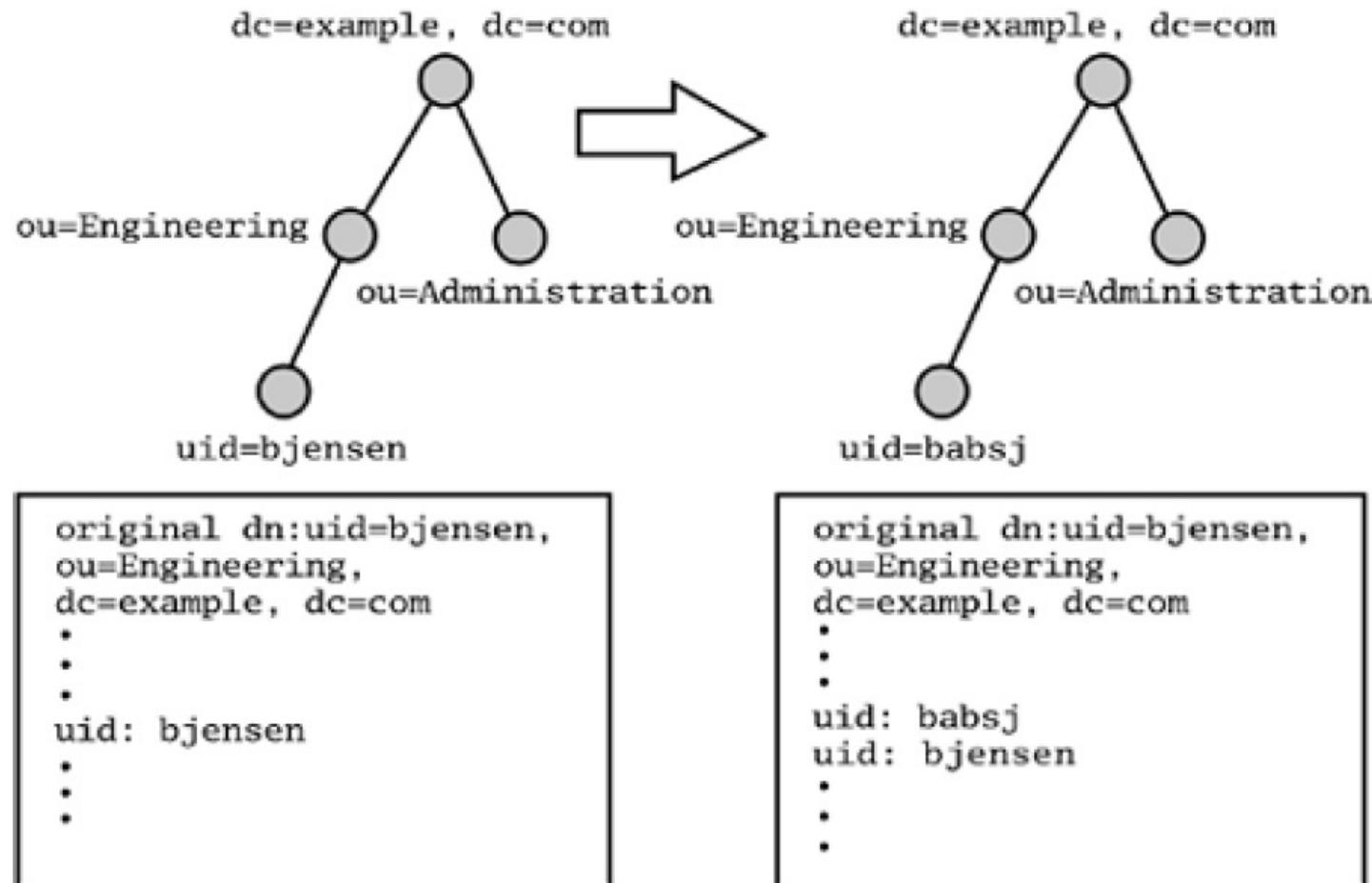
# Examples: What you can do with LDAP update operations contd.

- Renaming an entry, deleteoldrdn=true



# Examples: What you can do with LDAP update operations contd.

- Renaming an entry, deleteoldrdn=false



# LDAP Security Model

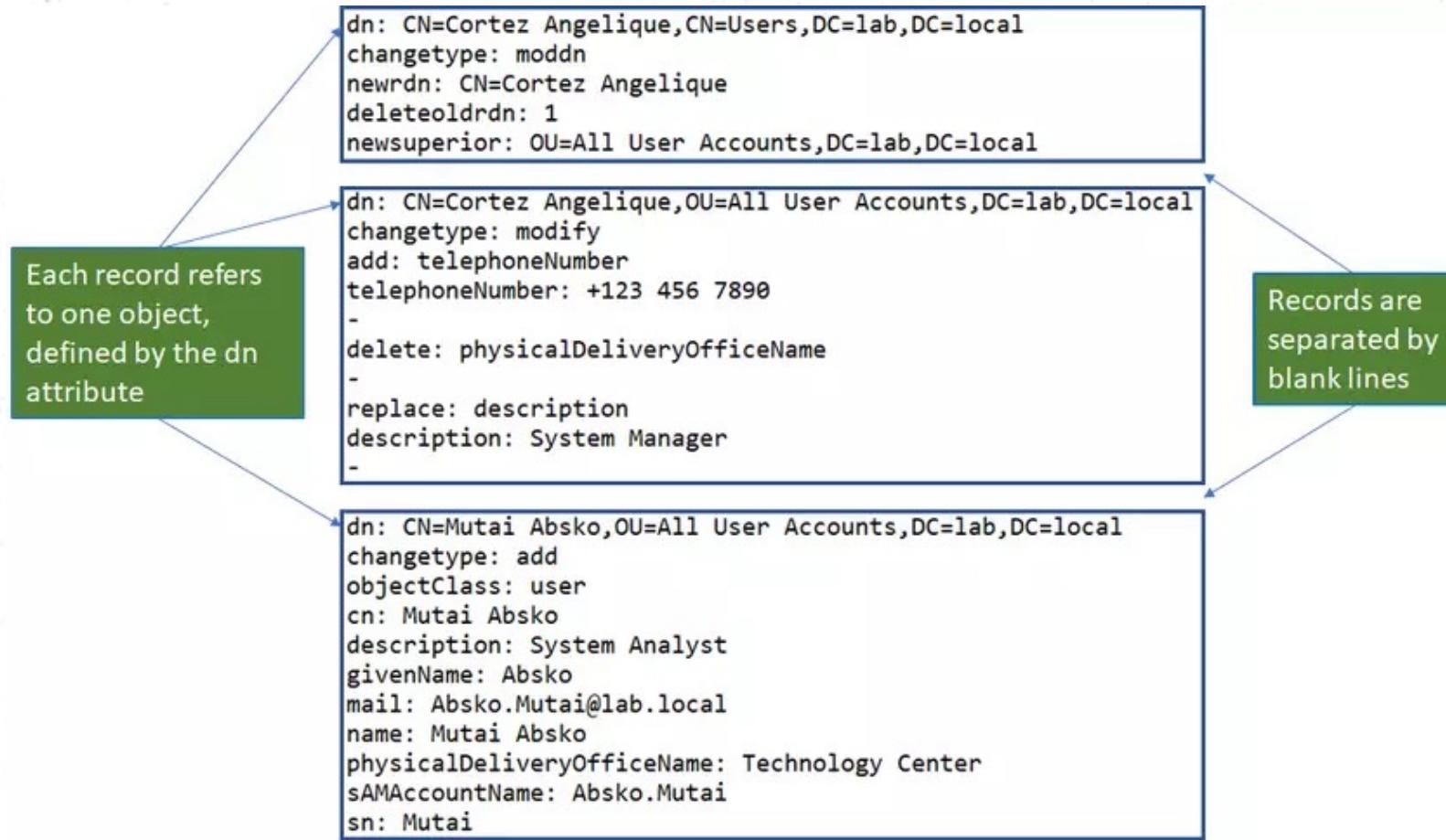
- Security = Encryption + Authentication + Access Control
- No access control in LDAP (yet...).
- Simple authentication (in LDAPv2).
  - DN and password provided.
  - Clear text or BASE 64 encoded.
- Simple authentication over SSL/TLS (in LDAPv3).
- Simple authentication and Security Layer SASL (RFC 2222).
  - Parameters: DN, mechanism, credentials.
  - Provides cross protocol authentication calls.
  - Encryption can be optionally negotiated.
  - `ldap_sasl_bind()` (ver3 call).
  - `ldap://<ldap_server>/?supportedsaslmechanisms`

# LDIF

- Stands for LDAP Data Interchange Format.
- LDIF is a standard text-based format for describing directory entries.
- Defined in RFC 2894.
- LDIF allows to export our directory data and import them to another directory server.
- Only ASCII (LDAPv3 uses UTF-8).
- An LDIF file is
  - A collection of entries separated from each other by blank lines . . . . .
  - A mapping of attribute names to values
  - A collection of directives that instruct the parser how to process the information
- The data in the LDIF file must obey the schema rules of your LDAP directory

Deploy an LDAP/AD tree in an org all the objects that necessary to create directory structure.

# LDIF Format

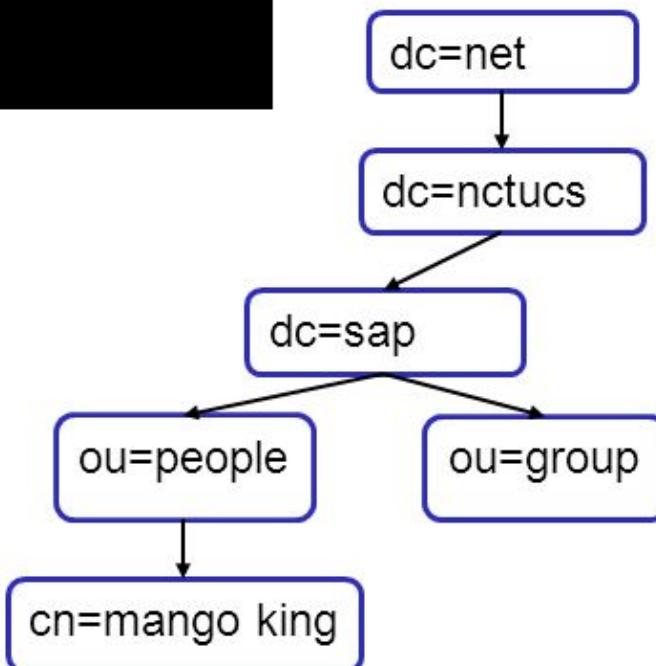


\* \*  
\* \*  
\* \*

## □ Sample LDIF

```
# sample entry
dn: cn=mango king,ou=people,dc=sap,dc=nctucs,dc=net
objectClass: person
cn: mango king
sn: king
telephoneNumber: 689-5566
```

dc: domain component  
ou: organizational unit  
cn: common name  
dn: distinguished name  
rdn: relative dn



# LDIF Example

```
dn: CN=Cox Paul,OU=All User Accounts,DC=labs,DC=local
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Cox Paul
sn: Cox
# This is their job title
description: IT Manager
givenName: Paul
distinguishedName:
CN=Cox Paul,OU=All User Accounts,DC=labs,DC=local
instanceType: 4
whenCreated: 20190111112006.0Z
whenChanged: 20190904114116.0Z
uSNCreated: 25067
memberOf:
CN=Professional Services Department,OU=All Groups,DC=labs,DC=local
memberOf: CN=DnsAdmins,CN=Users,DC=labs,DC=local
uSNChanged: 336116
name: Cox Paul
objectGUID:: Qxqi/AVGCUyDMSThWIKRHw==
---SNIP---
```

\* \*  
\* \*  
\* \*

Changetype

Comment

Multi-value  
attribute

Encoded value

# Schema Design

# Schema Design

- Similar to database systems, a schema is a set of rules that determines what data can be stored in the directory.
- **Purposes of a schema:**
- *Ensures that poorly designed applications “play by the rules” and do not store redundant data in the directory.*
  - If every directory-enabled application stored a person's name in a different directory attribute. The result would be wasted storage space.
- *Can be used to impose constraints on the size, range and format of data values stored in the directory.*
  - Ex: According to the Internet mail standards, e-mail address values should use a restricted set of characters and should conform to a specific format (addr@domain).
- *Can help slow the effect of directory entropy.*
  - Up to a certain level schema rules help in preventing disorder within your directory service.

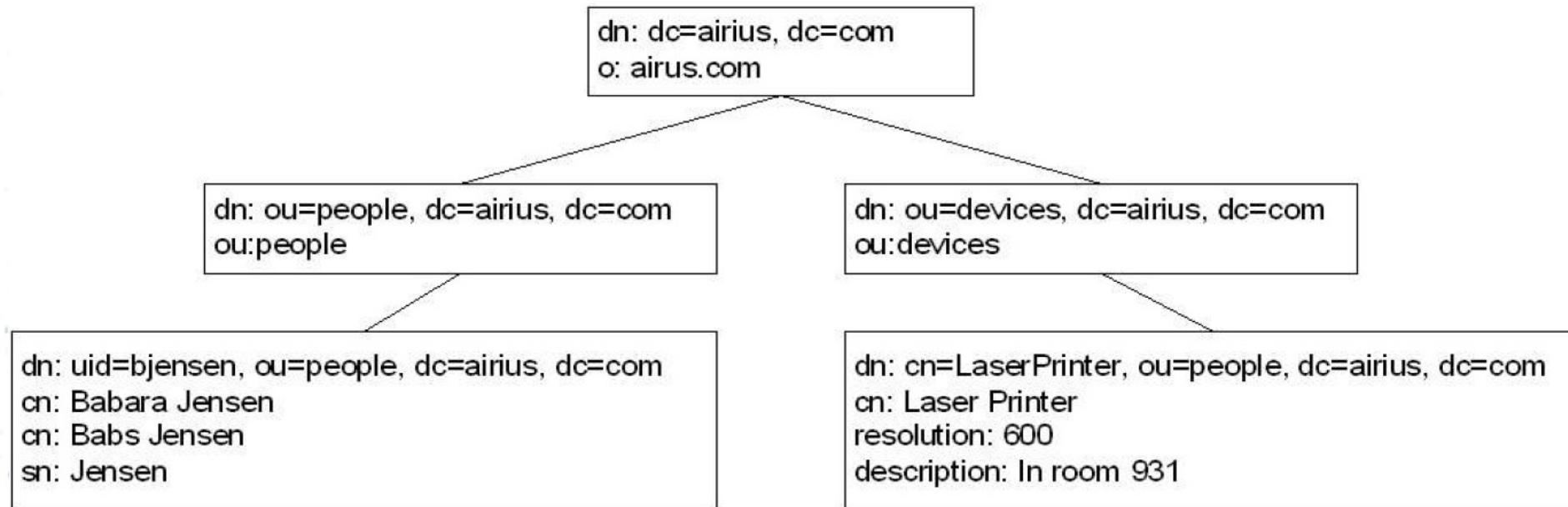
# Schema Design contd.

- LDAP based schemas consist of:
  - Attribute types.
  - Attribute syntaxes.
  - Matching rules.
  - Object classes.

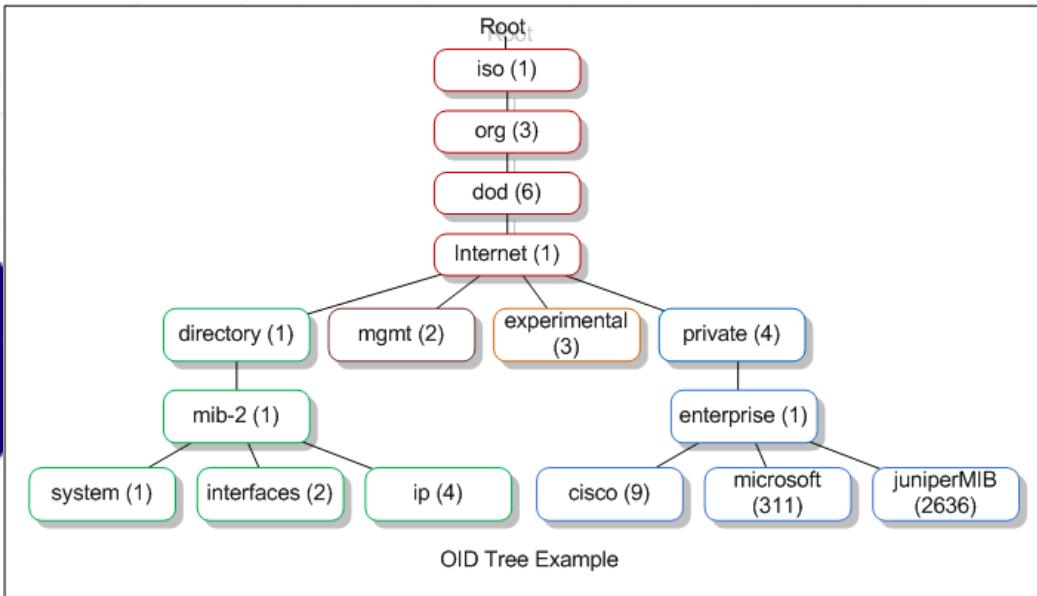
⋮  
⋮  
⋮

# Attributes

- As we have discussed, in a DIT, the directory entries contain a collection of attribute types and values.
- Attribute types (or simply attributes) hold specific data elements such as a name, business phone number etc.



- Each attribute type is having a *definition*.
- The *definition* contains the following:
  - A name that uniquely identifies the attribute type.
  - An OID that also uniquely identifies the attribute.
  - A textual description.
  - An indication of whether the attribute is single or multi-valued.
  - An associated attribute syntax and set of *matching rules*.
  - An attribute usage indicator.
  - Restrictions on the range or size of the values that may be stored in the attribute.
  - Indication of whether the attribute can be modified by regular applications.



# Attributes contd.

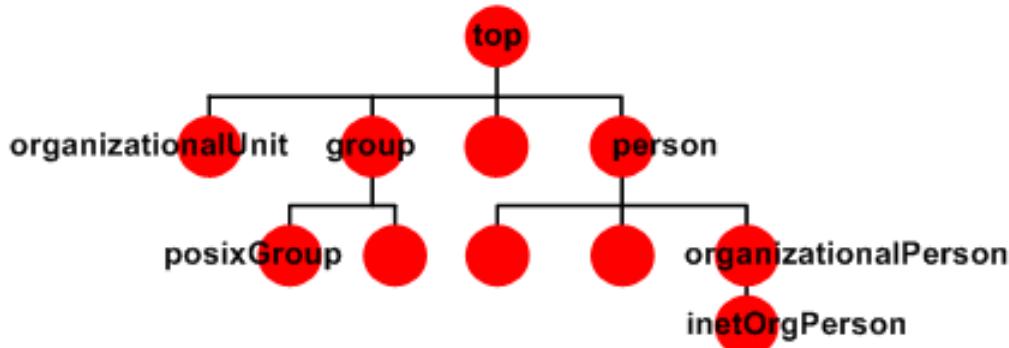
- Matching rules: govern comparisons and searches.
- Attribute usage indicator: whether for applications or for operation of the directory service itself.

# Properties of Attributes

- They are not case sensitive.
  - Ex: cn and CN are the same.
- Only ASCII characters and the hyphen are allowed.
  - Valid: cn, sn, postal-Address.
  - Invalid: -cn, hello#there.
- Name must start with a character.
- Unique across entire directory.
- Applications generally refer to an attribute using its name, rather than its OID.
- Attributes have one or more attribute values associated with them.
- Operational attributes are used by the directory itself for system related purposes.
  - Ex: modifyTimeStamp: date/ time an entry was last modified.

# Attribute Syntaxes and Matching Rules

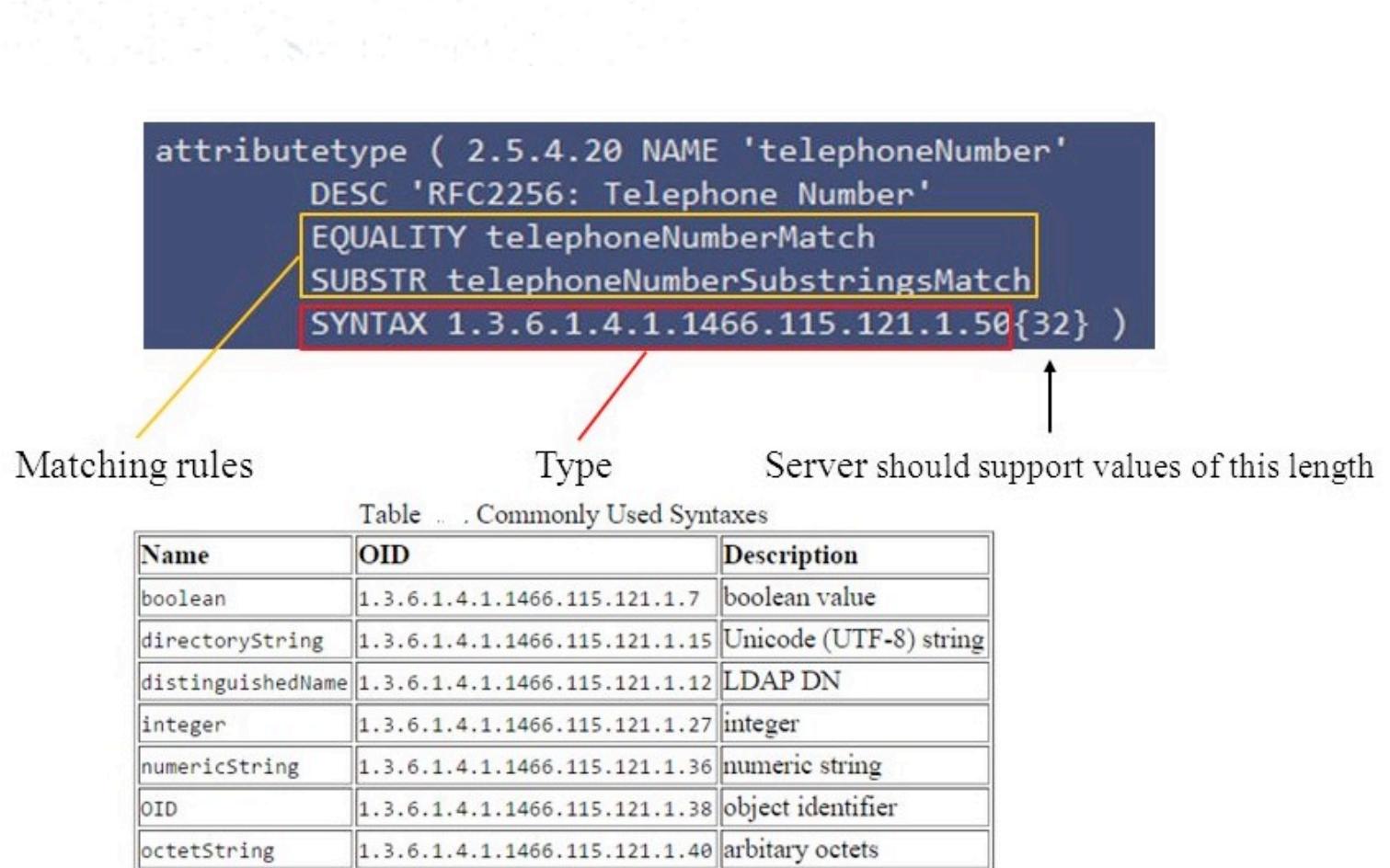
- Each attribute type has an associated:
  - Attribute syntax that specifies exactly how data values are represented and how comparisons between values are made.
  - The rules for making attribute value comparisons are called matching rules.



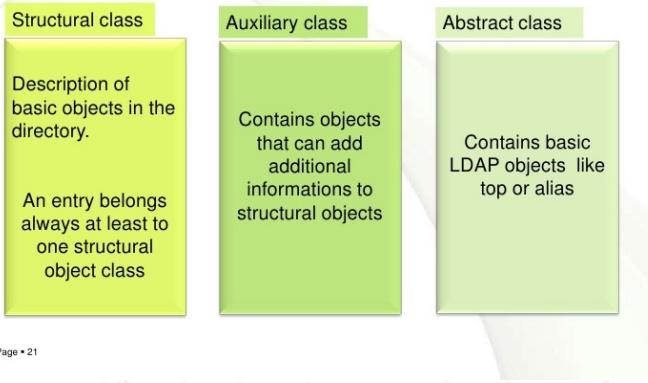
- Object classes are used to group related information (or attributes).
- Each directory entry belongs to one or more object classes.
- The names of the object classes to which an entry belongs are always listed as values for a special multi-valued attribute called *objectclass*.
- The set of object classes associated with an entry serves the following needs:
  - Determines which attribute types must be included in the entry.
  - Determines which attribute types may be included in the entry.

