Basic Internet Applications

- An application is anything useful that runs over a transport protocol, or even over raw IP
- We've already seen some: DNS and DHCP for example. Routing protocols also run over UDP or TCP
- Other basic apps include
  - Telnet and SSH
  - FTP
  - SMTP
  - SNMP

Types of Application

- The Internet is intrinsically a peer-to-peer network
  - peer = "one that is of equal standing with another"
  - anybody can send packets to anybody
- Applications are often classified as client/server or peer-to-peer (p2p)
  - client/server: a client program starts by asking the server to respond; client and server have different roles
  - p2p: each system starts by discovering the others. Systems may act as clients and servers for each other
  - Even a p2p application probably needs some designated servers (e.g. Skype login server)
  - Some applications are hard to classify (think about this when we discuss SMTP)

Telnet

- Insecure line-mode interaction over the network (remote login)
  - sends what you type, over TCP
  - returns what the other end responds with
  - more or less transparent transmission of ASCII characters
  - login password travels in the clear, hence highly discouraged unless you want your password made public
  - Telnet server listens on TCP port 23_{10}

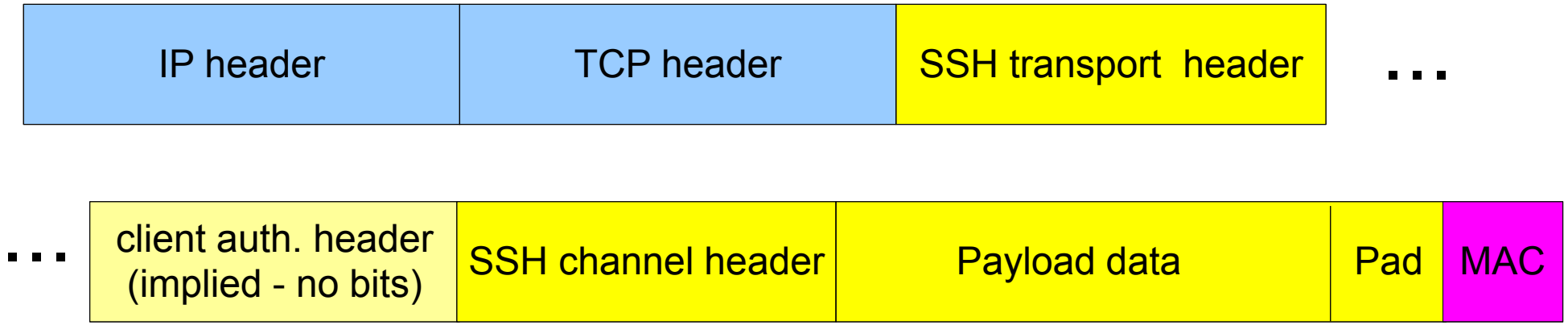
Secure Shell (a.k.a. SSH)

- Secure line-mode interaction
  - Can also be used for secure file transfer
  - SSH server listens on TCP port 22_{10}
  - Remote user is authenticated using public key cryptography
  - Server and client software establish an encrypted channel
  - Interaction (or file transfer) uses that channel

SSH Architecture

- Three main components:
  - SSH Transport Layer Protocol
    - Runs over TCP
    - Provides server authentication, data confidentiality (encryption), and data integrity
  - User Authentication
    - Runs over SSH Transport Layer
    - Authenticates the client-side user to the server
  - Connection Protocol
    - Runs over an encrypted, authenticated SSH transport connection
    - Multiplexes the connection into several logical channels

Notional Message Structure



- Blue - unprotected
- Yellow - authenticated and encrypted
- Magenta - message authentication code
- Notional view, because
  - SSH messages may be streamed across multiple TCP segments
  - Payloads for several channels may come in sequence
  - SSH headers are rather simple (and there is no auth. header)

SSH Messages

- All start with a code byte, e.g. a channel header + data is simply:

```
byte      SSH_MSG_CHANNEL_DATA
uint32    recipient channel
string    data
```

where `string` is a `uint32` containing the number of data bytes, followed by the data

