

# Gestione di Rete

## Paradigmi e Protocolli per la Gestione di Rete

# 1. Introduction

## 1. Introduction

### 1.1 Motivation

### 1.2 Terminology and Basic Concepts

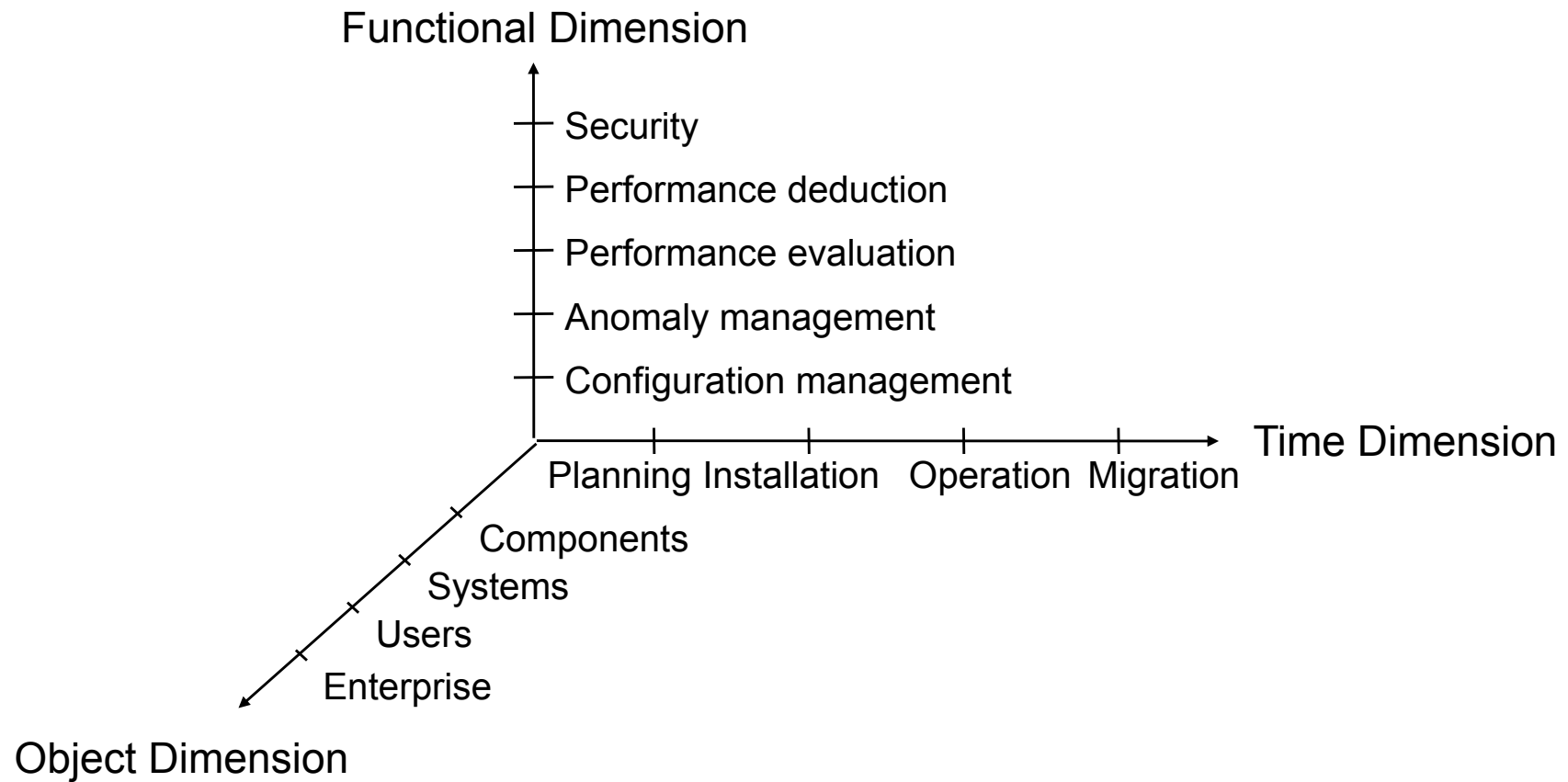
### 1.3 Abstract Syntax Notation One

## 2. Internet Management

# 1.1 Motivation: Why Do We Need Management ?

- Current situation:
  - increasing meaning of strategic resources "information".
  - a computer network is no longer only a supporting item in an enterprise, but takes even more frequently a key position.
  - the number of interconnected computers rose dramatically in the past few years. This process will probably continue to persist.
  - Complexity and functionality of the components grows in correspondence with the performance of the available hardware.
- Demand:
  - Permanent availability of network services with optimal quality.
  - Cost reduction for the network infrastructure of the company.
- Necessity:
  - computer-aided management of heterogeneous networks.

# Network Management Dimensions

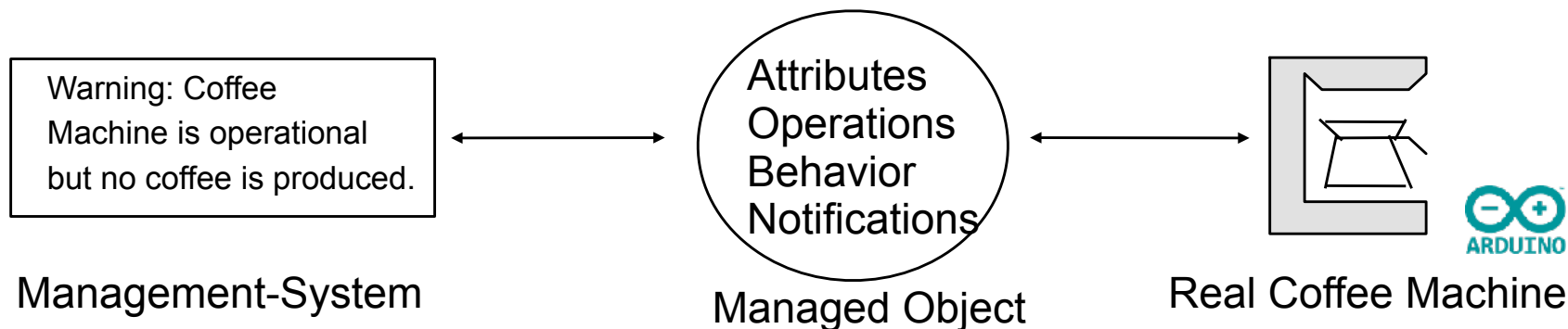


## 1.2 Terminology and Fundamental Concepts

- Control, co-ordination and monitoring of resources takes place via the manipulation from so-called managed objects:

"A *managed object* is the abstracted view of a resource that presents its properties as seen by (and for the purpose of) management." (ISO 7498-4)

- Managed objects are an abstract representation of a real resource.
- The boundary of a managed object specifies which details are accessible to a management system and which ones are shielded (black box).



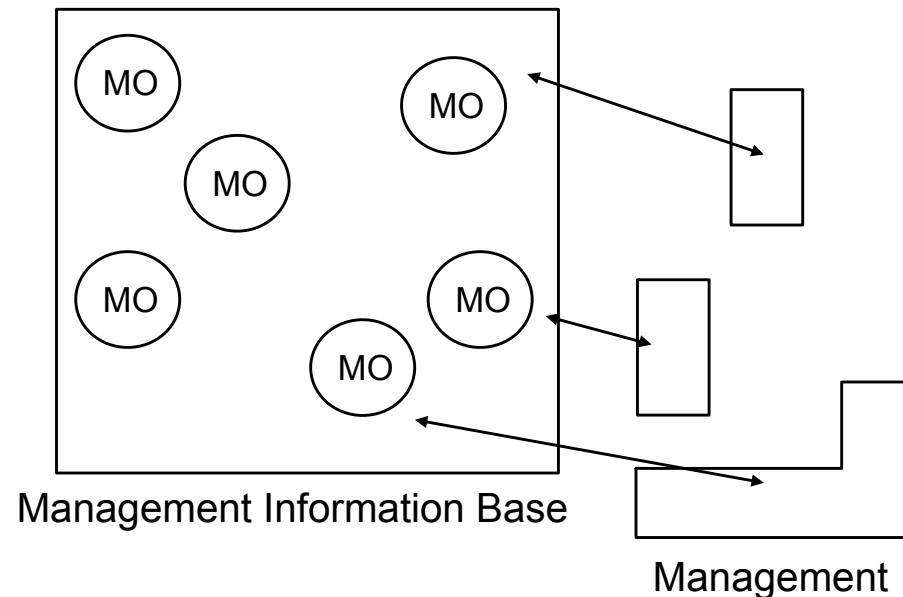
- Managed objects do not necessarily correspond to objects, as one knows from object-oriented programming. Simple variables correspond to the MOs in the Internet management.

# Managed Objects (MO)

- Attributes:
  - Attributes describe the state/condition of managed objects.
  - Attributes can change when the condition of the real object changes.
  - Attributes can be manipulated by means of management operations.
- Operations:
  - Make it possible to access a managed object. Typical operations are get, set, create and delete.
  - The number and type of operations influence the object performance and complexity.
- Behavior:
  - Determines the semantics and interaction with the real resource.
  - The behavior of managed objects is normally defined in plain English.
- Notifications:
  - The quantity and type of the messages, which can be generated by pre-defined situations by a managed object when specific situations occur.

# Management Information Base (MIB)

- The union of all managed objects contained in a system forms the *Management Information Base* (MIB) of the system:  
"The set of managed objects within a system, together with their attributes, constitutes that system's management information base." (ISO 7498-4)
- This is the first interpretation of the term "Management Information Base" (more definitions will follow).



- A MIB should be known both to the implementer and the manager.

# MIB Modularity

- Managed objects of a system are usually defined in multiple MIB definitions.
- Modules have been introduced in MIBs for enabling design modularity:
  - Different modules can be defined by different teams.
  - Management functionality can be gradually extended and modified.
  - Different systems can support different MIB modules/releases.
  - Vendors can extend the management functionality by means of proprietary MIBs.
  - MIBs are defined using a specification language

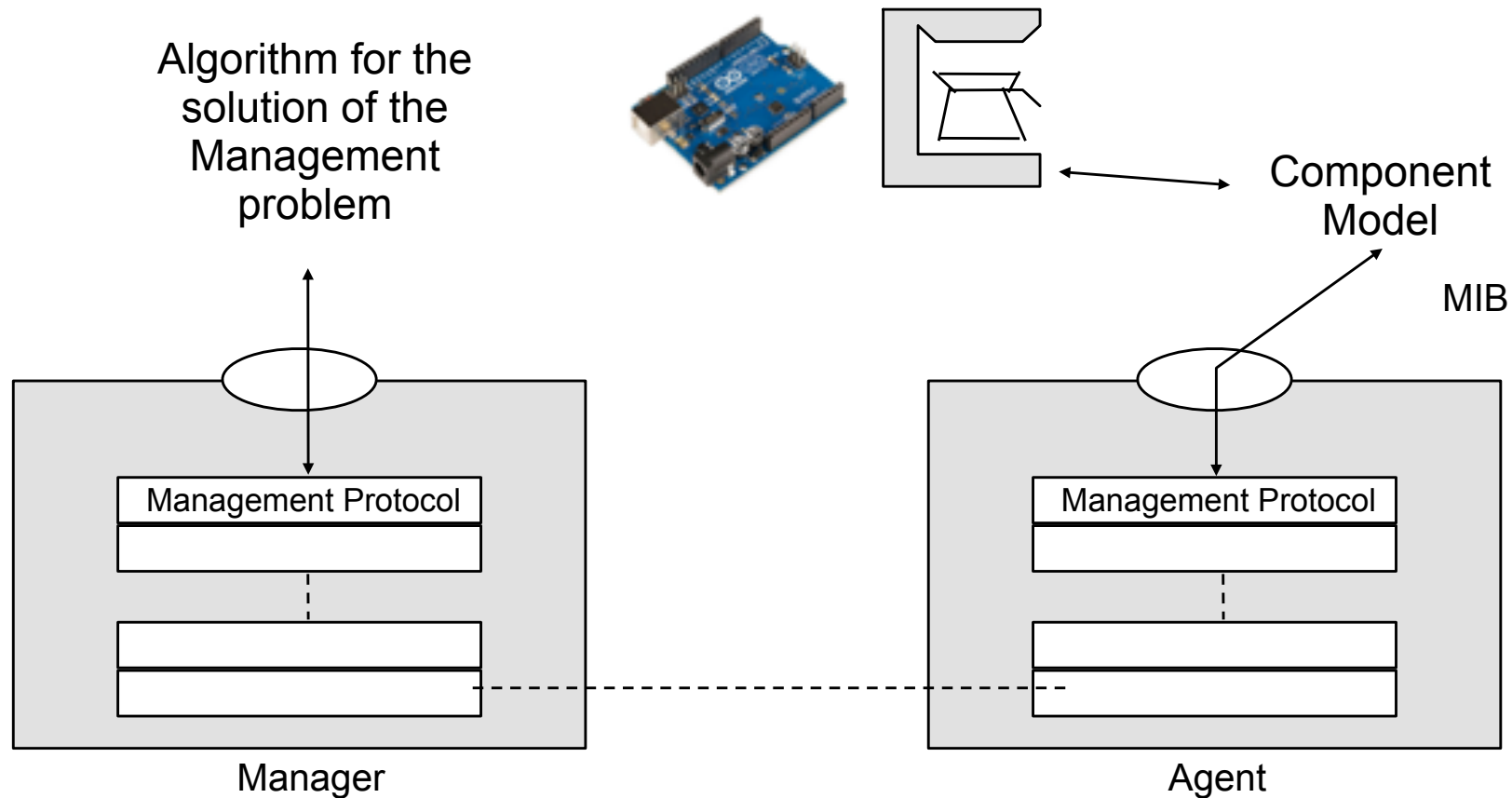


# Manager/Agent Paradigm

- Agent:
  - Implements the MOs MIB by accessing the real resources.
  - Receives requests from a manager, processes them and transmits appropriate responses.
  - Dispatches notifications about important changes in status in the MIB.
  - Protects MOs against unauthorised accesses using access control rules and communication authentication with the partner.
- Manager:
  - Exercises control: it controls functions hence it is the crucial instance.
  - Starts up management operations by appropriate protocol operations for the manipulation of MOs.
  - Receives messages from agents and passes them on (for handling) to appropriate applications.

# Management Protocol

- Management applications and MOs are not often on same node.
- A management protocol implements access to distant managed objects by encoding management data that is then secured during the transfer.



# Functional Areas (FCAPS) [1/2]

- Management applications can be divided into 5 function areas:
- Fault management:
  - Error detection, isolation, and repair.
- Configuration management:
  - Production and administration of configuration information.
  - Name administration.
  - Start, check and termination of services.
- Account management:
  - Entry of consumption (usage) data.
  - Distribution and monitoring of contingents.
  - Customer billing for resource consumption.

## Functional Areas (FCAPS) [2/2]

- Performance management:
  - Statistic data collection.
  - Determination of the system performance.
  - Systems modifications for increase in efficiency.
- Security management:
  - Production and verification of security policies.
  - Generation and distribution of passwords and accounts.
  - Report and analysis of security-relevant events.
- These 5 functional areas according to the initial letters of the English terms normally under the contraction FCAPS.
- These functional areas are not mutually independent (data measurement has often impact on system configuration).
- Basic functions (e.g. monitoring of a counter for threshold values) often reside in different functional areas.

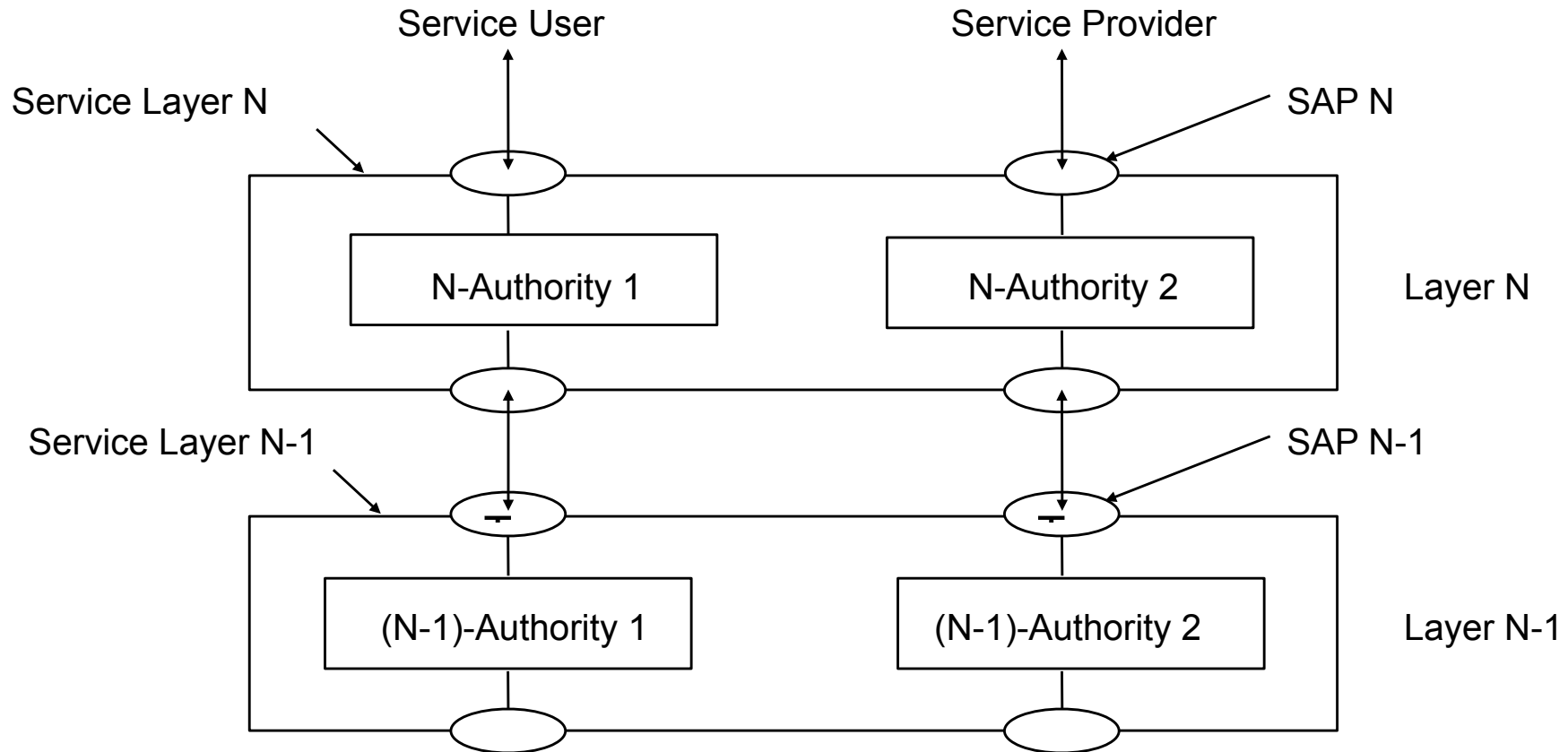
# Management Architectures Overview

- Structure of the management information:
  - defines the rules of the description of Managed Objects.
    - Identification and designation of MOs.
    - Composition of MOs.
    - Behaviour of MOs.
    - Relations to other MOs.
    - Possible operations and internal messages of the MOs.
  - Definition of the datatypes, structure and syntax for the description of the MOs.
  - The quantity of the descriptions of MOs in accordance with these rules defines the *Management Information Base* (MIB)
- Management Protocols and Services:
  - Defines the services and enable the access to remote MOs.
  - Several protocols can be used for the implementation of the defined services.
  - The service primitive and the appropriate protocol operations influence considerably the efficiency and the complexity of the management system.

# Services and Protocols: Some Definitions

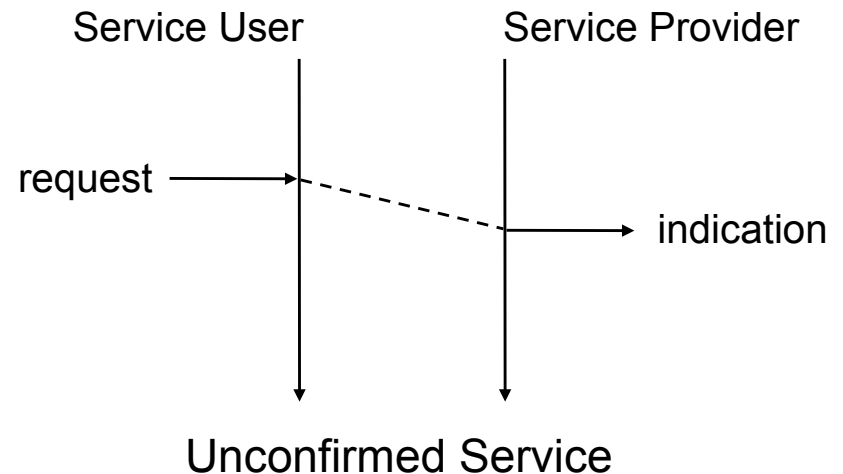
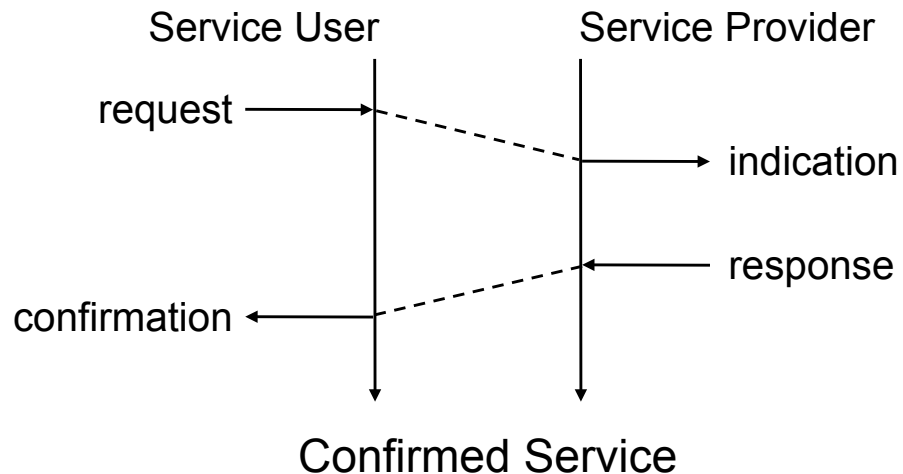
- Service
  - It is defined as an abstract function supplied by a network
- Service Primitive
  - The individual elementary functions are called service-primitives. Typical ISO/OSI services are:
    - request                      Service Request
    - indication                  Indication that a service was requested
    - response                    Reaction of the service to a service request
    - confirm                    Acknowledgement that a requested service was provided
- Service Access Point (SAP)
  - The interfaces over which the service primitive can be access as service access points.
- Entities
  - The services furnished by so-called instances.
- Protocol
  - The rules and the restrictions according to which instances interact with other instances.

# Representation and Layering of Services



- The definition of layers is a fundamental principle for the structuring of communication systems.
- Services of a layer may only accept service primitives of services in adjacent layers.

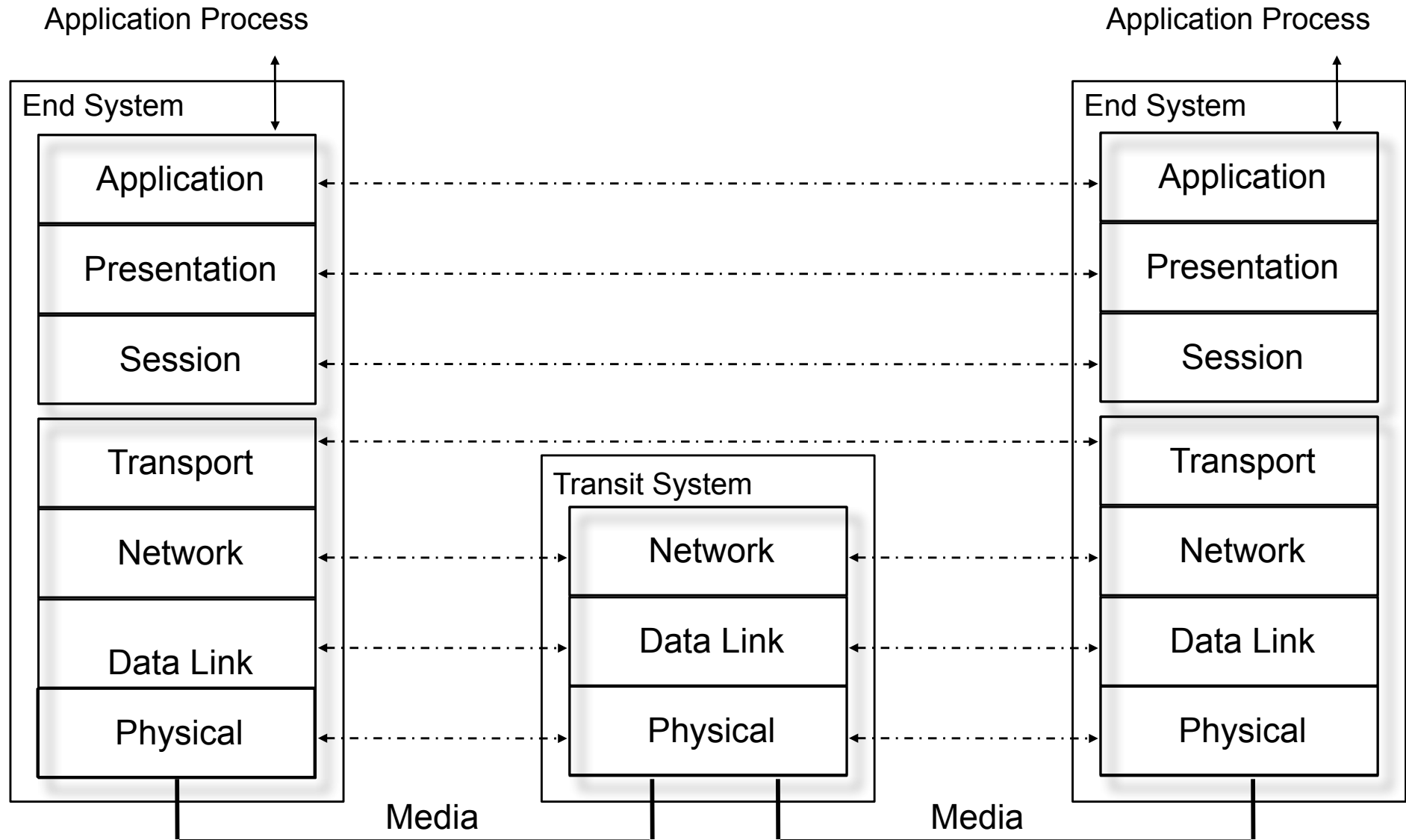
# Time Diagrams



- Time diagrams clarify the temporal and spatial connections between service primitives.
- Vertical axis are time axis, horizontal axis give the spatial distance between users and providers of services.
- Service requests of a confirmed service can result either in a positive or negative confirmation.
- Service requests of an unconfirmed service are not acknowledged.