# Object Classes contd.

- An object class holds the following information:
  - A name that uniquely identifies the class.
  - Ex: person, printer, etc.
  - A textual description.
  - An OID that uniquely identifies the class.
  - A set of mandatory (required) attribute types.
  - A set of allowed (optional) attribute types.
  - A kind (structural, auxiliary or abstract) which indicates how the class is used.



### Three types of object class



- Structural classes describe the basic aspects of an object:
  - Most object classes are structural classes.
  - All entries should belong to exactly one structural object class.
- Auxiliary classes place no restrictions on where an entry may be stored:
  - Used to add a set of related attributes to an entry that already belongs to a structural class.
  - Sometimes call “mix-in” classes. Ex: simpleSecurityObject, mailRecipient, and cacheObject.
- Abstract classes may not be used directly, but only as ancestors of derived classes:
  - Rare. Abstract classes, the third and last kind of object class, are rare and are used only for classes needed to support LDAP’s basic information model. Two examples of abstract classes are top and alias. Used only for classes needed to support LDAP’s basic information model.
    - Ex: top, alias

# Object Class Definitions...Examples

- objectclass ( OIDp 'printer'
  - SUP top
  - STRUCTURAL
  - MUST cn
  - MAY ( description \$ pagesPerMinute \$ languages ))
- objectclass ( OIDn 'networkDevice'
  - SUP top
  - AUXILLIARY
  - MUST ipaddress
  - MAY ( cn \$ connectionSpeed ))

# Object Class Inheritance

- One object class can be derived from another: *inheritance*.
- The class that inherits some of its characteristics from is called the superior class or superclass.
- LDAP does not support multiple inheritance...!!!

# Namespace Design

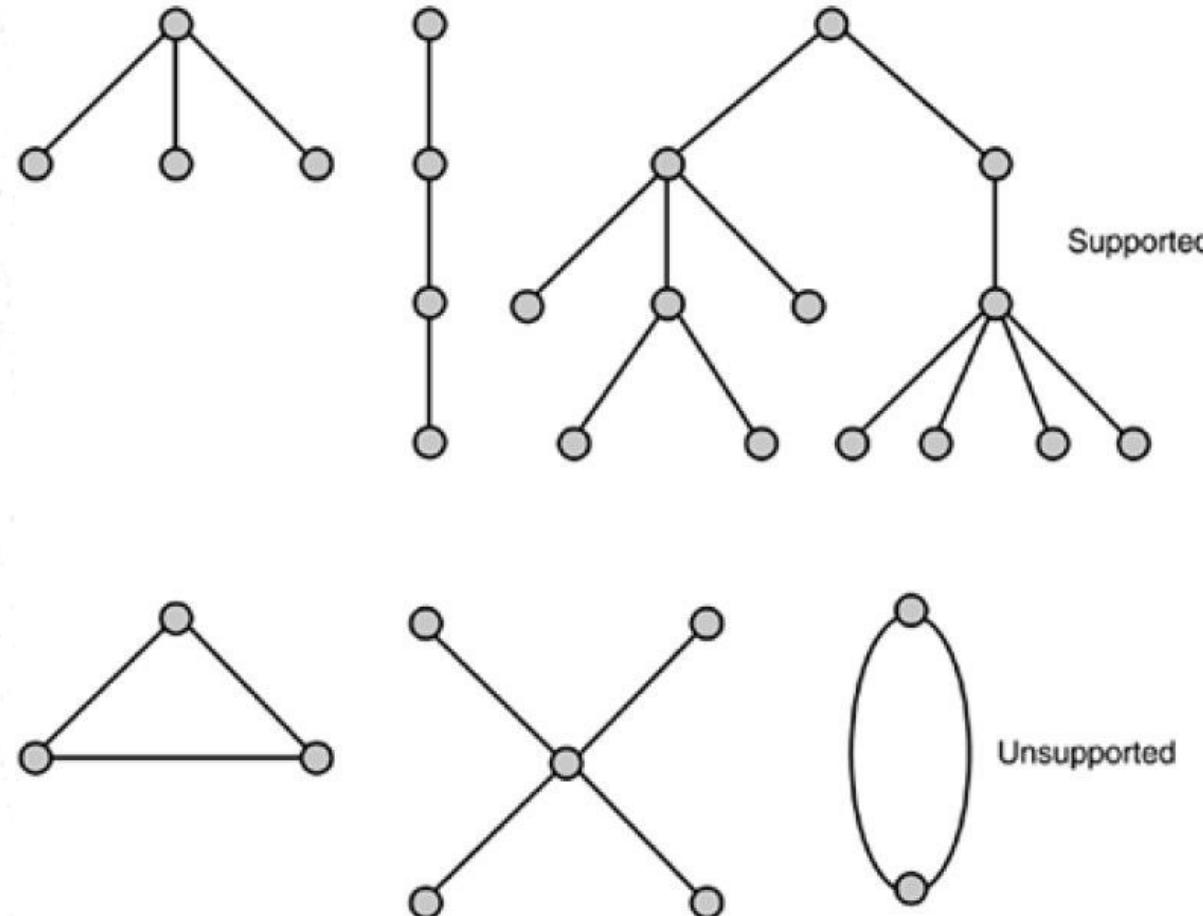
Your directory's namespace provides the basic means by which information is referenced in the directory

A properly designed namespace can lead to:

- Easier data maintenance.
- More flexibility in setting access control and replication policies.
- The ability to satisfy a wider variety of directory-enabled applications.
- More natural navigation through the directory.

- The LDAP model defines a flexible namespace framework, which means that you can almost certainly design a namespace to satisfy any requirements:
  - However, it also means that you have more choices to make than you might like.
- The LDAP namespace is inherited from the X.500 directory standard.
  - Tree-structured.
- The basic LDAP model is also hierarchical or tree-structured:
  - LDAP does not directly support namespaces in which an entry might be both a child and parent of the same entry:
  - alias and referral object classes can be used to construct such structures.
  - The seeAlso attribute indirectly supports such structures.

# Examples of Supported and Unsupported Structures



# Naming Entries in the Hierarchy

- Recall that an LDAP entry is a collection of attribute values.
  - The name of an entry is created by choosing one or more of these attribute values to form a Relative Distinguished Name (RDN).
  - However, multi-valued RDNs are discouraged.

<p>cn: Babara Jensen cn: Babs Jensen sn: Jensen title: Director uid: babs mail: babs@airius.com</p>	<p>cn=Babara Jensen uid=babs uid=babs+sn=Jensen mail=babs@airius.com</p>
Entry	Examples of RDNs

# Naming Entries in the Hierarchy cont.

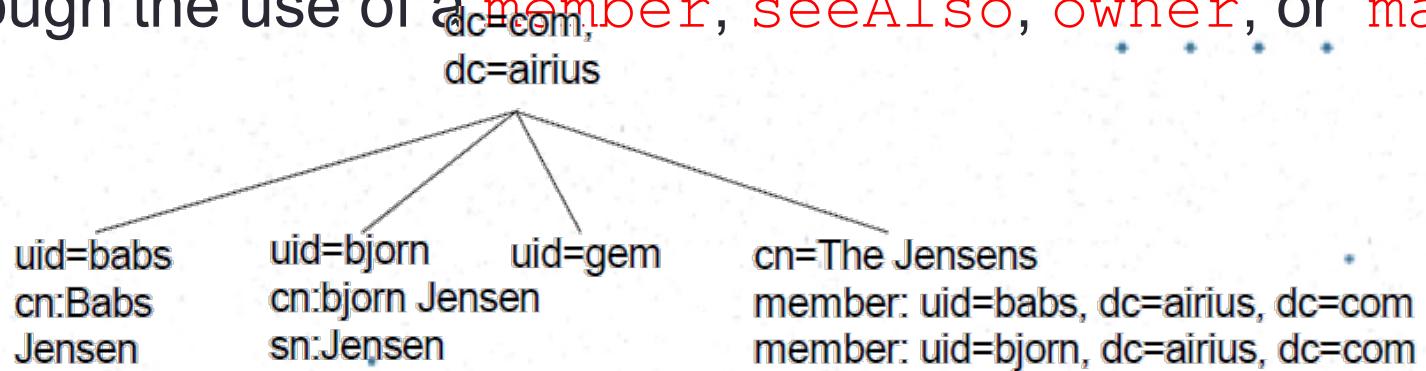
- When the RDN of an entry is chosen and its position relative to other entries in the hierarchy is determined,
  - forming the entry's full name is simple: Just combine the RDNs of the entry and all of its ancestor entries.
  - Resulting name is a Distinguished Name (DN).
  - DNs are written in little-endian order – Least significant component is written first.
    - E.g. cn=Barbara Jensen, o=Netscape, c=US

# Purposes of a Namespace

## Data Reference

A namespace provides the means by which **directory data** is referenced, which is important for two reasons.

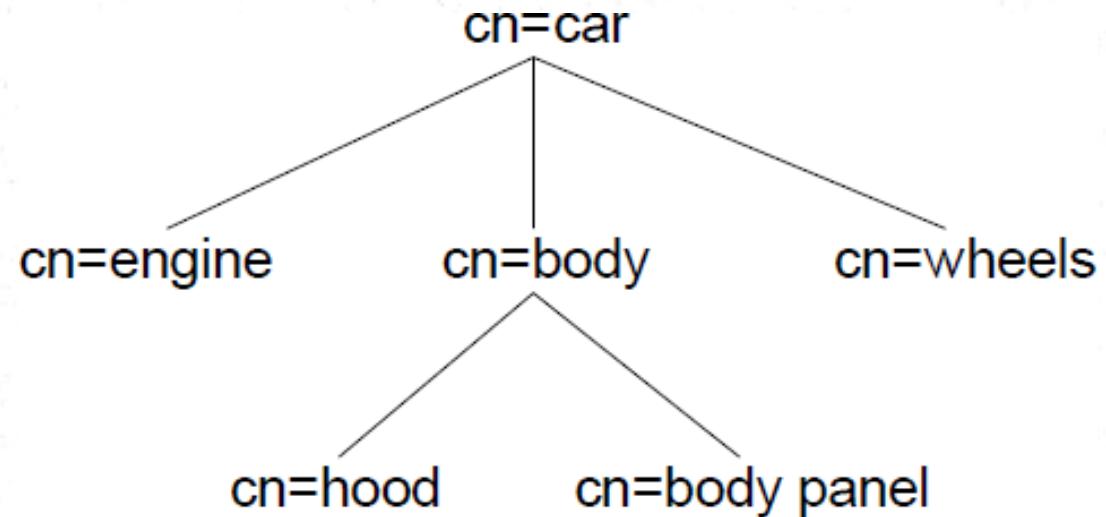
- First, there must be a way for directory clients to **refer unambiguously to a directory entry**.
- Second, the directory name provides a compact and efficient way to support groups of directory entries and directory entries that **refer to one another** (for example, through the use of a `member`, `seeAlso`, `owner`, or `manager` attributes).



# Purposes of a Namespace cont.

## Data Organization

For example, you might place all entries corresponding to people in one portion of the namespace, entries corresponding to devices in another etc. (further partitioning is possible).



# Purposes of a Namespace cont.

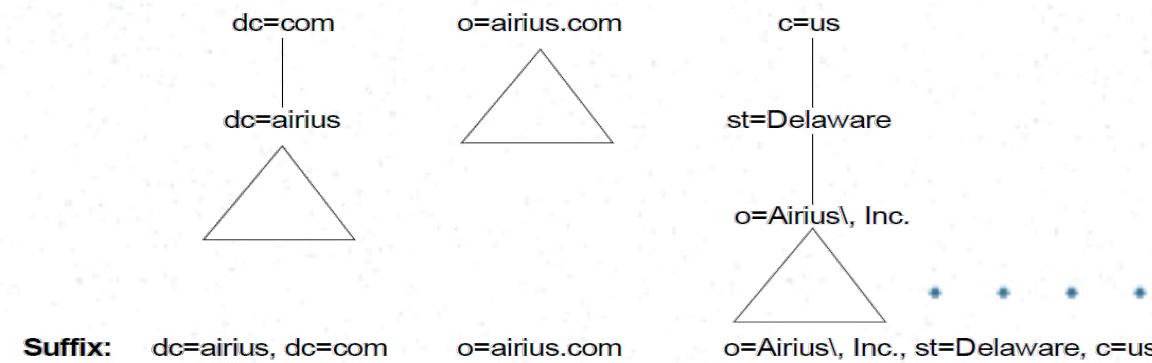
- Data partitioning:
  - A namespace enables directory data to be partitioned, or divided among multiple servers.
    - Partitioning can be performed only at a branch point in the directory.
- Data replication:
  - Choice of namespace design can constrain the replication scenarios your directory can support.
  - Most replication solutions require partitioning on branch points in the namespace.
- Access control:
  - Like replication, some products allow the setting of access control only at namespace branch points.
- Application support:
  - From a functional perspective, the namespace should support the applications that the directory is serving.

# Analyzing Namespace Needs

- Now that you have some idea of what you're going to do with your namespace after you define it, it's time to turn our attention to the **design process itself**.
- In this section we will look at the major areas and decisions you'll need to make when designing your namespace.
  - Choice of suffixes.
  - Flat or hierarchical namespace.
  - What attributes should you use to name entries.
  - Application requirements.
  - Administrative concerns.
  - Privacy concerns.
  - Future needs.
- We'll try to figure out what you gain and what you sacrifice at each above step.

# Choosing a Suffix

- The directory service may have only a local scope or it may be part of a larger or global directory system:
  - 3 methods are commonly used to choose the format of the suffix – RFC 2247, organization's DNS name itself or X.500 convention.
  - Directory's suffix should name the top of your department's tree otherwise.

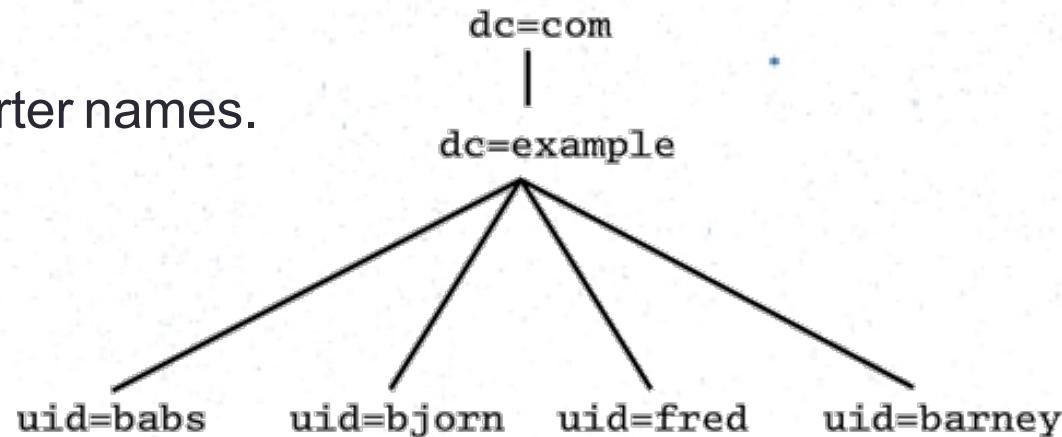


- A directory server may hold multiple suffixes:
  - You may want to design a service with multiple suffixes if you have two or more trees of information that do not have a natural common root.

# Flat and Hierarchical Schemes

## Flat Namespaces

- Less grouping and shorter names.

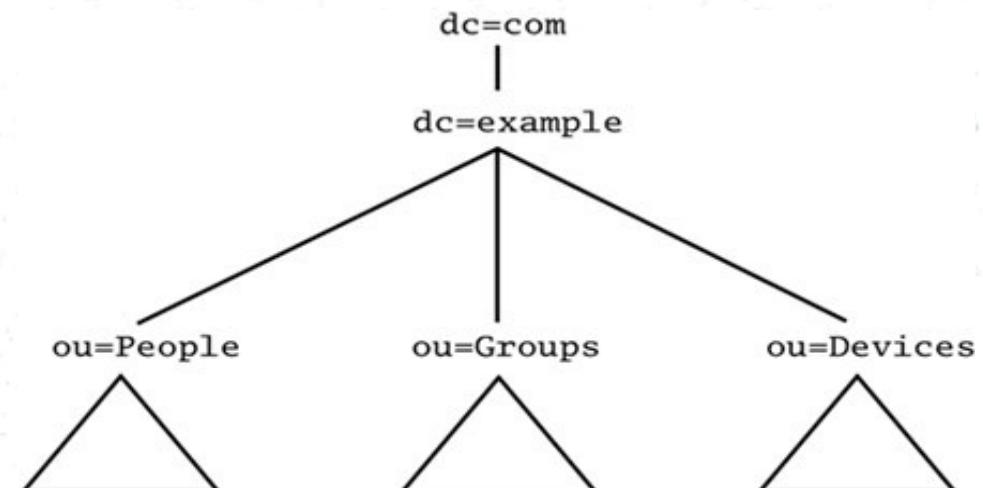


- As a guiding design principle, you should strive to mark a directory's namespace as flat as possible:
  - The flatter the namespace is, the **less likely names are to change**.
  - **Shorter names take up less space.**
    - E.g. Increasing the average name by only 20 bytes (the approximate result of adding an extra component) represents an increase of 2MB in a directory containing 100,000 entries..!
  - **Shorter names are easier to remember..!**

# Flat and Hierarchical Schemes

## Hierarchical Namespaces

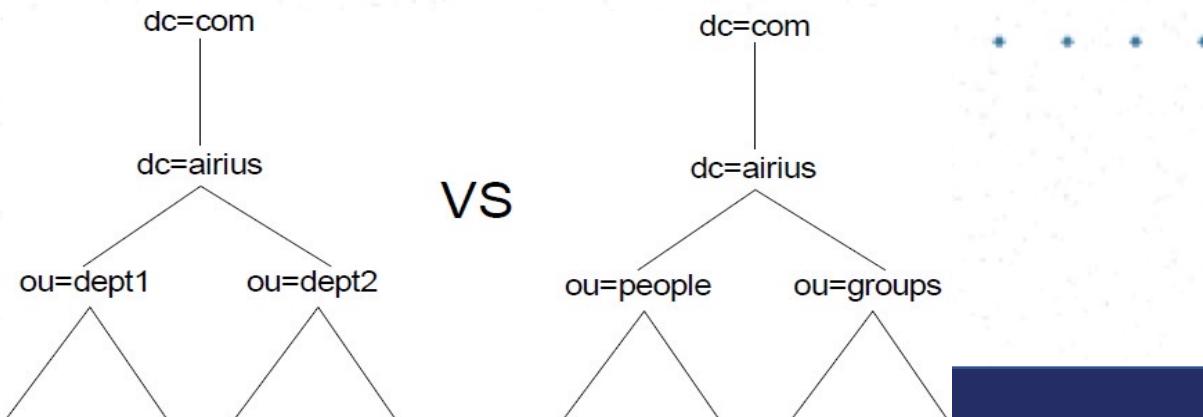
- Hierarchies can be used to **partition data amongst servers, enable replication, provide finer-grained access control, and browsing purposes.**
  - However, if a centralized directory is anticipated, there is no need to introduce hierarchies to enable data distribution.



# Flat and Hierarchical Schemes

## Hierarchical Namespaces cont.

- Design hierarchy in a way that **minimizes problematic name changes**.
- Name changes can be avoided if you are able to design your hierarchy based on information that is **not connected to directory information that is likely to change**.



# Naming Attributes

What attributes should you use to name entries

- Requirements imposed by LDAP on **naming attributes**:
  - The RDN of the entry must be chosen from one or more of the entry's attributes.
  - The RDN of the entry must be unique among all its siblings (other entries that have the same parent).
- Although these are the only restrictions, a **policy** which ensures **unique RDNs for people** is preferable (*unique names for people across the entire directory*):
  - Has the benefit that, even if an entry has to be moved, no name clashes will occur.

# Naming Attributes cont.

What attributes should you use to name entries

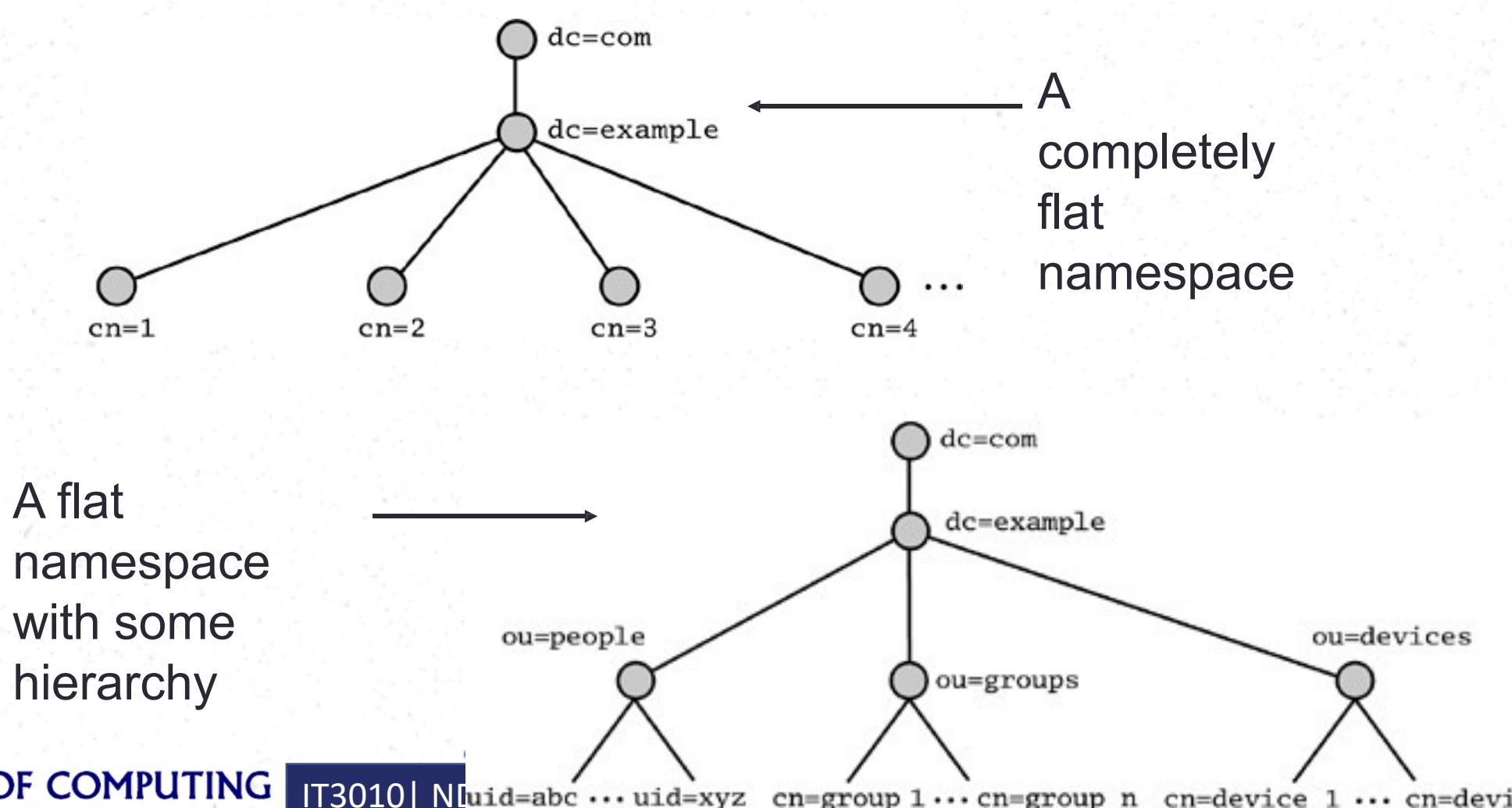
Two approaches to ensuring unique names (especially for people):

- Using existing unique names:
  - Employee IDs.
  - Unix login names.
- Constructing new identifiers:
  - Timestamping techniques.
  - Appending numbers to names.
  - Multivalued RDNs (strongly discouraged).