(`SSH_MSG_CHANNEL_DATA` has value 94_{10}. Many SSH message types are defined, each with a name and a corresponding numeric value)

Transport Establishment

- Two or three round trips, exchanging SSH messages of various types
- Version number exchange
  - version needs to be 2.0 today
- Key exchange
  - negotiate use of strongest mutually acceptable encryption algorithm
  - negotiate choice of Message Authentication Code (MAC) algorithm
  - server authenticates itself via shared secret **or** certificate
- Compression negotiation
  - built into key exchange dialogue
  - optional

User Authentication

- Transport negotiation creates a safe connection
  - Server is authenticated but client is unknown
  - Next step is to authenticate the client (user)
- Client sends SSH messages like

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name in ISO-10646 UTF-8 encoding
string    service name in US-ASCII
string    method name in US-ASCII
....      method specific fields
```

- After iteration to find a method that the server accepts, server will finally reply

```
byte      SSH_MSG_USERAUTH_SUCCESS
```

- The user is now authenticated on the safe connection
  - Hence, no authentication headers needed in following messages

SSH Cryptography and Authentication

- SSH can support many encryption algorithms
  - Must include 3DES-CBC
  - Should support AES128-CBC
- SSH can support many message integrity (MAC) algorithms
  - Must include HMAC-SHA1
- Client authentication methods include
  - Public key (client uses private key to sign authentication request)
  - Password (client sends text password, within SSH encryption)

SSH Channel Establishment

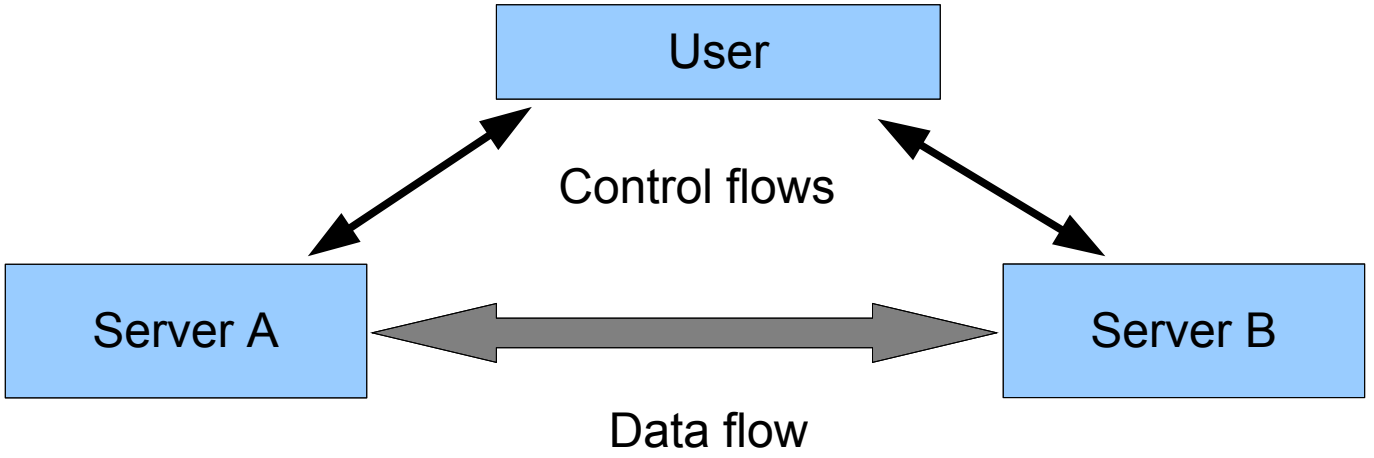
- Transport negotiation followed by user authentication creates a fully trustworthy connection
  - Final step before sending data is to open individual channels over that connection
  - The most common case is a remote login (shell) channel
  - Other options include X11, TCP/IP port forwarding, and secure FTP
- Opening a channel needs an SSH message such as