# ISO/OSI-Reference Model



# ISO/OSI Transport System [1/2]

- Physical Layer
  - Transport of a stream of bits over a media.
  - Transport depending on the characteristics of the media being used.
  - Representation of values 0 and 1 (e.g. voltage levels).
  - Synchronisation between senders and recipients.
  - Definition of standard plugs for media interconnection.
- Data Link Layer
  - Transport of a frame of bits.
  - Data communication between systems that share a common media.
  - Detection and recovery of transfer errors.
  - Flow control for handling traffic peaks (traffic jam).
  - Implementation usually in hardware on adapter cards (e.g. Ethernet card).

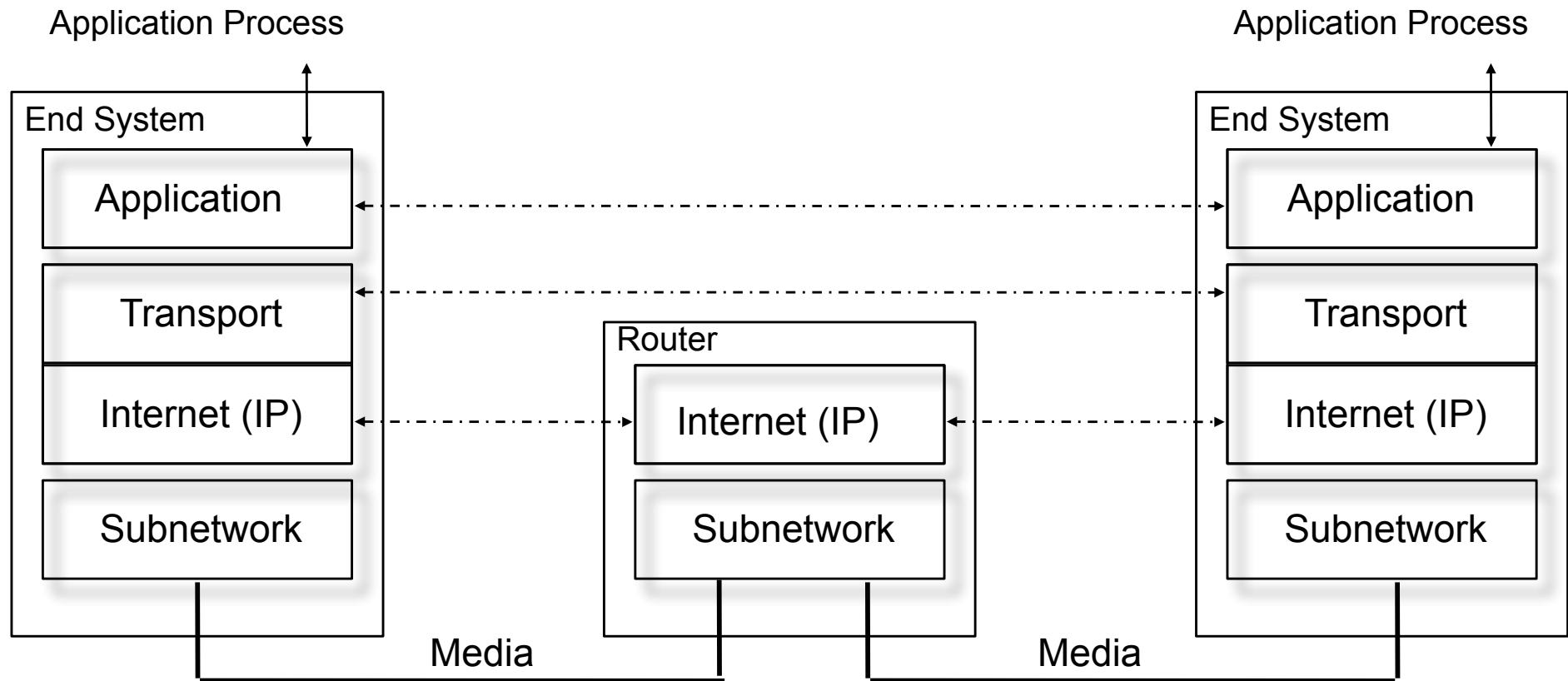
# ISO/OSI Transport System [2/2]

- Network Layer
  - Determination of a route through the network (routing).
  - Multiplex of network connections over a shared connection.
  - Error detection and recovery between end-systems.
  - Flow control between end-systems.
  - Division of a Packet in multiple frames.
- Transport Layer
  - End-to-end communication between applications.
  - Virtual connections over connectionless datagram services.
  - Error detection and recovery between applications.
  - Flow control between applications.
  - Concurrent usage of multiple services.

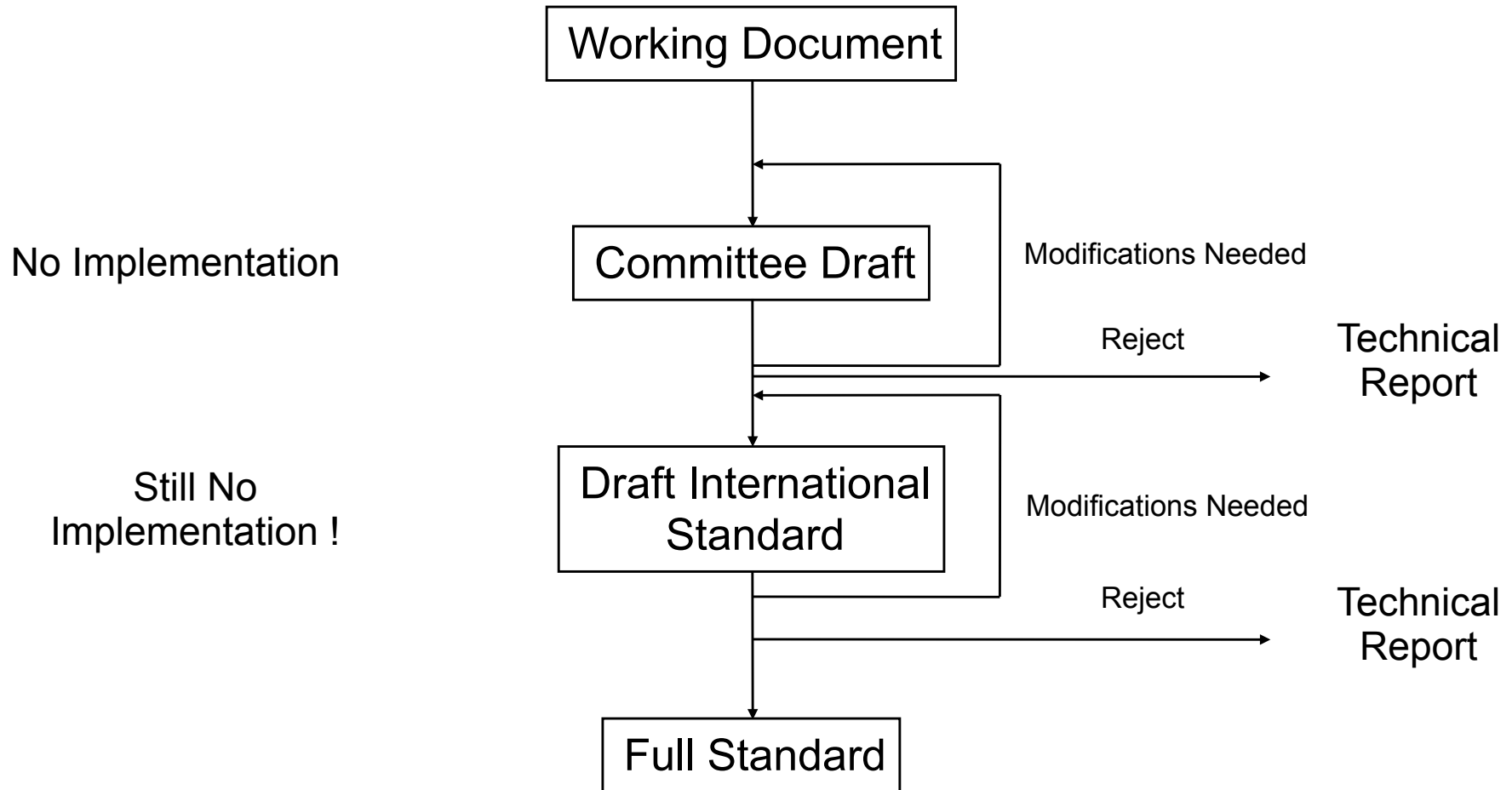
# ISO/OSI Higher Layers

- Session Layer
  - Synchronisation and co-ordination of communicating processes.
  - Session control (checkpoints for recovery).
- Presentation Layer
  - Transformation and adaptation of data presentations (e.g ASCII EBCDIC).
  - Serialisation of data structures for the purpose of transfer.
  - Data compression.
- Application Layer
  - Supply of fundamental services, which can be used directly by any application including (but not limited to):
  - File transfer, virtual terminals, name space administration, database access, network management, electronic communication networks, process and print control...

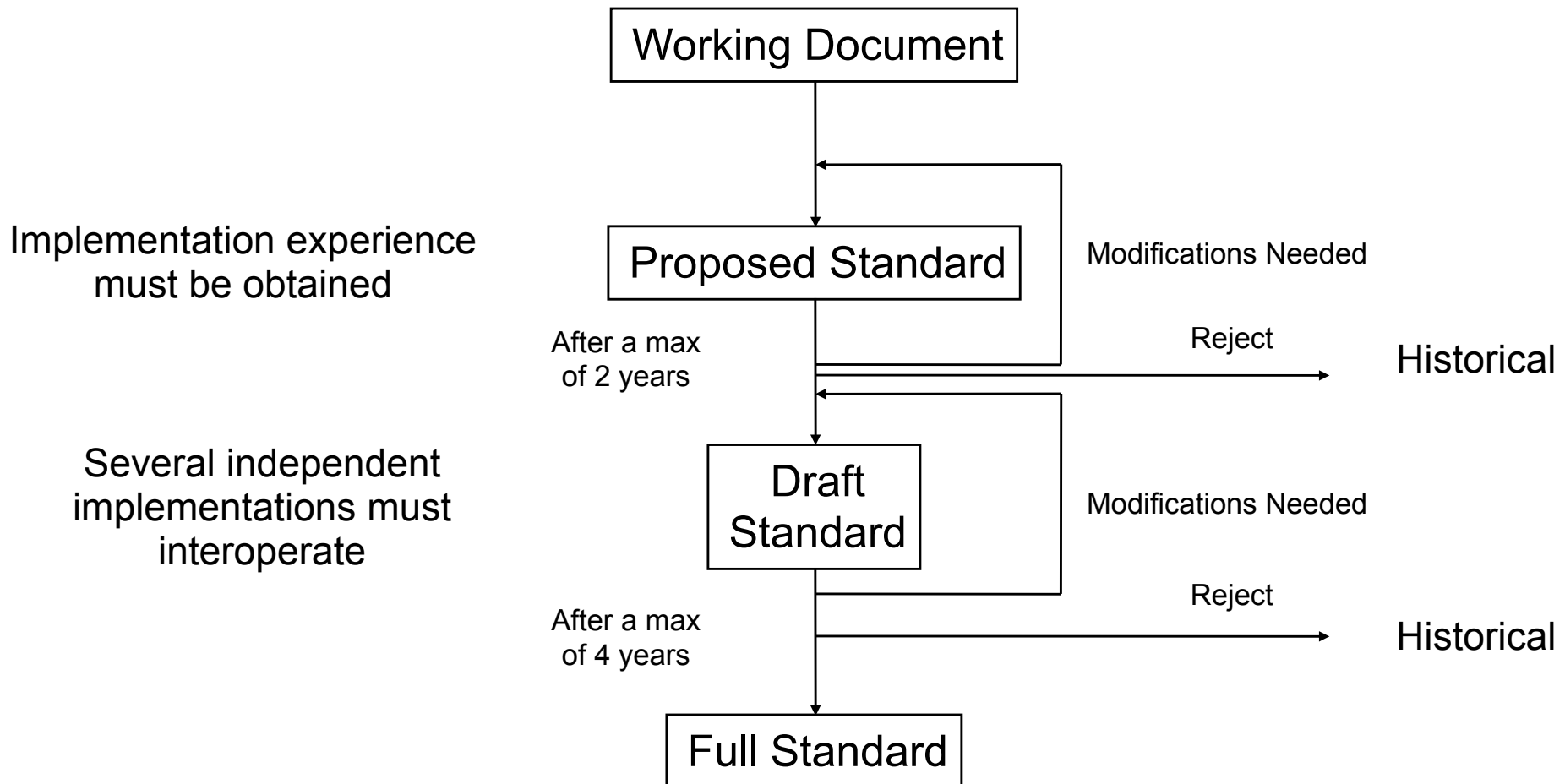
# Internet Layer Model



# ISO Standardisation



# IETF Standardisation



## 1.3 Overview: Abstract Syntax Notation One

- Abstract Syntax Notation One (ASN.1) is a syntax user for the definition of data structures and message formats.
- ASN.1 goals:
  - Exchange of information between machines with different hardware architectures (8/16/32/64 bit, little/big-endian).
  - Independence from existing programming languages (language neutral).
  - Coding of the data during the transfer should be selectable between senders and recipients (negotiation).
- Separation of the data presentation from the application-specific data structure representation.
- The abstract syntax defines the data structures during the transfer and determines in which form these data structures will serially transfer over a network.



## Little vs Big Endian [1/3]

LSB: least-significant byte / MSB: most-significant byte

MSB		LSB	
10101001	01001010	10101011	10001100

- **Big Endian:** load/store the MSB first
  - i.e., in the lowest address location
- **Little Endian:** load/store the LSB first
  - i.e., in the lowest address location

Network byte order is big endian, or most significant byte first

## Little vs Big Endian [2/3]

- Endianness matters when you store a multi-byte value to memory.
- Processors can be either Big (Motorola 68k, PowerPC) or Little endian (Intel x64, Apple Silicon)

### NAME

`htonl, htons, htonll, ntohl, ntohs, ntohll` – convert values between host and network byte order

### LIBRARY

Standard C Library (`libc`, `-lc`)

### SYNOPSIS

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

```
uint64_t  
htonll(uint64_t hostlonglong);
```

```
uint32_t  
htonl(uint32_t hostlong);
```

```
uint16_t  
htons(uint16_t hostshort);
```

```
uint64_t  
ntohll(uint64_t netlonglong);
```

```
uint32_t  
ntohl(uint32_t netlong);
```

```
uint16_t  
ntohs(uint16_t netshort);
```

# Little vs Big Endian [3/3]

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

int main(int argc, char *argv[]) {
    unsigned int a = 0x12345678;

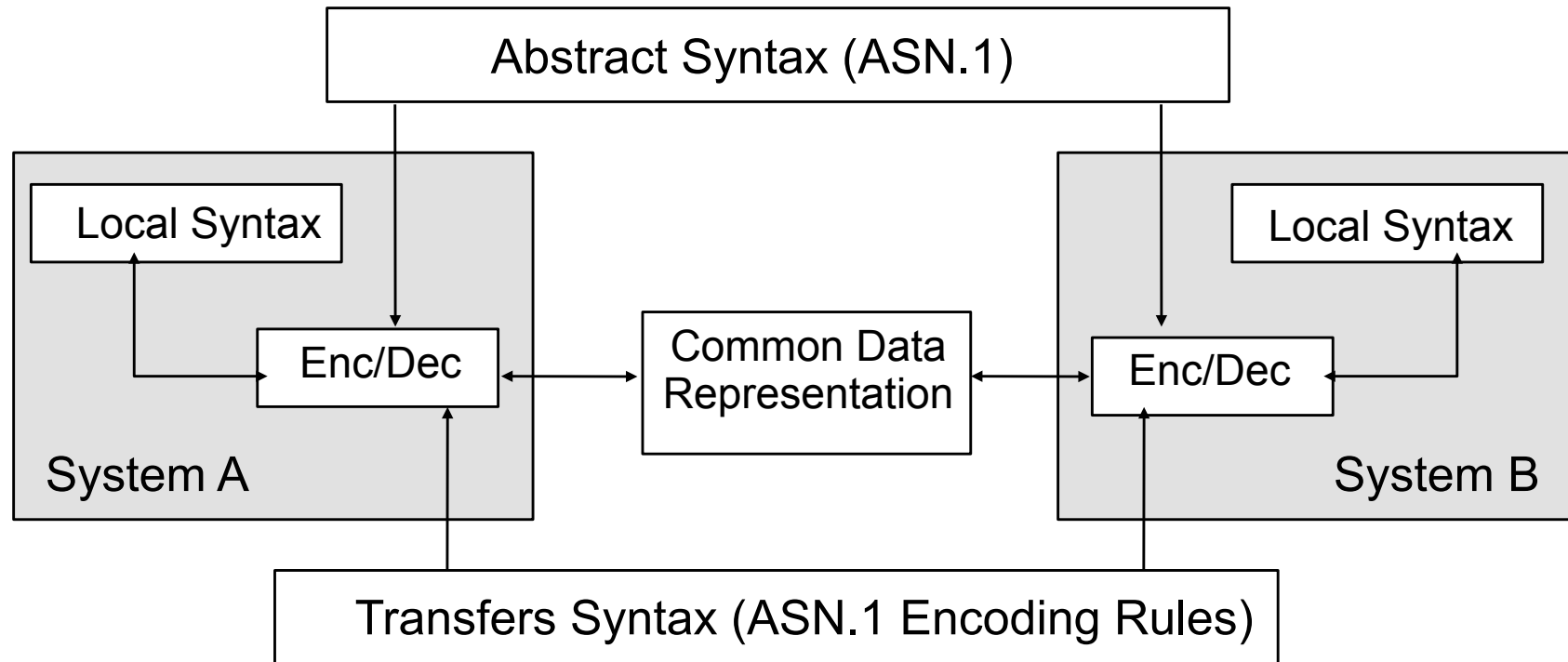
    printf("Host Representation:    0x%X\n", a);
    printf("Network Representation: 0x%X\n", htonl(a));

    return(0);
}
```

```
$ ./endian
Host Representation:    0x12345678
Network Representation: 0x78563412
```

Question: on what platform (little or big endian) did I run this program ?

# Abstract Syntax and Transfer Syntax



- ASN.1 defines a standardized abstract syntax.
- ASN.1 permits several encoding rules that transform the abstract syntax into a byte stream suitable for transfer. *BER* (Basic Encoding Rules) defines the mapping between abstract and transfer syntax.
- Applications normally use a local syntax depending on the programming language being used.

# Primitive ASN.1 Datatypes

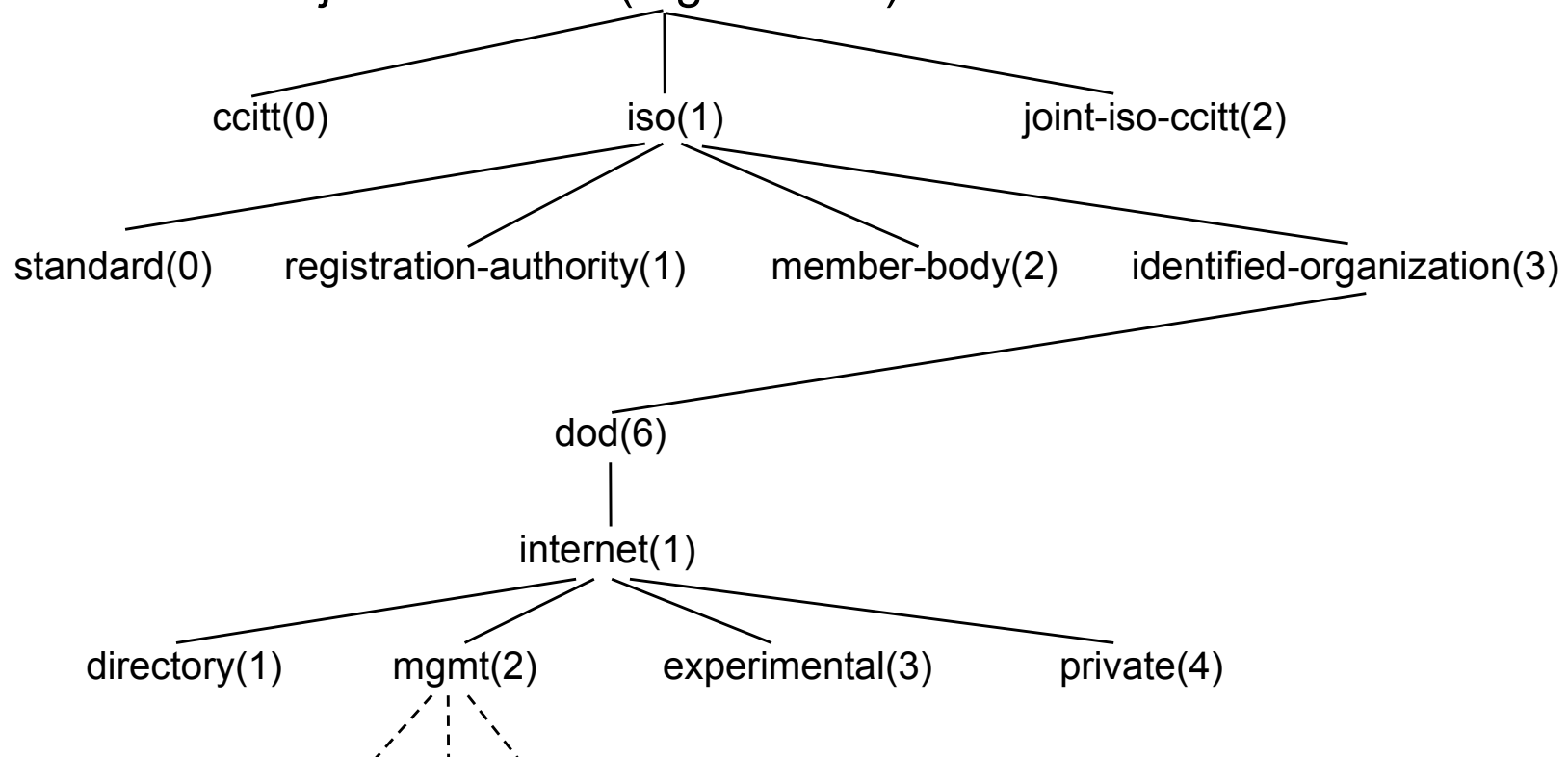
- Names of ASN.1 datatypes begin with a uppercase letter.
- Names of ASN.1 values (constants) begin with a lowercase letter.
- ASN.1 keywords and macro names consists only of uppercase letters.
- Comments are enclosed between `--` (e.g. `-- This is a comment --`).
- **BOOLEAN:**
  - Can only take the predefined values TRUE and FALSE.
- **INTEGER:**
  - Covers all the possible integer numbers. No delimitation of the number range.
- **BIT STRING:**
  - A sequence of bits. The length does not have to be divisible by 8.
- **OCTET STRING:**
  - A sequence of octets (bytes). It is the base type for different character sets and other derived types (GeneralizedTime, UTCTime).

# Primitive ASN.1-Datatypes

- **ENUMERATED:**
  - Type of enumerating. Possible values must be determined by the definition of derived datatypes.
- **OBJECT IDENTIFIER:**
  - Unique identification of a node in the ISO registration tree.
  - Path of the root of the tree to the target node.
- **ObjectDescriptor:**
  - A character string for the identification of a node in the Registration tree.
  - Not necessarily unique.
- **ANY:**
  - any ASN.1-datatype (Union of all ASN.1 datatypes as C 'void').
- **EXTERNAL:**
  - Data not described using an ASN.1 definition.
- **NULL:**
  - A substitute symbol, in order to indicate in an assembled datatype the absence of a value.

# ISO Registration Tree

- Used for uniquely identifying definitions, documents, objects...
- Hierarchical structure, similar to hierarchical file systems.
- All nodes of a level identified by a unique number.
- The path from the root of the registration tree to a node results in a numerical sequence called Object Identifier (e.g. 1.3.6.1).



# Assembled ASN.1 Datatypes

- **SEQUENCE:**
  - Corresponds to structures in C or records in Pascal.
  - The sequence of the items in a SEQUENCE is fixed.
- **SET:**
  - Similar to a SEQUENCE, with the difference that the sequence of the elements is not specified.
- **SEQUENCE OF:**
  - Ordered quantity (list) of homogeneous data.
- **SET OF:**
  - Unordered quantity of homogeneous data.
- **CHOICE:**
  - Type of selection, similar to the C union.
- **REAL:**
  - It consists of the `INTEGER` datatype extended with mantissa and exponent.



# Reduced Datatypes

- Definition of further datatypes by restricting the scope of existing datatypes.
- Exact syntax dependent on the underlying primitive datatype.

- Examples:

`LottoNumber ::= INTEGER (1..90)`

`MD5Key ::= OCTET STRING (SIZE (16))`

`IPAddress ::= OCTET STRING (SIZE (4|16))`

`Counter32 ::= INTEGER (0..4294967295)`

`Integer32 ::= INTEGER (-2147483648..2147483647)`

`Unsigned64 ::= INTEGER (0..18446744073709551615)`

- Restrictions of the scope are applied to derived datatypes (e.g `SEQUENCE OF MD5Key`).
- The restriction of the `INTEGER` datatype makes sense as today's computers internally usually operate with 32-bit or 64-bit numbers.

# Some Definitions of Types and Values

- Type definitions:

Number ::= INTEGER

DateAndTime ::= UTCTime

ID ::= OBJECT Identifier

- Value definitions :

ok BOOLEAN ::= TRUE

seven Number ::= 7

now DateAndTime ::= "971105012200-0100"

- Implicit Value Definitions :

Lotto ::= INTEGER { first(1), last(49) }

AccessRight ::= BIT STRING { read(1), write(2), execute(3) }

MaskAccessRight ::= { read, execute }

Sex ::= ENUMERATED { female(1), male(0) }

## A Complex Example [1/2]

```
Message ::= SEQUENCE {
    version INTEGER,
    community OCTET STRING,
    data ANY          -- e.g. PDUs if no authentication
}

PDUs ::= CHOICE {
    get-request          GetRequest-PDU,
    get-next-request    GetNextRequest-PDU,
    get-response        GetResponse-PDU,
    set-request          SetRequest-PDU
}

GetRequest-PDU      ::= [ 0 ] IMPLICIT PDU
GetNextRequest-PDU  ::= [ 1 ] IMPLICIT PDU
GetResponse-PDU     ::= [ 2 ] IMPLICIT PDU
SetRequest-PDU       ::= [ 3 ] IMPLICIT PDU
```

## A Complex Example [2/2]