# Other Considerations

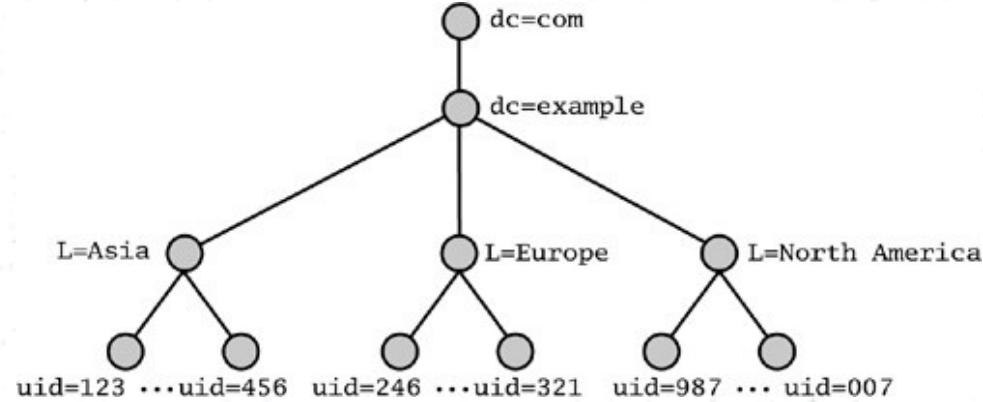
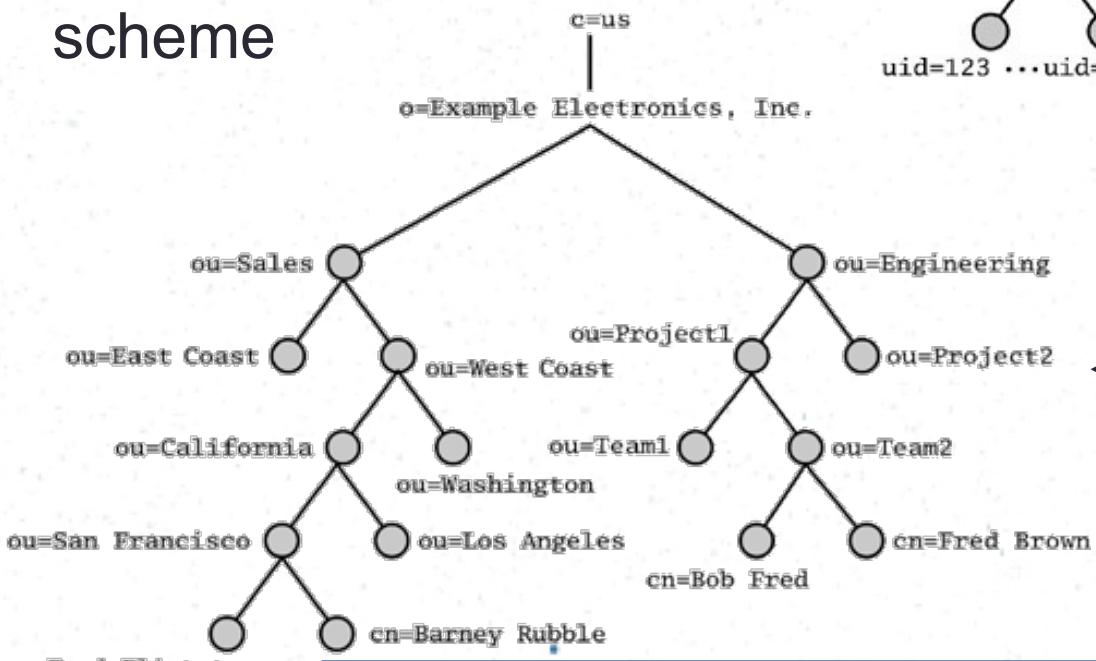
- **Applications considerations**
  - Typically, a directory is required to support one or more directory enabled applications.
- **Administrative considerations**
  - What is the impact of your namespace design on administrative tasks, such as adding/deleting entries, name changes or migration across directory server platforms.
- **Privacy considerations**
  - You should not reveal any information through the namespace that you do not intend to reveal..!
- **Anticipate the future** (situations that precipitate a namespace redesign):
  - Choosing the wrong naming attribute.
  - A directory starts out under central administrative control, but you later decide to hand over some portion of the data.
  - If you choose a hierarchical namespace with a hierarchy based on a geographical, organizational or any scheme that is bound to change.

# Some Example Namespaces



# Some Example Namespaces

A hierarchical namespace based on a geographical scheme



A hierarchical namespace based on organizational structure

# Topology Design

# Topology Design

- Partitions
- Gluing partitions
  - Referrals
  - Chaining
- Name resolution
- Authentication in a distributed directory
- Directory partitioning in openLDAP

•  
•  
•  
•

# What is Topology Design?

- LDAP directory services are designed to support a distributed directory
- The directory's topology describes:
  - the way you divide your directory tree among physical servers and
  - How you allocate those servers among your organisation's physical locations
- Benefits:
  - Achieve optimal performance for directory enabled applications
  - Increase directory availability
  - Better managed directory

\* \*  
\* \*  
\* \*

# Directory Topology Overview

- When you divide a single directory into manageable chunks and assign them to separate servers, you are partitioning the directory:
  - The directory itself is responsible for hiding partitioning details from users
  - Each server is responsible for only a portion of a tree
    - DNS operates in the same manner
  - In X.500 terminology, the unit of division is known as a naming context , partition or suffix
  - A directory partition is a complete sub-tree of the DIT
  - All entries in a partition must share a common ancestor known as the partition root

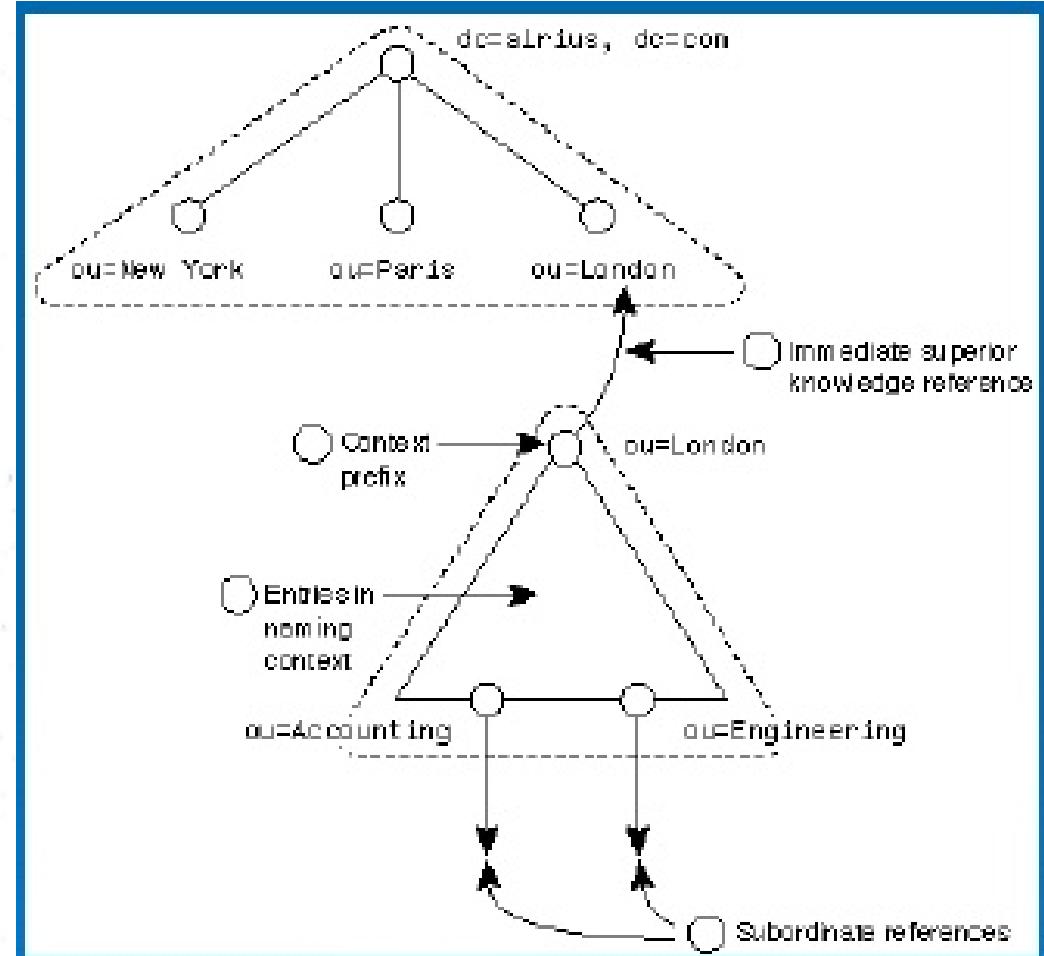


# Directory Topology Overview

- It is also possible to exclude directories from a partition
- Using these principles you can divide a single large directory into a number of smaller partitions

## Gluing the Directory Together

- We need some way to describe the relationships between directory partitions:
  - LDAP and X.500 defines these relationships in terms of knowledge references
- There are 2 types of knowledge references:
  - Immediate superior knowledge references: Point upward in the DIT toward the root and ties the naming context to its ancestor
  - Subordinate references: Point downward in the DIT to other partitions



# Name Resolution

- Name resolution is the process by which a directory maps a DN provided by a client to an actual object in the directory
- Name resolution is used in the following circumstances:
  - When locating the base object of an LDAP search or compare operation
  - When locating an entry to modify, delete, rename or bind as
  - When locating the parent of an entry to be added to the directory
- A DN presented in this fashion is called a purported name:
  - The client claims that it exists and its up to the directory system to check whether this is true

\* \*  
\* \*  
\* \*

## Name Resolution (Example)

1. The client presents the purported name
2. Since server 1 does not hold a naming context that could contain the entry, the immediate superior knowledge reference allows the name resolution process to walk up the tree to server 2
3. Server 2 contains a subordinate reference for the naming context `ou=engineering, dc=airius, dc=com`, so the name resolution process goes to server 3
4. Server 3 holds a subordinate reference for the naming context `ou=PCB design, ou=engineering, dc=airius, dc=com`, so the process walks down again
5. Finally, server 4 is consulted to check whether the entry exists

## Handling Distribution in the Client

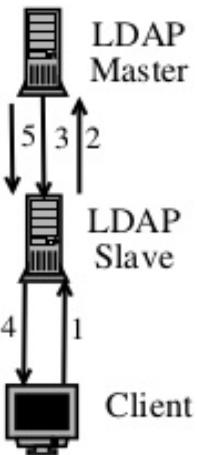
- LDAP referrals and search result continuation references are pieces of knowledge reference information sent from an LDAP server to an LDAP client:
  - Indicates that other servers need to be contacted to fulfil request
- LDAP referrals – When client performs add, modify, delete, moddn, compare or search operation with a base-level scope and the server does not hold target entry, an LDAP referral is returned
- Search result continuation references – When search scope is not base-level and the server has knowledge of other subordinate directory partitions, search result continuation references are returned
- Similar – Whereas referrals are contained in LDAP result messages, search result continuation references are contained in search result messages
- Normally, client libraries automatically handle processing of references

\* \*  
\* \*  
\* \*

# Structure of a LDAP Referral

- The information in a LDAP referral is in the format of a LDAP Uniform Resource Locator (URL)
- A referral gives the following information:
  - The hostname of the server to contact
  - The port number of the server
  - The base DN (search operation) or target DN (add, delete, compare or moddn operation)
    - Optional: if absent, the client should use the same base DN it used when performing the operation that resulted in the referral
- E.g. if a client searches the subtree dc=airius, dc=com for all entries with a surname smith, the referral would be returned as the following LDAP URL:

ldap://server3.airius.com:389/ou=engineering, dc=airius, dc=com



1. Client sends modification to replica
2. Replica forwards request to master
3. Master returns result to replica
4. Replica forwards result to client
5. Master updates replica

### Handling Distribution in the Server - Chaining

- When chaining is used, the client starts by submitting an operation to a server in the usual way:
  - If the server is unable to completely process the request because it does not hold all the required data, it chases the referral by contacting other servers on behalf of the client
  - It returns the combined results to the client when the other servers have completed the operation

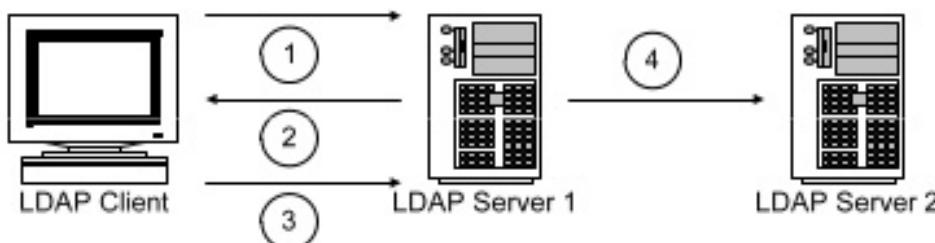
## Deciding Between Client or Server Distribution Handling

- Chaining in general reduces client complexity, but at the cost of increased server complexity:
  - Opposite is true if client is required to handle references
- You may not have a choice about which method you use:
  - openLDAP only supports client-side processing

•  
•  
•  
•

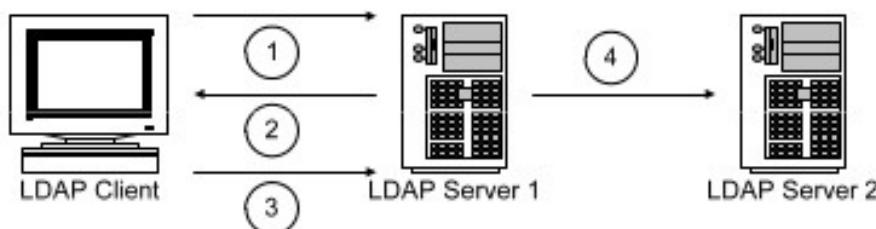
# Authentication in a Distributed Directory

- The server or servers that ultimately handle a client request must verify the identity of the client so that they can enforce restrictions
  - This is true even if the server handling the request is not the server to which the client originally authenticated
- Authentication in a chained environment is accomplished as follows:
  1. The directory client connects to server 1 and authenticates
  2. If the client's authentication credentials are successfully verified, server 1 returns a success code to the client
  3. The client submits a search operation to server 1
  4. Server 1 determines that it does not hold the appropriate partition, so it chains the operation to server 2



## Authentication in a Distributed Directory

- There are 2 ways server 2 might learn the clients identity:
  - Server 1 might tell server 2 who the client is and server 2 might choose to believe server 1
  - Server 1 might pass on to server 2 the client's identity and authentication credentials:
    - Server 2 could then independently verify the credentials
- • •



# Security Implications

- When using referrals, you should take precautions to ensure that you have absolute control over the referrals contained in your directory:
  - Additionally, you should allow referrals only to directories you trust
- Important because some directory clients may choose to automatically resubmit referred operations and authenticate to the referred-to server using the same credentials used when authenticating to the original server:
  - This means that if a rogue referral were placed in a directory, the directory clients may be tricked into submitting their authentication credentials
  - Similar principles apply for chaining – only trust remote servers under your direct control



# When to Partition?

- Factors favouring a directory with larger number of partitions:
  - The number of entries is too great for a single partition or server
  - Your directory-enabled applications tend only to read and modify entries from the local workgroup
  - The amount of update traffic to a single partition is too large
  - Partitioning allows data and update traffic to remain local and avoid spanning WAN links
  - Directory use will expand significantly in the future, beyond the point where a single partition is feasible
- •
- •
- •

# Replication Design

# Replication

- Replication design
- Replication scope
- Consistency and convergence
- Replication strategies



# Replication Design

- Replication increases the availability of your directory server:
  - Increased reliability - Protects your service from equipment failures, network partitioning etc...
  - Improved read performance – Reduces server, network load, thus reducing latency at clients and better load distribution
  - Improved write performance – Multi-master replication
- Security purposes:
  - By selectively replicating portions of a directory, the replica could function as a public directory service



# Replication Concepts

- Suppliers, consumers and replication agreements
- The unit of replication
- Consistency and convergence
- Incremental and total updates
- Initial population of a replica
- Replication strategies



## Suppliers, Consumers and Replication Agreements

- The terms supplier and consumer refer to the source and destination of replication updates:
  - These roles are not mutually exclusive
- The configuration information that tells a supplier server about a consumer server (and vice versa) is termed a replication agreement:
  - includes the unit of replication
  - the hostname and port of the remote server and
  - scheduling information
  - In essence, the replication agreement defines WHAT is to be replicated, WHERE it is to be sent and HOW it is to be done

# The Unit of Replication

- Define subtree – specify DN at the top of a subtree and replicate all entries subordinate to it
- Filtered replication (unsupported by openldap):
  - Define select entries in subtree – based on objectclass definitions:
    - X.500 defines this ability as the specification filter component of the unit of replication
  - Define select attribute types – based on attributetype definitions:
    - X.500 defines this as the attribute selection component of the unit of replication

• • •

# Consistency and Convergence

- Consistency describes how closely the contents of replicated servers match each other at a given point in time:
  - A strongly consistent replica is one that provides the same information as its supplier at all times
  - A weakly consistent replica is permitted to diverge from its supplier for some period of time:
    - All practical directory systems use weakly consistent replicas
- We say that a supplier and consumer have converged when they contain the same data

• •  
• •  
• •

# Initial Population of a Replica

- The consumer (replica) contains no data when it is initially configured:
  - Or, in the event that a replica becomes damaged, it must be brought back into synchronisation
- Directory vendors accomplish replica initialisation through similar techniques:
  - X.500 Directory Information Shadowing Protocol supports a total update strategy
  - Most LDAP products, including openLDAP, use LDAP itself to initialise the replica:
    - sending a series of delete operations to remove undesired entries and a series of add operations to populate the directory

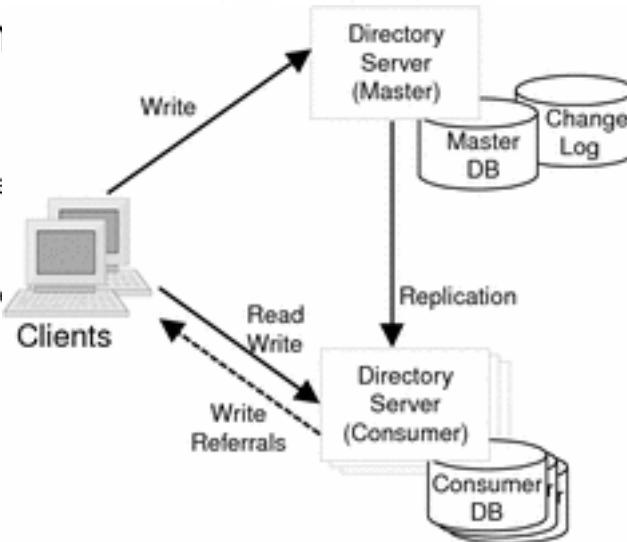
\* \*  
\* \*  
\* \*

# Replication Strategies

- The term replication strategy refers to the way updates flow from server to server and the way servers interact when propagating updates
- But after a client has successfully modified, deleted, added or renamed an entry, how does the server make it visible to its replicas?
- There are 3 main approaches toward solving this problem:
  - Single-master replication
  - Floating-master replication (unsupported by openldap) and
  - Multi-master replication (unsupported by openldap)

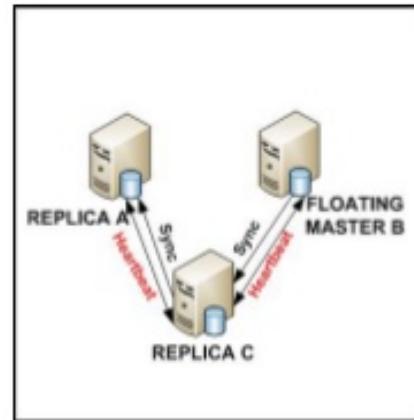
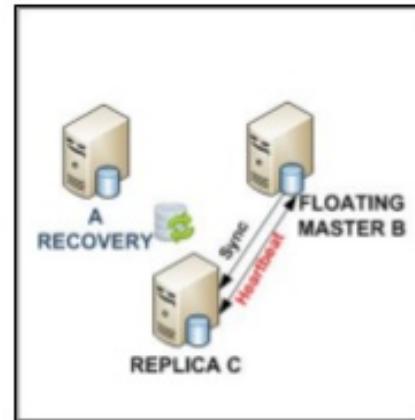
## Single Master Replication

- In single-master replication, there is one and only one server that contains a writable copy of a given directory entry.
  - Replicas are all read-only.
    - Advantage: since directory applications in general perform more read operations.
    - Disadvantage: single point of failure as the read-write server.
- What if the client needs to write to the directory via the
  - Referrals
  - Chaining



## Floating-Master Replication

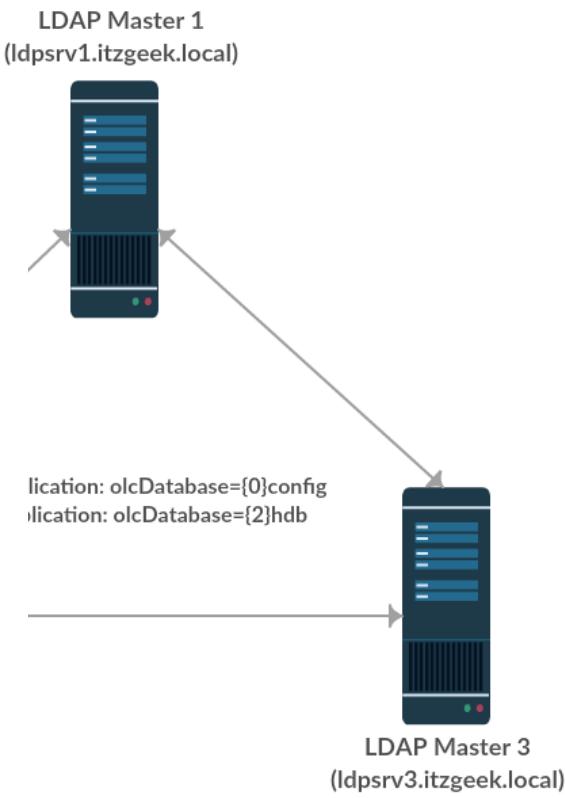
- Like single-master replication, floating-master replication maintains only a single read-write server:
  - When the read-write server becomes unavailable, a voting-based algorithm is used to elect a read-write master from the remaining replicas
  - Avoids single-point of failure
- Complications in synchronisation arise when 2 masters are rejoined:
  - Requires an update conflict resolution policy
    - **Last writer wins policy** - timestamps
    - Business rules – precedence between departments
  - Never used in directory products



# Multi-Master Replication

## Multi-master Replication

- In multi-master replication, there may be more than one read-write server available:
  - Clients submit a write operation to any of the read-write replicas.
    - Improves write performance
  - It becomes the responsibility of the set of cooperating servers to ensure that changes are eventually propagated
- Like floating master replication, this approach requires an update conflict resolution policy



LDAP Multi-Master Replication

# Thank You



# IT3010

## Network Design and Management

### Lecture 07

#### Socket Basics

**Shashika Lokuliyan**

Faculty of Computing  
Department of CSE



**SLIIT**

*Discover Your Future<sup>1</sup>*

## References:

Stevens, Fenner, Rudoff, *UNIX Network Programming*, vol. 1, Chapter 3.

⋮  
⋮  
⋮  
⋮  
⋮

## OBJECTIVE:

To discuss:

- Socket address structures.
- Byte manipulation functions.
- Address conversion functions.
- Read and write on a TCP socket.
- Example.

⋮  
⋮  
⋮  
⋮  
⋮

# Sockets Introduction

- Each socket has an address.
  - When a socket is created, it does not contain information about the protocol port number or IP address.
  - An application that wants to use a socket must specify the port number and IP address.
- Most socket functions require a pointer to socket address structure as an argument.
  - Note, each supported protocol defines its own socket address structure.

## Socket Address Structure

- An IPv4 socket address structure or Internet socket address structure is named **sockaddr\_in**.
  - Defined by including <netinet/in.h> header.
- POSIX = Portable Operating System Interface is a family of standards developed by IEEE, which are also adopted by ISO; Read Section 1.10 of Stevens' book.