```
byte      SSH_MSG_CHANNEL_OPEN
string    "session"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
```

  - SSH channels run a simple window mechanism to avoid buffer overflows (but rely on TCP for flow control & retransmission)

FTP: File Transfer Protocol

- Same generation as Telnet, i.e. insecure (passwords in the clear, no crypto, etc.)
- FTP client (user) and server exchange control messages and data over separate TCP connections
  - Commands and replies are sent in ASCII text using Telnet format
  - FTP server listens on TCP ports 21 (control) and 20 (data)
- FTP user can request file transfer between two other systems



Important FTP commands

- USER - username for login
- PASS - password for login (unprotected)
- CWD - cd
- QUIT
- PORT - change host address and port number for incoming data from its default value
- PASV (“passive”) - tell server to wait for data connection (instead of initiating it)
  - PORT and PASV can combine to start “triangle” transfer
- TYPE - Binary, ASCII, etc. (ASCII is 7-bit characters!)
- RETR pathname (“retrieve”) - open and send a file
- STORE pathname - receive and store a file

Secure File Transfer

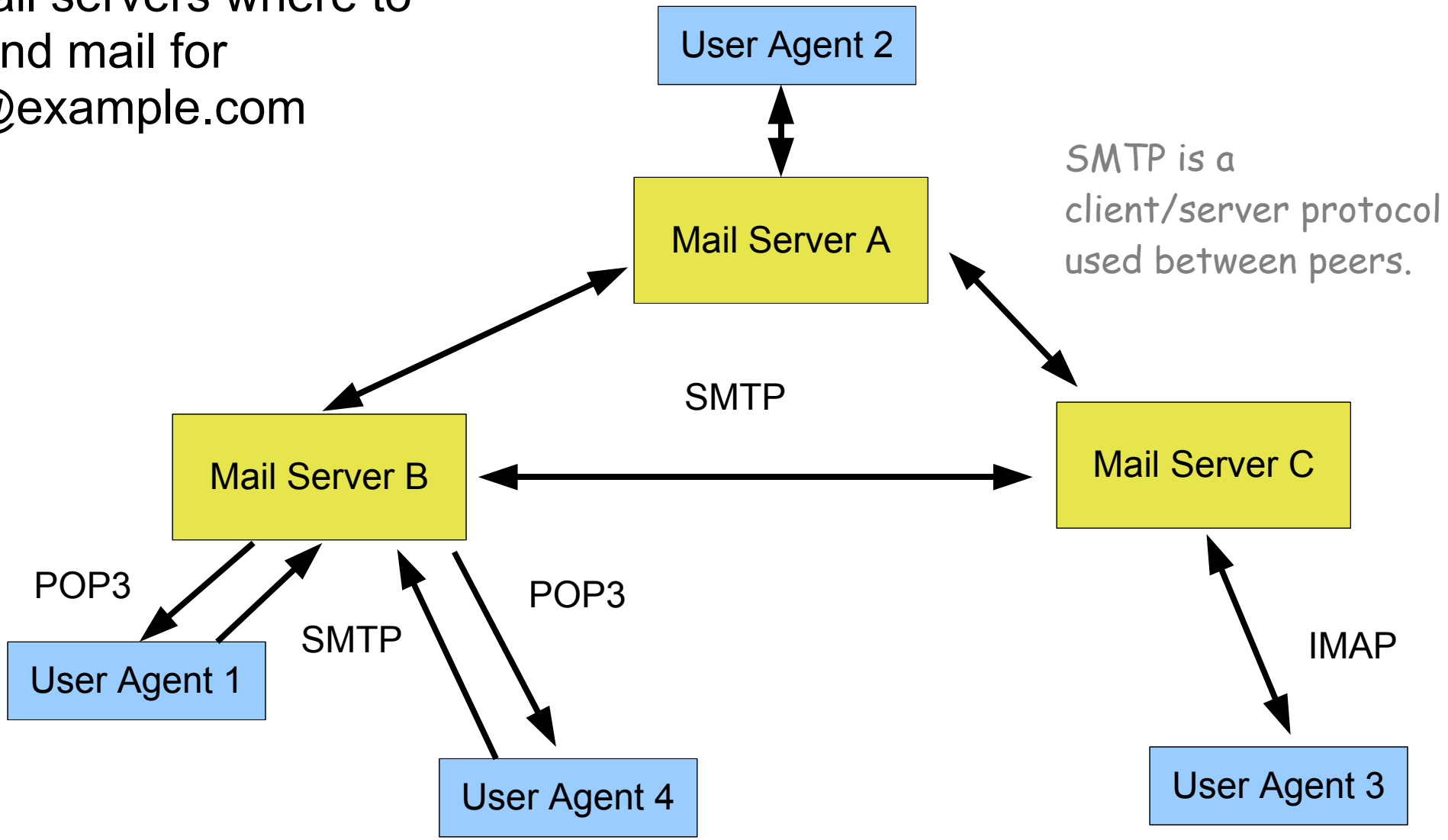
- Standard FTP is unprotected
- SCP is an old solution (remote copy over SSH)
- SFTP is sometimes
  - Simple File Transfer Protocol (obsolete, insecure)