```
PDU ::= SEQUENCE {
    request-id      INTEGER,
    error-status    INTEGER {
                        noError(0), tooBig(1),
                        noSuchName(2), badValue(3),
                        readOnly(4), genErr(5)
                    },
    error-index      INTEGER,
    variable-bindings VarBindList
}

VarBindList ::= SEQUENCE OF VarBind

VarBind ::= SEQUENCE {
    name      ObjectName,
    value     ObjectSyntax
}
```

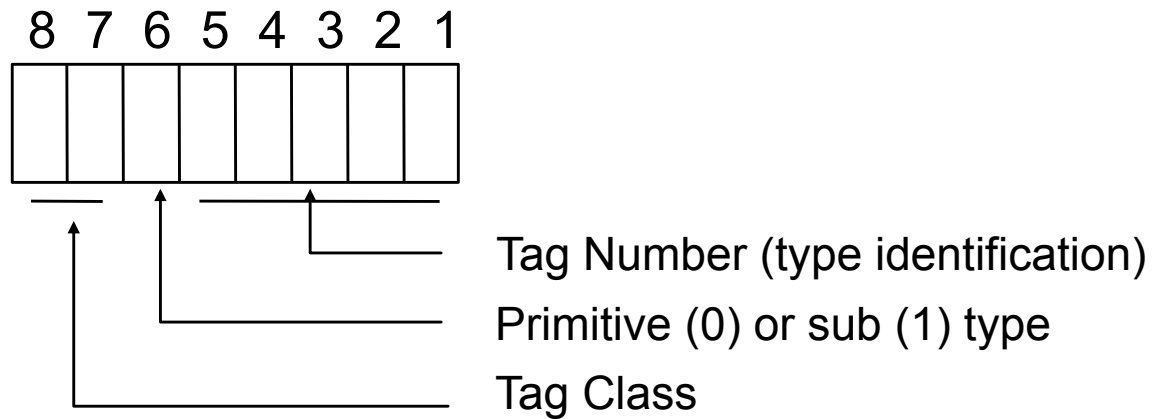
# Basic Encoding Rules (BER)

- The Basic Encoding Rules determine how a ASN.1 datatype can be represented as a string of bytes.
- Based on tag/length/value coding (TLV) algorithm, where the each variable is identified by one tag, the length of the value in bytes and the value of those bytes.
- The TLV coding permits a recipient to reconstruct the type of a message from the received byte stream.
- BER coding is a little inefficient as there is often unnecessary information to be transferred.
- The use of OPTIONAL fields further complicated the BER definition.
- BER also defines the transmission direction of the bit stream other than the coding the ASN.1 datatypes:



# Coding Tags Classes

- Each tags is coded in a byte:



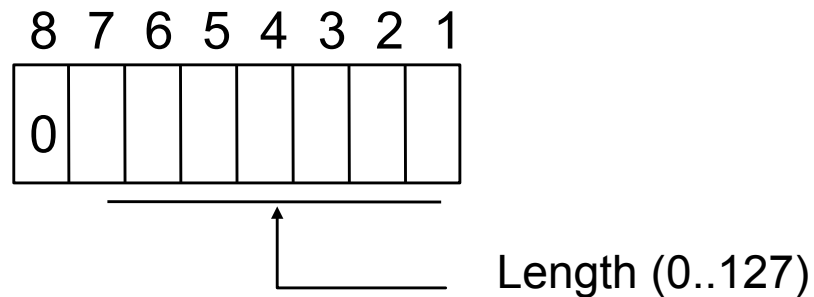
- Tag classes:

	Bit 8	Bit 7
UNIVERSAL	0	0
APPLICATION	0	1
CONTEXT-SPECIFIC	1	0
PRIVATE	1	1

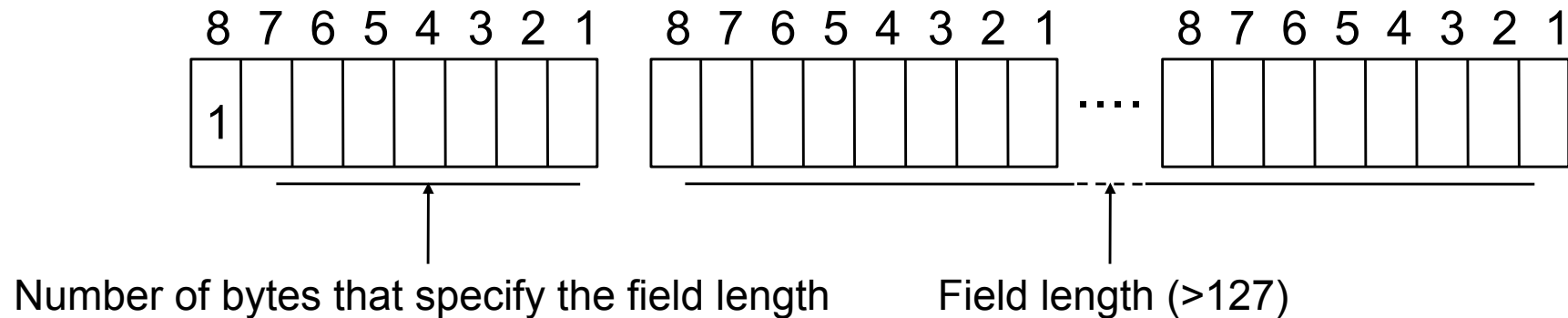
# Coding Field Length

- The length field indicates the length of the directly following value.

- Length within 0..127:



- Length > 127 :



# Value Coding

- For each primitive ASN.1 type there is a rule that allows values to be translated into a stream of bytes and vice-versa.
- The rules for INTEGER and OCTET STRING are simple.
- The rules for OBJECT IDENTIFIER are relatively complex.
- Assembled values (SEQUENCE, SEQUENCE OF) are easily represented by coding each individual item.
- With CHOICE constructs only the available value is transferred, therefore the associated tag must be unique.
- For further details:
  - D. Steedman: *Abstract Syntax Notation One (ASN.1) - The Tutorial and Reference*, Technology Appraisals, 1990



## Example of a BER Coded Message

30 1B	SEQUENCE, Length 27
02 01 00	INTEGER, Length 1, "0"
04 06 70 75 62 6C 69 63	OCTET STRING, Length 6, "public"
A1 0E	GetNextRequest-PDU, Length 14
02 04 36 A2 8F 07	INTEGER, Length 4, "916623111"
02 01 00	INTEGER, Length 1, "0"
02 01 00	INTEGER, Length 1, "0"
30 00	SEQUENCE OF, Length 0

- Length of the BER encoding must be well known (no dummy values) when a value is coded. With some restrictions it is also possible to specify the length after the value.
- The decoding is more difficult when the length is specified after the value.
- Coding the primitive values is not always as simple as in the example (some datatypes can be encoded in both short and long form).

# An ASN.1 Compiler [1/2]

ASN.1

```
CertainStructure ::= SEQUENCE {  
    tag    VisibleString,  
    val1   INTEGER,  
    val2   INTEGER OPTIONAL,  
    reals  SET OF REAL  
}
```

C

```
typedef struct CertainStructure {  
    VisibleString_t tag;  
    int            val1;  
    int            *val2;    /* OPTIONAL */  
    A_SET_OF(double) reals;  
} CertainStructure_t;
```

# An ASN.1 Compiler [2/2]

## Encoding and Decoding Data

```
CertainStructure_t *cs = 0;  
ber_decode(0, &asn_DEF_CertainStructure, &cs, buffer, buffer_length);  
cs->val1 = 123;    /* Modify the contents */  
ber_encode(&asn_DEF_CertainStructure, cs, write_handle, 0);
```

Online ASN.1 Compiler

<http://lionet.info/asn1c/asn1c.cgi>

## 2. Internet Management

1. Introduction

### 2. **Internet Management**

2.1 Overview

2.2 Structure the Management Information (SMIv2)

2.3 Fundamental MIBs

2.4 Simple Network Management Protocol Version 1 (SNMPv1)

2.5 Simple Network Management Protocol Version 2c (SNMPv2c)

2.6 Simple Network Management Protocol Version 3 (SNMPv3)

2.7 MIB Implementation and Agent Extensibility Protocol (AgentX)

## 2.1 Overview

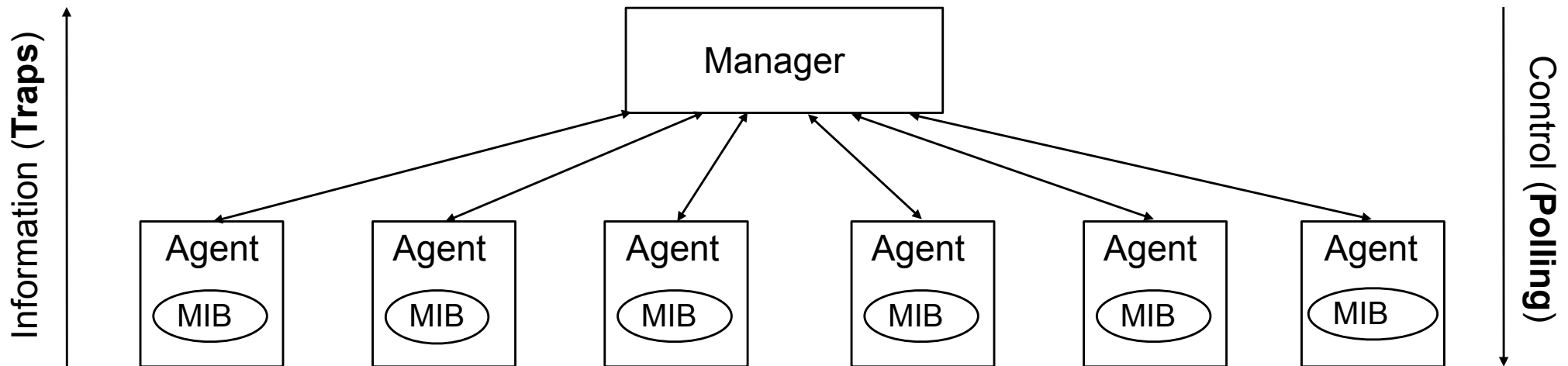
1987	Simple Gateway Monitoring Protocol (SGMP)	
1987	High-level Entity Management System (HEMS)	
1988	Simple Network Management Protocol (SNMPv1)	proposed
1990	Simple Network Management Protocol (SNMPv1)	standard 15, 16
1991	Management Information Base II	standard 17
1993	SNMP Version 2 (Party/Party/Context)	standard
1996	SNMP Version 2 (Communities)	standard
1998	SNMP Version 3 (User-based)	standard

- SNMPv1 has a large spreading particularly in data communication.
- The attempts for the standardisation of SNMPv2 failed.
- SNMPv3 with SNMPv1 has been accepted by a large community of network manufacturers.
- The user community has accepted SNMPv3 very well in terms of support and development.

# SNMP Development Goals

- Minimization of the number and complexity of the management functions, which are implemented by an agent:
  - Reduction of development costs for management agents (simple applications).
  - Ubiquity: use the same management technology for all devices (printers or Cray).
  - Application extensibility: development of new management functions without the need to modify the agents.
- Extensibility by defining new MIBs.
- Independence from existing computer or network architectures.
- Robustness by a simple, connectionless transport service (UDP).
- No dependency on other network services.
- Addition of management to new/existing devices/applications should be inexpensive, simple to develop and of limited functionality.
- Unfortunately some of these original goals have been lost: the term "simple" refers to the protocol and not to the specifications or the implementation of management applications.

# Trap Directed Polling



- SNMP managers poll in regular intervals the SNMP agents.
- Agents can signal exceptional cases to a manager by sending a trap.
- The SNMP manager can adapt the polling strategy upon the receipt of traps (trap directed polling).
- SNMP is a strictly centralised model, where the manager implements the whole functionality and responsibility.

# SNMP Application Areas

- SNMP can be used not only for network management:
  - control and monitoring of production processes.
  - control and monitoring of complex computer systems.
  - monitoring of complex application programs (relational databases, SAP R/3 components...).



- Many good SNMP toolkits are available on the market.
- Very few applications are available for solving complex management problems.
- The implementation of special applications or the conversion of local procedure guidelines is generally relatively complex and expensive.



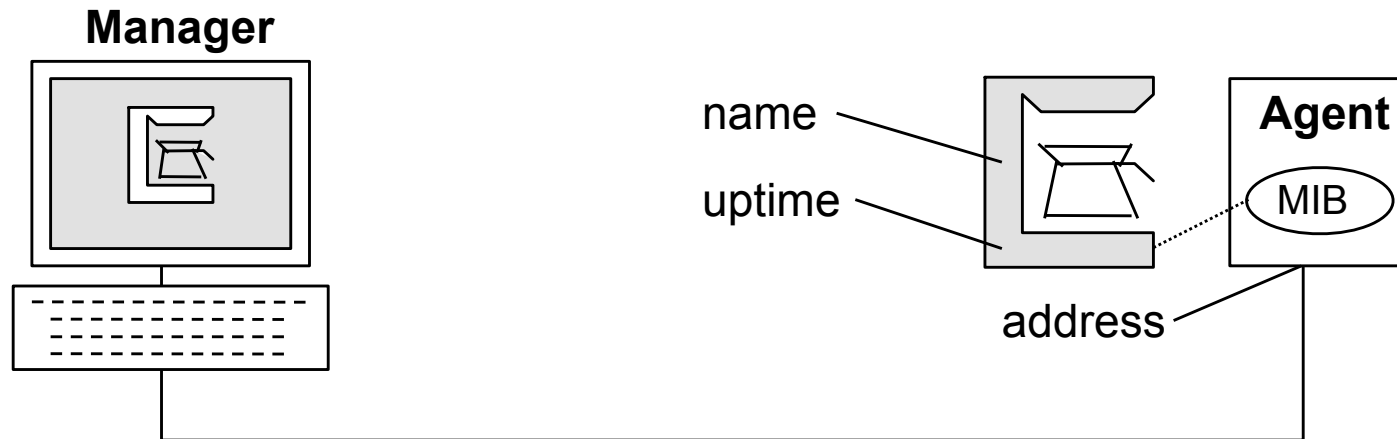
## 2.2 Structure the Management Information (SMIv2)

- The current information model known as "*Structure of Management Information version 2*" (SMIv2) is defined and based on simple typed variables.
- SMIv2 is based on extended subset of ASN.1 (1998).
- Each variable has a primitive, not assembled ASN.1 datatype:
  - INTEGER, OCTET STRING, OBJECT IDENTIFIER, NULL
  - Integer32, Unsigned32, Gauge32, Counter32, Counter64, IpAddress, TimeTicks, Opaque
- It does not implement complex data structures and operations on the variables.
- Variables are either **scalars** (exactly one instance) or columns in a “conceptual” two dimensional **table** (zero or several variables).
- On the variables only "**read**" and "**write**" operations can be applied. However the SNMP protocol permits the manipulation of lists of variables.
- SMIv2 management information Bases (MIBs) are defined using special ASN.1 macros.
- It leverages the complexity of new MIBs definitions: definition of basic functionality and primitive types to be used in new MIBs.

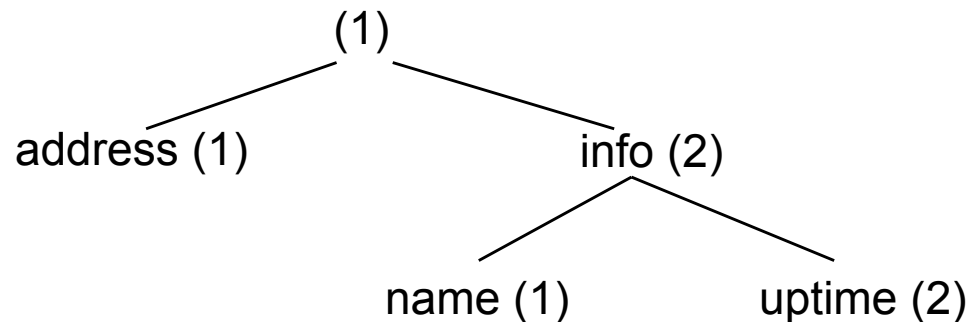
# SMIv2 Basic Datatypes (RFC 2578)

SMIv2	SMIv1	Description
INTEGER	INTEGER	Integer Numbers (-2147483648..2147483647)
OCTET STRING	OCTET STRING	Sequence of bytes (octets).
OBJECT IDENTIFIER	OBJECT IDENTIFIER	Unique identifier.
Integer32	INTEGER	32 bit Integers (-2147483648..2147483647)
Unsigned32	–	32 bit Positive Integers (0..4294967295)
Gauge32	Gauge	“Thermometer“ Integer (0..4294967295)
Counter32	Counter	32 bit non decreasing counter (0..4294967295)
Counter64	–	64 bit non decreasing counter (0..18446744073709551615)
TimeTicks	TimeTicks	Time in 1/100th of seconds
IpAddress	IpAddress	4 Byte IPv4 Address
Opaque	Opaque	Unspecified ASN.1 Type (not recommended)
BITS	–	Bits in a OCTET STRING
–	NetworkAddress	Network Address (not recommended)

# A MIB Use Case



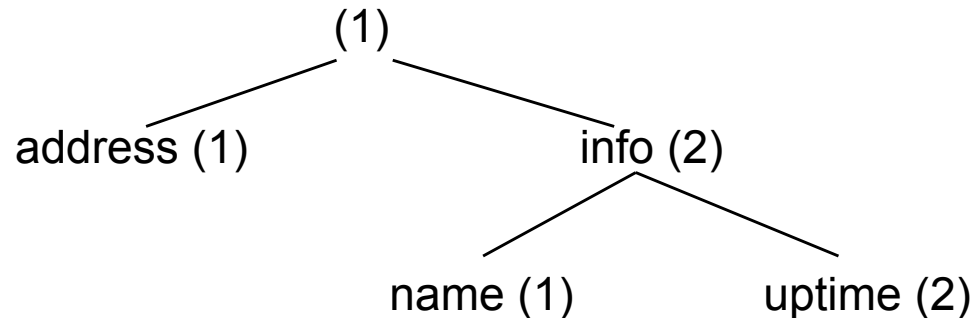
- Definition of the variables in the ISO Registration tree.
- Nodes are defined for naming purposes.
- The leaf of the tree represent the managed objects (i.e. “the meat”).
- Sub nodes can be used in order to logically organise the object types.



# Object Identifier and Instance Identifier

- In the registration tree each object can be identified by means of a unique object identifier.
- Concrete developments (instance) of a type of object are unique designated by a so-called *Instance Identifier*.
- A unique instance identifier is obtained by attaching an instance identifiers to the object identifier.
- Scalar object have basically **only one instance**, where the instance identifier has basically the value 0 (e.g. sysName.**0**).
- Instance identifiers for non-scalar variables are derived from the unique naming of a conceptual table.
- As object identifier can have up to 128 elements, hence instance names cannot be infinitely complex.

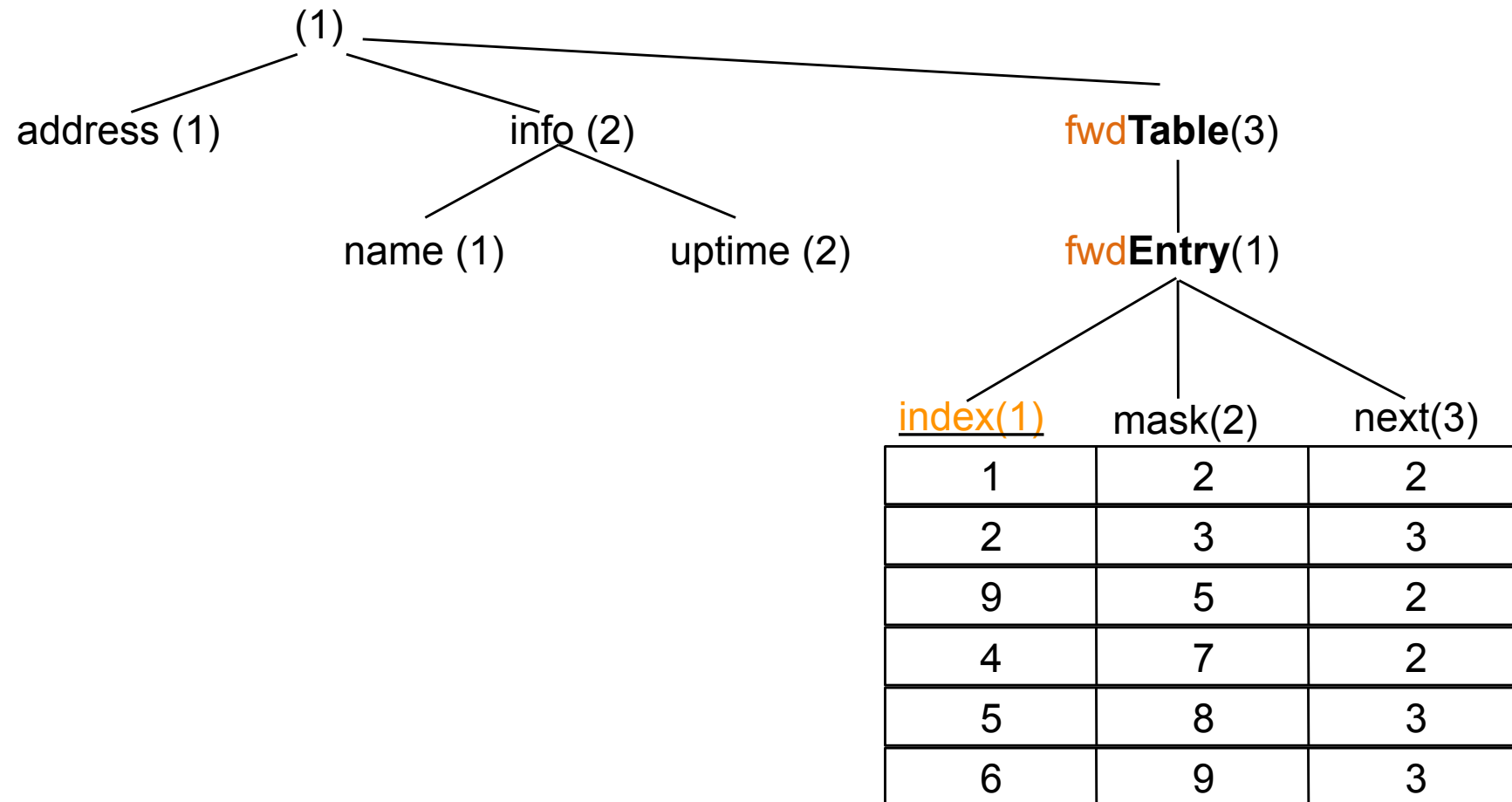
# Example of Object and Instance Identifiers



Object Identifier	Instance Identifier	Type	Value
1.1	0	IpAddress	10.1.2.1
1.2.1	0	OCTET STRING	"FilterFresh"
1.2.2	0	TimeTicks	54321

- MIB nodes names are relevant for human users only.
- Descriptors must be unique within a MIB module, although can be used several times in different MIB modules (one gets unique descriptors by the combining module names and descriptors).

# Extension of the Example MIB with a Routing table



For matter of simplicity in the above example addresses are represented using natural numbers.

# Identification of Table Entries

- Tables are defined basically with two "auxiliary nodes":
  - the first node defines the table and is of type `SEQUENCE OF`.
  - the second node defines an entry (a row) in the table and is of type `SEQUENCE`.
  - this is the only permitted use of `SEQUENCE` and `SEQUENCE OF` in SNMP SMIv2.
- The result of the column and instance identifier (code of the table) is a unique object identifier for each table entry.
- Table Example (convention `OID => value`):

1.3.1.1.1 => 1	1.3.1.3.1 => 2	1.3.1.2.4 => 7
1.3.1.2.1 => 2	1.3.1.1.4 => 4	1.3.1.2.7 =>
not existing		

# Tables Naming [1/3]

- Table naming is very important as it affects the way tables are accessed.
- Two kind of tables naming:
  - Use row numbers (not being used by SNMP).

1	2	2
2	3	3
3	5	2
4	7	2
5	8	3

← This is row number 3

- Use an index column (the SNMP way).

↓ This is the index column

1	2	2
2	3	3
3	5	2
4	7	2
5	8	3



## Tables Naming [2/3]

- A table index is not necessarily (but often is) an INTEGER. For instance the routingTable uses an IP address as table index.
- A table index can be made of several components:

- X . C . I1 . I2 ..... In

OID of the table

Column number

Index value 1

Index value n

routingTable

policy (2) 1 = low cost  
2 = high reliability

destination (1)		next (3)
130.89.16.23	1	130.89.16.23
130.89.16.23	2	130.89.16.127
192.168.10.12	1	172.16.1.18
192.168.10.12	2	172.16.1.12

# Tables Naming: Complex Table Indexes [3/3]

- An IP Routing table is the combination of IP address and the IP netmask necessary to satisfy the routing rules.

- The individual bytes of the IP address are specified as individual sub identifiers.

- Example:

$$\begin{array}{c}
 \text{Instance Identifier} \\
 \hline
 1.3.1.1 \quad 134.169.35.1 \quad 255.255.255.0 \Rightarrow 134.169.35.1 \\
 \begin{array}{cc}
 \hline
 \text{net} & \text{mask} \\
 \hline
 \end{array}
 \end{array}$$

$$1.3.1.3 \quad 134.169.34.0 \quad 255.255.255.0 \Rightarrow 134.169.34.15$$

Diagram illustrating the structure of a forwarding table:

```

graph TD
    fwdTable3[fwdTable(3)] --> fwdEntry1[fwdEntry(1)]
    fwdEntry1 --> net1[net(1)]
    fwdEntry1 --> mask2[mask(2)]
    fwdEntry1 --> next3[next(3)]
  
```

<u>net(1)</u>	<u>mask(2)</u>	<u>next(3)</u>
0.0.0.0	255.255.255.0	134.169.34.1
127.0.0.1	255.0.0.0	127.0.0.1
134.169.34.0	255.255.255.0	134.169.34.15
134.169.35.1	255.255.255.0	134.169.34.18
134.169.35.2	255.255.255.0	134.169.34.18

# MIB Module

- Similar object types are combined into MIB modules.
- Each MIB module must have a unique name (uppercase letters).
- MIB modules are (almost) normal ASN.1 modules and obey to the lexical ASN.1 rules.
- Definitions can be imported by other MIB modules with the help of the ASN.1 `IMPORT` statement.
- All used ASN.1 SMI Macros must be explicitly imported

```
COFFEE-MIB DEFINITIONS ::= BEGIN
```

```
IMPORT      MODULE-IDENTITY, OBJECT-TYPE, enterprises,  
            IpAddress, TimeTicks    FROM SNMPv2-SMI;
```

```
...
```

```
END
```

# Module-Identities (RFC 2578)

```
<descriptor> MODULE-IDENTITY
    LAST-UPDATED <ExtUTCTime>
    ORGANIZATION <Text>
    CONTACT-INFO <Text>
    DESCRIPTION  <Text>
    [ REVISION   <ExtUTCTime>
      DESCRIPTION <Text> ] *
 ::= <ObjectIdentifier>
```

- Defines administrative information e.g. contact information and version number.
- the REVISION and DESCRIPTION clauses are not mandatory and can occur several times.
- ExtUTCTime contains a date in the format „YYMMDDHHMMZ“ (UTC) or „YYYYMMDDHHMMZ“, e.g.. „9502192015Z“ or „199502192015Z“.

# Module-Identities (RFC 2578)

```
IF-MIB DEFINITIONS ::= BEGIN

IMPORTS ...

ifMIB MODULE-IDENTITY

    LAST-UPDATED "9611031355Z"

    ORGANIZATION "IETF Interface MIB Working Group"

    CONTACT-INFO "  Keith McCloghrie          408-526-5260
                  Cisco Systems, Inc.      kzm@cisco.com
                  170 West Tasman Drive
                  San Jose, CA 95134-1706, US"

    DESCRIPTION "The MIB module to of describe generic objects for network interface
                  sub-layers. This MIB is an updated version of MIB II's ifTable,
                  and incorporates the extensions defined in RFC 1229."