# Structures for handling internet addresses

```
struct in_addr {  
    in_addr_t s_addr;           // 32-bit IPv4 address  
};  
  
* * *  
struct sockaddr_in {  
    uint8_t      sin_len;        // length of structure (16)  
    sa_family_t   sin_family;    // AF_INET  
    in_port_t     sin_port;       // 16-bit TCP or UDP port  
    // number * * * * *  
    struct in_addr sin_addr;     // 32-bit IPv4 address  
    char         sin_zero[8];     // unused  
};  
                                /* network byte ordered */
```

# Posix.1g datatypes

Datatype	Description	Header
int8_t	signed 8-bit integer	<sys/types.h>
uint8_t	unsigned 8-bit integer	<sys/types.h>
int16_t	signed 16-bit integer	<sys/types.h>
uint16_t	unsigned 16-bit integer	<sys/types.h>
int32_t	signed 32-bit integer	<sys/types.h>
uint32_t	unsigned 32-bit integer	<sys/types.h>
sa_family_t	address family of socket address structure	<sys/socket.h>
socklen_t	length of socket address structure, normally uint32_t	<sys/socket.h>
in_addr_t	IPv4 address, normally uint32_t	<netinet/in.h>
in_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

# IPv4 Socket Address Structure (cont.)

## Internet address

- struct **in\_addr** is the Internet address structure.
- s\_addr is a 32-bit IPv4 address in *network byte ordered* (big-endian byte ordering).

## Internet Socket Address

- struct **sockaddr\_in** is the Internet socket address structure.
- POSIX.1g requires the Internet socket address structure to contain at least:
  1. sin\_family
  2. sin\_port
  3. sin\_addr

. . . . .

- 1. *sin\_port*: 16 bit TCP or UDP port number in *network byte ordered*.

# IPv4 Socket Address Structure (cont.)

- **sin\_addr**: 32-bit IPv4 address in *network byte ordered*.
  - Can be accessed in two ways.
  - if serv is defined as Internet socket address structure:

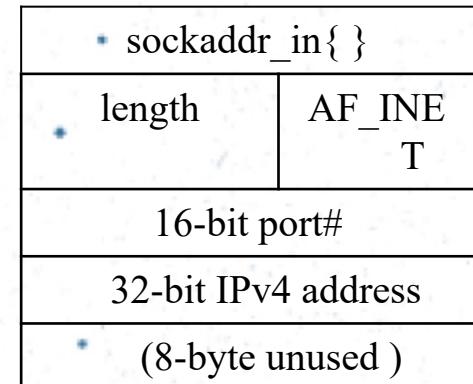
```
struct sockaddr_in serv;
```

then:
    - accessed as an in\_addr structure by:  
`serv.sin_addr;`
    - accessed as an in\_addr\_t by:  
`serv.sin_addr.s_addr;`
- **sin\_len** specifies the length of the socket address structure (fixed length: 16 bytes).
  - Simplifies the handling of variable-length socket address structures.
  - We need not set or examine it, except for routing sockets.
  - Used within kernel by routines that deal with socket address structures from various protocol families.

# IPv4 Socket Address Structure (cont.)

- *sin\_family*: Address family constants.
  - AF\_xxx: Address Family constants are used in the socket address structures.
  - PF\_xxx: Protocol Family constants are used to create a socket.
  - Both can be used interchangably, since they are set to same values.
- *sin\_zero* [8]: is unused, *always* set it to 0.
  - The entire structure (not just the sin\_zero) is always set to 0 at the beginning.

family	description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unixdomain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key socket



# Generic Socket Address Structure

- Socket address structures are *always* passed by reference to any of the socket functions.
- But the socket functions (such as bind ()) may be used in any of the supported protocol families.
- Problem: how to declare type of pointer that is passed.
  - ANSI C uses void \*
  - Solution chosen in 1982 was to define a *generic* Socket Address Structure in the <sys/socket.h>.

```
#include <sys/socket.h>
struct sockaddr {
    uint8_t    sa_len;
    sa_family_t sa_family;    /*address family:AF_xxx  value*/
    char       sa_data[14];    /*protocol specific address*/
};
```

- The socket functions are defined as taking a pointer to the generic socket address structure.
- From the programmer's view, the only use of these generic socket address structures is to cast to protocol-specific structures.

# Generic Socket Address Structure (cont.)

## Example

ANSI C function prototype for bind function:

```
int bind (int, struct sockaddr *, socklen_t);  
.  
* must use a pointer casting to the generic socket address structure */  
struct sockaddr_in serv;  
.  
* fill in serv {} */  
bind (sockfd, (struct sockaddr *) &serv, sizeof(serv));  
.
```

- If the cast (struct sockaddr \*) is omitted, C compiler generates a warning “incompatible pointer type”.
  - You may check this on your Linux system.

# Value Result Arguments

- Socket address structures can be passed in two directions:
  - From the process to the kernel.
  - From the kernel to the process.
- Socket address structure is always passed by *reference* (a pointer to the structure) to any socket functions.
- The length of the structure is also passed as an argument.
  - The way in which the length is passed depends on the directions:
    - Process to kernel.
    - Kernel to process.

# Value Result Arguments (cont.)

## Direction from process to kernel

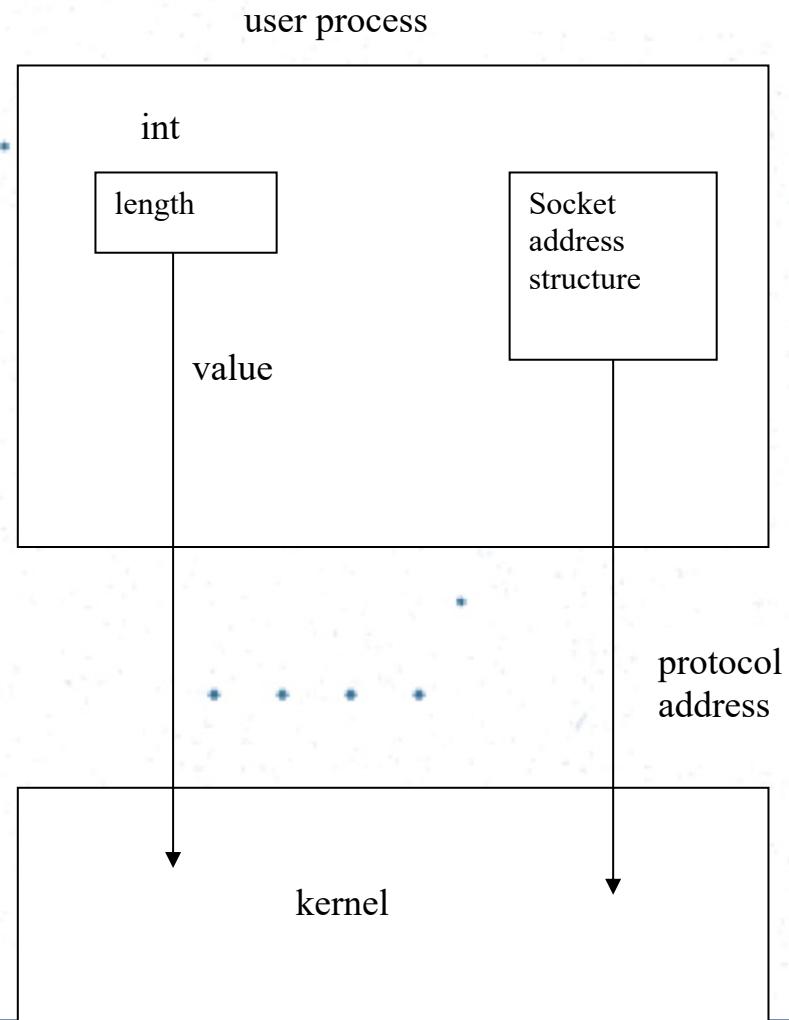
- Functions **bind()**, **connect()**, and **sendto()**, pass a pointer to the socket address structure and the integer size for the structure (from process to kernel).

## Example

```
struct sockaddr_in serv
/* fill in serv {} */

connect (sockfd, (struct sockaddr *)
&serv, sizeof(serv))
```

- Kernel knows exactly how much data to copy from the process to the kernel from sizeof(serv).



# Value Result Arguments (cont.)

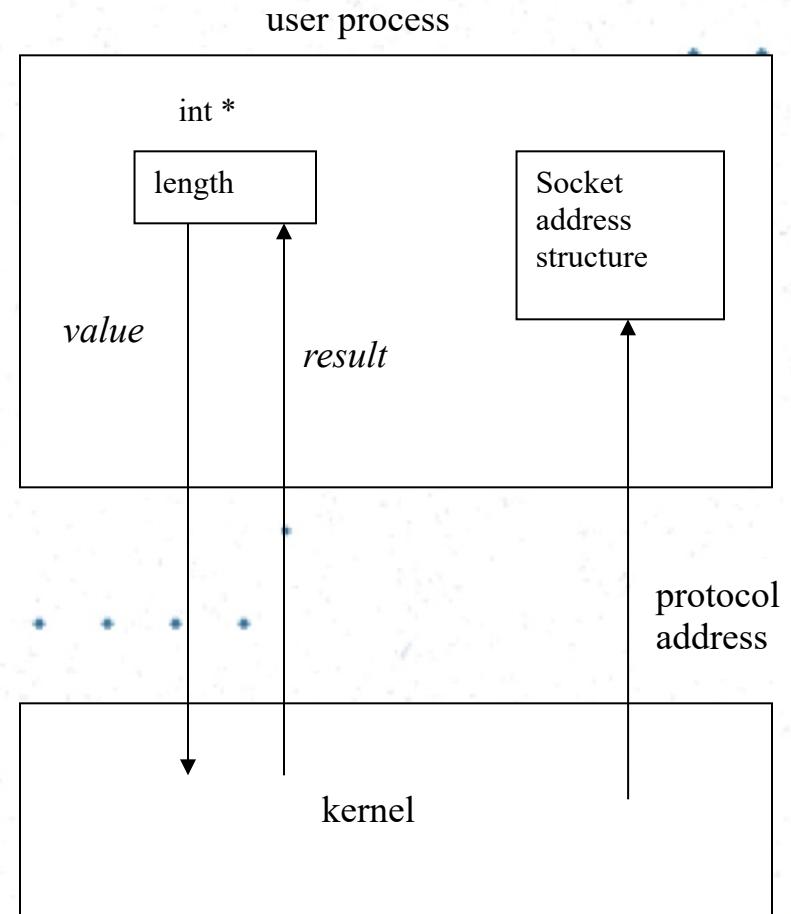
## Direction from kernel to process

- Functions `accept()`, `recvfrom()`, `getsockname()`, and `getpeername()` pass socket address structure from kernel to process.
- Here, we pass a *pointer* to the socket address structure along with a *pointer* to an integer containing the size of the structure.

## Example

```
struct sockaddr_un    cli; /*Unix domain*/  
socklen_t len;  
  
len = sizeof(cli);      /*len is a value*/  
getpeername (unixfd, (struct sockaddr *) &cli, &len);  
/*len may have changed*/
```

- When the function is called: the *len value* let the kernel know the size of the socket address structure.
- When the function returns: the kernel tells the process how much information it actually stored in the structure through a *result* in the *len*.
- For a fixed length socket address structure (e.g., IPv4), the result returned by the kernel will always be a fixed size, i.e., 16 bytes.



# Byte Ordering Functions

- There are two ways to store multi-bytes of data in memory:
  - *big-endian* byte order: high-byte is stored in lower address.

Address A	Address A+1
High-order byte	Low-order byte

- *little-endian* byte order: high-byte is stored in high address.

Address A+1	Address A
High-order byte	Low-order byte

- *host byte order*: the byte ordering used by a given system
- *network byte order*: the byte ordering used by networking protocols.
- Internet protocols use *big-endian* byte-order.

# Byte Ordering Functions

- Host computer may use either *big-endian* or *little-endian* byte ordering;
  - IBM System 370, Motorola 680x0, most RISC: *big-endian*.
  - Intel 80x86, Pentium, VAX: *little-endian*
  - PowerPC: Support both byte orderings. (*bi-endian*)
- Our concern is then converting between *host byte-order* and *network byte-order*.
  - We do not care about the actual values (*big endian* or *little endian*) for the host byte order and the network byte order.
  - We must call the appropriate function to convert a given value between the host and network byte order.
  - Systems that have the same byte ordering as the Internet protocol usually define converting functions as null macros.

# Byte Ordering Functions (cont.)

- Use these four functions to convert between these two byte orders; h=host, n=network, s=short, l = long.

```
#include      <netinet/in.h>
/*both return value in network byte order */
uint16_t htons (uint16_t host16bitvalue);
uint32_t htonl (uint32_t host32bitvalue);

Example
servaddr.sin_port = htons(13);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
// or htonl(0x860702DB); for 134.7.2.219

// or htonl(0b10000110000001110000001011001011)
/* both return value in host byte order */
uint16_t ntohs (uint16_t net16bitvalue);
uint32_t ntohl (uint32_t net32bitvalue);
```

Example

```
printf("port number %d\n", ntohs(servaddr.sin_port));
```

hton()	Host to Network Short	16-bit binary to big-endian
htonl()	Host to Network Long	32-bit binary to big-endian
ntoh()	Network to Host Short	16 bit to host format
ntohl()	Network to Host Long	32 bit to host format

# Program byteorder.c (from book)

```
#include <stdio.h>
#define CPU_VENDOR_OS "i386-pc-bsdi3.0"
int
main(int argc, char **argv)
{
    union {
        short s; // 2 bytes
        char c[sizeof(short)]; //2 chars or 2 bytes
    } un;
    un.s = 0x0102; //how these two bytes stored in memory depends on the host machine endian type
    printf("%s: ", CPU_VENDOR_OS);
    if (sizeof(short) == 2) {
        if (un.c[0] == 1 && un.c[1] == 2) //with the defined union, you can access the
two bytes one after
            printf("big-endian\n"); // the other.
        else if (un.c[0] == 2 && un.c[1] == 1)
            printf("little-endian\n");
        else
            printf("unknown\n");
    } else
        printf("sizeof(short) = %d\n", sizeof(short));
    exit(0);
}
```

## Example:

Consider a 2-byte data 0102H stored in memory location 100 and 101.

*big-endian*: [101] = 02H, [100] = 01H

*little-endian*: [100] = 02H, [101] = 01H

# Byte Manipulation Functions

Two groups of functions that operate on multibyte fields without interpreting the data, and without assuming that the data is null-terminated C string.

- Functions with names beginning with b (for byte). → `bzero()`, `bcopy ()`, `bcmpl()` → from 4.2BSD.
- Functions with names beginning with mem (for memory) → `memset()`, `memcpy()`, `memcmp()` → ANSI C standard.

```
#include <string.h>
/* sets the specified number of bytes to 0 in the destination */
void bzero (void *dest, size_t nbytes);
Example
bzero(&servaddr, sizeof(servaddr));

/* moves the specified number of bytes from the source to the destination */
void bcopy (const void *src, void *dest, size_t nbytes);

/* returns 0 if equal, nonzero if unequal */
int bcmp (const void *ptr1, const void *ptr2, size_t nbytes);
```

- `bzero()` function is used to initialize a socket address structure to 0.
- Note, the memory pointed to by the const pointer is read but not modified by the function.

# Byte Manipulation Functions (cont.)

```
/* sets the specified number of bytes to the value c in the destination */  
void *memset (void *dest, int c, size_t len);
```

## Example

```
memset(char * &servaddr, 0, sizeof(servaddr));
```

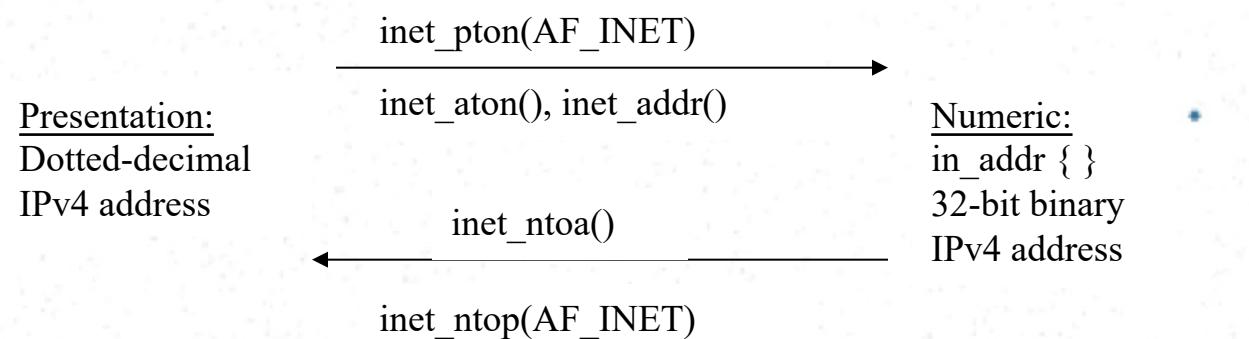
```
/* moves the specified number of bytes from the source to the destination */  
void *memcpy (void *dest, const void *src, size_t nbytes);
```

```
/* returns 0 if equal, nonzero if unequal */  
int memcmp (const void *ptr1, const void *ptr2, size_t nbytes);
```

- **bzero()**, **bcopy()**, and **bcmp()** are still provided by any system that supports the socket functions.
- **memset()**, **memcpy()**, and **memcmp()** are from ANSI C standard → better for portability.

# Presentation format of IP address

- There are two groups of Internet address conversion functions.
  - Address-conversion functions that *convert Internet addresses in ASCII strings (human readable format, text representation (i.e., “206.62.226.33”) to network-byte-ordered binary values (binary values that are stored in socket address structures): `inet_addr()`, `inet_aton()`, and `inet_nton()`.*
  - Address-conversion functions that *convert Internet addresses in network-ordered binary values to ASCII strings: `inet_ntoa()`, and `inet_ntop()`.*
- `inet_aton()`, `inet_ntoa()`, and `inet_addr()` are used for IPv4.
- `inet_ntop()` and `inet_ntop()` handle both IPv4 and IPv6.



# Presentation format of IP address (cont.)

```
#include <arpa/inet.h>
```

## inet\_aton()

```
/* returns 1 if string was valid, 0 on error */  
int inet_aton (const char *strptr, struct in_addr *addrptr);
```

Returns: 1 if successfull, 0 on error

- \* Converts the C character string (e.g., 206.62.226.33) pointed to by *strptr* into its 32-bit binary network byte ordered value stored through the pointer *addrptr*.

## inet\_addr()

```
/* returns 32-bit binary network byte ordered IPv4 address; INADDR_NONE if error */  
int_addr_t inet_addr (const char *strptr);
```

- \* Does the same conversion as *inet\_aton ()*, but returning the 32-bit binary network byte ordered value as the return value.
- \* The function returns INADDR\_NONE ( typically 32 one-bits) on an error.
- \* Problem: The function can not handle string 255.255.255.255 which should result in 32 one-bits.
- \* *inet\_addr()* is deprecated → use *inet\_aton ()*.

## inet\_ntoa()

```
/* returns pointer to dotted-decimal string */  
char *inet_ntoa (struct in_addr inaddr);
```

- \* Converts a 32-bit binary network byte ordered IPv4 address into its corresponding dotted-decimal string.

The string is returned in a statically allocated buffer, which subsequent calls will overwrite.

This function takes a structure as its argument, not a pointer to a structure, which is rare.

# Presentation format of IP address (cont.)

```
#include <arpa/inet.h>
```

Compatible with both IPv4 and IPv6 addresses.

## inet\_ntop()

```
/* returns 1 if OK, 0 if input not a valid presentation format, -1 on error */
int inet_ntop (int family, const void *addrptr, char *strptr, size_t len);
```

- Convert the string pointed to by *strptr*, and store the binary result through the pointer *addrptr*.
- Letters *p* and *n* stand for *presentation* and *numeric*.
- family* argument is either **AF\_INET** or **AF\_INET6**.

If *family* is not supported, both functions return an error, **EAFNOSUPPORT**.

## inet\_ntop()

```
/* returns pointer to result if OK, NULL on error */
const void *inet_ntop (int family, const void *addrptr, char *strptr, size_t len);
```

- convert the numeric form (*addrptr*) into presentation (*strptr*).
- len* is the size of the destination, to prevent the function from overflowing the caller's buffer. If too small error **ENOSPC** returned.

Caller must allocate memory for the destination and specify its size. On success, this pointer is the return value of the function.

# Void pointer

- In C **General Purpose Pointer** is called as void Pointer.
- It does not have any data type associated with it
- It can store address of any type of variable
- A void pointer is a C convention for a raw address.
- The compiler has no idea what type of object a void Pointer really points to

# Presentation format of IP address (cont.)

## Example:

```
Char *cp="134.7.2.219";
struct sockaddr_in serv;
char *ptr;
serv.sin_addr.s_addr = inet_addr(cp);

//Replace inet_addr() with inet_nton()
```

```
ptr = inet_ntoa(serv.sin_addr);
char str[INET_ADDRSTRLEN];

//Replace inet_ntoa() with inet_ntop()
```

Note:

```
#include <netinet/in.h>
* contains the following definition:
#define INET_ADDRSTRLEN 16      /*for IPv4 dotted decimal */
#define INET6_ADDRSTRLEN 46     /*for IPv6 Hex string */
```

object called *serv*

```
struct sockaddr_in {
    sa_family_t   sin_family;
    in_port_t     sin_port;
    struct in_addr sin_addr;
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;
};
```

# Reading and writing to/from sockets

- A *read()* or *write()* on a stream socket (e.g., TCP sockets) might input or output fewer bytes than requested (not an error);

Reason: buffer limits might be reached for the socket in the kernel.

- For the caller, need to call the *read()* or *write()* again to input or output the remaining bytes.

Example: code to read input from socket and output to the screen

```
char recvline[MAXLINE + 1];
while ((n = read(sockfd, recvline, MAXLINE)) > 0) {
    recvline[n] = 0; /* null terminate */
    if (fputs(recvline, stdout) == EOF)
        err_sys("fputs error");
}
if (n < 0)
    err_sys("read error");
```

- The book provides the following three functions:

- *readn()*, to read *n* bytes from a stream socket file descriptor and store it in a buffer.

```
ssize_t readn(int filedes, void *buff, size_t nbytes);
```

- *readline()*, to read a text line from a stream socket file descriptor and store it in a buffer.

```
ssize_t readline(int filedes, void *buff, size_t maxlen);
```

- *writen()*, to write *n* bytes to a stream socket file descriptor from a buffer.

```
ssize_t writen(int filedes, const void *buff, size_t nbytes);
```

All return: number of bytes read or written, -1 on error

# readn ()

```
/* returns number of bytes read, -1 or error */
#include "unp.h"
ssize_t /* Read "n" bytes from a descriptor. */
readn(int fd, void *vptr, size_t n)
{
    size_t nleft;
    ssize_t nread;
    char *ptr;
    ptr = vptr;
    nleft = n;
    while (nleft > 0) {
        if ((nread = read(fd, ptr, nleft)) < 0) {
            if (errno == EINTR)
                nread = 0; /* and call read() again */
            else
                return(-1);
        } else if (nread == 0)
            break; /* EOF */
        nleft -= nread;
        ptr += nread;
    }
    return(n - nleft); /* return >= 0 */
} /* end readn */

/* readn wrapper function */
ssize_t
Readn(int fd, void *ptr, size_t nbytes)
{
    size_t n;
    if ((n = readn(fd, ptr, nbytes)) < 0)
        err_sys("readn error");
    return(n);
}
```

# written()

```
/* returns number of bytes written, -1 or error */
#include "unp.h"
ssize_t /* Write "n" bytes to a descriptor. */
written(int fd, const void *vptr, size_t n)
{
    size_t nleft;
    ssize_t nwritten;
    const char *ptr;
    ptr = vptr;
    nleft = n;
    while (nleft > 0) {
        if ((nwritten = write(fd, ptr, nleft)) <= 0) {
            if (errno == EINTR)
                nwritten = 0; /* call write() again */
            else
                return(-1); /* error */
        }
        nleft -= nwritten;
        ptr += nwritten;
    }
    return (n);
} /* end written */

/* writen wrapper function */
void
Writen(int fd, void *ptr, size_t nbytes)
{
    if (written(fd, ptr, nbytes) != nbytes)
        err_sys("written error");
}
```

# readline Version 1

```
/* returns number of bytes read, -1 or error */
#include    "unp.h"
ssize_t
readline(int fd, void *vptr, size_t maxlen)
{
    ssize_t n, rc;          char c, *ptr;
    ptr = vptr;
    for (n = 1; n < maxlen; n++) {
again:   if ((rc = read(fd, &c, 1)) == 1) {
            *ptr++ = c;
            if (c == '\n')
                break; /* newline is stored, like fgets() */
        } else if (rc == 0) {
            if (n == 1)
                return(0); /* EOF, no data read */
            else
                break; /* EOF, some data was read */
        } else {
            if (errno == EINTR)
                goto again;
            return(-1); /* error, errno set by read() */
        }
    }
    *ptr = 0; /* null terminate like fgets() */
    return(n);
} /* end readline */
```

```
ssize_t
Readline(int fd, void *ptr, size_t maxlen)
{
    ssize_t n;
    if ((n = readline(fd, ptr, maxlen)) < 0)
        err_sys("readline error");
    return(n);
}
```

# readline Version 2

```
/* returns number of bytes read, -1 or error */

#include "unp.h"
static int read_cnt = 0;
static char *read_ptr;
static char read_buf[MAXLINE];
static ssize_t

my_read (int fd, char *ptr)
{
    if (read_cnt <= 0) {
again:
    if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
        if (errno == EINTR)
            goto again;
        return(-1);
    } else if (read_cnt == 0)
        return(0);
    read_ptr = read_buf;
}
    read_cnt--;
    *ptr = *read_ptr++;
    return(1);
}
```

The internal function **my\_read** reads up to **MAXLINE** characters at a time and then returns them, one at a time.