  - SSH File Transfer Protocol (available with SSH, but not formally standardised, and not FTP over SSH)
- There is of course a way of securing FTP with TLS (RFC 4217)

SMTP: Simple Mail Transfer Protocol

- Simple? Not really
  - 76 pages in the RFC, plus another 51 pages for mail message format
- Another TCP application (port 25)
- Used for one mail server to forward mails to another, and for user agents to submit mail to their own server
  - Not used for mail delivery to user agents
- SMTP transports a mail object
  - A mail object contains an envelope and content.
  - The content is what you can see with 'view message source' in most mail agents
  - The envelope is formed by a series of SMTP commands expressed in 7-bit ASCII

Mail overview

MX (mail exchange) records in the DNS tell mail servers where to send mail for *@example.com



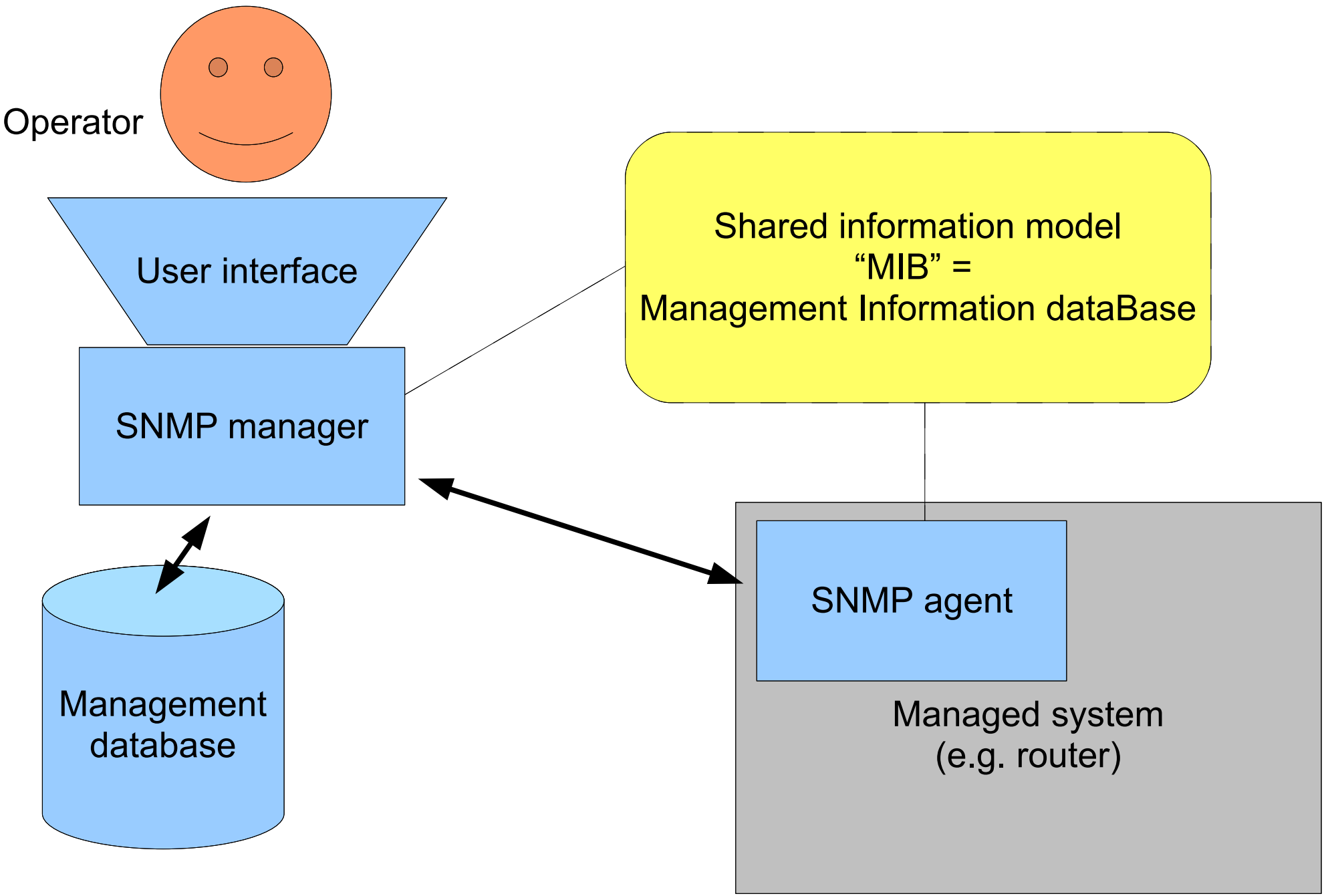
SMTP commands (simplified)

- EHLO - opening command from client side
  - SMTP servers take client role when sending
  - HELO - obsolete version of EHLO
- MAIL FROM: <reverse-path>
  - <reverse-path> is the source mail address, to be used for returning errors - not for normal replies
- RCPT TO: <user@example.com>
  - destination mail address
  - multiple recipients = multiple RCPT commands
- DATA
  - Start of message body