    REVISION      "9602282155Z"

    DESCRIPTION "Revisions made by the Interfaces MIB WG"

    REVISION      "9311082155Z"

    DESCRIPTION "Initial revision, published as part of RFC 1573."

    ::= { mib-2 31 }

...

END
```

# Object Identities (RFC 2578)

```
<descriptor> OBJECT-IDENTITY
    STATUS          <Status>
    DESCRIPTION     <Text>
    [ REFERENCE     <Text> ]
    ::= <ObjectIdentifier>
```

- Defines and registers an object identifier value.
- Permits the allocation of any node within the registration tree.
- The STATUS clause defines whether the allocated node is "obsolete" "current", or "deprecated".
- The optional REFERENCE is used to refer to further information (similar to HTML hyperlinks).

# Example of Object Identities (RFC 2578, RFC 1906)

zeroDotZero OBJECT-IDENTITY

STATUS current

DESCRIPTION

"A value used for null Identifiers."

::= { 0 0 }

snmpUDPDomain OBJECT-IDENTITY

STATUS current

DESCRIPTION

"The SNMPv2 over UDP transport domain. The corresponding  
transport address is of type SnmpUDPAddress."

::= { snmpDomains 1 }

snmpIPXDomain OBJECT-IDENTITY

STATUS current

DESCRIPTION

"The SNMPv2 over IPX transport domain. The corresponding  
transport address is of type SnmpIPXAddress."

::= { snmpDomains 5 }

# Object Types (RFC 2578)

```
<descriptor> OBJECT-TYPE
    SYNTAX          <Syntax>
    [ UNITS          <Text> ]
    MAX-ACCESS      <Access>
    STATUS          <Status>
    DESCRIPTION     <Text>
    [ REFERENCE      <Text> ]
    [ INDEX          <Index> ]
    [ AUGMENTS       <Index> ]
    [ DEFVAL         <Value> ]
    ::= <ObjectIdentifier>
```

- Macro for the definition of object types and conceptual tables.
- The INDEX and AUGMENTS clauses are permitted only for the definition by tables.
- Exactly one of the above clauses must be specified during table definition.



# Example for ObjectTypes (RFC 2012)

tcpRtoMin OBJECT-TYPE

SYNTAX Integer32

UNITS "milliseconds"

MAX-ACCESS read-only

STATUS current

DESCRIPTION

"The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBO and quantity of described in RFC 793."

::= { tcp 2 }

# Example for ObjectTypes (RFC 1907)

## `sysORTable` OBJECT-TYPE

SYNTAX SEQUENCE OF SysOREntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"The (conceptual) table listing the capabilities of the local SNMPv2 entity acting in an agent role with respect to various MIB modules. SNMPv2 entities having dynamically-configurable support of MIB modules will have a dynamically-varying number of conceptual rows."

::= { system 9 }

## `sysOREntry` OBJECT-TYPE

SYNTAX SysOREntry

MAX-ACCESS not-accessible

STATUS current

DESCRIPTION

"An entry (conceptual row) in the sysORTable."

INDEX { sysORIndex }

::= { `sysORTable` 1 }

# Notification-Types (RFC 2578)