# readline Version 2 (cont.)

```
ssize_t readline(int fd, void *vptr, size_t maxlen)
{
    int    n, rc;
    char   c, *ptr;
    ptr = vptr;
    for (n = 1; n < maxlen; n++) {
        if ((rc = my_read(fd, &c)) == 1) {
            *ptr++ = c;
            if (c == '\n')
                break; /* newline is stored, like fgets() */
        } else if (rc == 0) {
            if (n == 1)
                return(0); /* EOF, no data read */
            else
                break; /* EOF, some data was read */
        } else
            return(-1); /* error, errno set by read() */
    }
    *ptr = 0; /* null terminate like fgets() */
    return(n);
} /* end readline */
```

A new function, readline, exposes the internal buffer state so that callers can check and see if more data was received beyond a single line.

- The only change to the `readline` function itself is to call `my_read` instead of `read`.

```
/* readline () wrapper function */
ssize_t
Readline(int fd, void *ptr, size_t maxlen)
{
    ssize_t      n;
    if ((n = readline(fd, ptr, maxlen)) < 0)
        err_sys("readline error");
    return(n);
}
```

# comparison

- **Version 1** of the readline() function calls the system's read() once for every byte of data → inefficient.
- **Version 2** of the readline () function put the data in user buffer by calling read() to obtain as much data as we can (up to MAXLINE) and then examine the user buffer 1 byte at a time → faster.
- The book reports big performance difference: For reading 2781-line file (135,816 bytes):
  - **Version 1:** 8.8 seconds, 135,816 system calls.
  - **Version 2:** 0.3 seconds, 34 system calls. (135,816/4096).
- Security and reliability are very important in network programming.
  - Make sure that the buffers cannot be overflowed.
  - Make sure that the return value from calling a function is checked.

# Client-Server Programs (Stevens' book)

## Daytime Client:

- Establishes a TCP connection with a server, read the reply message from the server, and print the message to the screen.
- Needs functions: `socket()`, `connect()`, `read()`.

## Daytime Server:

- Waits for connection from a client, sends back the current time and date in a human-readable format, and wait for another connection (infinite loop).
- Needs functions: `socket()`, `bind()`, `listen`, `accept()`, `write()`.

letters).

# Daytime Client

```
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int      sockfd, n;
6     char    recvline[MAXLINE + 1];
7     struct sockaddr_in          servaddr;
8     if (argc != 2)
9         err_quit("usage: a.out <IPaddress>");
10    /*create a TCP socket: Internet (AF_INET) stream (SOCK_STREAM) socket */
11    if((sockfd=socket(AF_INET,SOCK_STREAM,0))< 0)
12        err_sys("socket error");
13    bzero(&servaddr, sizeof(servaddr));
14    servaddr.sin_family = AF_INET;
15    servaddr.sin_port   = htons(13);/* daytime server */
16    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
17        err_quit("inet_ntop error for %s", argv[1]);
18    if (connect(sockfd, (SA *) & servaddr, sizeof(servaddr)) < 0)
19        err_sys("connect error");

20    while ((n =readn(sockfd, recvline, MAXLINE))> 0) {
21        recvline[n] = 0;           /* null terminate */
22        if (fputs(recvline, stdout) == EOF)
23            err_sys("fputs error");
24    }
25    if (n < 0)
26        err_sys("read error");
27    exit(0);
}
```

# Daytime Server

```
1 #include "unp.h"
2 #include <time.h>

3 int
4 main(int argc, char **argv)
5 {
6     int listenfd, connfd;
7     struct sockaddr_in servaddr;
8     char buff[MAXLINE];
9     time_t ticks;

10    listenfd = Socket(AF_INET, SOCK_STREAM, 0);

11    bzero(&servaddr, sizeof(servaddr));
12    servaddr.sin_family      = AF_INET;
13    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
14    servaddr.sin_port        = htons(13);           /* daytime server */

15    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
16    Listen(listenfd, LISTENQ);

17    for ( ; ; ) {
18        connfd = Accept(listenfd, (SA *) NULL, NULL);

19        ticks = time(NULL);
20        sprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
21        Write(connfd, buff, strlen(buff));

22        Close(connfd);
23    }
24 }
```



# IT3010

## Network Design and Management

### Lecture 08

#### Socket Functions

## Shashika Lokuliyan

Faculty of Computing  
Department of CSE



# SLIIT

*Discover Your Future<sup>1</sup>*

- ❖ During this lecture, we will learn TCP sockets

- Introduction to sockets

- TCP sockets

- Various functions related sockets

- Simple Server Design with TCP sockets

- Simple Client Design with TCP sockets

- ❖ we will also learn UDP sockets

- Introduction UDP sockets

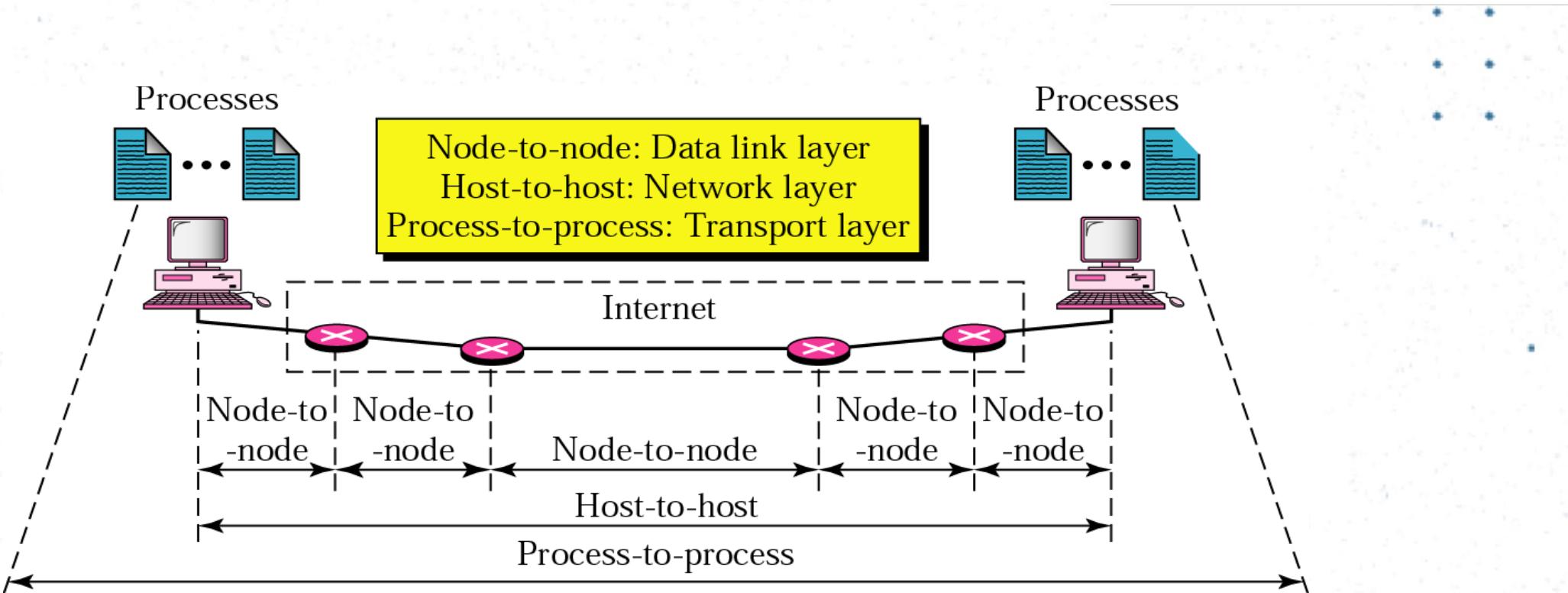
- Differences between TCP and UDP

- Applications of UDP

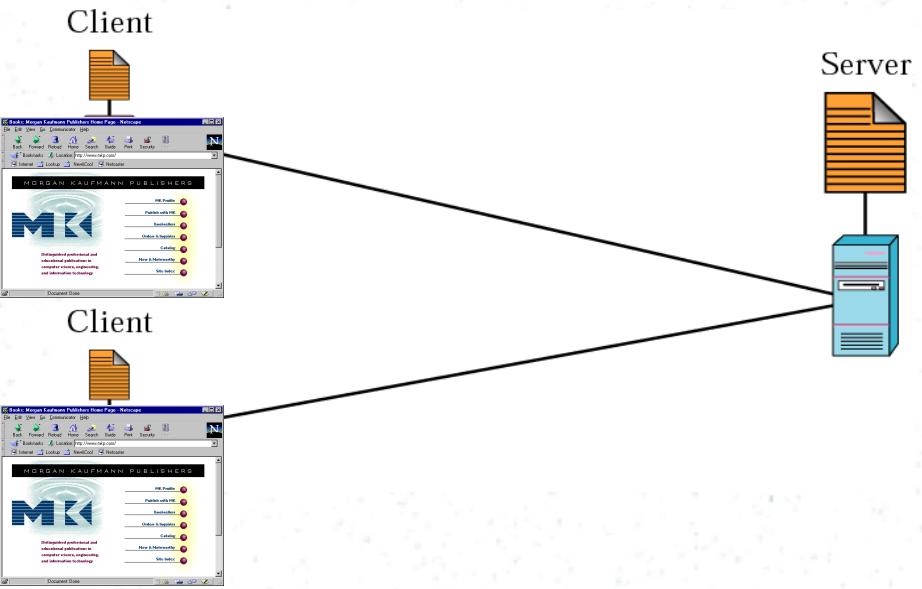
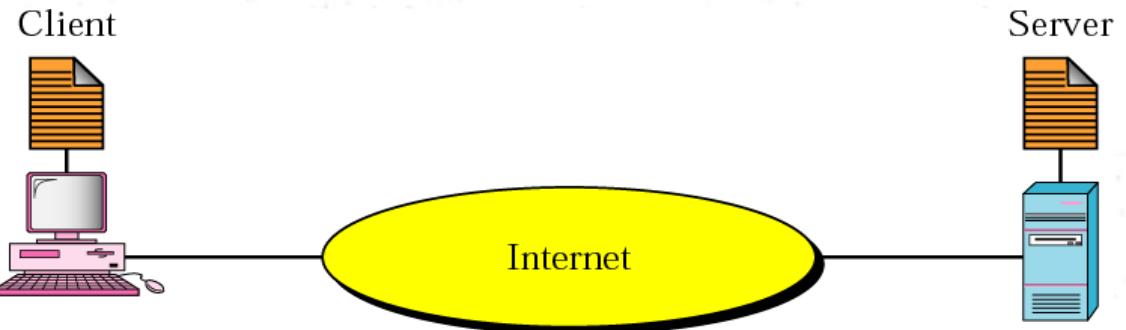
- Socket functions for Network programming using UDP

- Examples of Simple UDP Client and Server

# Types of data deliveries

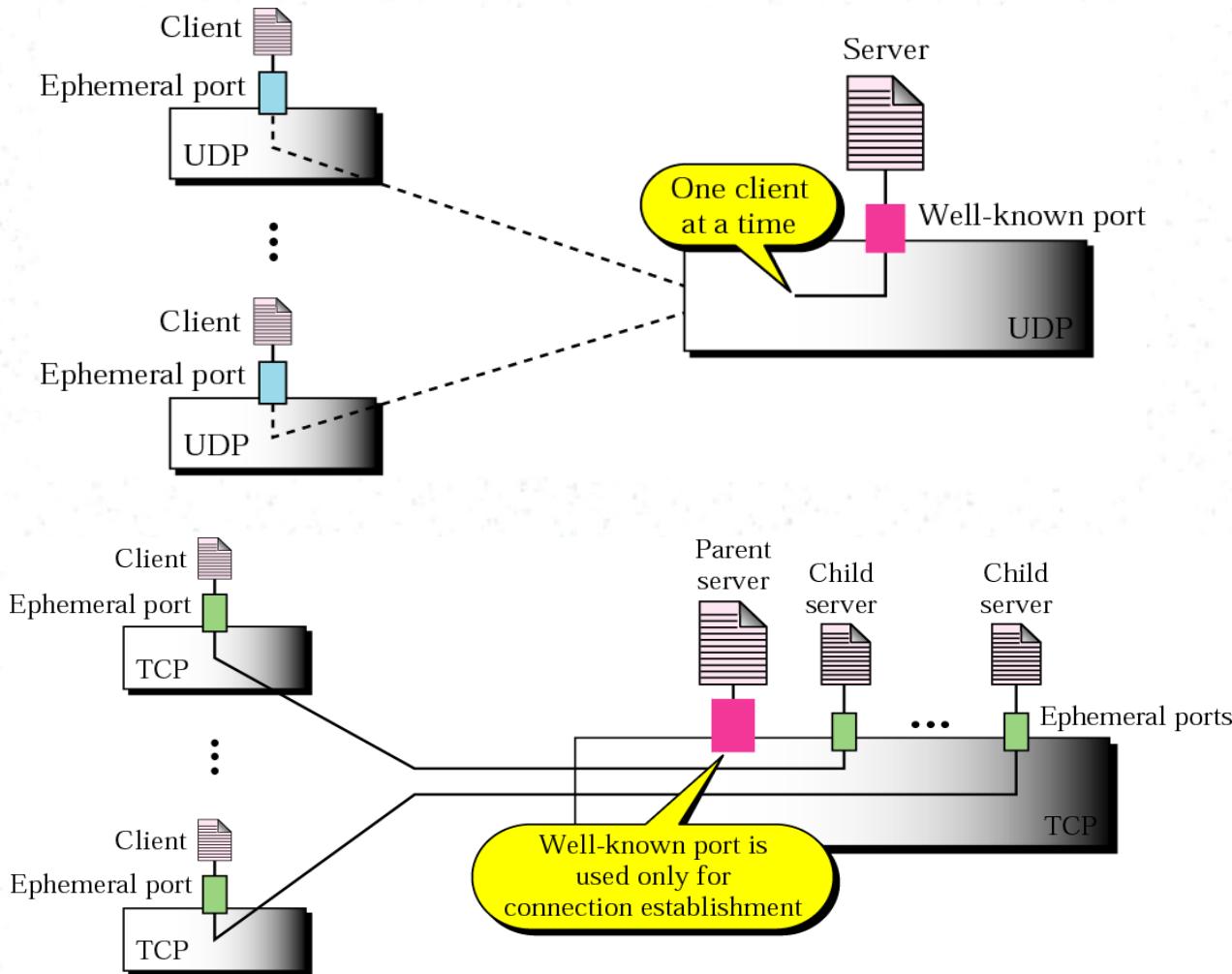


# Client-server model and Process-Process Interaction



- Client-Server interaction is a process-to-process interaction.
- Server could be a webserver, a mailserver, a database server, ftp server, telent server, ssh server etc.

# Well known services use well known ports



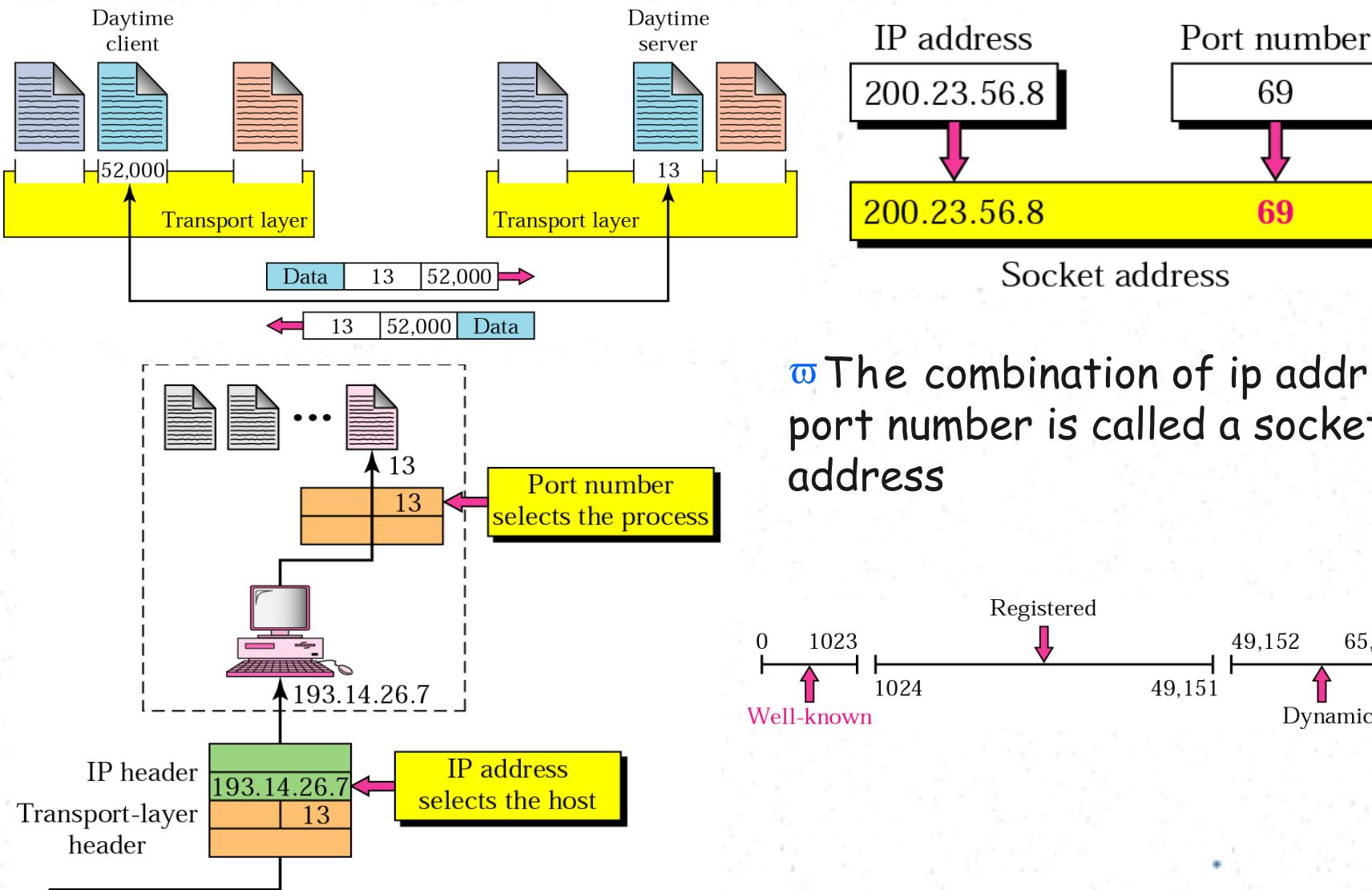
- Services could be supported by TCP or UDP, or others.
- Well known services use well known port numbers to be well identified.
- Servers can also use short lived ports to support concurrency
- Clients use temporary ports.

# /etc/services

☞ Port numbers of some well known services.

<i>echo</i>	<i>7/tcp</i>	
<i>echo</i>	<i>7/udp</i>	
<i>systat</i>	<i>11/tcp</i>	<i>users</i>
<i>daytime</i>	<i>13/tcp</i>	
<i>daytime</i>	<i>13/udp</i>	
<i>netstat</i>	<i>15/tcp</i>	
<i>ftp-data</i>	<i>20/tcp</i>	
<i>ftp</i>	<i>21/tcp</i>	
<i>ssh</i>	<i>22/tcp</i>	<i># Secure Shell</i>
<i>telnet</i>	<i>23/tcp</i>	
<i>smtp</i>	<i>25/tcp</i>	<i>mail</i>

# Port Numbers, IP & Socket Address



☞ The combination of ip addr and port number is called a socket address

# TCP Client/Server Interaction

Server starts by getting ready to receive client connections...

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int socket(int family, int type, int protocol);  
/* Create socket for incoming connections */  
if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)  
    DieWithError("socket() failed");  
    . . .
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:  
 . . .
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t  
addrlen);  
  
echoServAddr.sin_family = AF_INET; /* Internet address family */  
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any incoming interface */ echoServAddr.sin_port =  
htons(echoServPort); /* Local port */
```

```
if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0) DieWithError("bind()  
failed");
```

Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int listen(int sockfd, int backlog);
```

```
/* Mark the socket so it will listen for incoming connections */ if  
(listen(servSock, MAXPENDING) < 0)  
    DieWithError("listen() failed");
```

## Client

1. Create a TCP socket
  2. Establish connection
  3. Communicate
  4. Close the connection

## Server

1. Create a TCP socket
  2. Bind socket to a port
  3. Set socket to listen
  4. Repeatedly:
    - a. Accept new connection
    - b. Communicate
    - c. Close the connection

# TCP Client/Server Interaction

```
int accept(int sockfd,  
          struct sockaddr* cliaddr, int *addrlen);  
for (;;) /* Run forever */  
{  
    clntLen = sizeof(echoClntAddr);  
  
    if ((clntSock=accept(servSock,(struct sockaddr *)&echoClntAddr,&clntLen)) < 0) DieWithError("accept()  
        failed");
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

Server is now blocked waiting for connection from a client

Later, a client decides to talk to the server...

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
/* Create a reliable, stream socket using TCP */  
if ((sock = socket(PF_INET, SOCK_STREAM,  
IPPROTO_TCP)) < 0)  
    DieWithError("socket() failed");
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int connect(int sockfd, const struct sockaddr*  
servaddr, int addrlen);  
    echoServAddr.sin_family      = AF_INET;          /* Internet address family */  
    echoServAddr.sin_addr.s_addr = inet_addr(servIP); /* Server IP address */  
    echoServAddr.sin_port       = htons(echoServPort); /* Server port */  
  
if (connect(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)  
    DieWithError("connect() failed");
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
if ((clntSock=accept(servSock,(struct sockaddr  
*)&echoClntAddr,&clntLen)) < 0)  
    DieWithError("accept() failed");  
    . . . . .
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int send(SOCKET sockfd,  
        const char* buf, int buflen,int flags);  
  
echoStringLen = strlen(echoString); /* Determine input length */  
  
/* Send the string to the server */  
if (send(sock, echoString, echoStringLen, 0) != echoStringLen)  
    DieWithError("send() sent a different number of bytes than  
expected");
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int recv(SOCKET sockfd,  
        char* buf, int len,int flags);  
  
/* Receive message from client */  
if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)  
    DieWithError("recv() failed");  
    . . . . .
```

## Client

1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection

# TCP Client/Server Interaction

```
int close(int sockfd);
```

close(sock);

close(clntSocket)

⋮  
⋮  
⋮  
⋮

## Client

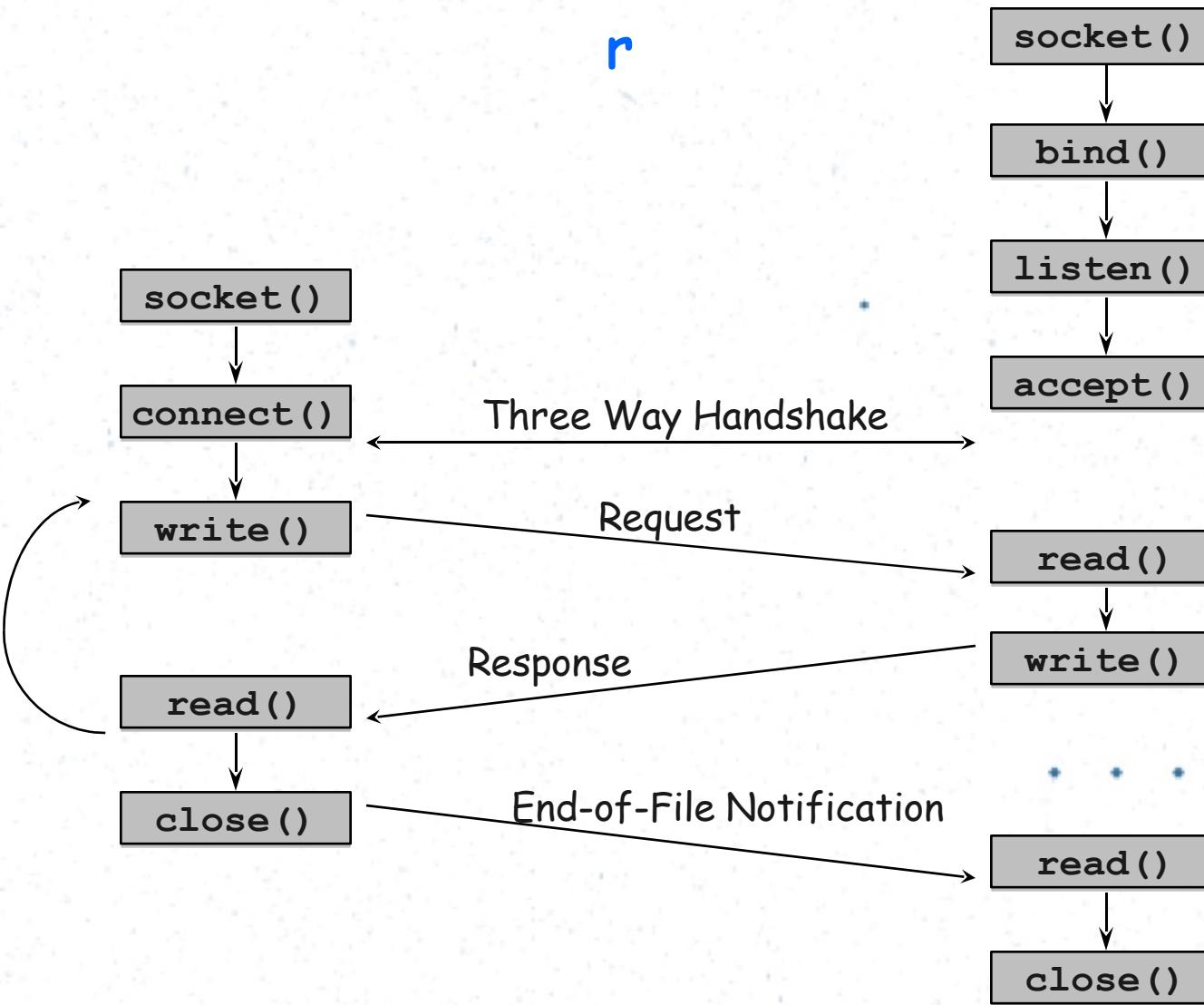
1. Create a TCP socket
2. Establish connection
3. Communicate
4. Close the connection

## Server

1. Create a TCP socket
2. Bind socket to a port
3. Set socket to listen
4. Repeatedly:
  - a. Accept new connection
  - b. Communicate
  - c. Close the connection



# Summary of Socket Calls For a TCP Client & Server



# socket()

- ❖ Creates an endpoint for communication

```
int socket(int family, int type, int protocol);
```

♣ *family*: specifies the protocol/address family for the socket:

- Typically, AF\_INET or AF\_UNIX (AF\_LOCAL)
- AF\_INET6 (for IPv6), AF\_X25 (X.25 Protocols), AF\_ATMPVC (ATM Private Virtual Circuits), AF\_PACKET (low level packet communications)
- AF\_ROUTE (access to routing tables), AF\_KEY (new, for encryption)
- There are more.