  - Originally 7 bit ASCII based; now “8 bit clear” is negotiable
  - End of body is **<CRLF>.<CRLF>**

SNMP: Simple Network Management Protocol

- Large networks don't run themselves - they need constant monitoring, and frequent configuration updates
- SNMP is one way this can be achieved from a central point
- SNMP features:
  - Real time status monitoring
  - Alerts when something goes wrong
  - SET commands for configuration
    (However, routers etc. are usually configured using a command line interface, typically over SSH)

SNMP Model


Operator
User interface
SNMP manager
Shared information model
“MIB” =
Management Information dataBase
Management
database
SNMP agent
Managed system
(e.g. router)

MIBs and SMI

- A MIB module describes in machine-readable form the information model for managing a particular device or protocol
  - MIBs are written in a format called SMI (Structure of Management Information) using ASN.1 syntax
  - ASN.1 (Abstract Syntax Notation 1) was part of OSI
  - A MIB module must be syntactically correct, just like a program, so that manager and agent can parse it
  - Manager and agent must use exactly the same MIB
  - The agent contains code to map MIB objects to and from real-world objects
  - The semantics of MIB objects is often expressed as a comment; that's where code has to be written

Background slide

Sample extract from the MIB for IP

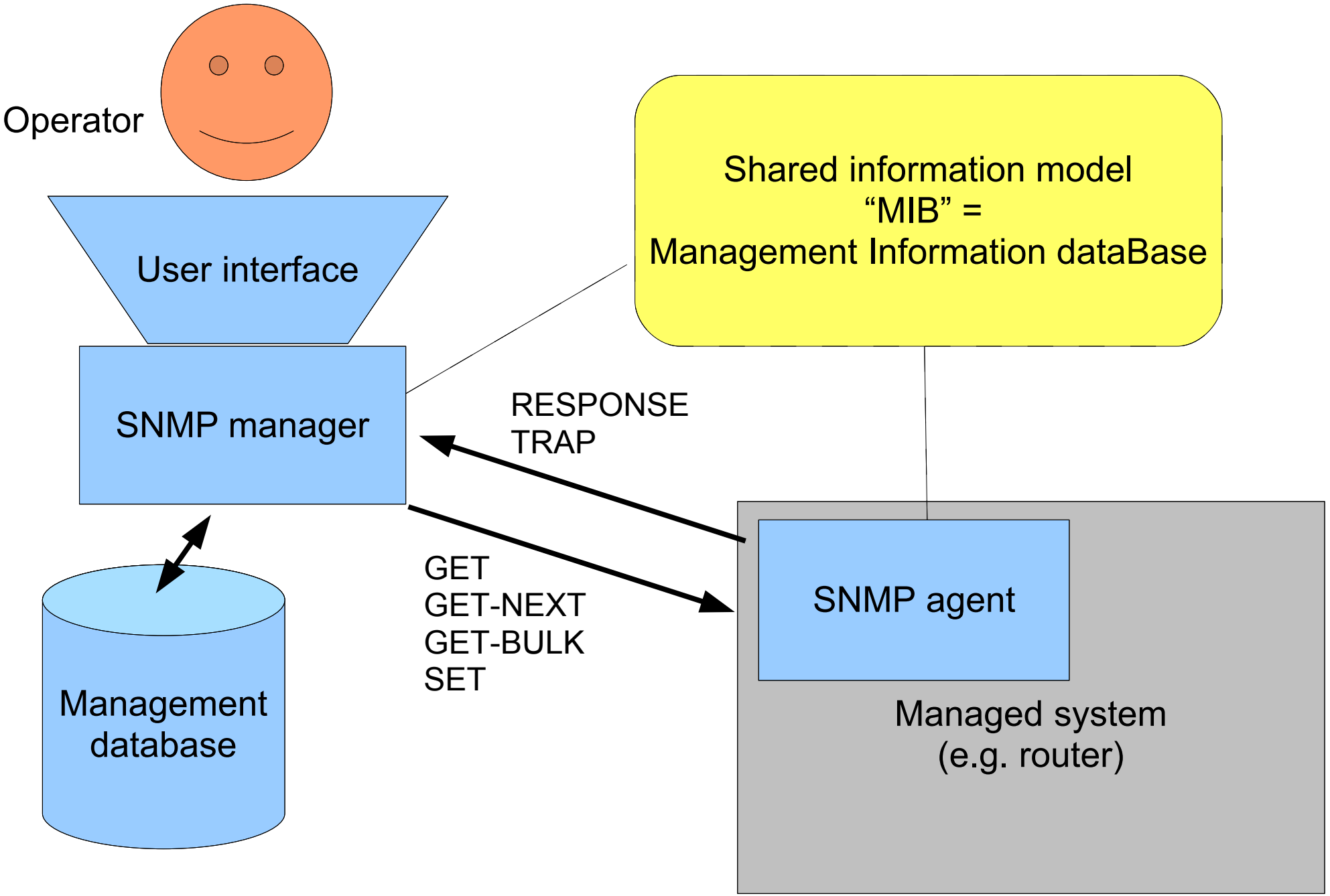
```
IP-MIB DEFINITIONS ::= BEGIN
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Integer32, Counter32, IpAddress,
    mib-2, Unsigned32, Counter64,
    zeroDotZero FROM SNMPv2-SMI
...
ipSystemStatsInAddrErrors OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS read-only
    STATUS      current
    DESCRIPTION
            "The number of input IP datagrams
  discarded because the IP address in their IP
  header's destination field was not a valid address
  to be received at this entity."
::= { ipSystemStatsEntry 9 }
```

SNMP messages

- Normally runs over UDP
  - short messages
  - must do no harm if lost or repeated, e.g. *set value=4* is OK, *increment value* is unsafe
- Message types (simplified)
  - GET
  - GET-NEXT
  - GET-BULK

  } (ask for object value(s)
  - RESPONSE (reply to a GET)
  - SET (set an object value)
  - TRAP (alert message from agent)
- Messages include object names and data values as appropriate (according to MIB syntax, mapped in a defined way into binary)

SNMP Message Flow



References

- Shay 11.5
- SSH - RFC 4251, 4252, 4253, 4254, 4256, 4250
- FTP - RFC 959 (and updates)
- SMTP - RFC 2821
  - RFC 2822 for message formats
- SNMP - RFC 3410 (SNMPv3 intro), RFC 3416 (protocol)
  - RFC 2578 (SMIv2)