```
<descriptor> NOTIFICATION-TYPE
    [ OBJECTS      <Objects> ]
    STATUS        <Status>
    DESCRIPTION   <Text>
    [ REFERENCE   <Text> ]
    ::= <ObjectIdentifier>
```

- Macro for the registration of an event.
- In case of event a manager or an agent can send an appropriate notification to another manager.
- The OBJECTS clause defines which MIB objects must be contained in the event description.
- The DESCRIPTION clause must describe which instances are meant in each case.

# Example for Notification Types (RFC 2233)

## linkDown NOTIFICATION-TYPE

OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }

STATUS current

### DESCRIPTION

"A linkDown trap signifies that the SNMPv2 entity, acting in an agent role, has detected that the ifOperStatus object for one of its communication links is about to enter the down state from some other state (but not from the notPresent state). This other state is indicated by the included value of ifOperStatus."

::= { snmpTraps 3 }

## linkUp NOTIFICATION-TYPE

OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }

STATUS current

### DESCRIPTION

"A linkDown trap signifies that the SNMPv2 entity, acting in an agent role, has detected that the ifOperStatus object for one of its communication links left the down state and transitioned into some other state (but not into the notPresent state). This other state is indicated by the included value of ifOperStatus."

::= { snmpTraps 4 }

# New Types from Textual Conventions

- Textual conventions allow new types to be derived from SMIv2 base types.
- However, additional types may not be derived from a textual convention.
- A `DISPLAY-HINT` clause defines a simple figure of the ASN.1 representation of a value into a format readable for humans.
- The `DISPLAY-HINT` clause can be used only together with the `INTEGER` and `OCTET STRING` datatype and from which it derives.
- A Textual convention can determine restrictions on the scope.
- A Textual convention cannot define an assembled type.

# Textual Conventions [1/2]

- Textual conventions are defined in RFC 2579.

`<descriptor> ::= TEXTUAL-CONVENTION`

`[ DISPLAY-HINT <Text> ]`

`STATUS <Status>`

`DESCRIPTION <Text>`

`[ REFERENCE <Text> ]`

`SYNTAX <Syntax>`

- The `DISPLAY-HINT` clause defines a bi-directional figure of the internally used representation on a representation readable for humans. .
- In the `SYNTAX` clause only base datatypes may be used (one can thus limit not existing Textual Conventions even further).
- All further semantics must be defined in the `DESCRIPTION` clause.

## Textual Conventions [2/2]

- The followings are the textual conventions defined in RFC 2579:
  - PhysAddress
  - MacAddress
  - TruthValue
  - AutonomousType
  - InstancePointer
  - VariablePointer
  - RowPointer
  - RowStatus
  - TimeStamp
  - TimeInterval
  - DateAndTime
  - StorageType
  - TDomain
  - TAddress

# INTEGER DISPLAY-HINTS

Format	Description
d	Representation of an Integer
d-<number>	Representation of `d` with a decimal point
o	Octal Representation
x	Hex Representation

- Example:
  - `""d""` stands for `""143""`
  - `""d-2""` stands for `""1.43""`
  - `""o""` stands for `""217""`
  - `""x""` stands for `""8F""`



# OCTET STRING DISPLAY-HINTS

- `[<repeat>]<number><format>[separator][terminator]`

Field	Description (similar to C/C++ <code>printf</code> )
<code>&lt;repeat&gt;</code>	Indicator for the specification repetition
<code>&lt;number&gt;</code>	# bytes in the following format field
<code>&lt;format&gt;</code>	Format (a ASCII, d Decimal, x Hexadecimal, o Octal, t UTF8)
<code>&lt;separator&gt;</code>	Separator among multiple values
<code>&lt;terminator&gt;</code>	Terminator specified at the end of the rule

- Example:
  - `“255a”` format for the ASCII characters `“aBc”` in the string `“aBc”`
  - `“1x:”` format for the ASCII characters `“aBc”` in the string `“61:42:63”`
  - `“0aH0ae0a10a10ao0a 1a”`  
format for the ASCII characters `“World”` in the string `“Hello World”`

# Example for Textual-Conventions (RFC 2579)

RunState ::= TEXTUAL-CONVENTION

STATUS current

DESCRIPTION

"This TC of describes the current execution state of  
a running application or process."

SYNTAX INTEGER {

running(1), runnable(2),

waiting(3), exiting(4), other(5)

}

MacAddress ::= TEXTUAL-CONVENTION

DISPLAY-HINT "1x:"

STATUS current

DESCRIPTION

"Represents an 802 MAC address represented in the  
'canonical' or the defined by IEEE 802.1a, i.e., as if it  
were transmitted least significant bit first, even though  
802.5 (in contrast to other 802.x protocols) requires MAC  
addresses to be transmitted most significant bit first."

SYNTAX OCTET STRING (SIZE (6))

# Example for Textual-Conventions (RFC 2579)

DateAndTime ::= TEXTUAL-CONVENTION

DISPLAY-HINT "2d-1d-1d,1d:1d:1d.1d,1a1d:1d"

STATUS current

DESCRIPTION

"A date-time specification.

field	octets	contents	range
-----	-----	-----	-----
1	1-2	year	0..65536
2	3	month	1..12
3	4	day	1..31
4	5	hour	0..23
5	6	minutes	0..59
6	7	seconds	0..60
		(use 60 for leap-second)	
7	8	deci-seconds	0..9
8	9	direction from UTC	'+' / '-'
9	10	hours from UTC	0..11
10	11	minutes from UTC	0..59

For example, Tuesday May 26, 1992 at 1:30:15 PM EDT would be displayed as:

1992-5-26,13:30:15.0,-4:0

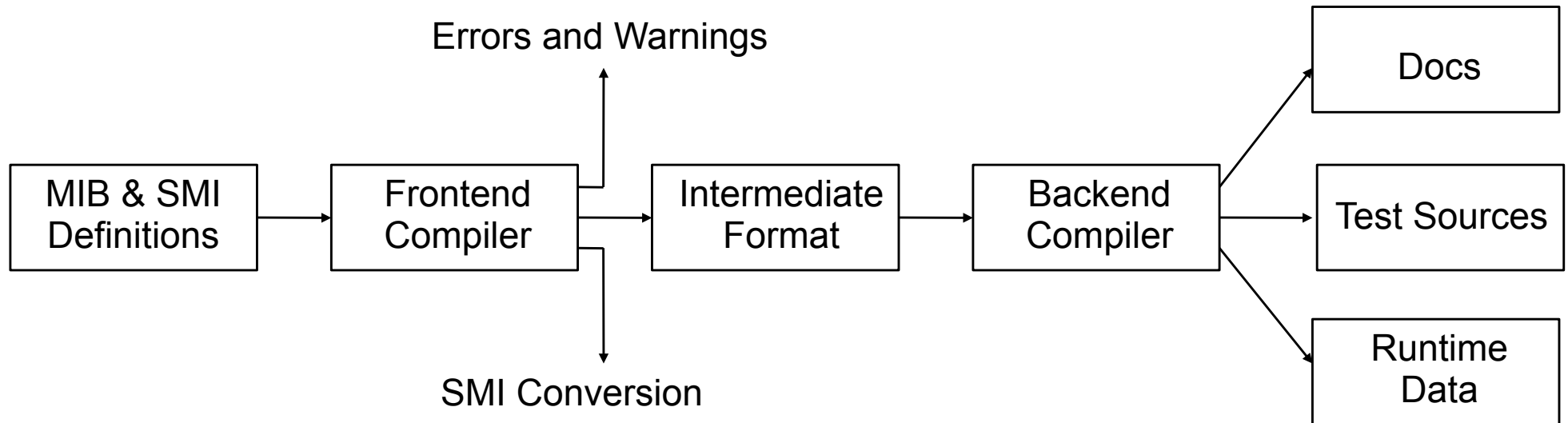
Note that if only local time is known, then timezone information (fields 8-10) is not present."

SYNTAX OCTET STRING (SIZE (8 | 11))

# Further SMIv2 Macros

- OBJECT-GROUPS
  - It enables the definition of groups of related object types.
  - This macro can be used in the MODULE-COMPLIANCE macro.
- NOTIFICATION-GROUPS
  - It enables the definition of groups of related notification types.
  - This macro can be used in the MODULE-COMPLIANCE macro.
- MODULE-COMPLIANCE
  - It defines one or more constraints that a MIB implementations must fulfil.
- AGENT-CAPABILITIES
  - It describes the capabilities of a real MIB implementation.

# MIB-Compiler

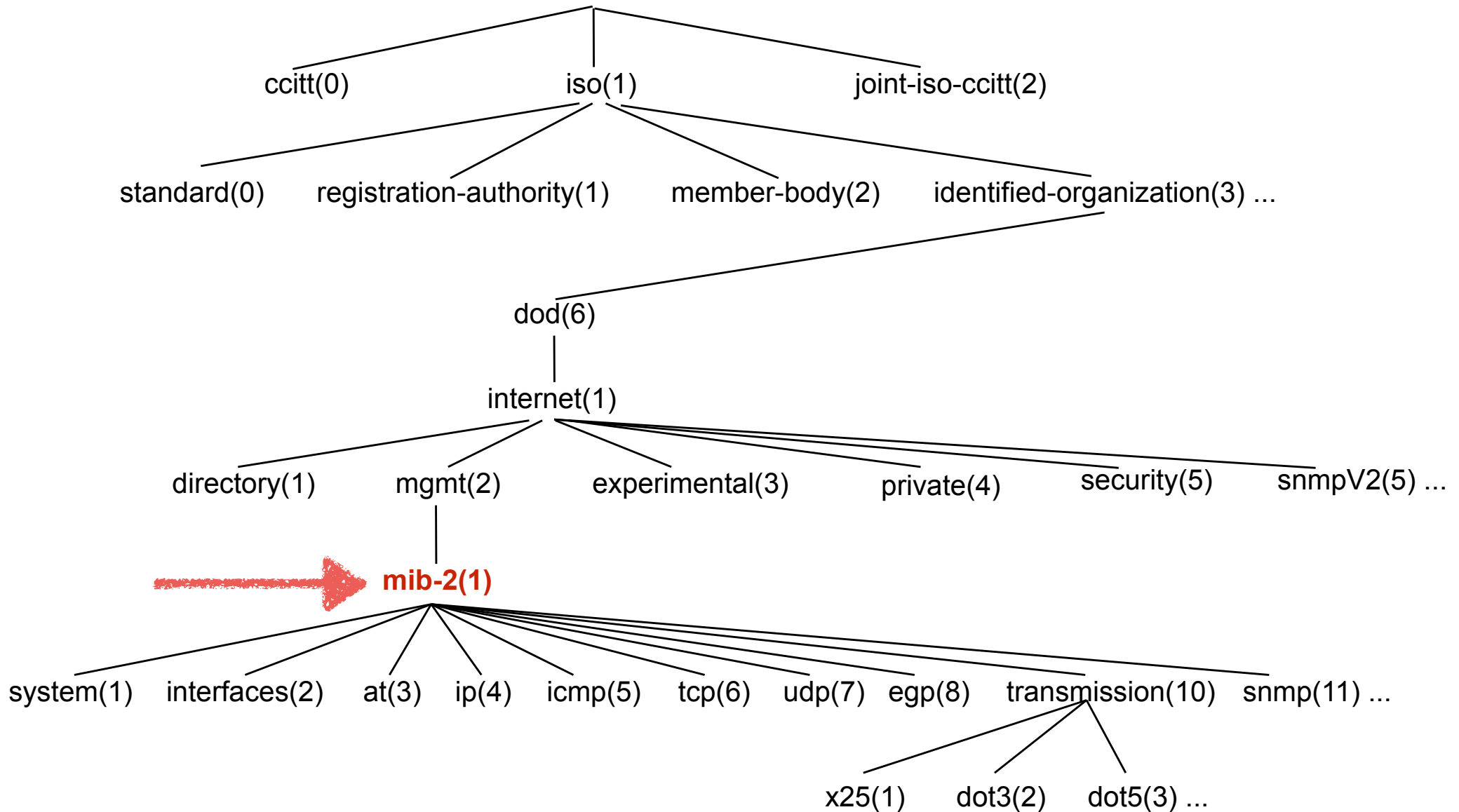


- Backend-Compiler can produce the following outputs:
  - Documentation (hypertext versions of MIB modules, diagrams)
  - Source code for the semiautomatic implementation of agents
  - Test-cases for testing manager and agent implementations
  - Inputs for management applications, the MIB definitions needed at run-time.
- There is no standardised or generally accepted intermediate format.

## 2.3 Fundamental MIBs

- MIB-II (RFC 1213) defines object types for the Internet Protocols IP, ICMP, UDP, TCP, SNMP (and other definitions not relevant here). Basically it models the management of the TCP/IP protocol stack.
- Goals of the MIB-II definition:
  - Define basic error and configuration management for Internet protocols.
  - Very few and weak control objects.
  - Avoidance of redundant information in the MIB.
  - MIB implementation should not interfere with the normal network activities.
  - No implementation-dependent object types.
- Altogether 170 object types.
- Some MIB definitions turned out to be too simple and minimal (Routing table, Interface table).
- Some MIB definitions presuppose a 4-Byte address format, hence these tables must be redefined for IP version 6 (IPv6).

# Registration and Structure of MIB-II



## Relations Between MIBs [1/2]

	MIB-II	Host	Repeater	Bridge	RMON
Interface Statistics	X				
IP, TCP & UDP Statistics	X				
SNMP Statistics	X				
Host Job Counts		X			
Host File System Information		X			
Link Testing			X	X	
Network Traffic Statistics			X	X	X
Address Tables			X		X
Host Statistics			X		X

Overlap

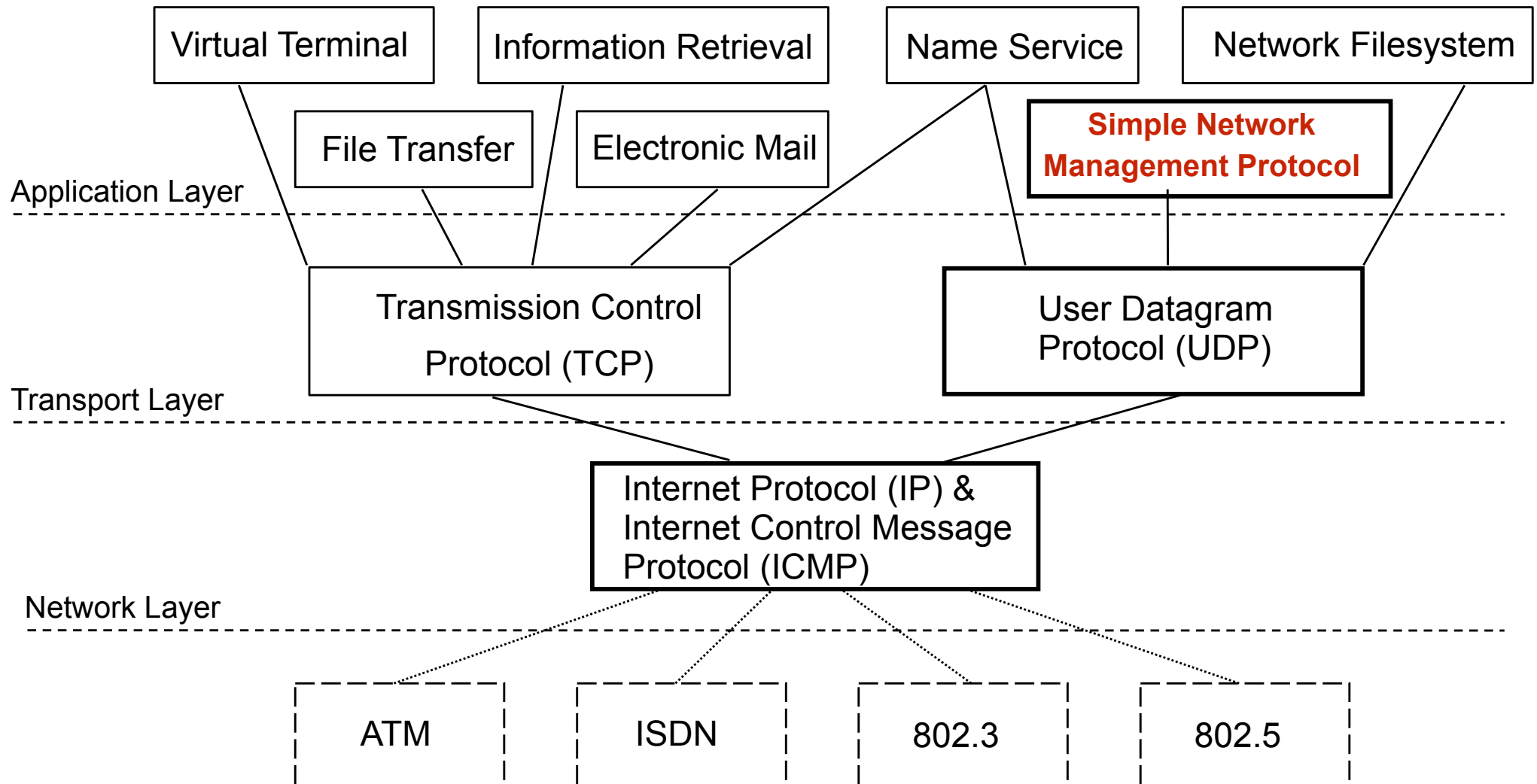




## Relations Between MIBs [2/2]

	MIB-II	Host	Repeater	Bridge	RMON
Historical Statistics				X	
Spanning Tree Performance			X		
Wide Area Link Performance			X		
Thresholds for any variable				X	
Configurable Statistics				X	
Traffic Matrix with all Nodes				X	
'Host Top N' Information					X
Packet/Protocol Analysis					X
Distributed Logging				X	

## 2.4 Simple Network Management Protocol Version 1



# Lexicographical Ordering

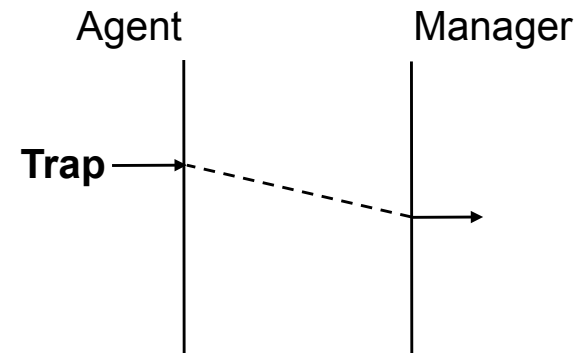
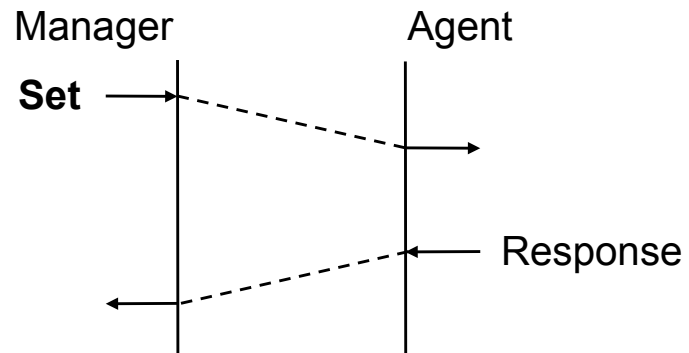
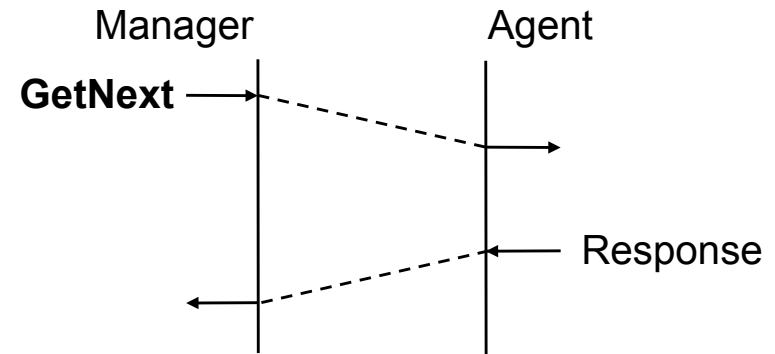
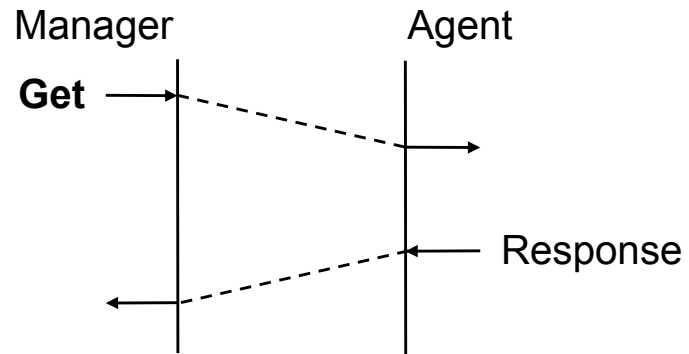
- MIB instances are arranged in the MIB according to their lexicographical ordering.
- The ordering is determined by the value of the object identifier that identify the instance.
- The SNMP log uses the lexicographical order, in order to read (walk) conceptual tables or unknown MIBs.

## Example of Lexicographical Ordering

Object Identifier:	Value:	Object Identifier	Value :
1.1.0	10.1.2.3	1.3.1.2.4	7
1.2.1.0	"FilterFresh"	1.3.1.2.5	8
1.2.2.0	54321	1.3.1.2.6	9
1.3.1.1.1	1	1.3.1.3.1	2
1.3.1.1.2	2	1.3.1.3.2	3
1.3.1.1.3	3	1.3.1.3.3	2
1.3.1.1.4	4	1.3.1.3.4	2
1.3.1.1.5	5	1.3.1.3.5	3
1.3.1.1.6	6	1.3.1.3.6	3
1.3.1.2.1	2		
1.3.1.2.2	3		
1.3.1.2.3	5		

- With this ordering the conceptual table structure is lost as the walk output is a list and no longer a table.
- the SNMP protocol operates only on this arranged list.

# SNMPv1 protocol operations (RFC 1157)



Note: the SNMP protocol can only exchange (a list of) scalars.

# SNMPv1 Message Format

## SNMP message:

version	community	SNMP PDU
---------	-----------	----------

## GetRequest, GetNextRequest, SetRequest:

PDU type	request-id	0	0	variable-bindings
----------	------------	---	---	-------------------

## GetResponse:

PDU type	request-id	error-status	error-index	variable-bindings
----------	------------	--------------	-------------	-------------------

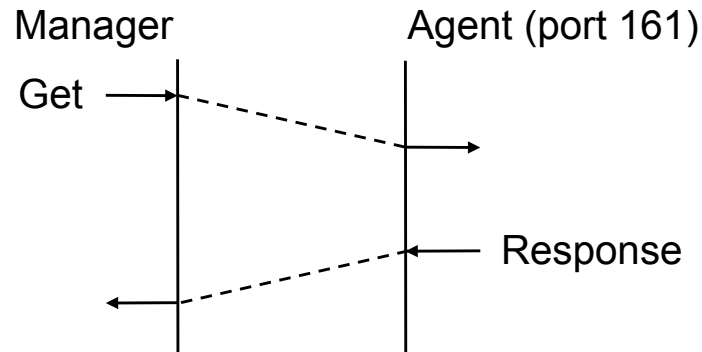
## Trap:

PDU type	enterprise	address	generic	specific	timestamp	vbs
----------	------------	---------	---------	----------	-----------	-----

## variable-bindings:

name <sub>1</sub>	value <sub>1</sub>	name <sub>2</sub>	value <sub>2</sub>	...	name <sub>n</sub>	value <sub>n</sub>
-------------------	--------------------	-------------------	--------------------	-----	-------------------	--------------------

# SNMPv1 Get Operation



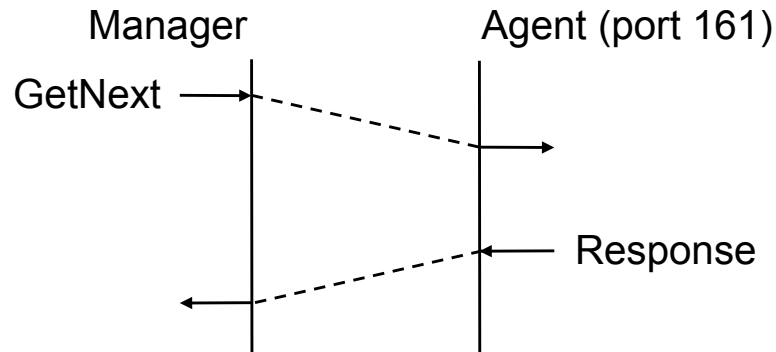
- The Get operation can be used for reading one or more variables.
- Possible errors when processing a GET operation:
  - noSuchName            the requested instance does not exist or is not a leaf.
  - tooBig                the result of the request does not fit not into the response (UDP).
  - genErr                any other error occurred.
- In the case of several errors occurred, only one error is signalled as error-index and error-status are unique in the PDU.

## Example of Get Operation

- `Get(1.1.0)`  
`Response(noError@0, 1.1.0=10.1.2.3)`
- `Get(1.2.0)`  
`Response(noSuchName@1, 1.2.0)`
- `Get(1.1)`  
`Response(noSuchName@1, 1.1)`
- `Get(1.1.0, 1.2.2.0)`  
`Response(noError@0, 1.1.0=10.1.2.3, 1.2.2.0=54321)`
- `Get(1.3.1.1.4, 1.3.1.3.4)`  
`Response(noError@0, 1.3.1.1.4=4, 1.3.1.3.4=2)`
- `Get(1.1.0, 1.2.2.0, 1.1)`  
`Response(noSuchName@3, 1.1.0, 1.2.2.0, 1.1)`



# SNMPv1 GetNext Operation

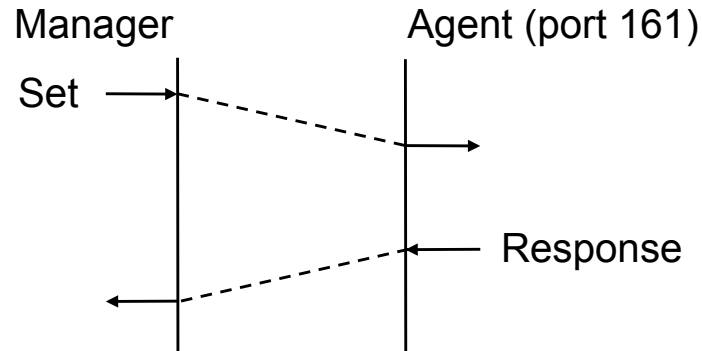


- It retrieves the object name and the value of the next instance. This operation is used to discover MIB structures and read tables.
- The GetNext operation allows MIB instances to be read in accordance to the lexicographical order.
- Using multiple/successive GetNext operations it is possible to read the complete MIB without knowing its structure.
- Possible errors when processing a GetNext Operation:
  - noSuchName the requested instance does not exist (= end of MIB).
  - tooBig the result of the request does not fit not into the response (UDP).
  - genErr any other error occurred.

# Example of GetNext Operation

- `GetNext(1.1.0)`  
`Response(noError@0, 1.2.1.0=FilterFresh)`
- `GetNext(1.2.1.0)`  
`Response(noError@0, 1.2.2.0=54321)`
- `GetNext(1.1)`  
`Response(noError@0, 1.1.0=10.1.2.3)`
- `GetNext(1.3.1.1.1)`  
`Response(noError@0, 1.3.1.1.2=2)`
- `GetNext(1.3.1.1.6)`  
`Response(noError@0, 1.3.1.2.1=2)`
- `GetNext(1.3.1.1.1, 1.3.1.2.1, 1.3.1.3.1)`  
`Response(noError@0, 1.3.1.1.2=2, 1.3.1.2.2=3, 1.3.1.3.2=3)`

# SNMPv1 Set Operation

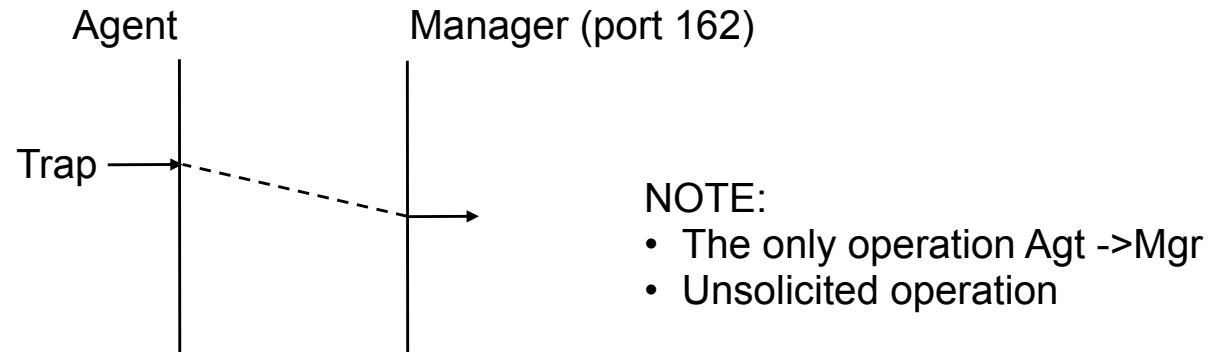


- The Set Operation writes values in one or more MIB instances.
- The Set Operation is atomic.
- With the help of the set operation new MIB instances can also be created, if the MIB definition permits (there is no standard procedure defined in SNMPv1 for instance creation).
- Possible errors when processing a Set operation:
  - noSuchName the requested instance does not exist and cannot be created.
  - badValue the specified value is of wrong type.
  - tooBig the result of the request does not fit not into the response (UDP).
  - genErr any other error occurred.
- The error code readOnly is also defined, but not usually used!

# Example of Set Operation

- `Set(1.2.1.0=HotJava)`  
`Response(noError@0, 1.2.1.0=HotJava)`
- `Set(1.1.0=foo.bar.com)`  
`Response(badValue@1, 1.1.0=foo.bar.com)`
- `Set(1.1.1=10.2.3.4)`  
`Response(noSuchName@1, 1.1.1=10.2.3.4)`
- `Set(1.2.1.0=HotJava, 1.1.0=foo.bar.com)`  
`Response(badValue@2, 1.2.1.0=HotJava, 1.1.0=foo.bar.com)`
- `Set(1.3.1.1.8.1=7, 1.3.1.2.7=2, 1.3.1.3.7=3)`  
`Response(noError@0, 1.3.1.1.8.1=7, 1.3.1.2.7=2, 1.3.1.3.7=3)`

# SNMPv1 Trap Operation

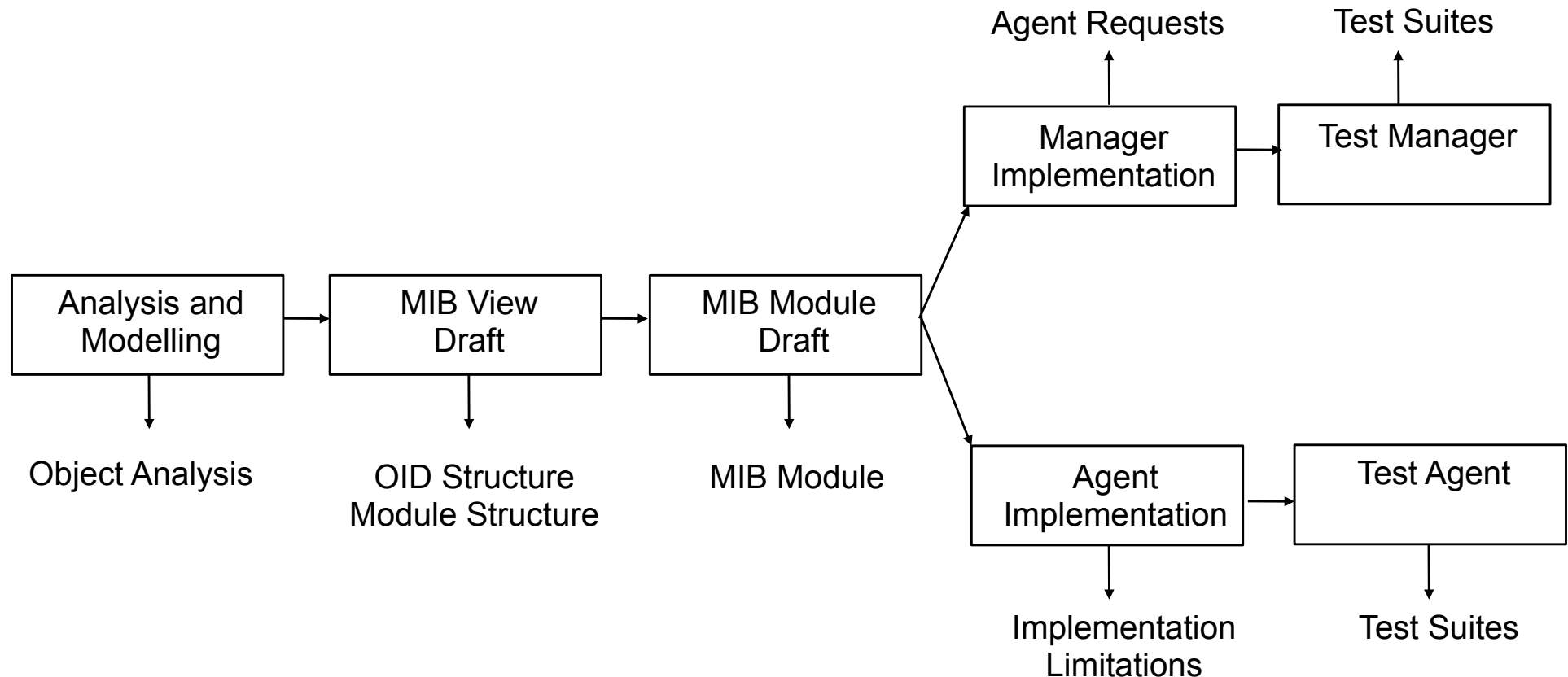


- With the trap operation and agent can emit an event and inform a manager. Note: a manager can be configured to discard traps!
- The receipt of a trap operation is not acknowledged thus is unreliable as it can be lost during the transfer.
- The production of traps can lead to so-called *trap storms*, if e.g. after a power failure all devices want to display the restart at the same time.
- Agents can be normally configured with the IP addresses of hosts where traps can be dispatched. However there is no standard technique in SNMPv1 for such agent configuration. Usually a configuration file (not the MIB) is used.
- Although if traps are used, polling is still necessary (for instance the agent might be down)

# Example of SNMPv1 Trap Operation

- ColdStart  
Trap(generic=0, specific=0)
- WarmStart  
Trap(generic=1, specific=0)
- LinkDown  
Trap(generic=2, specific=0, 1.3.6.1.2.1.2.2.1.1.2=2)
- LinkUp  
Trap(generic=3, specific=0, 1.3.6.1.2.1.2.2.1.1.2=2)
- AuthenticationFailure  
Trap(generic=4, specific=0)
- EnterpriseSpecific (QMS, qmsPtrErrorMsg)  
Trap(generic=6, specific=1, enterprise=1.3.6.1.4.1.480,  
1.3.6.1.4.1.480.2.1.1.1=out of paper)

# Agent MIB Implementation



- It is possible for have several iterative phases for the MIB definitions until it is in draft status.
- MIB definitions cannot however be further changed, if they were released.

## SNMP MIB II: Introduction

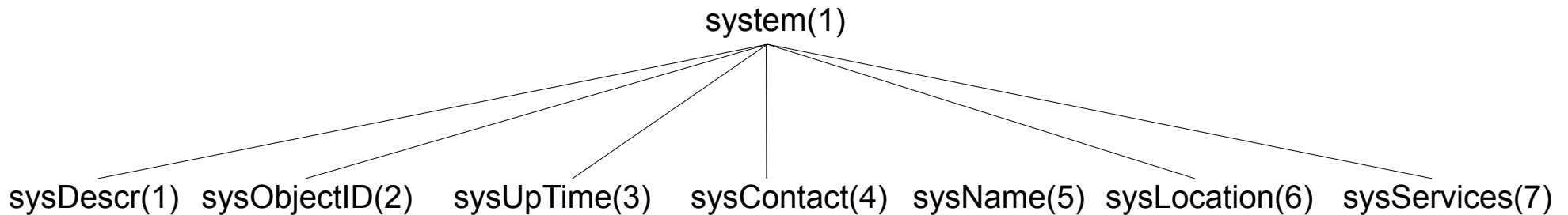
- MIB-II (RFC 1213) defines object types for the Internet Protocols IP, ICMP, UDP, TCP, SNMP (and other definitions not relevant here). Basically it models the management of the TCP/IP protocol stack.
- Altogether 170 object types.
- Some MIB definitions turned out to be too simple and minimal (Routing table, Interface table).
- Some MIB definitions presuppose a 4-Byte address format, hence these tables must be redefined for IP version 6 (IPv6).



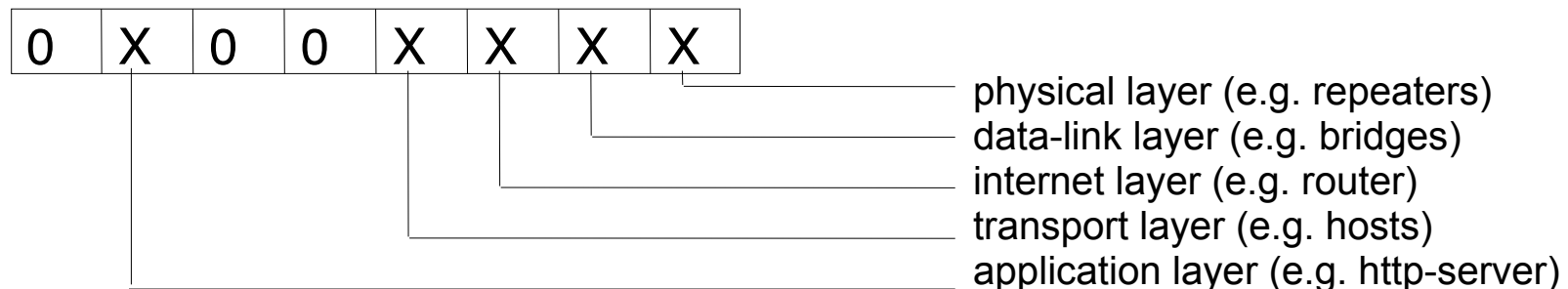
## SNMP MIB II: Goals

- Goals of the MIB-II definition:
  - Define basic error and configuration management for Internet protocols.
  - Very few and weak control objects.
  - Avoidance of redundant information in the MIB.
  - MIB implementation should not interfere with the normal network activities.
  - No implementation-dependent object types.

## "system" Group [1/2]



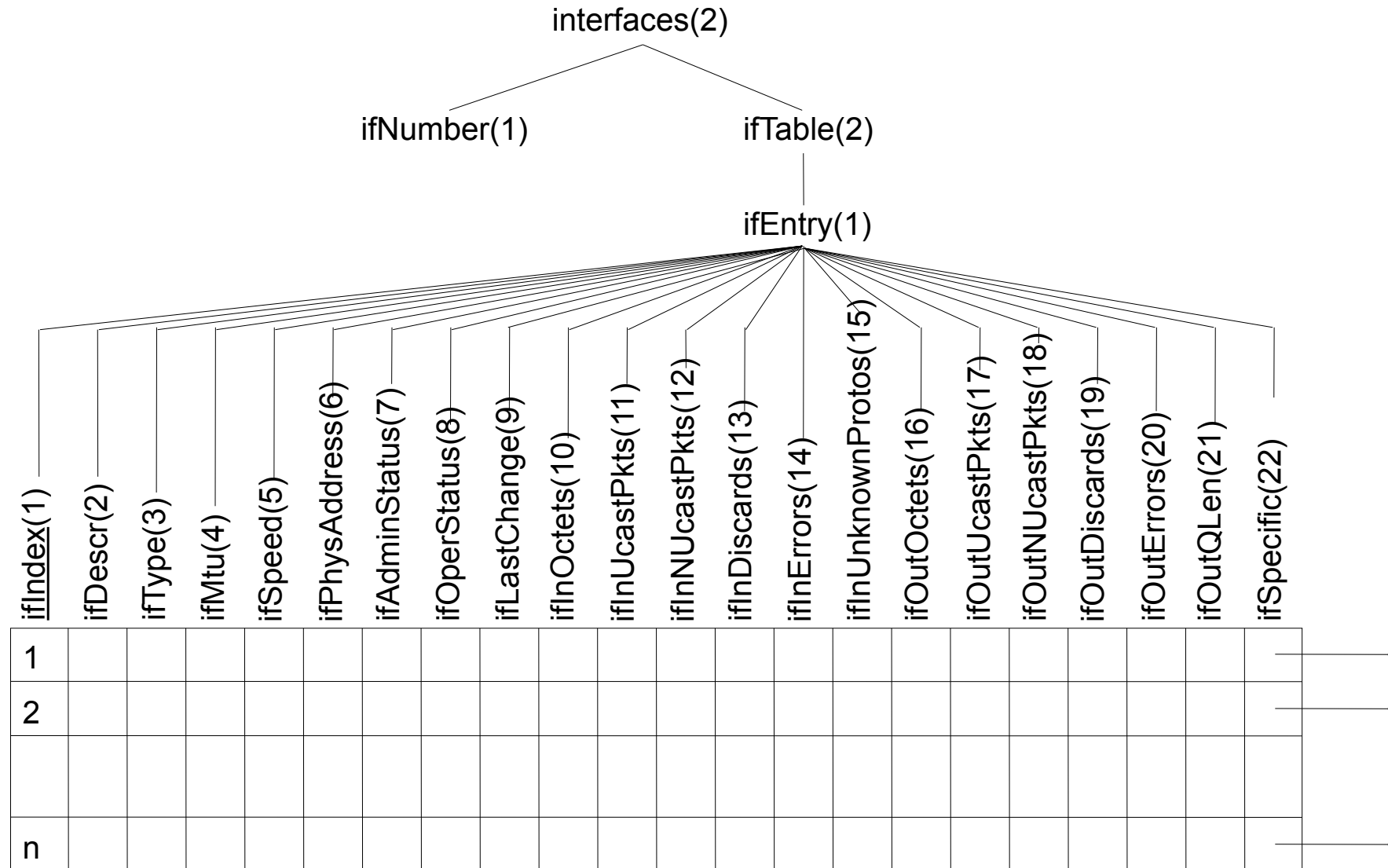
- `sysUpTime.0` is a very important variable as it is used for determining service discontinuities:
  - If  $\text{sysUpTime.0}_{t_1} > \text{sysUpTime.0}_{t_2}$  where  $t_2 > t_1$  then the agent has been reinitialised and management application rely on previous values.
- `sysServices` roughly reports the services supplied by the system:



## "system" Group [2/2]

- sysObjectld.0 has the format enterprises.<manufacturer>.<id>+ and it is used to identify manufacturer and model. For instance enterprises.9.1.208 identifies a Cisco (.9) 2600 router (.1.208).
- sysDescr.0 provides a precise description of the device (e.g. “Cisco Internetwork Operating System Software IOS (tm) C2600 Software (C2600-I-M), Version 12.2(23), RELEASE SOFTWARE (fc2) Copyright (c) 1986-2004 by cisco Systems, Inc.”)
- In a nutshell the system group is important for:
  - Device mapping (via sysObjectld.0, sysDescr.0, and sysLocation.0)
  - Counter wrapping check (sysUpTime.0)
  - Reporting problems about the device to the administrator (sysContact.0)

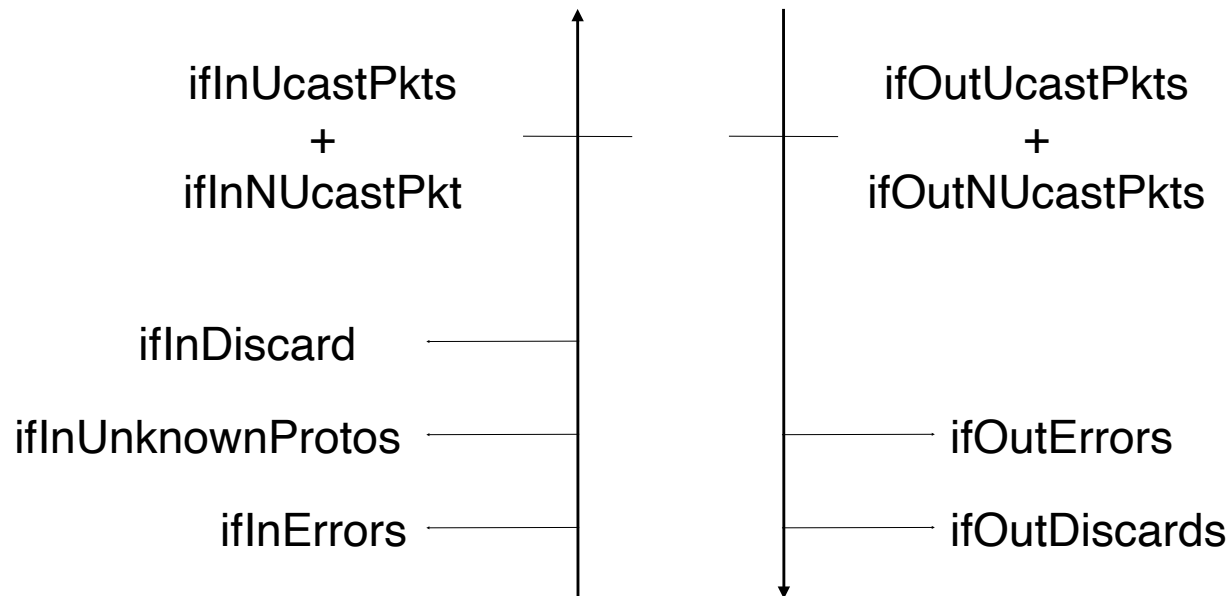
# "interface" Group



# "interface" Group Variables

- ifAdminStatus: the current administrative state of the interface. Values: up(1), down(2), testing(3). A value different from up means that the interface is **not physically present** on the system or that it's present but unavailable to the operating system (e.g. the driver has not been loaded).
- Caveat: SNMP MIB index holes
- ifOperStatus: the current operational state of the interface. Values: up(1), down(2), testing(3). It is similar to `ifconfig <device> up/down`.
- ifOutQLen: the length of the output packet queue (in packets). It is useful for knowing more about transmission speeds and throughput (buffer full means that the receiver is not as fast as the sender).
- ifLastChange contains the value of sysUpTime at the time the interface entered its current operational state. Useful for detecting when an interface changed state (e.g. cable connected).

## Case Diagram for the "interface" Group



- Case diagrams illustrate dependencies between Variables:
  - the number of packets delivered by a network interface to the next higher protocol layer: `ifInUcastPkts + ifInNUcastPkts`.
  - the number of packets received by the network:  
`(ifInUcastPkts + ifInNUcastPkts) + ifInDiscards + ifInUnknownProtos + ifInErrors`
  - the number of actually transmitted packets:  
`(ifOutUcastPkts + ifOutNUcastPkts) - ifOutErrors - ifOutDiscards`

## Using the "interface" Group [1/2]

- It is the base of SNMP-based monitoring.
- Many tools periodically poll values from interfaces (mostly `ifInOctets` and `ifOutOctets`).
- Values are aggregated and not divided per protocol, destination, AS. This is a major limitation if fine grained monitoring is required. The reason is that SNMP counters are basically the kernel counters 'exposed' via SNMP.
- Interface errors can be used for detecting communication problems, especially on WAN links.

## Using the "interface" Group [2/2]

- Packet size statistics are not reported however simple Octets/Packets statistics can be computed.
- Many manufacturers (e.g. Cisco, Juniper) report information about both physical and logical interfaces (also known as sub-interfaces). Others (e.g. Extreme) have the entry in the table but counters are always zero.
- Using the interface counters it is possible to produce reports about:
  - VLAN (Virtual LAN)
  - PVC (Private Virtual Circuit) on Frame Relay Links



## Using the "arp" Group

- Useful for accessing the arp (Address Resolution Protocol) table of a remote device.
- It can be used for identifying arp-poisoning attacks or misconfigured hosts (e.g. duplicated IP addresses).
- NOTE:
  - ARP and switch Forwarding tables are DIFFERENT concepts.
  - ARP is an IPv4-only concept (IPv6 uses multicast)
- Example:

```
RFC1213-MIB::atIfIndex.4.1.172.22.6.168 = INTEGER: 4
RFC1213-MIB::atIfIndex.4.1.172.22.7.255 = INTEGER: 4
RFC1213-MIB::atPhysAddress.4.1.172.22.6.168 = Hex-STRING: 00 40 F4 67 49 08
RFC1213-MIB::atPhysAddress.4.1.172.22.7.255 = Hex-STRING: FF FF FF FF FF FF
RFC1213-MIB::atNetAddress.4.1.172.22.6.168 = Network Address: AC:16:06:A8
RFC1213-MIB::atNetAddress.4.1.172.22.7.255 = Network Address: AC:16:07:FF
```

# SNMP Traps (RFC 3418 and RFC 2863)

- RFC 3418 and RFC 2863 define 4+1 traps:
  - » A SNMP agent sends a **coldStart** (1.3.6.1.6.3.1.1.5.1) trap when it is initialized (boot).
  - » A **warmStart** (1.3.6.1.6.3.1.1.5.2) is sent when such system is reinitialised (reboot).
  - » A **linkDown** (1.3.6.1.6.3.1.1.5.3) trap signifies that the SNMP agent detected that the ifOperStatus object for one of its communication links is about to enter the down state.
  - » A **linkUp** (1.3.6.1.6.3.1.1.5.4) trap that the SNMP agent detected that the ifOperStatus object for one of its communication links left the down state and transitioned into some other state.
  - » An **authenticationFailure** (1.3.6.1.6.3.1.1.5.5) trap signifies that the SNMP entity has received a protocol message that is not properly authenticated (e.g. bad community).

# Bridge MIB (RFC 1493)

- Useful for controlling the status of L2/L3 switches. Do not make the common mistake to believe that it is used only on bridges.
- It somehow complementary to the MIB II as it provides information the hosts connected to the switch ports.
- Common uses of the bridge MIB:
  - To know the MAC address of a host connected to the port X/unit Y of the switch `dot1dTpFdbTable.dot1dTpFdbAddress` (NOTE: the MIB II has the MAC address of the switch port).
  - The MAC/port association is the base for detecting the physical location of a host. In fact as switch ports are usually connected to wall sockets, this is a good method for know who's where (user -> computer -> switch port -> room/desk)
  - It keeps track of the “previous” MAC address (and the time) connected to a port so it is possible to track users as they move from a room to another.
  - It can be used for detecting ports with associated multiple MAC addresses (trunk) hence to detect users with multiple MACs (e.g. a user runs VMware on his PC, or a user has been infected by a virus/worm) or ports with a switch connected to it that the network policy could be violated.

# Get Port Mac Address

```
# snmpwalk -c public@1 14.32.6.17 dot1dTpFdbAddress
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.208.211.106.71.251 = Hex-STRING: 00 D0 D3 6A 47
```

```
# snmpwalk -c public@6 14.32.6.17 dot1dTpFdbAddress
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.2.185.144.76.102 = Hex-STRING: 00 02 B9 90 4C 66
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.2.253.106.170.243 = Hex-STRING: 00 02 FD 6A AA F3
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.16.13.56.16.0 = Hex-STRING: 00 10 0D 38 10 00
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.96.84.144.248.0 = Hex-STRING: 00 60 54 90 F8 00
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.208.2.214.120.10 = Hex-STRING: 00 D0 02 D6 78 0A
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.208.211.54.162.60 = Hex-STRING: 00 D0 D3 36 A2 3C
```

```
.1.3.6.1.2.1.17.4.3.1.1.0.224.30.159.10.210 = Hex-STRING: 00 E0 1E 9F 0A D2
```

## Note:

- the <community>@<id> means that MAC is searched on VLAN X
- The MAC is part of the OID.

# Get MAC Address Port [1/2]

```
# snmpwalk -c public@1 14.32.6.17 dot1dTpFdbPort
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.208.211.106.71.251 = INTEGER: 113
```

```
# snmpwalk -c public@6 14.32.6.17 dot1dTpFdbPort
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.2.185.144.76.102 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.2.253.106.170.243 = INTEGER: 113 <- this is not ifIndex
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.6.83.198.64.173 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.16.13.56.16.0 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.96.84.144.248.0 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.208.2.214.120.10 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.208.211.54.162.60 = INTEGER: 113
```

```
.1.3.6.1.2.1.17.4.3.1.2.0.224.30.159.10.210 = INTEGER: 65
```

```
nms-server2:/home/ccarring> snmpwalk -c public 14.32.6.17 dot1dBasePortIfIndex
```

```
.1.3.6.1.2.1.17.1.4.1.2.68 = INTEGER: 12
```

```
.1.3.6.1.2.1.17.1.4.1.2.69 = INTEGER: 13
```

```
.....
```

```
.1.3.6.1.2.1.17.1.4.1.2.113 = INTEGER: 57 <- this is ifIndex
```

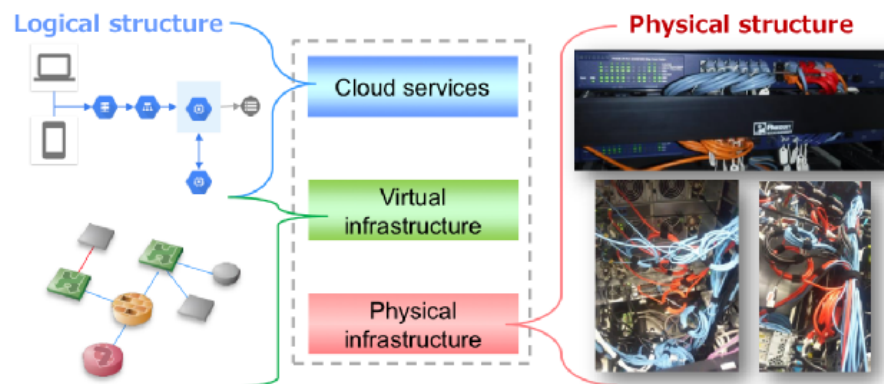
## Get MAC Address Port [2/2]

```
# snmpwalk -On -c public 14.32.6.17 ifName
.1.3.6.1.2.1.31.1.1.1.1.1 = STRING: sc0
.1.3.6.1.2.1.31.1.1.1.1.2 = STRING: sl0
.1.3.6.1.2.1.31.1.1.1.1.3 = STRING: me1
.1.3.6.1.2.1.31.1.1.1.1.4 = STRING: VLAN-1
.1.3.6.1.2.1.31.1.1.1.1.5 = STRING: VLAN-1002
.1.3.6.1.2.1.31.1.1.1.1.6 = STRING: VLAN-1004
.1.3.6.1.2.1.31.1.1.1.1.7 = STRING: VLAN-1005
.1.3.6.1.2.1.31.1.1.1.1.8 = STRING: VLAN-1003
.1.3.6.1.2.1.31.1.1.1.1.9 = STRING: 2/1
...
.1.3.6.1.2.1.31.1.1.1.1.55 = STRING: 2/47
.1.3.6.1.2.1.31.1.1.1.1.56 = STRING: 2/48
.1.3.6.1.2.1.31.1.1.1.1.57 = STRING: 2/49 (Slot 2, port 49)
.1.3.6.1.2.1.31.1.1.1.1.58 = STRING: 2/50
```

See: [http://www.cisco.com/warp/public/477/SNMP/cam\\_snmp.html](http://www.cisco.com/warp/public/477/SNMP/cam_snmp.html)

## Detecting Network Topology [1/2]

- System are combination of logical/physical structures.
  - » Logical structure provides visualisation.
  - » Physical structure has limited visualisation.
- Problem: how to detect the physical network topology and any changes in topology.



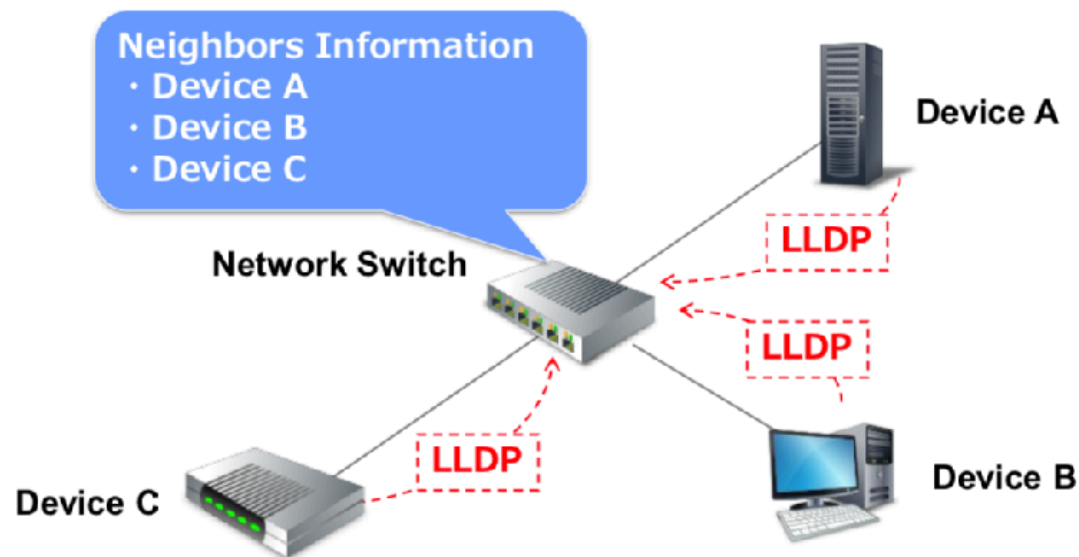
## Detecting Network Topology [2/2]

- Neighbouring information allows to discover adjacencies and this the topology.
- This information is present in the data link layer (layer 2).
- Vendors have their own protocols (e.g. Cisco has CDP Cisco Discovery Protocol) but the standard is LLDP Link Layer Discovery Protocol (RFC 2922)



# LLDP

- LLDP periodically send LLDP packets with multicast.
- Information on neighbour devices can be read using SNMP (LLDP-MIB).



## LLDP MIB [1/3]

IldpRemSysName	The system name of the remote system. (Hostname)
IldpRemSysDesc	The system description of the remote system. (OS Type, Model name ...etc)
IldpRemPortIdSubtype	The type of port identifier encoding used in the associated 'IldpRemPortId' object.
IldpRemPortId	The port component associated with the remote system. (Port name, MAC Address ...etc)
IldpRemPortDesc	The description of the given port associated with the remote system. (Port name ...etc)

### Note:

- Each entry has a timestamp of the time the LLDP packet has been received LLDP-MIN::LLDPRemoteTree.**[time]**.**[LocalPort Index]**.**[Entry]**. Note that often **[time]** is zero.
- You can have multiple entries (with different timestamps) per port.

Time

Local InterfaceId

Entry Id

## LLDP MIB [2/3]

\$ snmpwalk -v 2c -c public 172.16.67.210 1.0.8802.1.1.2.1.4.1.1.6

iso.0.8802.1.1.2.1.4.1.1.6.0.2103300.1 = INTEGER: 5

iso.0.8802.1.1.2.1.4.1.1.6.0.2103428.1 = INTEGER: 5

iso.0.8802.1.1.2.1.4.1.1.6.0.2103556.2 = INTEGER: 5

iso.0.8802.1.1.2.1.4.1.1.6.0.2103684.2 = INTEGER: 5

iso.0.8802.1.1.2.1.4.1.1.6.0.9437185.6 = INTEGER: 7

\$ snmpwalk -v 2c -c public 172.16.67.210 1.0.8802.1.1.2.1.4.1.1.7

iso.0.8802.1.1.2.1.4.1.1.7.0.2103300.1 = STRING: "TenGigabitEthernet 1/49"

iso.0.8802.1.1.2.1.4.1.1.7.0.2103428.1 = STRING: "TenGigabitEthernet 1/50"

iso.0.8802.1.1.2.1.4.1.1.7.0.2103556.2 = STRING: "TenGigabitEthernet 1/2"

iso.0.8802.1.1.2.1.4.1.1.7.0.2103684.2 = STRING: "TenGigabitEthernet 1/2"

iso.0.8802.1.1.2.1.4.1.1.7.0.9437185.6 = STRING: "42"

\$ snmpwalk -v 2c -c public 172.16.67.210 1.0.8802.1.1.2.1.4.1.1.8

iso.0.8802.1.1.2.1.4.1.1.8.0.2103300.1 = STRING: "TenGigabitEthernet 1/49"

iso.0.8802.1.1.2.1.4.1.1.8.0.2103428.1 = STRING: "TenGigabitEthernet 1/50"

iso.0.8802.1.1.2.1.4.1.1.8.0.2103556.2 = STRING: "TenGigabitEthernet 1/2"

iso.0.8802.1.1.2.1.4.1.1.8.0.2103684.2 = STRING: "TenGigabitEthernet 1/2"

iso.0.8802.1.1.2.1.4.1.1.8.0.9437185.6 = STRING: "42"

\$ snmpwalk -v 2c -c public 172.16.67.210 1.0.8802.1.1.2.1.4.1.1.9

iso.0.8802.1.1.2.1.4.1.1.9.0.2103300.1 = STRING: "swStorageAccessB7-1"

iso.0.8802.1.1.2.1.4.1.1.9.0.2103428.1 = STRING: "swStorageAccessB7-1"

iso.0.8802.1.1.2.1.4.1.1.9.0.2103556.2 = STRING: "swStorageLeaf1-1"

iso.0.8802.1.1.2.1.4.1.1.9.0.2103684.2 = STRING: "swStorageLeaf1-2"

iso.0.8802.1.1.2.1.4.1.1.9.0.9437185.6 = STRING: "swOobManagementB5-2"

\$ snmpwalk -v 2c -c public 172.16.67.210 1.0.8802.1.1.2.1.4.1.1.10

iso.0.8802.1.1.2.1.4.1.1.10.0.2103300.1 = STRING: "Dell EMC Real Time Operating System Software..."

iso.0.8802.1.1.2.1.4.1.1.10.0.2103428.1 = STRING: "Dell EMC Real Time Operating System Software..."

iso.0.8802.1.1.2.1.4.1.1.10.0.2103556.2 = STRING: "Dell Real Time Operating System Software..."

iso.0.8802.1.1.2.1.4.1.1.10.0.2103684.2 = STRING: "Dell Real Time Operating System Software..."

iso.0.8802.1.1.2.1.4.1.1.10.0.9437185.6 = STRING: "ProCurve J9022A Switch 2810-48G..."

lldpRemPortIdSubtype

lldpRemPortId

lldpRemPortDesc

lldpRemSysName

lldpRemSysDesc

# LLDP MIB [3/3]

SNMP Devices / swStorageAccessB7-2 (172.16.67.210) | Interfaces Topology MAC Addresses



## Topology

10 ▾

Index^	Interface Name	Speed	Status	Remote Device	Remote Port Description	Remote System Name	Remote System Description
<a href="#">2103300</a>	TenGigabitEthernet 1/49 <span>LLDP</span>	10 Gbit	Up	TenGigabitEthernet 1/49	TenGigabitEthernet 1/49	<a href="#">swStorageAccessB7-1</a>	Dell EMC Real Time Opera...
<a href="#">2103428</a>	TenGigabitEthernet 1/50 <span>LLDP</span>	10 Gbit	Up	TenGigabitEthernet 1/50	TenGigabitEthernet 1/50	<a href="#">swStorageAccessB7-1</a>	Dell EMC Real Time Opera...
<a href="#">2103556</a>	TenGigabitEthernet 1/51 <span>LLDP</span>	10 Gbit	Up	TenGigabitEthernet 1/2	TenGigabitEthernet 1/2	swStorageLeaf1-1	Dell Real Time Operating...
<a href="#">2103684</a>	TenGigabitEthernet 1/52 <span>LLDP</span>	10 Gbit	Up	TenGigabitEthernet 1/2	TenGigabitEthernet 1/2	swStorageLeaf1-2	Dell Real Time Operating...
<a href="#">9437185</a>	ManagementEthernet 1/1 <span>LLDP</span>	1 Gbit	Up	42	42	<a href="#">swOobManagementB5-2</a>	ProCurve J9022A Switch 2...

## Side note: SNMP vs. CLI Counters [1/4]

- It a common belief among the network administrator community that SNMP and CLI counters are basically a different view of same thing.
- Many administrators do like CLI counters more, as:
  - Are formatted for direct human consumption
    - 0 packets input, 0 packets output
  - Many implementations provide command to clear/reset counter
    - clear interface ethernet 3
- Note: the definition of what a given counter counts is dependent on vendor documentation

## Side note: SNMP vs. CLI Counters [2/4]

```
c4500#sh int e1
Ethernet1 is up, line protocol is down
Last clearing of "show interface" counters never
Output queue 0/40, 0 drops; input queue 0/75, 0 drops
0 packets input, 0 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    0 input packets with dribble condition detected
187352 packets output, 11347294 bytes, 0 underruns
187352 output errors, 0 collisions, 3 interface resets
```

- **Notes:**
  - CLI counters remain the basic way of life in element management.
  - Counters format/appearance change vendor to vendor (often even within the same manufacturer, e.g. Cisco IOS vs. CatOS vs. PIX).
  - Note: IOS, CatOS, and PIX are respectively the router, switch and firewall OS used by Cisco appliances.

## Side note: SNMP vs. CLI Counters [3/4]

- SNMP counters instead:
  - Allow you to compare apples to apples
    - Counters have standard definitions
      - as defined by IETF, IEEE, some vendors...
      - regardless of network element type or vendor
    - and globally unique, hard to pronounce names
      - 1.3.6.1.2.1.17.2.4 dot1dStpTopChanges
  - Have a well specified size
    - 32 or 64 bits wide (64 bit available in SNMP v2c or v3).
  - Counters do not necessarily (but often do) start at zero
    - Vendor implementation friendly.
  - Are not designed for directing human consumption
    - require a delta function to compute rate.

## Side note: SNMP vs. CLI Counters [4/4]

**dot1dTpPortInFrames** OBJECT-TYPE

SYNTAX Counter

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The **number of frames** that have been received by this port from its segment. Note that a frame received on the interface corresponding to this port is **only counted by this object if and only if it is for a protocol being processed by the local bridging function, including bridge management frames.**"

REFERENCE

"IEEE 802.1D-1990: Section 6.6.1.1.3"

- Note: good counters are generally derived from underlying protocol specification.



## How To Calculate Bandwidth Utilisation (%) with SNMP

Bandwidth  
Utilisation

$$\frac{(\Delta \text{ ifInOctets} + \Delta \text{ ifOutOctets}) \times 8 \times 100}{(\Delta \text{ time}) \times \text{ifSpeed}}$$

Input  
Utilisation

$$(\Delta \text{ ifInOctets}) \times 8 \times 100 / ((\Delta \text{ time}) \times \text{ifSpeed})$$

Output  
Utilisation

$$(\Delta \text{ ifOutOctets}) \times 8 \times 100 / ((\Delta \text{ time}) \times \text{ifSpeed})$$

See: [http://www.cisco.com/en/US/tech/tk648/tk362/technologies\\_tech\\_note09186a008009496e.shtml](http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a008009496e.shtml)

## What else can you do with SNMP?

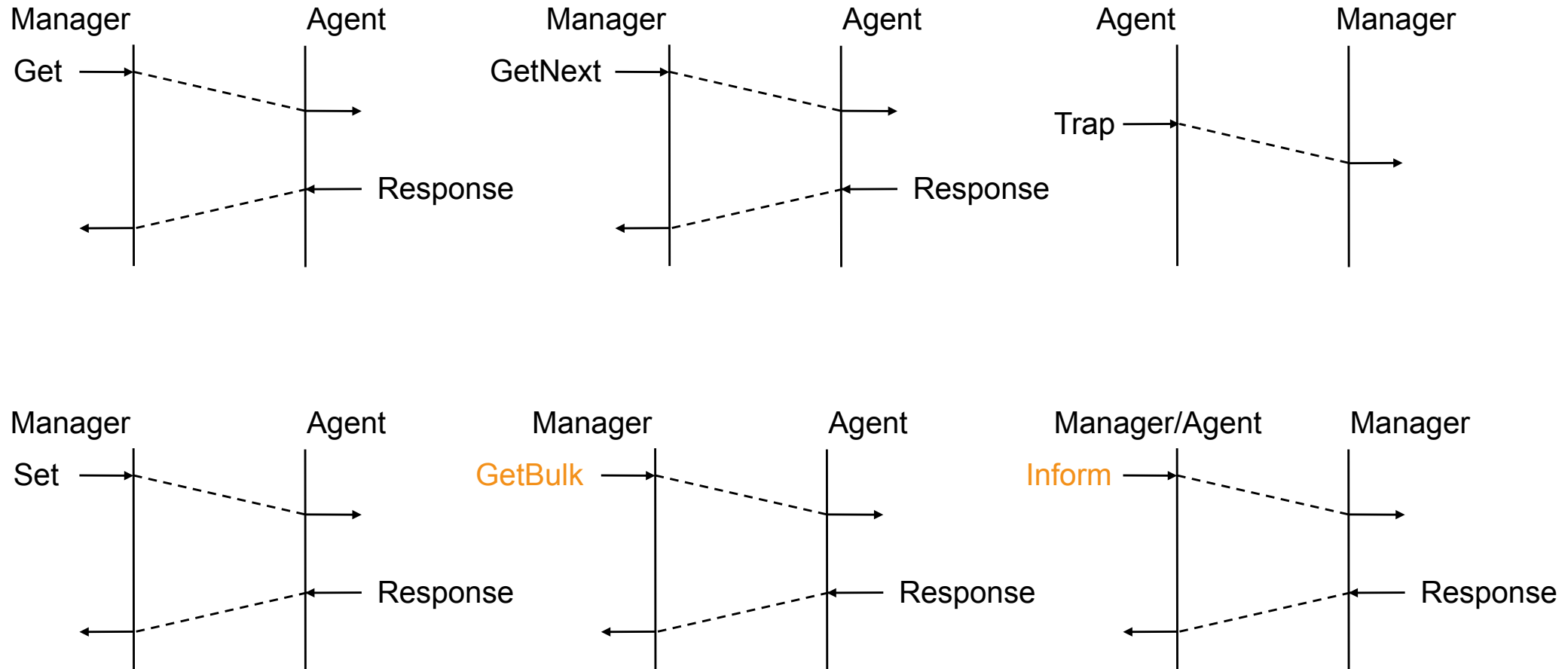
- Detect and clear hung TCP connections.
- Manipulate ARP entries.
- Get environmental temperature.
- Check CPU Utilisation.
- Monitor redundant/uninterruptible power supplies.
- Find P2P users (NAT table).
- Layout network topology (e.g. via CDP)

See: [http://www.cisco.com/en/US/tech/tk648/tk362/tk605/tsd\\_technology\\_support\\_sub-protocol\\_home.html](http://www.cisco.com/en/US/tech/tk648/tk362/tk605/tsd_technology_support_sub-protocol_home.html)

## 2.5 Simple Network Management Protocol Version 2c

- There are some variants of of SNMP Version 2:
  - SNMPv2p
    - » SNMPv2 version with party-based security model, historical
  - **SNMPv2c**
    - » SNMPv2 with trivial community-based security model
  - SNMPv2u
    - » SNMPv2 with a user-based security model, historical
  - SNMPv2\*
    - » SNMPv2 with security and administration model, historical
- The term SNMPv2 is ambiguous.
- Work on a solution of the security problems has blocked improvements of other protocol characteristics (too) for a long time.

# SNMPv2c protocol operations (RFC 1905)



# SNMPv2c Message Format

**SNMP message:**

version	community	SNMP PDU
---------	-----------	----------

**GetRequest, GetNextRequest, SetRequest, Trap, InformRequest:**

PDU type	request-id	0	0	variable-bindings
----------	------------	---	---	-------------------

**GetResponse:**

PDU type	request-id	error-status	error-index	variable-bindings
----------	------------	--------------	-------------	-------------------

**GetBulkRequest:**

PDU type	request-id	non-reps	max-reps	variable-bindings
----------	------------	----------	----------	-------------------

**variable-bindings:**

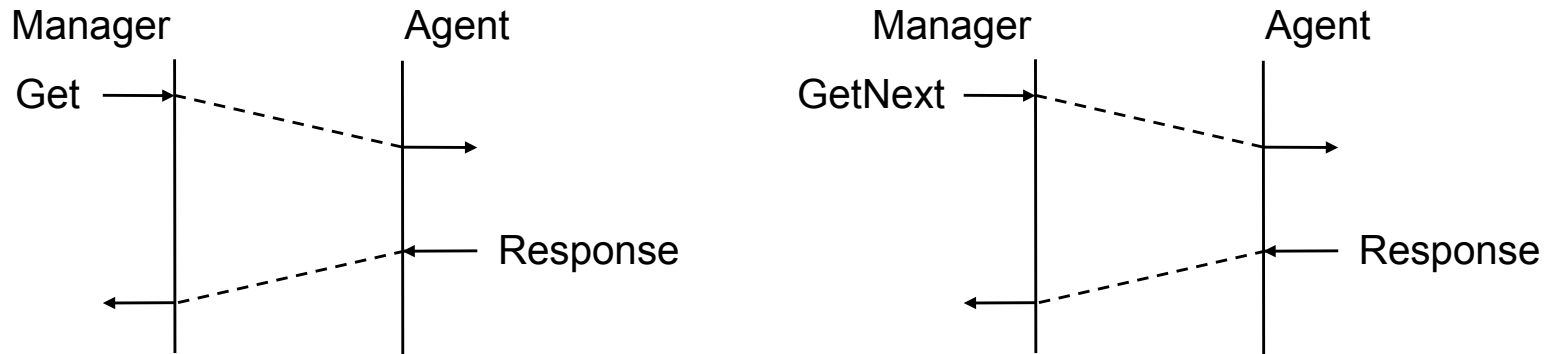
name <sub>1</sub>	value <sub>1</sub>	name <sub>2</sub>	value <sub>2</sub>	...	name <sub>n</sub>	value <sub>n</sub>
-------------------	--------------------	-------------------	--------------------	-----	-------------------	--------------------

## SNMPv2 Exceptions (RFC 1905)

SNMPv2 Exception	SNMPv1 Status	Used by
noSuchObject	noSuchName	Get
noSuchInstance	noSuchName	Get
endOfMibView	noSuchName	GetNext, GetBulk

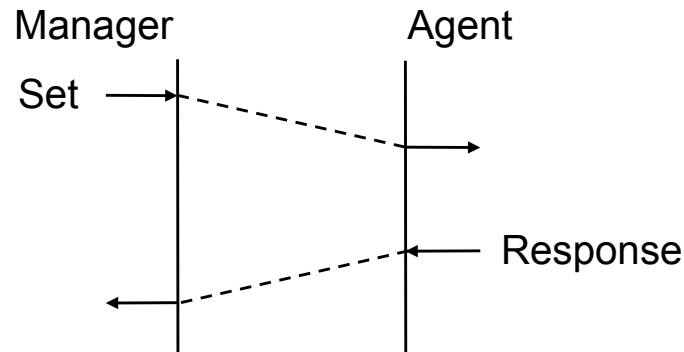
- Exceptions allow instance access errors to be signaled to MIB authorities, without causing the whole operation to fail (as happened in SNMPv1).
- Example:
  - `Get(1.1.0, 1.1.1, 1.2.0)`
  - `Response(noError@0, 1.1.0=10.1.2.3, 1.1.1=noSuchInstance, 1.2.0=noSuchObject)`
  - `GetNext(1.1, 1.5.42)`
  - `Response(noError@0, 1.1.0=10.1.2.3, 1.5.42=endOfMibView)`

# SNMPv2c Get and GetNext Operations



- Not existing MIB instances produce an exception and not an error.
- Similar to the equivalent SNMPv1 operations.

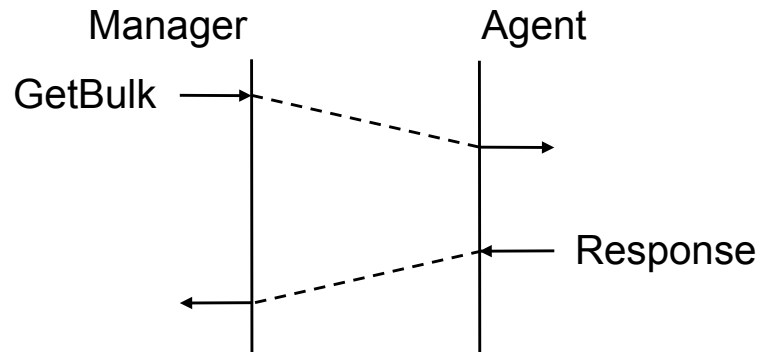
# SNMPv2c Set Operation



- There are 14 possible error codes during processing of set operations:
  - |                     |                   |                  |
|---------------------|-------------------|------------------|
| wrongValue          | wrongEncoding     | wrongType        |
| wrongLength         | inconsistentValue | noAccess         |
| notWritable         | noCreation        | inconsistentName |
| resourceUnavailable | commitFailed      | undoFailed       |
- There are two more error codes that have been defined but not really used:  
readOnly, authorizationError
- No support of error codes that depend on the object type.



# SNMPv2c GetBulk Operation

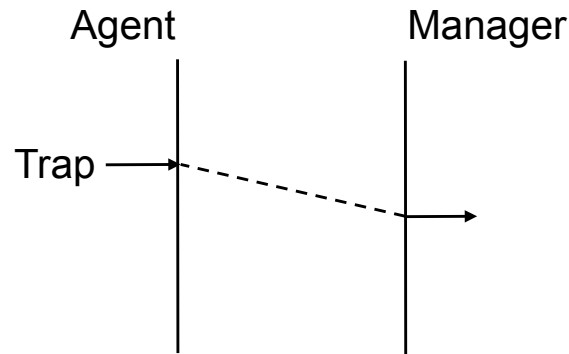


- An extension of the GetNext operation:
  - It returns the first N elements (non repetition) of the varbind list treated as normal GetNext operations.
  - The following items of the varbind list treated as repeated Get Next operation, whereby M (max repetition) indicates the max number of repetitions.
- The GetBulk operation is similar to the GetNext operation on the lexicographical arranged list of the MIB instances and has therefore no knowledge of table boundaries.

# Example of the GetBulk Operation

- `GetBulk(non-repeaters=0, max-repetitions=4, 1.1)`  
`Response(noError@0, 1.1.0=10.1.2.3, 1.2.1.0=FilterFresh,`  
`1.2.2.0=54321, 1.3.1.1.1=1)`
- `GetBulk(non-repeaters=1, max-repetitions=2`  
`1.2.2.0, 1.3.1.1, 1.3.1.2, 1.3.1.3)`  
`Response(noError@0, 1.2.2.0=54321,`  
`1.3.1.1.1=1, 1.3.1.2.1=2, 1.3.1.3.1=2,`  
`1.3.1.1.2=2, 1.3.1.2.2=3, 1.3.1.3.2=3)`
- Without knowledge about the length of a table it is difficult for the manager to select a suitable number for max repetitions:
  - if max-repetitions is too small, then there is no efficiency increase of GetBulk with respect to the GetNext operation .
  - if max-repetitions is too large, then a large number of unnecessary instances are read .
- The agent can possibly produce a response, which can either get lost in large/busy networks or not be processed at all by the manager (this causes the manager to retransmit the request).
- If max repetitions is large and reading the MIB instances is time-consuming, agents can receive multiple times the manager's request (e.g. due to retransmission) thus blocking the agent for some time.

# SNMPv2c Trap Operation



- It corresponds logically to the SNMPv1 Trap operation.
- Trap specific information (sysUpTime, trapType) is accommodated in the varbind list.
- Trap types are indicated by Object Identifier and not by a pair of numbers (generic, specific) as in SNMPv1.

# SNMPv1 vs. SNMPv2c Traps

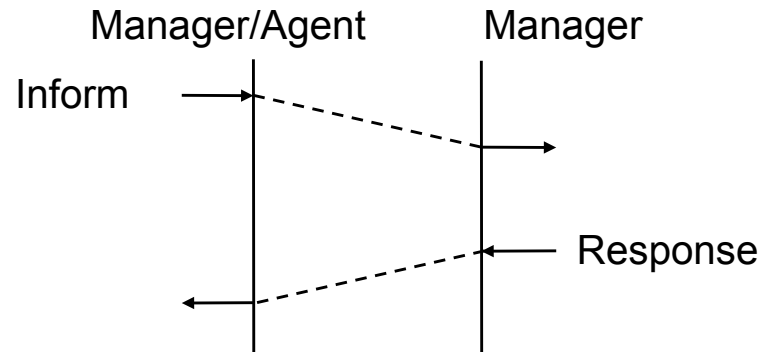
- In SNMPv2 MIBs may now include NOTIFICATION-TYPE macros.
- SNMPv1 Trap