♣ *type*: specifies the type of socket:

- Typically, SOCK\_STREAM (reliable, connection oriented bytestream); or
- SOCK\_DGRAM (unreliable, connectionless datagrams)
- SOCK\_RAW (access to raw network protocols), and there are more.

♣ *protocol*: which protocol to use within the family, most often set to 0 (default protocol)

- IPPROTO\_TCP for SOCK\_STREAM, or IPPROTO\_UDP for SOCK\_DGRAM

- ❖ Returns a 'socket descriptor' if successful, or -1/INVALID\_SOCKET on failure

# bind()

- ❖ Assigns a local protocol address (name) to a socket

```
int bind(int sockfd, const struct sockaddr  
*myaddr, socklen_t addrlen);
```

- ♣ sockfd : is socket descriptor from socket ()
- ♣ myaddr: protocol-specific structure containing the local address information. It is a pointer to address struct with:
  - port number and IP address
  - if port is 0, then host will pick ephemeral port not usually for server (exception RPC port-map)
  - IP address = INADDR\_ANY, indicates we want to bind to all of arlocal host's addresses.
- ♣ addrlen: the size of the structure pointed to by myaddr
- ❖ Returns 0 on success or -1/ SOCKET \_ERROR if it fails.

# listen()

- ❖ Makes a socket 'listen' for new connections

```
int listen(int sockfd, int backlog);
```

*sockfd*: *sockfd* is socket descriptor from `socket()`

*backlog*: specifies the length of the queue of waiting connections that the kernel should maintain. *backlog* is maximum number of *incomplete* connections

- historically 5
- rarely above 15 on a even moderate Web server!

- ❖ Returns 0 on success or -1/`SOCKET_ERROR` on failure

# accept()

- ❖ Accepts new connections from a listening socket

```
int accept(int sockfd,  
          struct sockaddr* cliaddr, int  
          *addrlen);
```

*sockfd*: *sockfd* is socket descriptor from `socket()`

*cliaddr*: protocol-specific structure containing the remote address information

*addrlen*: a pointer to the size of the structure pointed to by *cliaddr*, altered to indicate the size used

- ❖ Returns a new socket descriptor on success or `-1/INVALID_SOCKET` on failure

# connect()

- ❖ Used by TCP clients to establish a connection to a 'listening' socket

```
int connect(int sockfd, const struct sockaddr*  
servaddr, int addrlen);
```

♣ *sockfd*: *sockfd* is socket descriptor from `socket()`

♣ *servaddr*: protocol-specific structure containing the local address information.

♣ *servaddr* is a pointer to a structure with:

➢ port number and IP address

♣ *addrlen*: the size of the structure pointed to by *servaddr*

- ❖ client doesn't need `bind()`

OS will pick  
ephemeral port

- ❖ Returns 0 on success or -1/ `SOCKET_ERROR` on failure

# Thank you



# IT3010

## Network Design and Management

### Lecture 09

#### Concurrent Servers

## Shashika Lokuliyan

Faculty of Computing  
Department of CSE



# SLIIT

*Discover Your Future<sup>1</sup>*

## References:

1. Stevens, Fenner, Rudoff, *UNIX Network Programming*, vol. 1, Chapter 4.

⋮  
⋮  
⋮  
⋮  
⋮

## Objective:

To discuss:

- Socket functions
- Concurrent servers using fork().



# Concurrent Servers

- A server is a process that is waiting to be contacted by a client process so that the server can do something for the client.
- Two types of server processes:
  - (1) *Iterative servers*.

When a client's request can be handled by the server in a known, short amount of time, the server process handles the request itself
    - like daytime server.
  - (2) *Concurrent servers*.

When the amount of time to service a request depends on the request itself (the server does not know ahead of time how much effort it takes to handle each request), the server typically handles it in a concurrent fashion;
    - Otherwise the server may be tied up with one client → better to handle multiple clients at the same time.
- The simplest way to write a *concurrent server* under UNIX is to `fork()` a child process to handle each client.
  - The original server process can go back to sleep, waiting for the next client request.
  - Other way is using *thread* (discussed later)

# fork() function

```
#include <unistd.h>
/*returns 0 in child, process ID of child in parent, -1 on error */
pid_t fork (void);
```

- `fork()` is used in UNIX to create a new process → a copy of the process that was executing.
  - The process that executes `fork()` is called the *parent process*, and the new process is called the *child process*.
  - A process terminates when it executes last statement and asks the OS to delete it (`exit`).
  - At that point, the process may output data from child to parent (via `wait()`), and process' resources are deallocated by the OS.
  - Parent may also terminate execution of children processes (`abort()`).
- `fork()` is called once but returns two different values:
  - Returns with *value 0 (zero) in child*.
  - Returns with *process ID of the child in parent*.

# fork() function (cont.)

- If the child wants to know the PID of its parent, calls `getppid()`.
- If the parent wants to keep track of the PID of its children, records them.
- Parent can `wait()` for child to terminate and fetch its termination status.
- *All open descriptors in parent are shared* with child after `fork()` returns.
- Two typical uses of `fork()`:
  - A process makes copy of itself so that the copies do different tasks.
    - Typical for network servers.
  - A process wants to execute another program.
    - The process calls `fork()` to make a copy of itself, then calls `exec()` to replace itself with the new program.
    - Typical for programs such as shells.

# exec()

The only way in which an executable program file on disk can be executed by Unix is for an existing process to call one of the six exec functions. (We will often refer generically to "the exec function" when it does not matter which of the six is called.) exec replaces the current process image with the new program file, and this new program normally starts at the main function. The process ID does not change. We refer to the process that calls exec as the *calling process* and the newly executed program as the *new program*.

# fork() function (cont.)

## UNIX example

```
printf("one\n");
pid = fork ();
printf ("two\n");
```

PC

fork

```
printf("one\n");
pid = fork ();
printf ("two\n");
```

PC

```
if ( (pid = fork() ) == 0) {
    // Child process will enter here
} else (if pid > 0) {
    // Parent process will execute here
} else
    // fork() error
```

before

after

```
printf("one\n");
pid = fork ();
printf ("two\n");
```

PC

# Concurrent Servers (cont.)

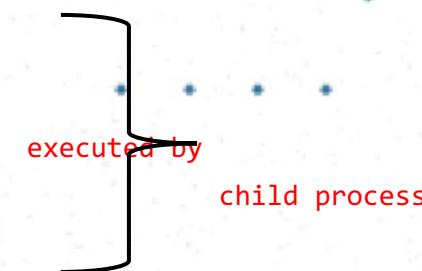
## Basic steps in concurrent server:

- When a connection is established → accept() returns.
- The server calls fork():
  - The child process serves the client (on the *connected socket*).
  - The parent waits for another connection (on the *listening socket*).
- The child that handles the server closes the *connected socket*.

## Outline for typical concurrent server:

```
pid_t pid;
int listenfd, connfd;
listenfd = Socket (...);

* * *
/* fill in sockaddr_in {} with server's well-known port */
Bind(listenfd, .. );
Listen(listenfd, LISTENQ);
for ( ; ; ) {
    Connfd = Accept(listenfd, .. ); /* probably block */
    if ( (pid = Fork() ) == 0) {
        Close(listenfd); /*child closes listening socket*/
        doit (connfd); /*process the request*/
        close (connfd); /*done with this client*/
        Exit(0); /*child terminates*/
    }
    Close(connfd); /*parent closes connected socket*/
}
```

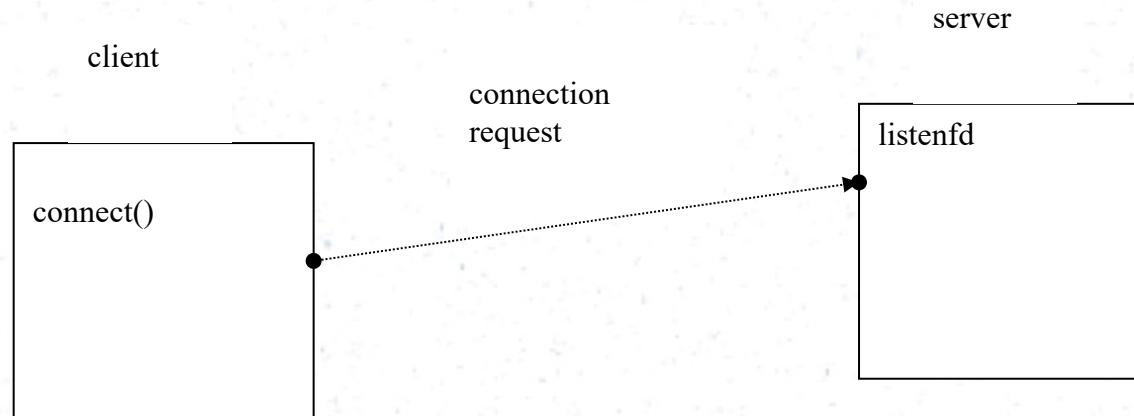


# Descriptor Reference Counts

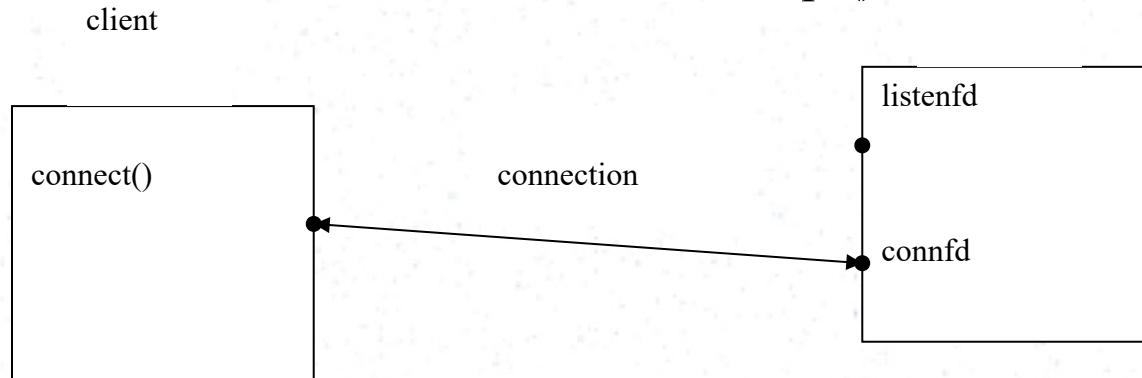
- A reference count is a total number of descriptors that are currently open that refer to this socket
  - This count is maintained in the file table entry.
  - listenfd: has reference count 1 after socket() returns.
  - connfd: has reference count 1 after accept() returns.
  - fork(): both descriptors are duplicated → both reference counts become 2.
  - close (fd): decrement reference count for fd by one.
- A real close on the descriptor takes place after reference count reaches 0.
  - This initiates TCP's four-packet connection termination sequence.
- If a server does not close each *connected socket* returned by accept():
  - The parent may run out of descriptors.
  - None of the client connections will be terminated, although the child has called close().

# Concurrent Servers (cont.)

Status of client-server before call to accept() returns:

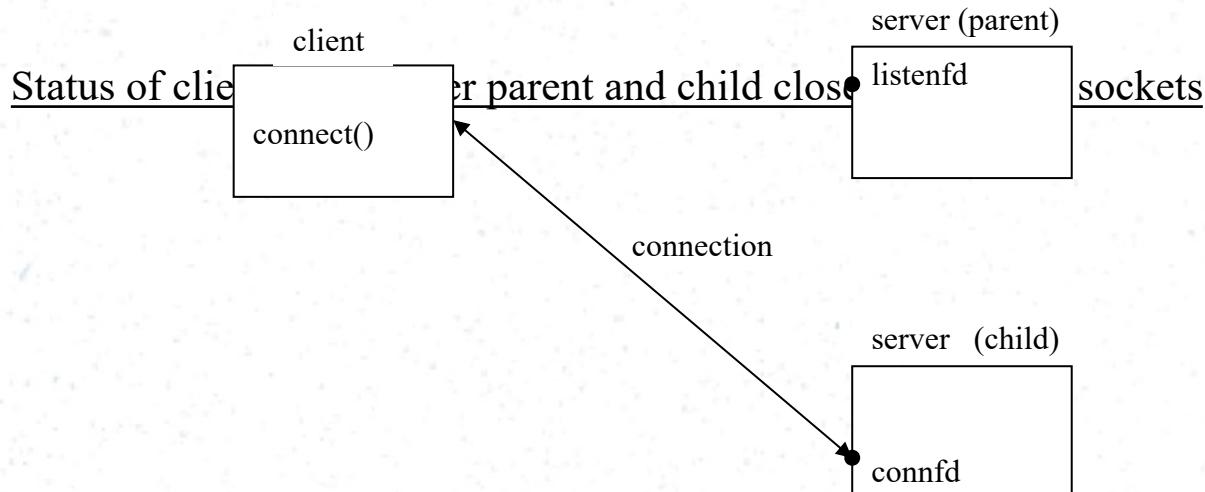
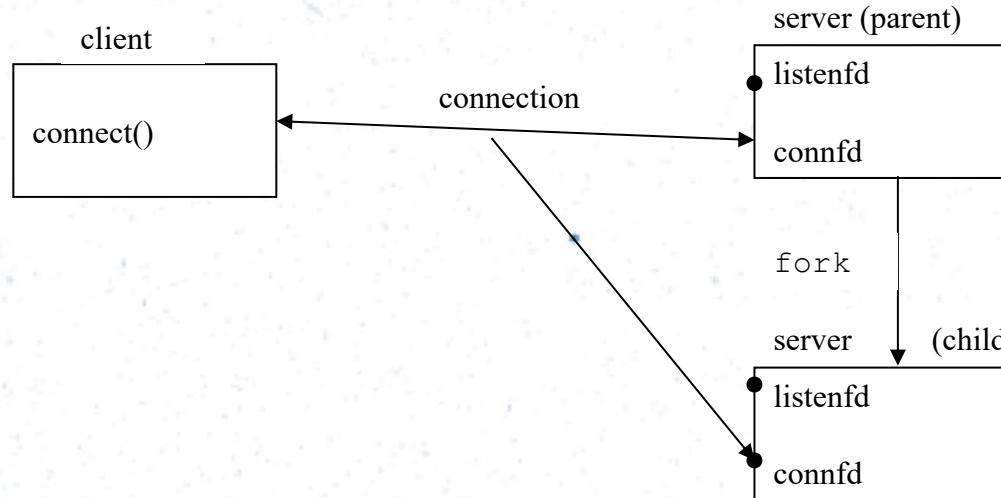


Status of client-server after return from accept():



# Concurrent Servers (cont.)

Status of client-server after fork() returns:

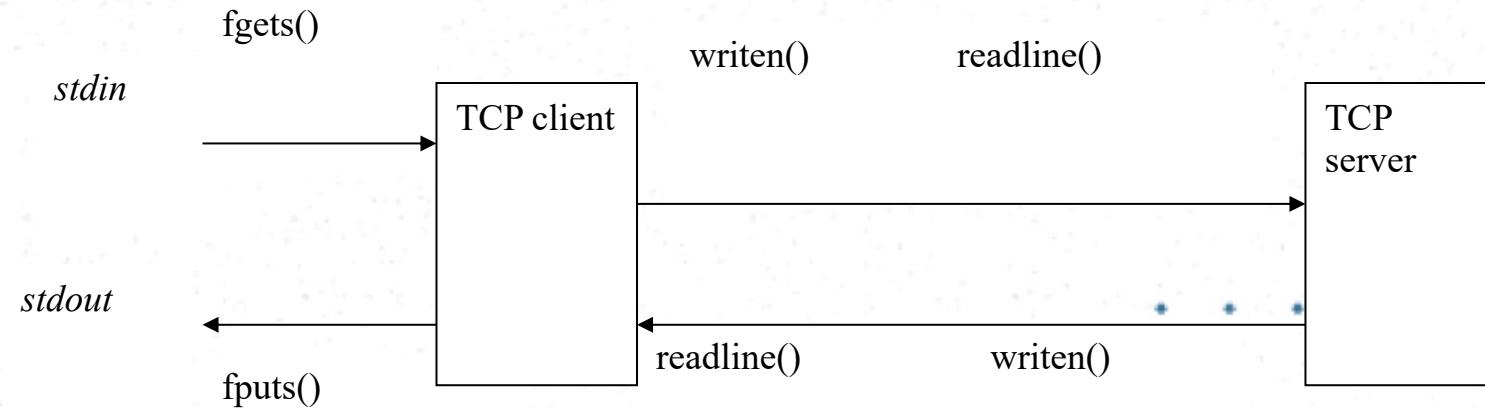


# TCP Client-Server Example

## Simple echo client and server

Perform the following steps:

1. The client reads a line of text from its standard input and writes the line to the server.
2. The server reads the line from its network input and echoes the line back to the client.
3. The client reads the echoed line and prints it on its standard output.



# TCP echo Server

```
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int listenfd, connfd;
6     pid_t childpid;
7     socklen_t clilen;
8     struct sockaddr_in cliaddr, servaddr;
9
10    listenfd = Socket(AF_INET, SOCK_STREAM, 0);
11    bzero(&servaddr, sizeof(servaddr));
12    servaddr.sin_family      = AF_INET;
13    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
14    servaddr.sin_port        = htons(SERV_PORT);
15    Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
16    Listen(listenfd, LISTENQ);
17
18    for ( ; ; ) {
19        clilen = sizeof(cliaddr);
20        connfd = Accept(listenfd, (SA *) &cliaddr, &clilen);
21        if ( (childpid = Fork()) == 0) { /* child process */
22            Close(listenfd); /* close listening socket */
23            str_echo(connfd); /* process the request */
24            exit(0);
25        }
26    }
27 }
```

Creates an endpoint for communication and returns a descriptor.

Function sets the entire servaddr socket address structure object to zero.

IPv4 address family

The socket will be bound to all local interfaces

When a socket is created with *socket()*, it exists in a name space (address family) but has no address assigned to it. *bind()* assigns the address specified by *servaddr* to the socket referred to by the file descriptor *listenfd*

Marks the socket referred to by *listenfd* as a passive socket, that is, as a socket that will be used to accept incoming connection requests using *accept()*.

Used to connect the server with incoming client connections.

Child process execution

# Fork()

```
#include <unistd.h>

pid_t fork(void);
```

- Creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent, except for few options.

- ❖ The **child has its own unique process ID**, and this PID does not match the ID of any existing process group (*setpgid()*).
  - ❖ The **child inherits copies of the parent's set of open file descriptors**.

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created, and errno is set appropriately.

## TCP echo Server (cont.)

- Line 12: We specify the Internet address for the server's bind() as the constant INADDR\_ANY → tells the system that we will accept() a connection destined for any local interface, in case the system is multihomed.
  - Line 13: The choice of TCP port number is arbitrary, must not conflict with any other server's port.
  - Notice the sequence: socket(), bind(), listen(), and accept() calls.
  - In concurrent server, notice that the parent closes the *connected* socket, and the child closes the *listening* socket.

# TCP echo Server (cont.)

## str\_echo function: echo lines on a socket

```
1 #include "unp.h"

2 void
3 str_echo(int sockfd)
4 {
5     ssize_t n;
6     char line[MAXLINE];
7
8     for ( ; ; ) {
9         if ( (n = readline(sockfd, line, MAXLINE)) == 0)
10            return; /* connection closed by other end */
11
12 }
```

*ssize\_t readline(int fd, void \*buf, size\_t count);*

Attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*. On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. If the client closes the connection the receipt of the client's FIN causes the child's *read()* to return 0.

*ssize\_t writen(int fd, const void \*buf, size\_t count);*

Writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

- In the str\_echo() function, the readline() function returns 0 if the client closes its connection (the server will receive a FIN).

# TCP echo client

```
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int sockfd;
6     struct sockaddr_in servaddr;
7
8     if (argc != 2)
9         err_quit("usage: tcpcli <IPaddress>");
10
11    sockfd = Socket(AF_INET, SOCK_STREAM, 0);
12
13    bzero(&servaddr, sizeof(servaddr));
14    servaddr.sin_family = AF_INET;
15    servaddr.sin_port = htons(SERV_PORT);
16    inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
17
18    connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
19
20    str_cli(stdin, sockfd); /* client function*/
21    exit(0);
22 }
```

# TCP echo client (cont.)

## str\_cli function: client processing loop

```
1      #include      "unp.h"
2
3      void
4      str_cli(FILE *fp, int sockfd)
5      {
6          char      sendline[MAXLINE], recvline[MAXLINE];
7
8          while(fgets(sendline, MAXLINE, fp) != NULL) {
9              writen(sockfd, sendline, strlen(sendline));
10             if (readline(sockfd, recvline, MAXLINE) == 0)
11                 err_quit("str_cli: server terminated prematurely");
12         }
13     }
```

*char \*fgets(char \*s, int size, FILE \*stream)*

Reads in at most one less than *size* characters from *stream* and stores them into the buffer pointed to by *s*. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte (aq\0aq) is stored after the last character in the buffer.