```
myLinkDown TRAP-TYPE
    ENTERPRISE myEnterprise
    VARIABLES { ifIndex }
    DESCRIPTION
        "A myLinkDown trap signifies that the sending SNMP application
        entity recognises a failure in one of the communications links
        represented in the agent's configuration."
    ::= 2
```

- SNMPv2 Trap

```
linkUp NOTIFICATION-TYPE
    OBJECTS { ifIndex }
    STATUS current
    DESCRIPTION
        "A linkUp trap means that the entity has detected that the ifOperStatus
        object has changed to Up"
    ::= { snmpTraps 4 }
```

# SNMPv2c Inform Operation



- The structure of the PDU corresponds to a SNMPv2 Trap PDU.
- It allows (new) managers to talk each other (SNMPv1 limited interaction to agent-manager or vice-versa).
- The receipt of a Inform message is acknowledged with a Response message.
- Example:  

```
Inform(1.2.2.0=54321, 1.4.1.0=1.4.2.43,  
       1.3.1.2.2=16, 1.3.1.3.2=3)  
Response(noError@0, 1.2.2.0=54321, 1.4.1.0=1.4.2.43,  
         1.3.1.2.2=16, 1.3.1.3.2=3)
```

# SNMPv2c and SNMPv1 Error Codes

SNMPv2	SNMPv1	Comment
noError	noError	all operations
tooBig	tooBig	Get, GetNext, Set, Inform
noSuchName	noSuchName	Get, GetNext, Set (only with SNMPv1)
badValue	badValue	Set (only with SNMPv1)
readOnly	readOnly	not used
genErr	genErr	Get, GetNext, GetBulk, Set
wrongValue	badValue	Set (only with SNMPv2c)
wrongEncoding	badValue	Set (only with SNMPv2c)
wrongType	badValue	Set (only with SNMPv2c)
wrongLength	badValue	Set (only with SNMPv2c)
inconsistentValue	badValue	Set (only with SNMPv2c)
noAccess	noSuchName	Set (only with SNMPv2c)
notWritable	noSuchName	Set (only with SNMPv2c)
noCreation	noSuchName	Set (only with SNMPv2c)
inconsistentName	noSuchName	Set (only with SNMPv2c)
resourceUnavailable	genErr	Set (only with SNMPv2c)
commitFailed	genErr	Set (only with SNMPv2c)
undoFailed	genErr	Set (only with SNMPv2c)
authorizationError	noSuchName	Not used

# 64 Bit Counters (RFC 2233)

```
IF-MIB DEFINITIONS ::= BEGIN
```

```
IMPORTS
```

```
MODULE-IDENTITY, OBJECT-TYPE, Counter32, Gauge32, Counter64,  
Integer32, TimeTicks, mib-2, NOTIFICATION-TYPE FROM SNMPv2-SMI
```

```
IfEntry ::=
```

```
SEQUENCE {  
    ifIndex          InterfaceIndex,  
    ifDescr          DisplayString,  
    ifType           IANAifType,  
    ifMtu            Integer32,  
    ifSpeed          Gauge32,  
    ifPhysAddress    PhysAddress,  
    ifAdminStatus    INTEGER,  
    ifOperStatus     INTEGER,  
    ifLastChange     TimeTicks,  
    ifInOctets       Counter32,  
    ifInUcastPkts    Counter32,  
    ifInNUcastPkts   Counter32, -- deprecated  
    ifInDiscards     Counter32,  
    ifInErrors       Counter32,  
    ifInUnknownProtos Counter32,  
    ifOutOctets      Counter32,  
    ifOutUcastPkts   Counter32,  
    ifOutNUcastPkts  Counter32, -- deprecated  
    ifOutDiscards    Counter32,  
    ifOutErrors      Counter32,  
    ifOutQLen        Gauge32, -- deprecated  
    ifSpecific       OBJECT IDENTIFIER -- deprecated  
}
```

```
IfXEntry ::=
```

```
SEQUENCE {  
    ifName           DisplayString,  
    ifInMulticastPkts Counter32,  
    ifInBroadcastPkts Counter32,  
    ifOutMulticastPkts Counter32,  
    ifOutBroadcastPkts Counter32,  
    ifHCInOctets     Counter64,  
    ifHCInUcastPkts  Counter64,  
    ifHCInMulticastPkts Counter64,  
    ifHCInBroadcastPkts Counter64,  
    ifHCOctets       Counter64,  
    ifHCOUcastPkts   Counter64,  
    ifHCOmulticastPkts Counter64,  
    ifHCOBroadcastPkts Counter64,  
    ifLinkUpDownTrapEnable INTEGER,  
    ifHighSpeed      Gauge32,  
    ifPromiscuousMode TruthValue,  
    ifConnectorPresent TruthValue,  
    ifAlias           DisplayString,  
    ifCounterDiscontinuityTime TimeStamp  
}
```

# SNMP v2 vs SNMP v1

- Improved Performance via the Get-Bulk PDU.
- 64 bit counters (main improvement).
- Definition of additional datatypes and formalisms based on implementation experience of SNMPv1 agents/managers.
- Transport Service Independence: mappings for SNMPv2 have been defined for several transports and not for just UDP (TCP can also be used). In practice everybody still uses UDP.
- Redefined the Trap PDU:
  - It has the same format of the other PDUs
  - It may be sent to zero, one or many managers

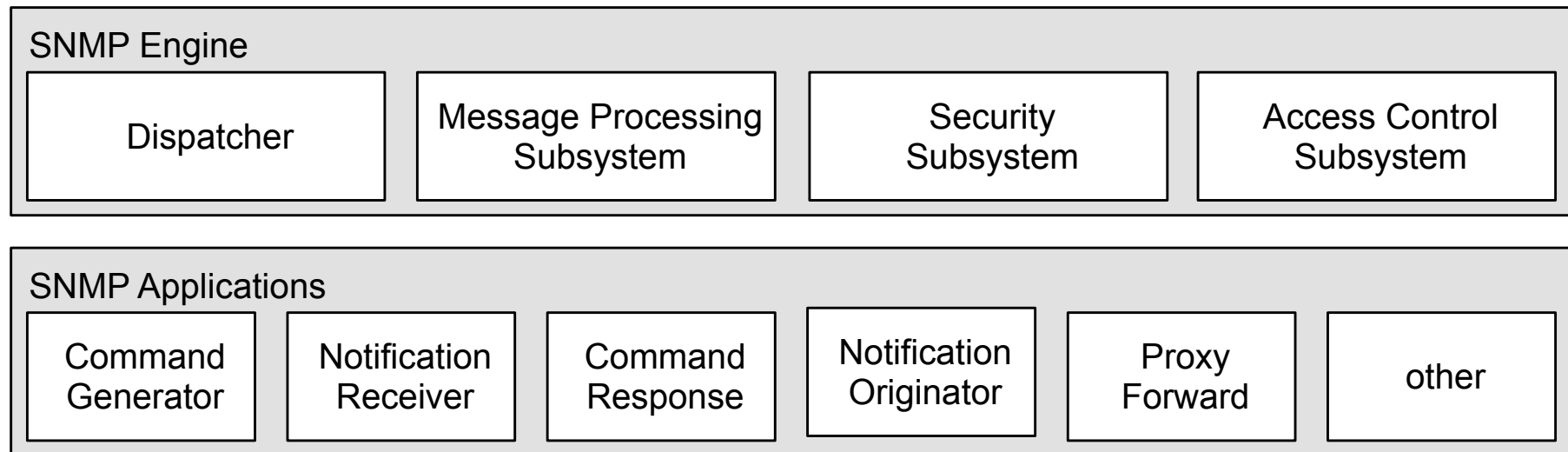


## 2.6 SNMPv3

- Design goals of SNMPv3:
  - Issue of secure SET protocol operations.
  - Definition (hopefully) of a long-living architecture model
  - Support of cheap simple and more expensive complex implementations (scalability).
  - Independence of the standards
  - Use of existing material (mostly MIBs) when possible (design reuse)
  - SNMP is to remain as simply as possible
- Several (commercial and open source) implementation available.
- Spreading in real networks still relatively small (most network devices still use SNMPv2c) due to configuration headaches.

# Architectural Model of SNMPv3 (RFC 2571)

## SNMP Entity

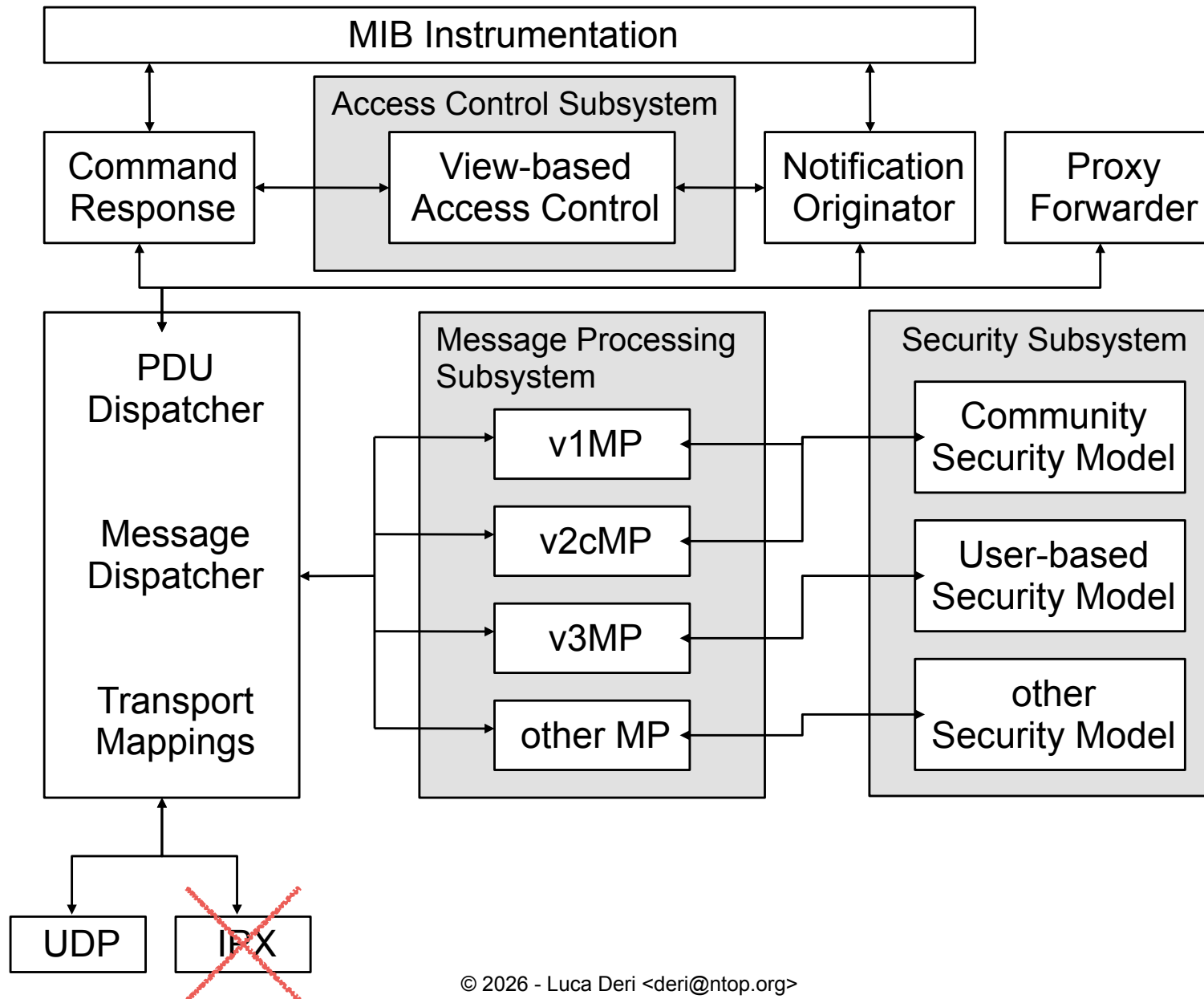


- The SNMP engine of a SNMP entity consists of several subsystems and a dispatcher.
- The manager/agent model is replaced by a number of smaller “applications”.
- The modularity permits incremental advancement of SNMP by means of SNMP Context (RFC 2571)

# SNMP Context (RFC 2571)

- A context is a quantity of management information that a SNMP Entity can have accessed to. For each subsystem:
  - A SNMP-Entity has potentially access to several contexts.
  - The same information can be present in several contexts.
- In a management domain an instance of a Managed Objects is uniquely identified by the following items:
  - the identification of the SNMP engines in a SNMP Entity (e.g. „xzy“).
  - the name of the context in a SNMP Entity (e.g. „board1“).
  - the identification of the type of the Managed Objects (e.g. „IF-MIB::ifDescr“).
  - the identification of the Instance (e.g. „1“).
- Note: the identification of an SNMP engine does not have to do anything with their addressing.

# SNMPv3 Agent in SNMPv3: Architectural Model



# SNMPv3 Message Format (RFC 2572)

## SNMPv3Message:

msgVersion	msgGlobalData	msgSecurityParameter	msgData (scopedPDU)
------------	---------------	----------------------	---------------------

## MsgGlobalData:

msgID	msgMaxSize	msgFlags	msgSecurityModel
-------	------------	----------	------------------

## UsmSecurityParameter:

msgEngineID	msgEngineBoots	msgEngineTime	msgUserName	msgAuthParams	msgPrivParams
-------------	----------------	---------------	-------------	---------------	---------------

## ScopedPDU:

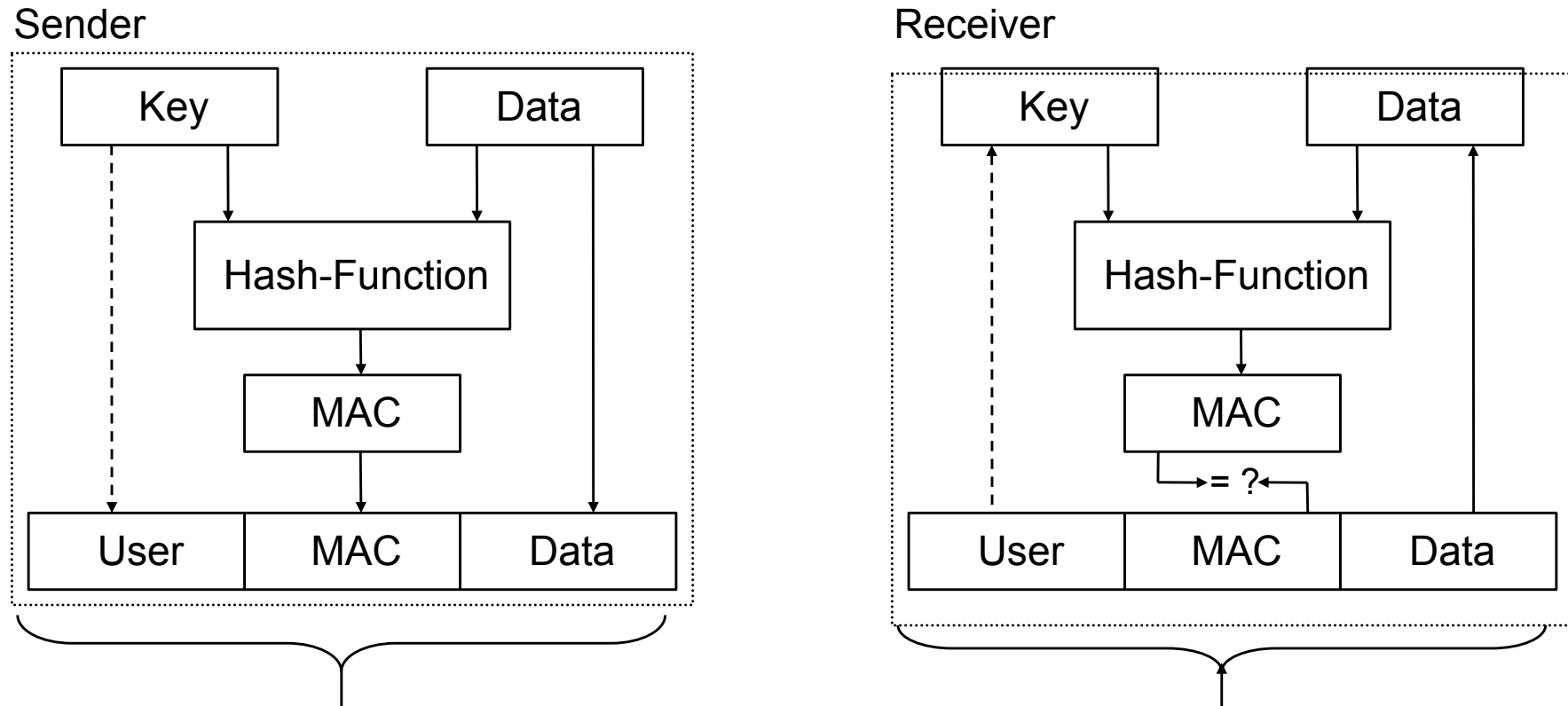
contextEngineID	contextName	SNMPv2 PDU (as defined in RFC 1905)
-----------------	-------------	-------------------------------------

- Security information are in the centre of the message.
- msgData contain either a ScopedPDU or an encoded ScopedPDU.
- msgID is used for the association of responses to pending inquiries.
- msgSecurityParameter depends on msgSecurityModel.

# Security Issues

- Blow you can find the questions which must be answered when a decision whether an operation has to be performed:
  - Is the received message authentic?
  - Who (requester name) would like to get the operation executed?
  - Which objects are affected by the operation?
  - Which access rights has the requester regarding the objects concerned?
- Questions 1 and 2 are answered by the measures to the protection of the messages (authentication, encoding).
- Questions 3 and 4 are answered by a model to the access supervision (Unix-like).

# Data Integrity and Authentication [1/2]



- Authentication with Message Authentication Code (MAC) is efficient to implement.
- The Hash function must be cryptographically strong and a "good" MAC producer.
- The MD5 algorithm (RFC 1321) can be implemented in software with acceptable performance (128 bit digest).
- The Secure Hash algorithm 1 (SHA-1) is considered stronger of MD5 (see next slide).

## Data Integrity and Authentication [2/2]

- Encoding is a way of translating between different formats. Like converting a Spanish recipe for cake into English.
- Encryption is a way of protecting data behind a secret.
- Hashing is a way of permanently converting from one recognisable thing to something uniform and simple. Notes:
  - 1) hashing is unidirectional, i.e. you cannot revert the hash to the original data.
  - 2) multiple different input data can produce the same hash value.

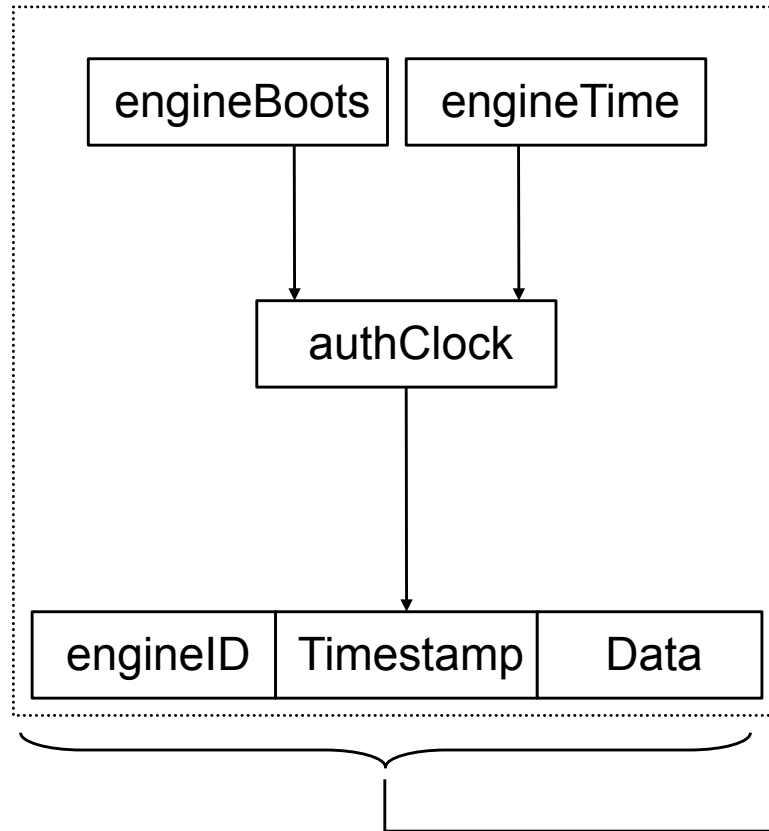
Algorithm	Family	Bits	Introduced	Deprecated	Replaced
SHA-1	Hash	160 (hash)	1995	2011	SHA-256
MD5	Hash	128 (hash)	1992	2011	SHA-256
DES	Encryption	56 (key)	1991	2018	AES-256
AES	Encryption	128 (key)	1976	2002	AES-256

- Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2 (RFC 9155)

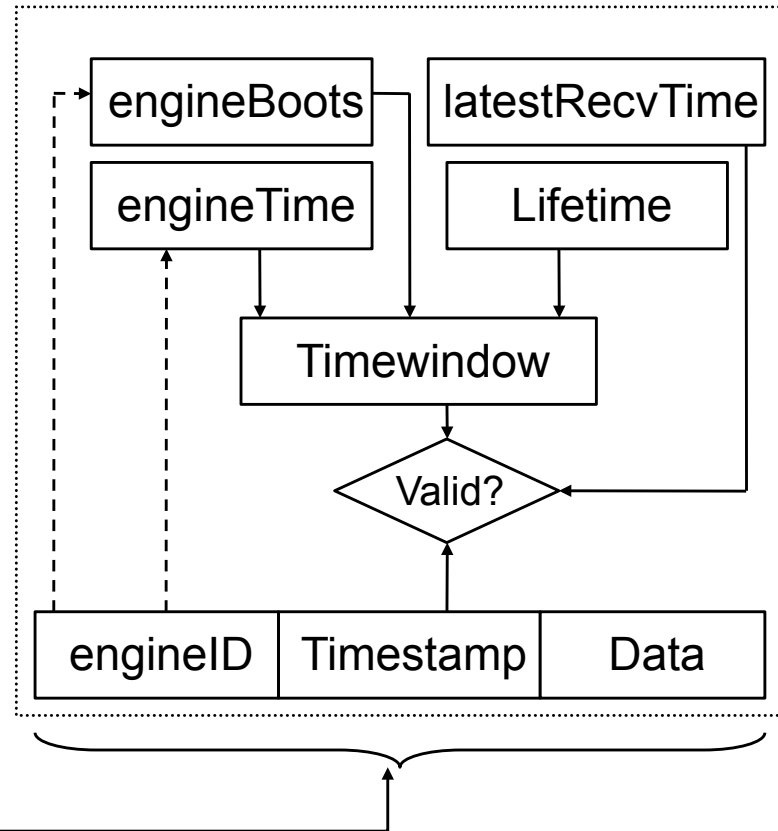


# Protection Against Repetitions of Old Messages

Authoritative Engine

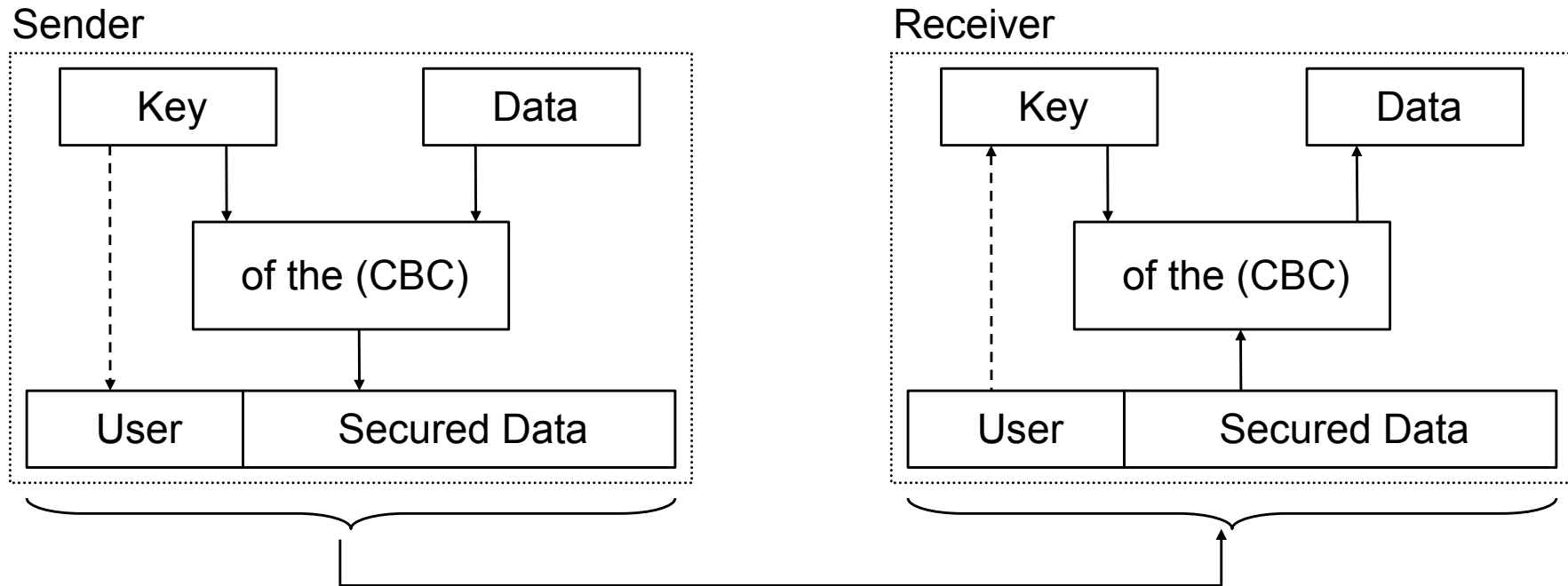


Receiver



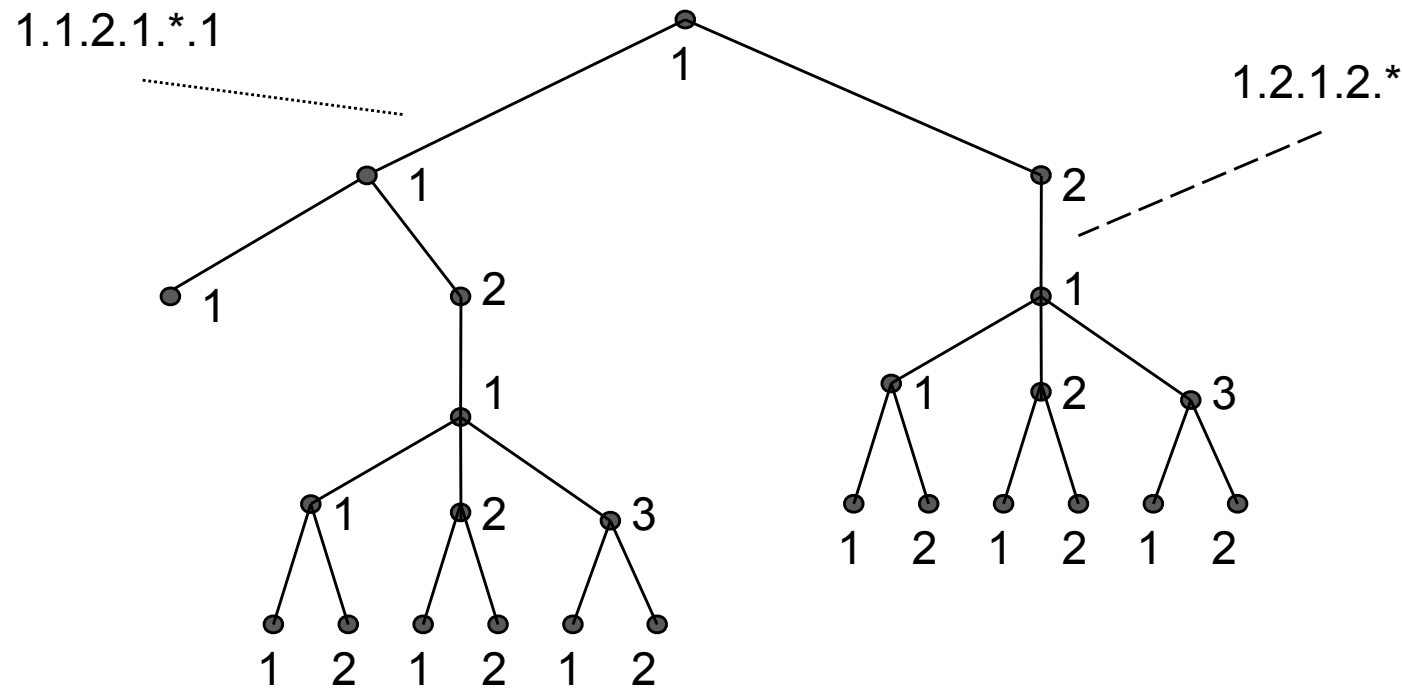
- A recipient must know the "time-of-day" of the authoritative SNMP engine for the message.
- If the received message is situated in the validity interval and is "younger" than the last valid message, then the message will become processed and the clocks adapted.
- Before the beginning of authenticated communication the **clocks must be synchronized**.

# Protection Against Sniffing



- For protecting against sniffing the ScopedPDU can be optionally encoded.
- Data Encryption Standard (of the) in Cipher Block Chaining Modus (CBC) is used for encryption.
- Encryption is relatively complex and should only be used in area/situations where an encoding is really necessarily.
- SNMPv3 permits "relatively protected" code modification without encryption (by using message digest).

# MIB Views (RFC 2575)

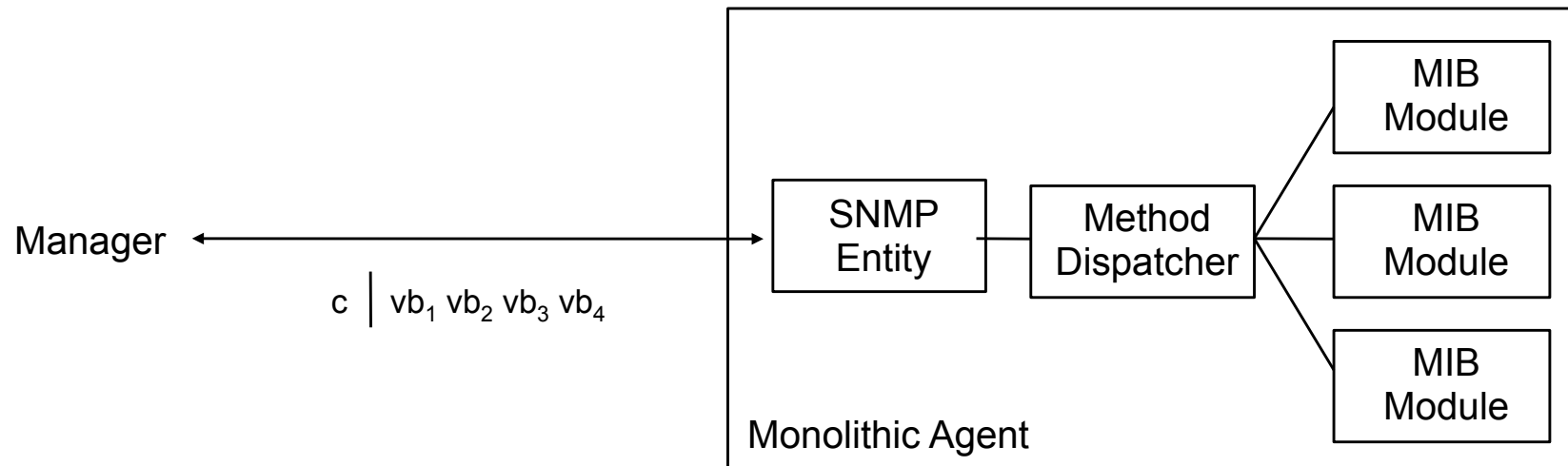


- A view subtree is the quantity of all MIB objects, which possess common OID prefix.
- A view tree family is the combination of one view subtree OID prefix with a filter (bitmask), which determines whether an item of the prefix is significant or not.
- A view is an ordered set of view tree families.
- It defines the access rights for read view, write view and notify view.

# MIB Name Conventions

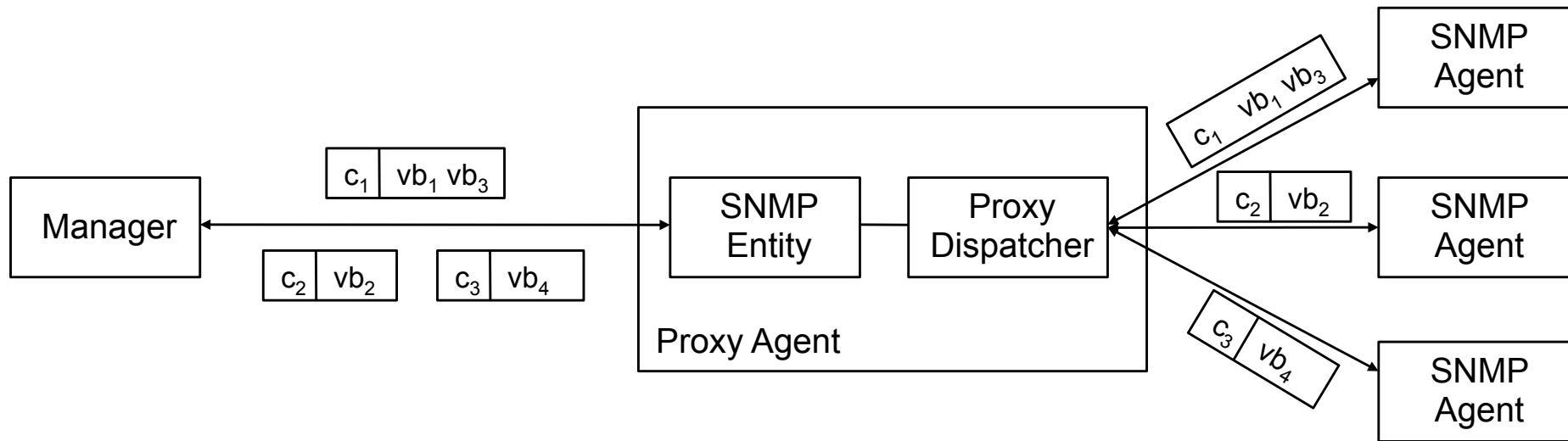
- Similar definitions should be registered together in the registration tree.
- Names of object types should begin the logical grouping with a common prefix, that suggest (e.g. sysDescr, sysUpTime).
- Names for counter are to be selected in the Plural form.
- Names of conceptual tables should possess the ending Table (e.g. ifTable).
- Names of lines of a conceptual table should possess the ending entry (e.g. ifEntry).
- All items of a conceptual table should use common prefix in the name (e.g. ifType, ifDescr).

# Monolithic Agents



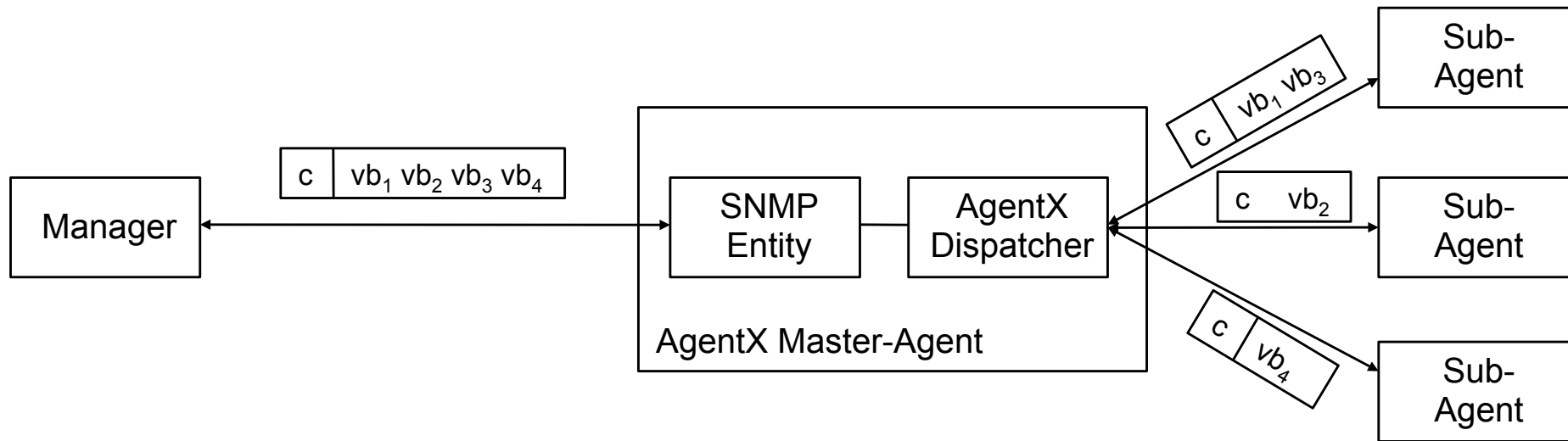
- A monolithic agent is normally implemented by an individual process which contains the SNMP protocol machine and the MIB instrumentation.
- The supported MIB modules is determined at compilation time.
- The method dispatchers is called during processing of SNMP messages, which can either read or modify values from relevant resources.

# Proxy-Agents



- SNMP Proxy agents permit managers to access other SNMP agents that are not reachable directly (e.g. behind a firewall) or that are reachable using non IP protocols (e.g. IPX).
- Management applications must (usually) select the appropriate community string or context in order to enable the proxy to reach the agents (no transparency).
- Proxy are important for the implementation of firewalls or for conversion between different SNMP protocol versions.

# Extensible Agents



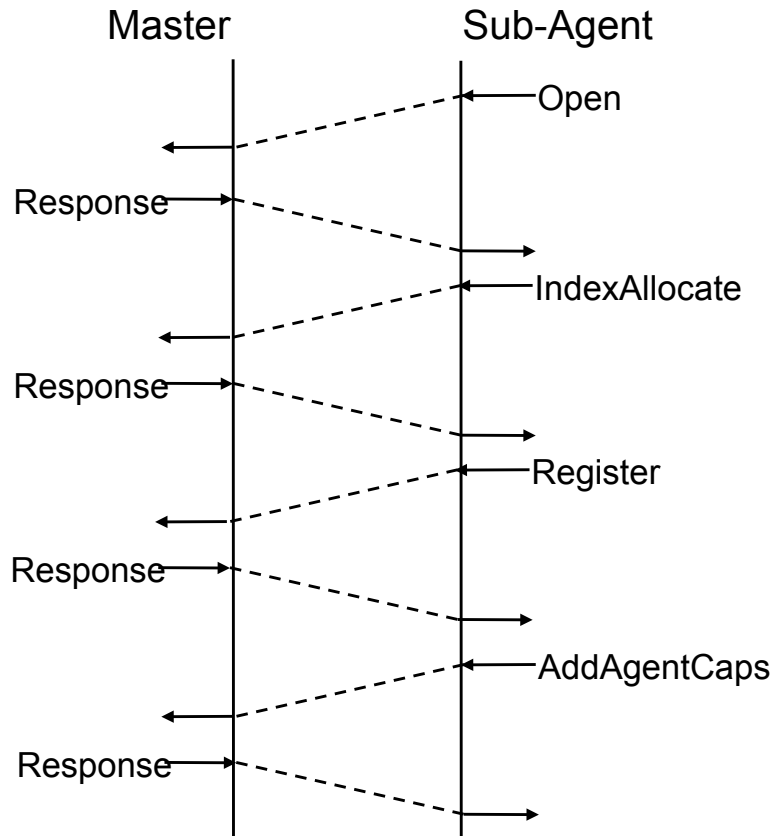
- Extensible SNMP agents separate the SNMP protocol machine (master agent) from the MIB instrumentation (subagent).
- MIB modules can be added by starting further subagents dynamically at runtime.
- Expandable agents are transparent for management applications.
- A special protocol regulate communications between the master agent and the subagents

# AgentX-Protocol Version 1 (RFC 2257)

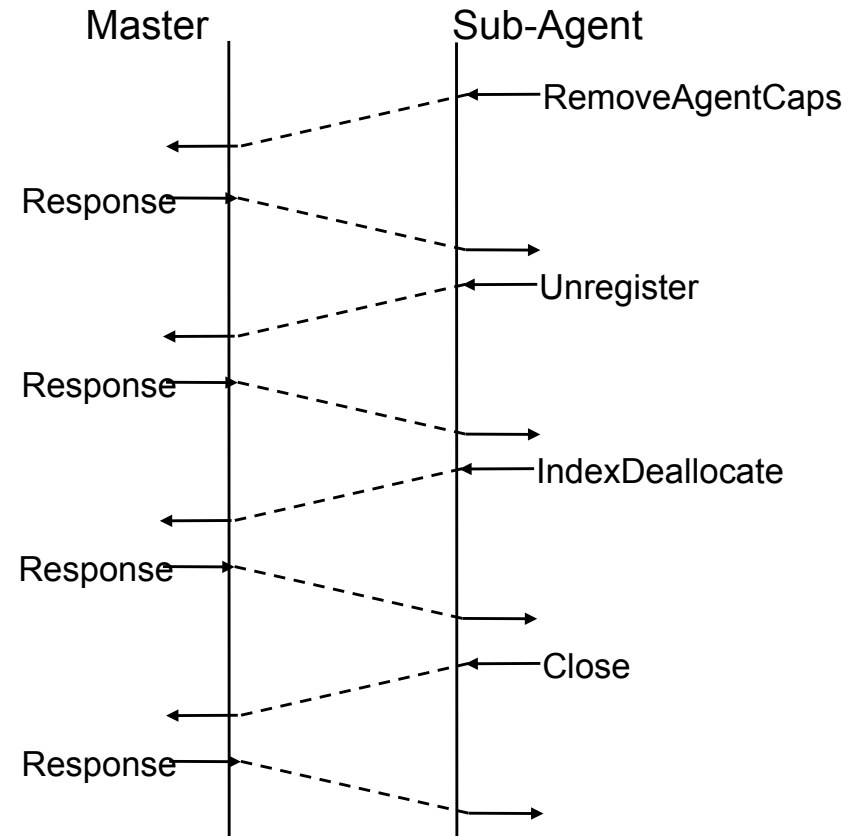
- The AgentX protocol is a new standard protocol for the implementation of expandable SNMP agents.
- AgentX Message Coding:
  - No ASN.1 coding.
  - Compact representation of object identifier values by coding repetitive OID prefixes.
  - Byte order is selected by the subagent (no transformations necessarily, if master agent and subagent on the same system).
- AgentX Message Transport:
  - TCP connections to the port 705.  
(It is possible to have several AgentX sessions over the same TCP connection)
  - UNIX Domain Sockets (/var/agentx/master).
  - Can be likewise used other local (not standardised) IPC mechanisms.



# Administrative AgentX Protocol Operations



- AgentX Session Establishment
- Index Allocation
- MIB Registration
- Registration of the Agent Capabilities

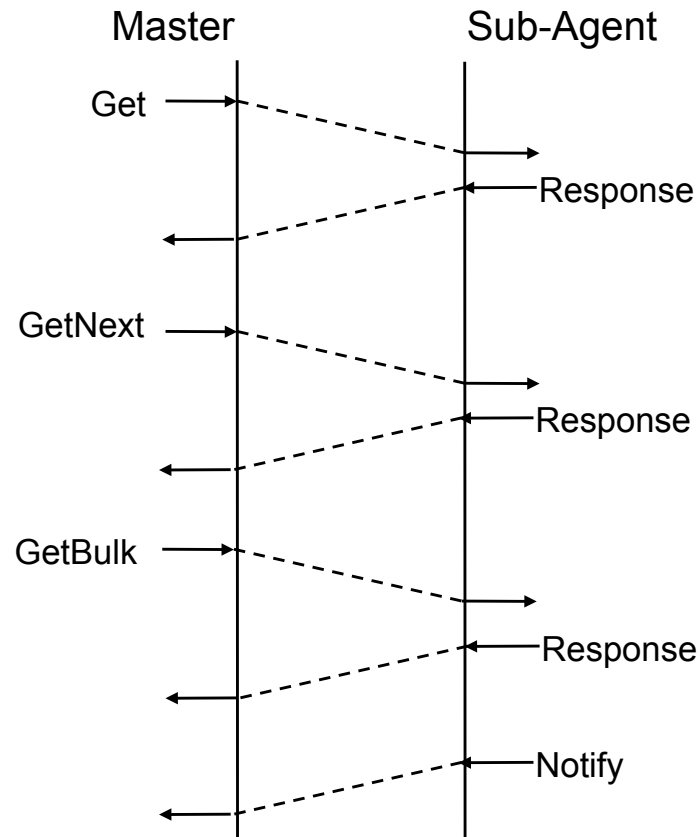


- Deregistration of the Agent Capabilities
- MIB Deregistration
- Free of allocated indexes
- AgentX Session Termination

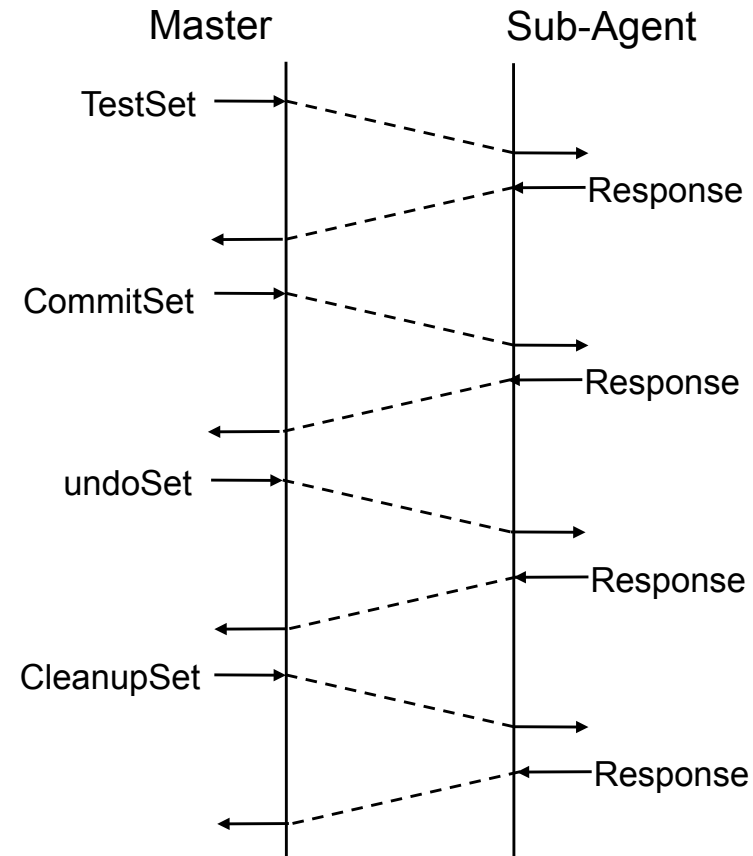
# Index-Allocation, OLD Registration, Scoping

- Index allocation for common tables between subagents:
  - Allocation of specific (private) indexes.
  - Allocation of indexes not used at present.
  - Allocation of indexes no longer in use.
- OLD Registration:
  - Registration of individual instances (instance level registration)
    - 1.3.6.1.2.1.2.2.1.1.42 (ifIndex.42)
    - 1.3.6.1.2.1.2.2.1.2.42 (ifDescr.42)
    - 1.3.6.1.2.1.2.2.1.3.42 (ifType.42)
  - Registration of MIB Ranges:
    - 1.3.6.1.2.1.2.2.1.[1-22].42 (ifIndex.42 - ifSpecific.42)
- Scoping:
  - AgentX can specify scoping with GetBulk operations (similar to CMIP Scope).

# AgentX Protocol Operations for SNMP Operations



- SNMP-operations correspond to AgentX operations.
- A SNMP operation can concern several subagents.



- Atomicity of SNMP SET operations is guaranteed by the AgentX protocol.