Sends the line to the server

Reads the line echoed back from the server

*int fputs(const char \*s, FILE \*stream);*

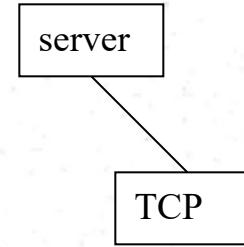
Writes the string *s* to *stream*, without its terminating null byte (aq\0aq).

- Fgets() wrapper function checks for an error, and aborts if one occurs.
- Fgets() returns NULL only when EOF is encountered.

# TCP Example – Normal Condition

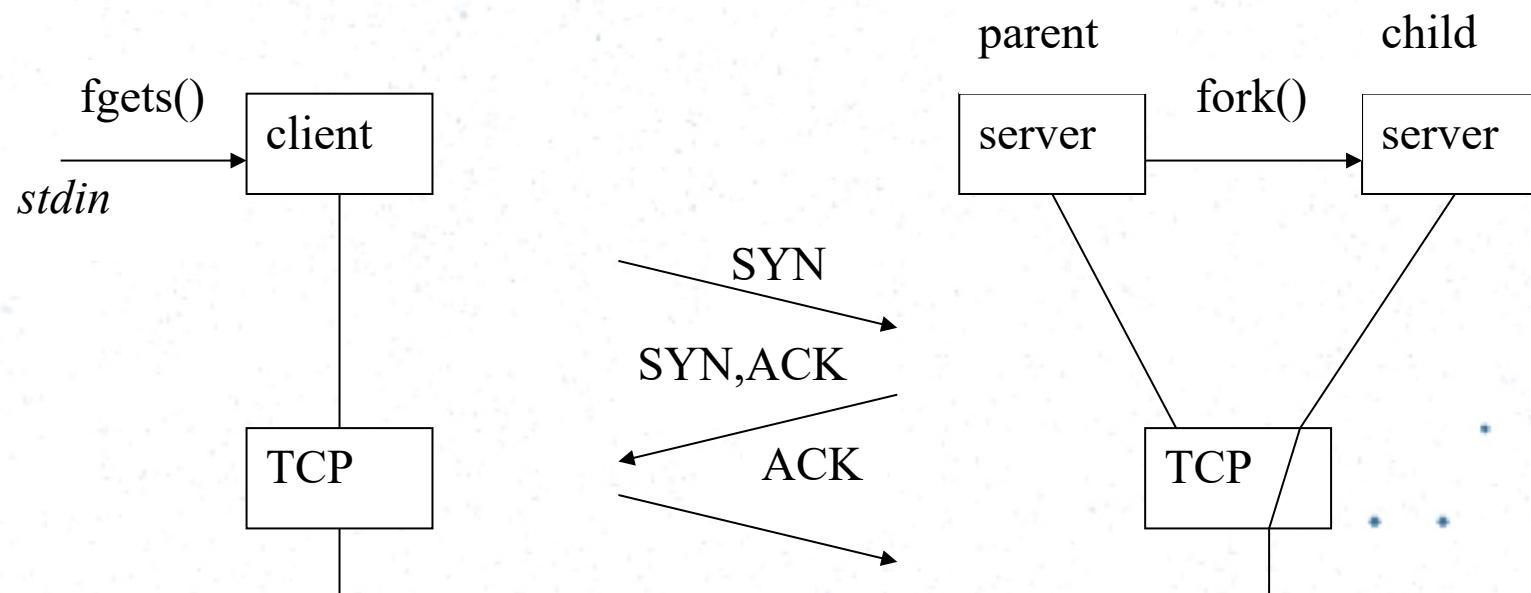
## Normal Startup:

- (1) Run the server on the background.
  - Server starts: socket(), bind(), listen(), accept().
  - The server blocks in the call to accept().
- (2) Run the client (on the same host).
  - client starts: socket(), connect().
    - connect() causes a 3-way handshake.
    - 3-way handshake completed → the server's accept() returns, and connection is established.
- (3) The following steps, then take place:
  - Client calls str\_cli(), and blocks in a call to fgets() → returns after we type in a line input.
  - When accept() returns, server calls fork().
  - The child server calls str\_echo(), and blocks in a call to read() (inside readline() ) → returns after a line is sent from the client.
  - The parent server calls accept() again, and blocks, waiting for the next client connection.
  - At this point, we have three blocked processes: parent server process blocks in accept(), child server process blocks in read(), and client process waiting for input (from keyboard).



## Normal Condition (cont.)

- `connect()` returns after the second segment of the 3-way handshake is received by the client.
- `accept()` returns only after the third segment of the handshake is received by the server → one-half of the RTT after `connect()` returns.



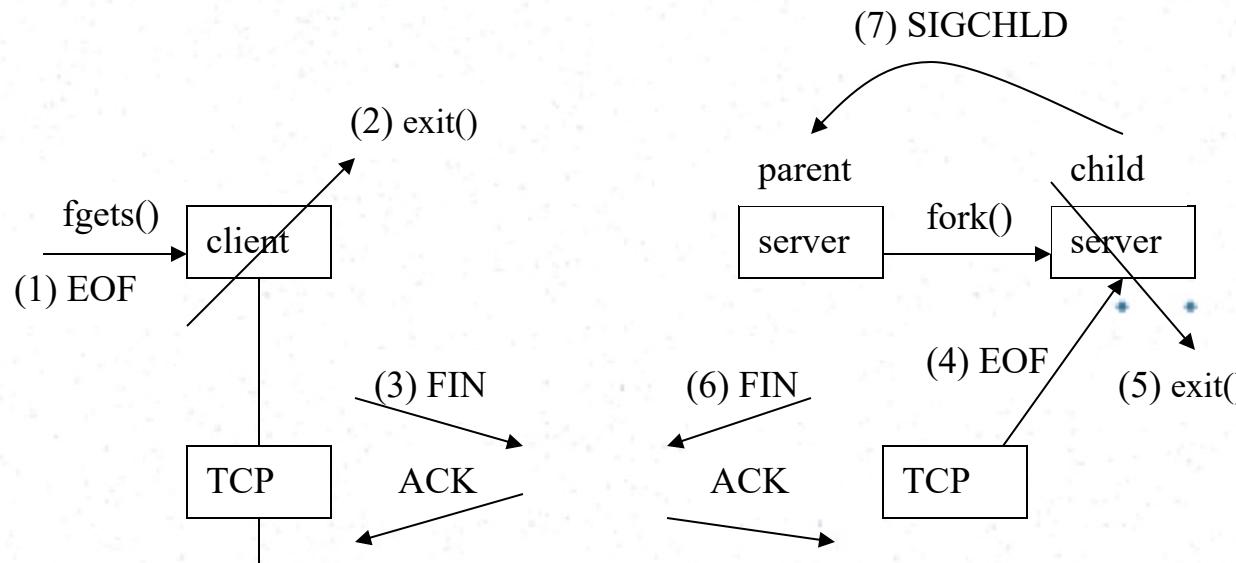
# Normal Termination

- (1) We type our EOF character (Control-D) to the client's standard input.  
fgets() returns NULL in str\_cli(), and str\_cli() returns to caller (main function of TCP client).
- (2) TCP client calls exit (0) to terminate.  
Client's socket is automatically closed by kernel when client process terminates.
- (3) Client's TCP sends FIN, server's TCP responds with ACK.
- (4) The readline() in the server gets an EOF on its read(), and readline() returns to its caller, str\_echo(). This returns to the main function of TCP server (child process).
- (5) The child calls exit (0) to terminate.  
Child's socket is automatically closed by kernel when child process terminates.
- (6) Server's TCP sends FIN, client TCP responds with ACK.

## Normal Termination (cont.)

(7) The **SIGCHLD** signal is sent to parent.

- Since parent does not catch this signal (in this program), the default action is for the signal to be ignored.
- Child becomes a zombie process and remains so until parent terminates.
  - Checked with **ps** command.
- When the parent terminates, the zombie process is inherited by **init** (kernel process) which will clean it up.
- NOTE: the parent process should handle the **SIGCHLD** signal.



# Posix Signal Handling

"Portable Operating System Interface", is a family of standards specified by the IEEE for maintaining compatibility between operating systems.

- A *signal* is a notification to a process that an event has occurred.
  - Also called *software interrupt*;
  - A signal occurs *asynchronously* (process does not know ahead of time exactly when a signal will occur.)
- Signals can be sent:
  - By one process to another process (or to itself)
  - By the kernel to a process.
- **SIGCHLD** signal is sent by the kernel to the parent of the terminating process.
- Every signal has a *disposition* (*action* associated with the signal)
  - Set by `sigaction()` function.

# Posix Signal Handling

- There are three choices for *disposition*:
  - We provide a *signal handler* function that is called whenever a specific signal is caught.
    - `void handler (int signo)`
    - Note: **SIGKILL** and **SIGSTOP** can not be caught.
  - We can *ignore* a signal by setting its disposition to **SIG\_IGN**.
    - Example: `signal (signo, SIG_IGN)`.
    - Note: **SIGKILL** and **SIGSTOP** can not be ignored.
  - We can set the *default* disposition for a signal by setting its disposition to **SIG\_DFL**.
    - Normally to terminate a process on receipt of a signal.
- Disposition for a signal can be established either by using **signal()** function or **sigaction()** function.
  - **signal()** is a historical function that predates Posix.1.
  - **sigaction()** is a Posix standard.

# signal () function

```
1 #include "unp.h"
2 Sigfunc *
3 signal (int signo, Sigfunc *func)
4 {
5     struct sigaction act, oact;
6     act.sa_handler = func;
7     sigemptyset(&act.sa_mask);
8     act.sa_flags = 0;
9     if (sigaction(signo, &act, &oact) < 0)
10    return(SIG_ERR);
11    return(oact.sa_handler);
12 }/* end signal */
```

Defined as `typedef void Sigfunc(int);`  
stating that signal handlers are functions with an integer argument and  
the function returns nothing (void)..

signal(signal name, signal handler)

`sa_handler` is a member of the `sigaction` structure is set too the `func` argument.

No additional signals will be blocked while the signal handler is running.

When the flag is set, a system call interrupted by this signal will be automatically restarted by kernel.

Call `sigaction` and then return the old action for the signal as the return value of the `signal` function.

An easier way to set the disposition of a signal is to call the signal function. The first argument is the `signal name` and the second argument is either a pointer to a function or one of the constant `SIG_ING` or `SIG_DEF`. Different implementations provide different signal semantics when it is called, providing backward compatibility, whereas POSIX explicitly spells out the semantics when `sigaction` is called.

## Example on establishing signal handler

```
/* called before we fork() Just after the listen( ) function
call*/
```

```
Signal (SIGCHLD, sig_chld);
```

It must be done sometime before we fork the first child and needs to be done only once.

## **sigaction()**

```
int sigaction(int signum, const struct sigaction *act, struct sigaction  
*oldact);
```

The **sigaction()** system call is used to change the action taken by a process on receipt of a specific signal.

If *act* is non-NULL, the new action for signal signum is installed from act.  
If *oldact* is non-NULL, the previous action is saved in oldact.

The sigaction structure is defined as something like:

```
struct sigaction {  
    void        (*sa_handler)(int);  
    void        (*sa_sigaction)(int, siginfo_t *, void *);  
    sigset_t    sa_mask;  
    int         sa_flags;  
    void        (*sa_restorer)(void);  
};
```

## Signal semantic for Posix-compliant system

- 1) Once a signal handler is installed, it remains installed.
- 2) Once a signal handler is executing, the signal being delivered is blocked.
- 3) If a signal is generated one or more times while it is blocked, it is normally delivered only one time after the signal is unblocked.
  - By default, **UNIX signals are not queued.**
  - It is possible to selectively block and unblock a set of signals using the **sigprocmask( )** function. Allows to protect a critical region of code by preventing certain signals from being caught earlier than that region of code is executing.

## Handling SIGCHLD Signals

- The purpose of the zombie state is to maintain information about the **child** for the **parent** to fetch at some later time.
  - E.g. the process ID of the child, its termination status and information on resource utilisation of the child.
- If a process terminates, and that process has children in the zombie state, the parent process ID of all the zombie children is set to 1 (the *init* process), which will inherit the children and clean them up.
- We do not want to leave zombies around → they take up spaces in the kernel.
- The parent process must wait for its children to prevent them from becoming zombies.

# Example of SIGCHLD signal handler using wait()

```
1 #include "unp.h"
2 void
3 sig_chld(int signo)
4 {
5     pid_t    pid;
6     int      stat;
7     pid = wait(&stat);
8     printf("child %d terminated\n",pid);
9 }
```

signal handler function example with wait() function.

wait for process to change state

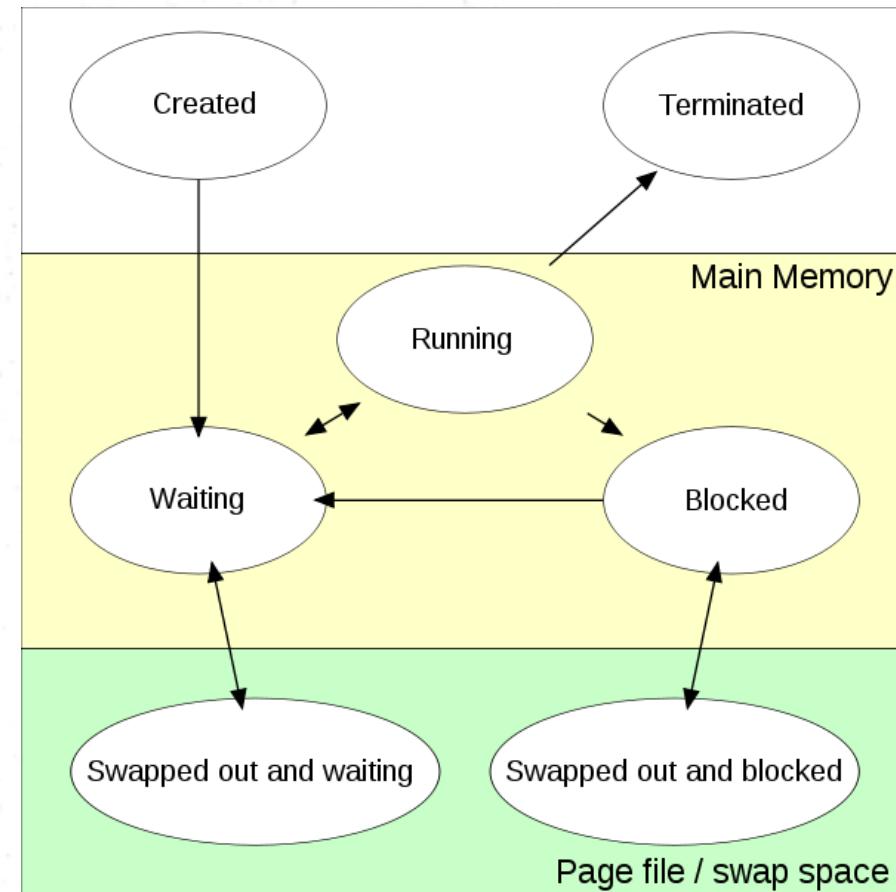
returns the exit status of the child

on success, returns the process ID of the terminated child; on error, -1 is returned.

All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: *the child terminated; the child was stopped by a signal; or the child was resumed by a signal*. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state. If a child has already changed state, then these calls return immediately. Otherwise, they block until either a child changes state or a signal handler interrupts the call.

## State diagram

The various process states, displayed in a state diagram, with arrows indicating possible transitions between states - as can be seen, some processes are stored in main memory, and some are stored in secondary (virtual) memory.



## Termination steps

- The client is terminated by the use of EOF character and the client TCP sends a FIN to the server and it responds with an ACK.
- The receipt of the FIN delivers and EOF to the child's pending `readline` and the child terminates.
- The parent is blocked in its call to accept when the `SIGCHLD` signal is delivered. The `sig_chld` function executes ( our signal handler) , `wait` fetches the child's PID and termination status, and `printf` is called from the signal handler and the signal handler returns
- Since the signal was caught by the parent while the parent was in a slow system call (`accept ()` ), the kernel causes the `accept` to return and an error of `EINTR` (interrupt system call). The parent does not handle this error, so it aborts.

# Handling Interrupted System calls

- *Slow system call*: system call that can block forever, e.g., `accept()`, `read()`, `write()`.
- When a process is blocked in a slow system call, and the process catches a signal and the signal handler returns, the system call can return an error `EINTR`.
- We must handle interrupted system calls.
  - Some kernels automatically restart some interrupted system calls.
  - For portability, we must be prepared for slow system calls to return `EINTR`.
- To handle the interrupted `accept()` ( or `read()`, `write()`, `select()`, and `open()` ), use the following:

```
for ( ; ; ) {
    clilen = sizeof (cliaddr);
    if((connfd=accept(listenfd,(SA *) &cliaddr, &clilen)) < 0) {
        if (errno == EINTR)
            continue;
        else
            err_sys ("accept error");
    }
}
```

- Note, for `connect()`, once it returns error (including `EINTR`), we cannot call it again.
  - You have to `close()` the socket, and create a new socket.

## wait() and waitpid() functions

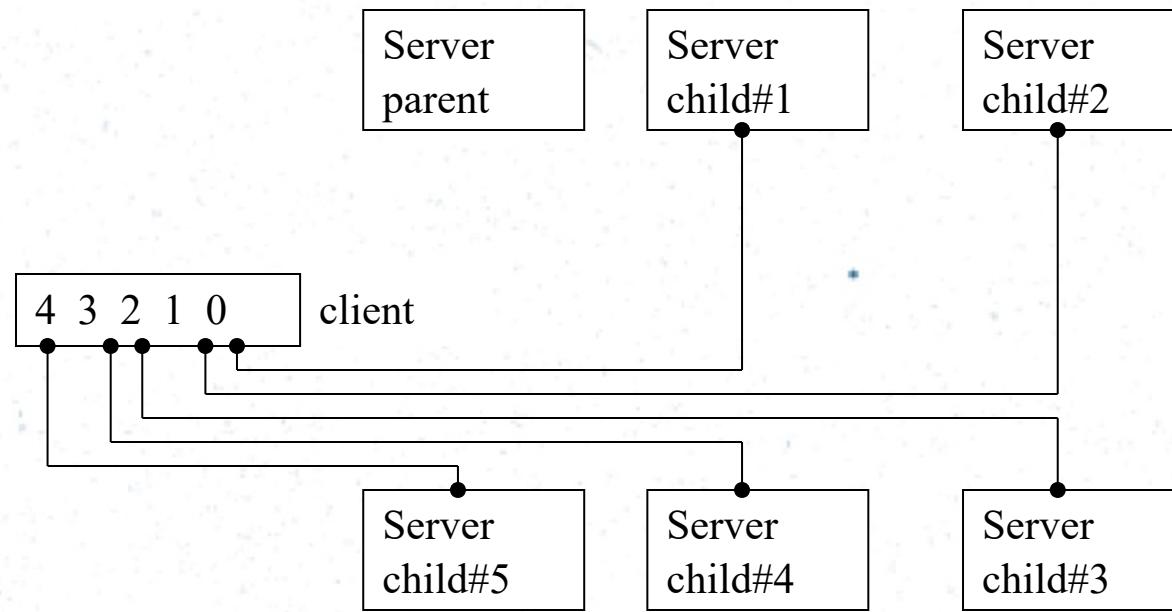
```
#include <sys/wait.h>
/*return process ID if OK, 0 or -1 on error*/
pid_t wait (int *statloc);
pid_t waitpid (pid_t pid, int *statloc, int option);
```

- `wait()` and `waitpid()` both return two values:
  - The process ID of the terminated child (return value of the function).
  - The termination status of the child (an integer) through the `statloc` pointer.
- If there are no terminated children for the process calling `wait()`, but the process has one or more children that are still executing, then `wait()` blocks until the first of the existing children
- `terminates.waitpid()` provides options:
  - Which child process to wait for.
    - $pid > 0$ , wait for a child process whose id is  $pid$ .
    - $pid = -1$  means wait for the first child to terminate like in `wait()`.
  - Whether or not to block.
    - $option = WNOHANG$  means block *only if* there are terminated children.
- Thus, `wait()` waits for *any* process that terminates, while `waitpid()` allows us to specify for *which* process or group.
- `wait()` blocks the calling program when there are some child processes executing, but `waitpid()` (with `WNOHANG` option) returns 0 when there is no terminated child.

## wait() vs. waitpid() – Example / Client Program

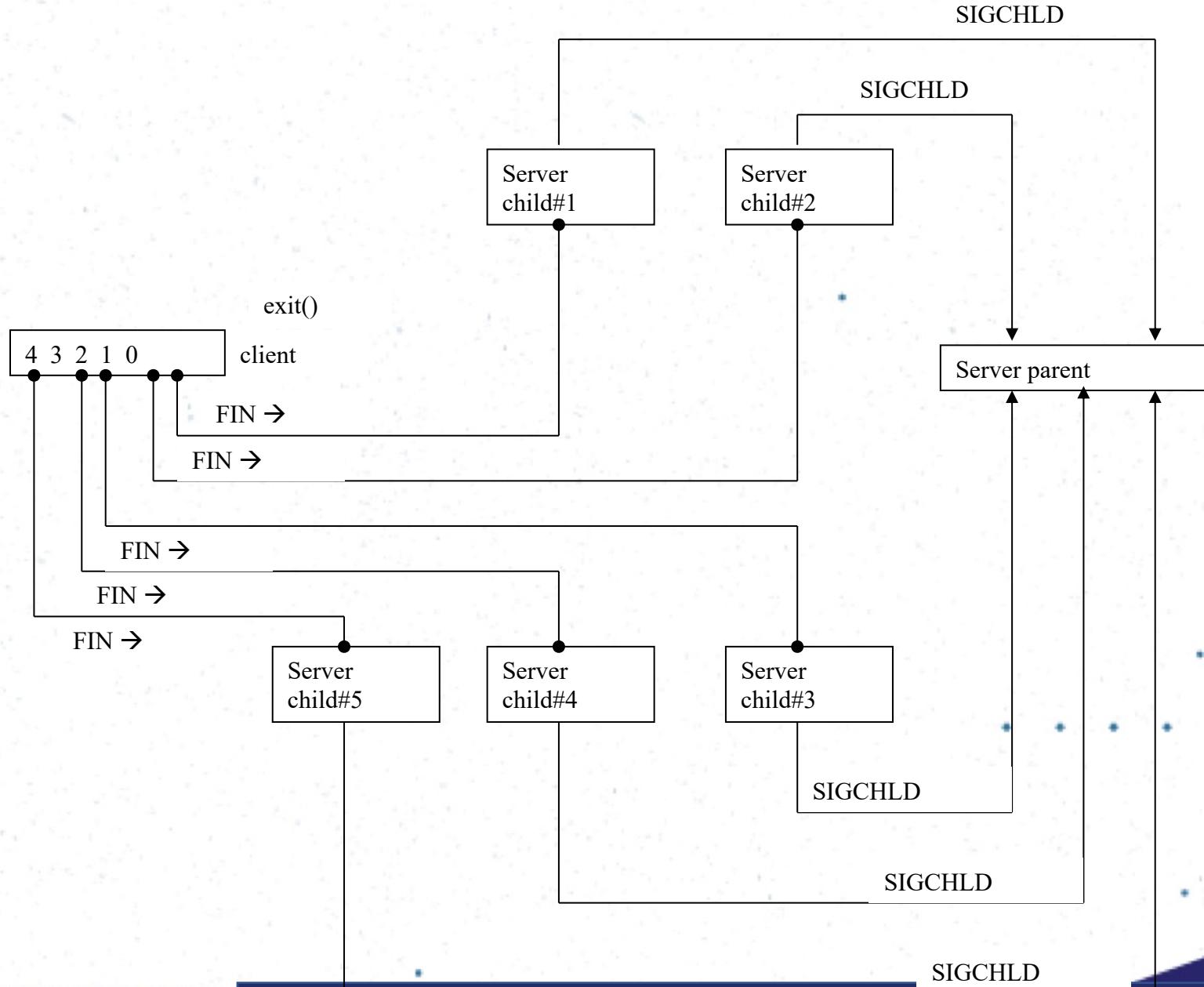
```
1 #include "unp.h"
2 int
3 main(int argc, char **argv)
4 {
5     int      i, sockfd[5];
6     struct sockaddr_in servaddr;
7     if (argc != 2)
8         err_quit("usage: tcpcli <IPaddress>");
9     // 5 clients are connected to the server; server creates 5 child processes
10    for (i = 0; i < 5; i++) {
11        sockfd[i]=Socket(AF_INET, SOCK_STREAM, 0);
12        bzero(&servaddr, sizeof(servaddr));
13        servaddr.sin_family = AF_INET;
14        servaddr.sin_port = htons(SERV_PORT);
15        inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
16        Connect(sockfd[i], (SA *) &servaddr,sizeof(servaddr));
17    str_cli(stdin, sockfd[0]); /* only one call */
18    exit(0);
19 }
```

## wait() vs. waitpid() (cont.)



- Client terminates, closing all 5 connections, terminating all five child processes
  - It causes 5 SIGCHLD signals to be delivered to the parent at about the same time.
- `wait()` is insufficient for preventing zombies:
  - All 5 signals are generated before the signal handler is executed.
  - The signal handler is executed only one time because Unix signals are normally not queued.

## wait() vs. waitpid() (cont.)



```
linux % tcperv03 &  
[1] 20419  
linux % tcpcli04 127.0.0.1  
hello                                     we type this  
hello                                     and it is echoed  
^D                                         we then type our EOF character  
child 20426 terminated                   output by server  
* * * * *  
* * * * *
```

The first thing we notice is that only one `printf` is output, when we expect all five children to have terminated. If we execute `ps`, we see that the other four children still exist as zombies.

```
PID TTY TIME CMD  
20419 pts/6 00:00:00 tcperv03  
20421 pts/6 00:00:00 tcperv03 <defunct>  
20422 pts/6 00:00:00 tcperv03 <defunct>  
20423 pts/6 00:00:00 tcperv03 <defunct>  
20424 pts/6 00:00:00 tcperv03 <defunct>  
* * * * *
```

## Use `waitpid()` to prevent zombies

```
void  
sig_chld(int signo)  
{  
    pid_t pid;  
    int    stat;  
  
    . . .  
  
    while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)  
        printf("child %d terminated \n", pid);  
  
    return;  
}
```

This version works because we call `waitpid` within a loop, fetching the status of any of the children that have terminated.

`WNOHANG` option tells `waitpid` not to block if there are running children that have not yet terminated.

`Wait` function cannot be called in a loop, because there is no way to prevent `wait` from blocking if there are running children that have not yet terminated.

# Important

- Must catch the **SIGCHLD** signal when *forking* child process.
- Must handle interrupted system calls when we catch signals.
- A **SIGCHLD** handler must be coded correctly using *waitpid* to prevent any zombies from being left behind.

# TCP server that handles EINTR from accept()

```
#include      "unp.h"
int
main(int argc, char **argv)
{
    void      sig_chld(int);
    ...
    listenfd = Socket(AF_INET, SOCK_STREAM, 0);
    ...
    Bind( ... );
    Listen(... );
    Signal(SIGCHLD, &sig_chld); /*must call waitpid() */
    for(;;) {
        clilen = sizeof(cliaddr);
        if ((connfd = accept(listenfd, (SA *) &cliaddr, &clilen)) < 0) {
            if (errno == EINTR)
                continue; /* back to for() */
            else
                err_sys("accept error");
        }
        if ((childpid = Fork()) == 0) /* child process */
            Close(listenfd);/* close listening socket */
            str_echo(connfd);           /* process the request */
            exit(0);
        }
        Close(connfd); /* parent closes connected socket */
    }
}
```

## Interruption of system calls and library functions by signal handlers

If a signal handler is invoked while a system call or library function call is blocked, then either:

- the call is automatically restarted after the signal handler returns; or
- the call fails with the error **EINTR**.

Which of these two behaviours occurs depends on the interface and whether or not the signal handler was established using the **SA\_RESTART** flag (*see sigaction()*). The details vary across UNIX systems. If a blocked call to one of the interfaces is interrupted by a signal handler, then the call will be automatically restarted after the signal handler returns if the **SA\_RESTART** flag was used; otherwise the call will fail with the error **EINTR**:



## Connection Abort before accept() returns

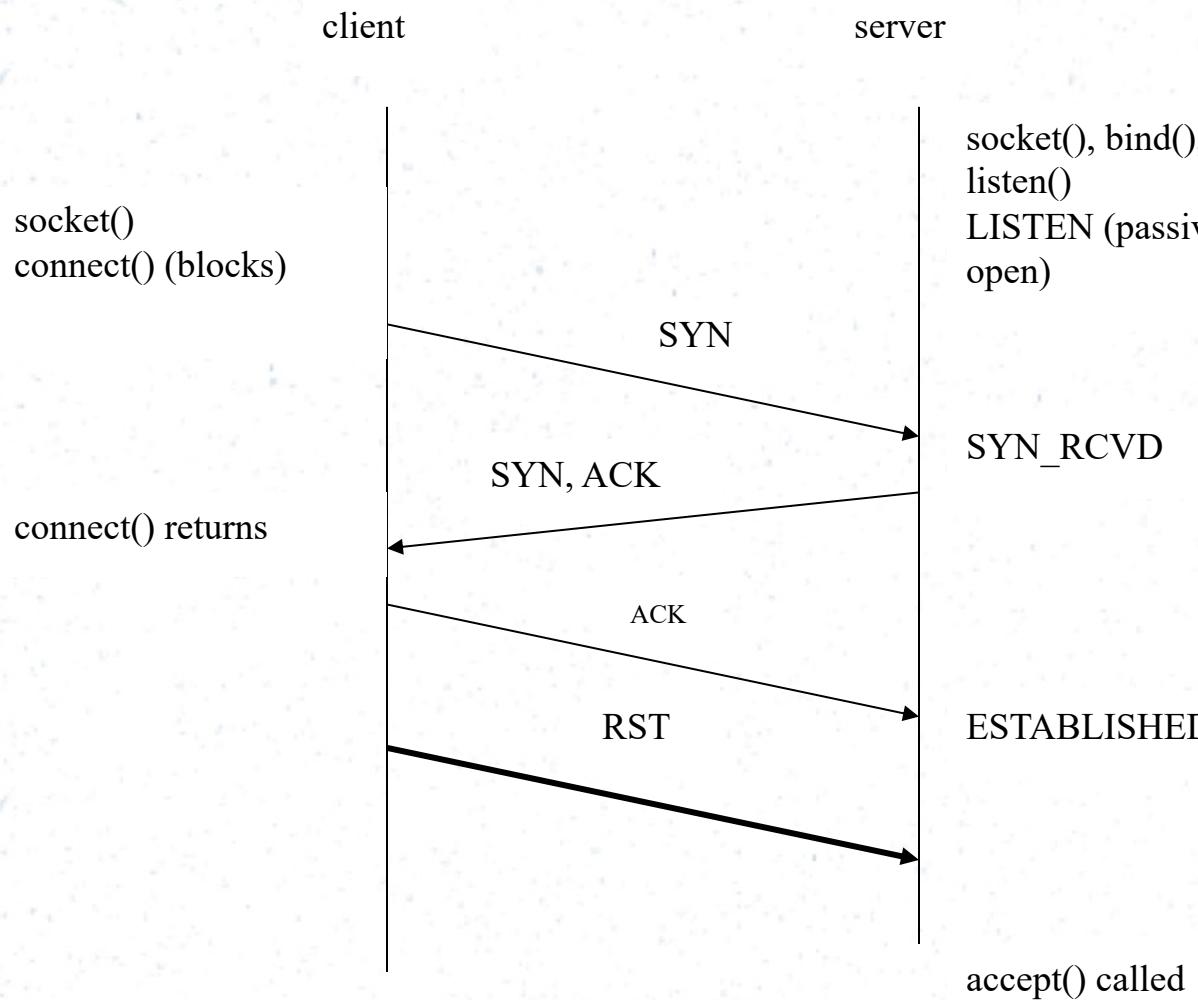
### Scenario (in a busy server):

- 3-way handshake completes, connection establishes.
- Client side: sends **RST**.
- Server side:
  - Connection is queued by its TCP.
  - Waiting for server process to call `accept()`, and **RST** arrives.

### Handling:

- Implementation dependent;
- Posix.1g: `accept()` should return with **ECONNABORTED**.
  - We should just call `accept()` again.

# Connection Abort before accept() returns (cont.)



Implementations handle the aborted connection completely within the kernel, and the server process never sees it. But returns an error to the process as the return from *accept()*.

example,

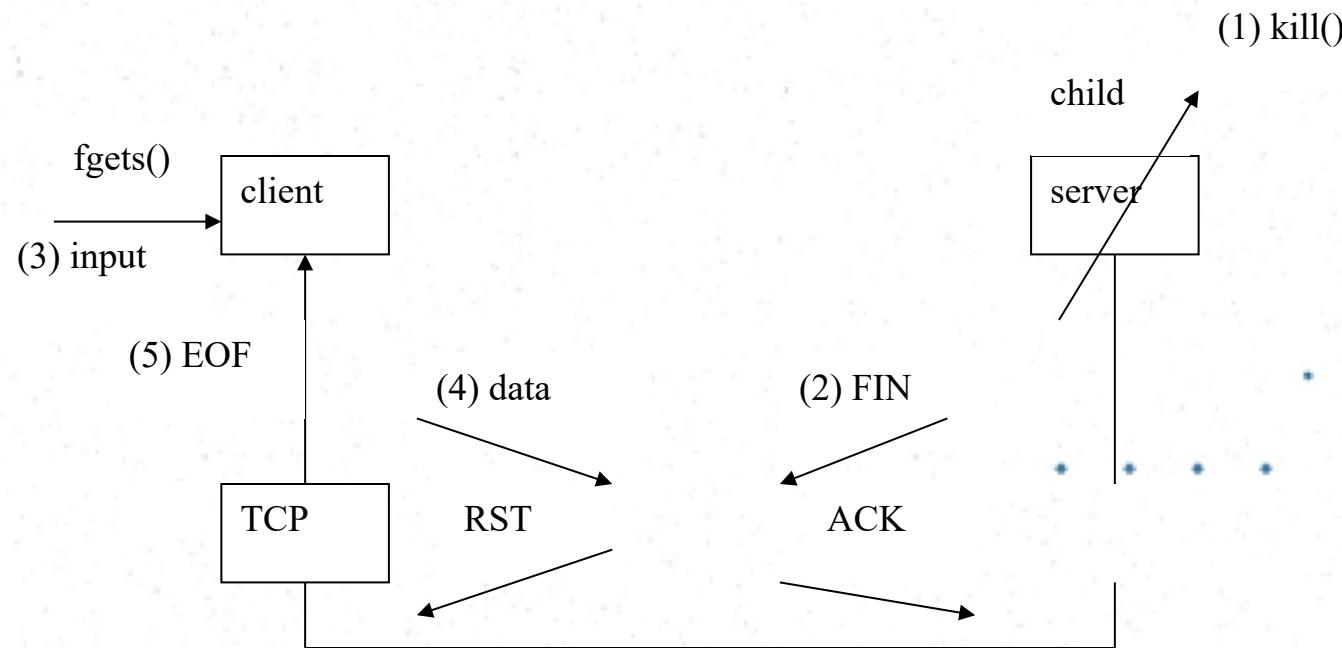
POSIX – **ECONNABORTED**  
SVR4 - **EPROTO**

# Termination of Server Process - child

## Scenario:

- Start client-server, and kill the server child process
  - This simulates the crashing of the server process.

Note: crashing of the *server process* is different than crashing of the *server host*.



## Termination of Server Process (cont.)

- (1) Server process aborts and the server's kernel closes all its open descriptors.
- (2) Server's TCP starts a connection termination with client's TCP.
  - Server's TCP sends FIN, client's TCP responds with ACK → first half of TCP connection termination.
  - SIGCHLD signal is sent to parent server and handled correctly.
  - Server's closed TCP connection is blocked forever in a FIN\_WAIT\_2 state until client does something.
  - **Problem:** client process is blocked in the call to fgets() waiting for a line from the terminal.
- (3) We can still type a line of input in the client, and the client calls writen() to send a line to server.

This is allowed by TCP because the receipt of the FIN by client TCP only indicates that the server process has closed its end of the connection and will not be sending any data.

The FIN does not indicate the server process has terminated. This is known as TCP's half close.

- (4) Client's TCP sends the data to the server.
  - Server's TCP receives the data, and responds with a RST.
- (5) But the client process will not see the RST, because it calls readline() immediately after the call to writen().
  - Therefore, readline() returns 0 (EOF, from the previously received FIN)  
→ print '*server terminated prematurely*'.

- If the *readline* happens before the RTS is received, the result is an unexpected EOF in the client as in example.
- But if the RTS arrives first, the result is an ECONNRESET (“*Connection reset by peer*”) error return from *readline*.

## Termination of Server Process (cont.)

The problem in this example:

- The client is blocked in the call to fgets() when the FIN arrives on the socket. Client is really working with two descriptors; the

Client is really working with two descriptors; the socket and the user input, so it should block on input from either the sources.

## Solution:

- Use `select()` or `poll()` functions so that the client is notified immediately (with FIN) once the server child is killed.
    - We will discuss these functions later.
  - If the client process ignores the return of 0, and tries to send more data to server, the client process will receive a **SIGPIPE**.
    - Because client TCP has received RTS from server.

# SIGPIPE signal

- **Scenario:** the client needs to performs two writes to server before reading anything back, with the first write eliciting the **RST** (e.g., when the child server is killed).
  - *When a process writes to a socket that has received an RST, the SIGPIPE signal is sent to the process.*
- The **default action** of the **SIGPIPE** signal is to **terminate the process**.
  - So the process must catch the signal to avoid being involuntarily terminated.
  - If the process either catches the signal or ignores the signal, the `write()` operation returns **EPIPE**.
  - The problem with program terminated by **SIGPIPE**: normally nothing is output.
  - Recommended way to handle **SIGPIPE** depends on what the application wants to do when this occurs.

## Example: modification of client str\_cli function

```
1 #include      "unp.h"  
  
2 void  
3 str_cli(FILE *fp, int sockfd)  
4 {  
5     char        sendline[MAXLINE], recvline[MAXLINE];  
6     while(Fgets(sendline, MAXLINE, fp) != NULL) {  
*     *     *  
7         Writen(sockfd, sendline, 1); //write to the socket  
*     *     *  
8         sleep(1);   //expect socket to receive RST during this time interval  
*     *     *  
9         Writen(sockfd, sendline+1,strlen(sendline)-1); // generates SIGPIPE  
*     *     *  
10        if (Readline(sockfd, recvline, MAXLINE) == 0)  
11            err_quit("str_cli: server terminated prematurely");  
*     *     *  
12        Fputs(recvline, stdout);  
*     *     *  
13    }  
*     *     *  
14}
```

If we run the client on our Linux host, we get:

```
linux % tcpclill 127.0.0.1
```

```
hi
```

*there we type this line*

```
hi
```

*there this is echoed by the server*

```
bye
```

*then we type this line*

```
Broken pipe
```

*this is printed by the shell*

The recommended way to handle **SIGPIPE** depends on what the application wants to do when this occurs. If there is nothing special to do, then setting the signal disposition to **SIG\_IGN** is easy, assuming that subsequent output operations will catch the error of **EPIPE** and terminate.

# Crashing of Server Host (without rebooting)

**Scenario:** Start client and server on two different hosts. After the two are connected correctly, disconnect the server host from the network.

- Nothing happens at client when server host crashes if the client process is idle (blocked in the call to `fgets()` on stdin).

(1) We enter a line of input.

When the server host crashes, nothing is sent out on the existing network connections.

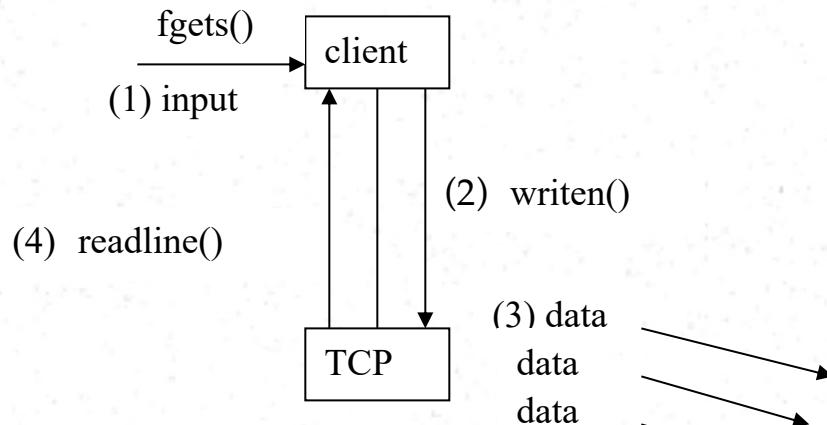
(2) Client calls `written()` to send a line to server and blocks in the call to `readline()` for reply.

(3) TCP buffers the data and tries to send it. `written()` does not return an error.

- Client's TCP tries and tries to send data, eventually times out.

(4) `readline()` in client returns **ETIMEOUT** error: *Connection timed out that may take 9 minutes, retransmitting the data segments 12 times.*

- If the intermediate router determined that the server host is unreachable and responds with an ICMP “*destination unreachable*” providing error **EHOSTUNREACH** or **ENETUNREACH**.



Solution for the waste of time:  
Place a timeout on the call to `readline()`, which will be discussed in a later lecture.

# Crashing and Rebooting of Server Host

**Scenario:** Start client and server on two different hosts. After the two are connected correctly, disconnect the server host from the network, and then reboot it before the client sends data.

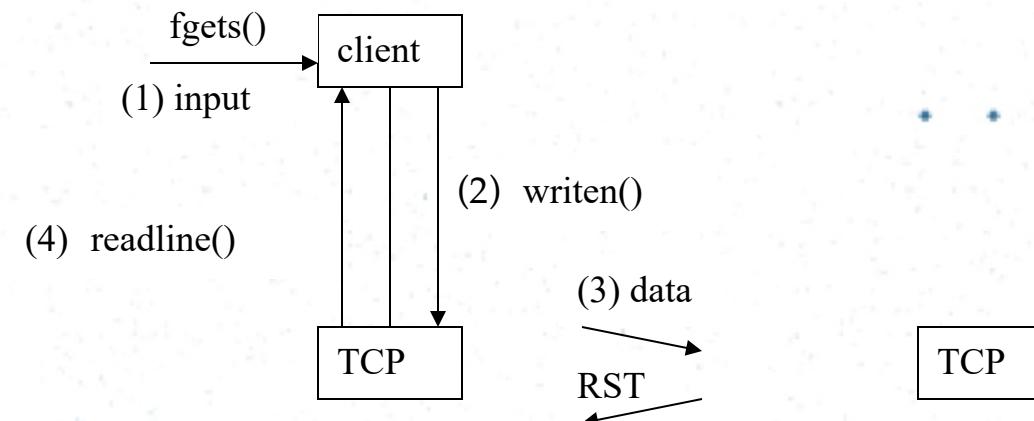
- Nothing happened at client when server host crashes and reboots if the client process is idle (blocked in call to `fgets()` on stdin).

- (1) We enter a line of input.
- (2) Client calls `written()` to send a line to server.
- (3) TCP buffers the data and tries to send it. `written()` does not return an error.

Client's TCP sends the data, server's TCP responds with RST because the TCP loses all information about connections that existed before crashes after the reboot.

- (4) `readline()` in client returns **ECONNRESET** error: “*Connection reset by peer*”

- Note: The client should detect the crashing of the server host using **SO\_KEEPALIVE** socket option (discussed later)



## Shutdown of Server Host

**Scenario:** Start client and server on two different hosts. After the two are connected correctly, shutdown the server host (while the server process is running).

- In Unix, the *init* process sends the **SIGTERM** signal (catchable) to all processes, waits for several seconds, and sends **SIGKILL** signal (not catchable) to any process still running. Providing short amount of time to clean up and terminate.
- TCP module on server's kernel starts a connection termination with client's TCP module. (Server's TCP sends FIN, client's TCP responds with ACK.)
- The only difference between this case and server host crashing and rebooting is that here the client can be notified of server shutdown. (But we do not catch this notification, since client is blocked in `fgets()` on `stdin`.)
- When data is sent by the client, we have the same scenario as before, depending whether server is up or down.

**Note:** Client must use `select()` or `poll()` function so that the client can detect the termination of the server process as soon as it occurs.

## Server is not running

- Server TCP responds to client TCP's **SYN** with **RST**.
- Client's call to `connect()` returns **ECONNREFUSED** error:  
    “Connection refused”

• •  
• •  
• •  
• •

## No network connectivity between client and server

- Client's call to `connect()` times out (normally after 75 seconds).
- If an ICMP error is returned (e.g., “host unreachable”), **EHOSTUNREACH** returned.
- If no ICMP error is returned, `connect()` returns **ETIMEOUT**.
- Some older implementations incorrectly abort connection attempt as soon as ICMP error received.

# Additional Materials

## Other issues to consider

- 1) For the client-server programs, who should *talk* first? For better security, let the client *talk* first.
- 2) Consider an iterative connection-oriented server and assuming that the client is expected to *talk* first. What will happen if a client makes a connection to the server but never sends a request?
  - Server deadlock.
  - How is the deadlock detected?
  - We may also have *starvation* in clients.
  - The standard solution is using timeout, concurrent server, and reconnect.
- 3) What type of data do you use?
  - Text
  - ASCII
  - Binary
  - Which data type is the most efficient to use?

## Data Format

- We must be careful on the format of the data exchanged between the client and the server.
  - If we pass decimal values across the socket (instead of text strings), it does not work when:
    - the client and server are run on hosts with different byte orders.
    - The hosts do not agree on the size of *long* integers.

```
struct args {  
    long      arg1;  
    long      arg2;  
};  
struct result {  
    long      sum;  
};
```

# Passing Binary Structures between Client and Server

**Client**

```
void str_cli(FILE *fp, int sockfd)
{
    char sendline[MAXLINE];
    struct args args_obj;
    struct result result_obj;

    while (fgets(sendline, MAXLINE, fp) != NULL) {
        if (sscanf(sendline, "%ld%ld", &args_obj.arg1, &args_obj.arg2) != 2) {
            printf("invalid input: %s", sendline);
            continue;
        }
        Writen(sockfd, &args_obj, sizeof(args_obj));
        if(Readn(sockfd, &result_obj, sizeof(result))== 0)
            err_quit("str_cli: server terminated prematurely");
        printf("%ld\n", result_obj.sum);
    }
}
```

- ❖ `sscanf` converts the two arguments from text strings to binary
  - ❖ We call `writen` to send the structure to the server.
  - ❖ We call `readm` to read the ready, and print the result using `prin`

## str\_echo function that adds two binary integers

### Server

```
void  
str_echo (int sockfd)  
{  
    ssize_t      n;  
    struct args args_obj;  
    struct result      result_obj;  
  
    . . .  
    for ( ; ; ) {  
        if((n=Readn(sockfd, &args_obj, sizeof(args_obj)))==0)  
            return; /* connection closed by other end  
        */  
  
        . . .  
        result_obj.sum = args_obj.arg1 + args_obj.arg2;  
        Writen(sockfd, &result_obj, sizeof(result_obj));  
    }  
}
```

We read the arguments by calling `readn`, calculate and store the sum, and call `writen` to send back the result structure.

If we run the client and server on two machines of the same architecture, everything works fine. Here is the client interaction:

But when the client and server are on two machines of different architectures (say the server is on the big-endian SPARC system freebsd and the client is on the little endian Intel system linux), it does not work.

```
linux % tcpcli09 206.168.112.96
1 2                                     we type this
3                                     and it works
-22 -77                                then we type this
-16777314                               and it does not work
```

# Passing text strings between Client and Server

```
// str_echo function modified, that adds two numbers
void
str_echo(int sockfd)
{
    long          arg1, arg2;
    ssize_t        n;
    char          line[MAXLINE];
    for ( ; ; ) {
        if((n = Readline(sockfd, line, MAXLINE))== 0)
            return; /* connection closed by other end */
        if (sscanf(line, "%ld%ld", &arg1, &arg2)== 2)
            snprintf(line, sizeof(line), "%ld\n", arg1 + arg2);
        else
            snprintf(line, sizeof(line), "input Error\n");
        n = strlen(line);
        Writen(sockfd, line, n);
    }
}
```

Server

- sscanf is used to convert from text string to long integers.
- snprintf is used to convert the result into a text string.
- The client-server works fine regardless of the byte ordering of the client and server hosts.

## Common solutions to data format problem

1. Pass all the numeric data as a text string → assuming that both hosts have the same character set.
  2. Explicitly define the binary formats of the supported datatypes (number of bits, big or little endian) and pass all data between the client and the server in this format.
- 