Lecture 2
summary of Java SE & Network Programming – section 2

presentation

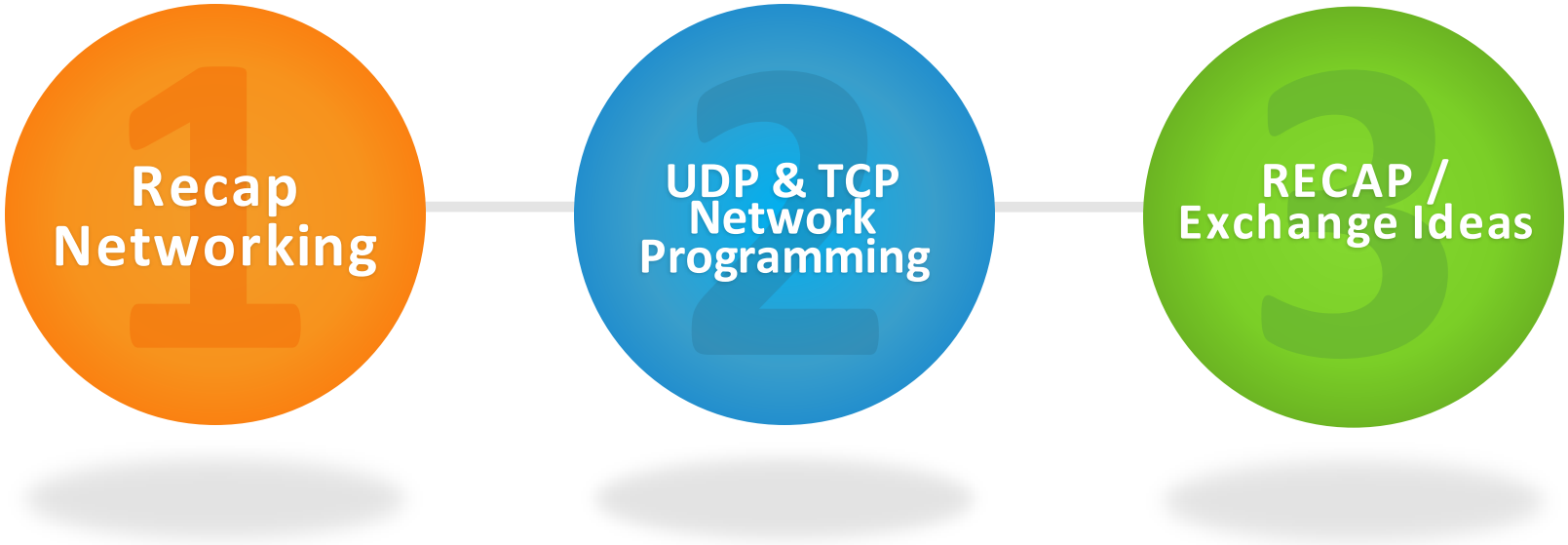
DAD – Distributed Applications Development

Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro

Cristian Toma – Business Card

Agenda for Lecture 2

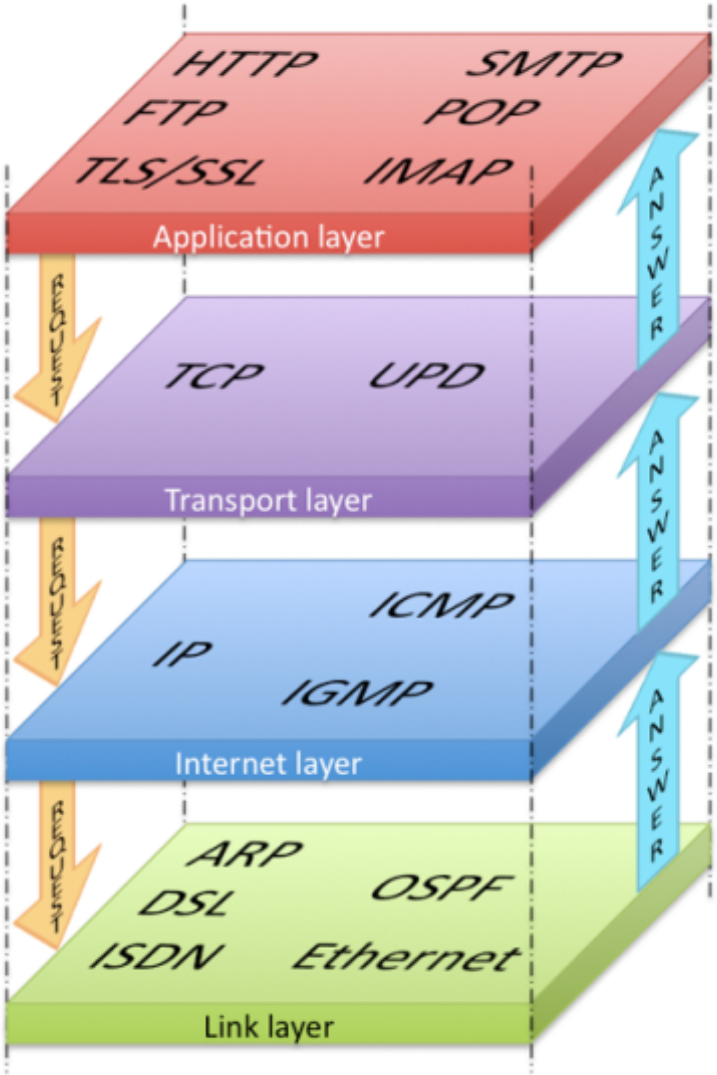


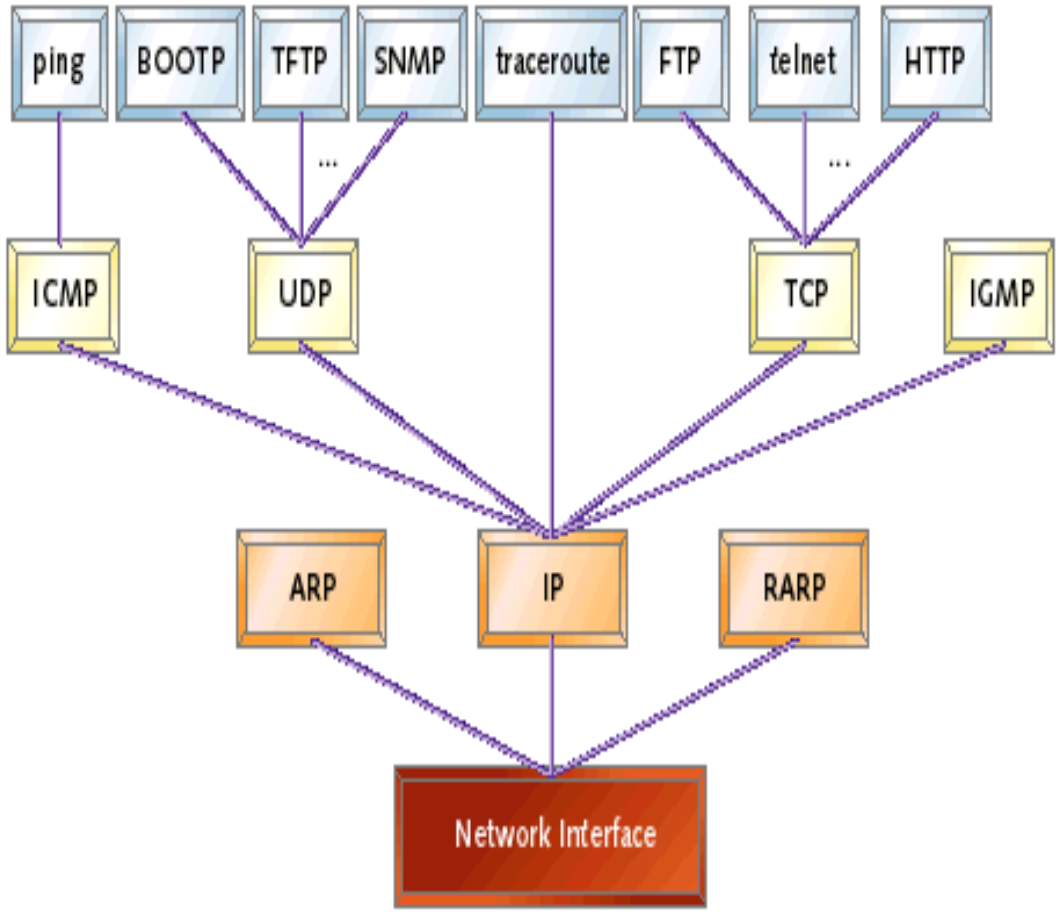
Networking IP, UDP and TCP programming, TCP/IP state machine, SNMP, SMTP

Networking Recapitulation

1. Networking TCP/IP Stack

HOW TCP/IP Works:



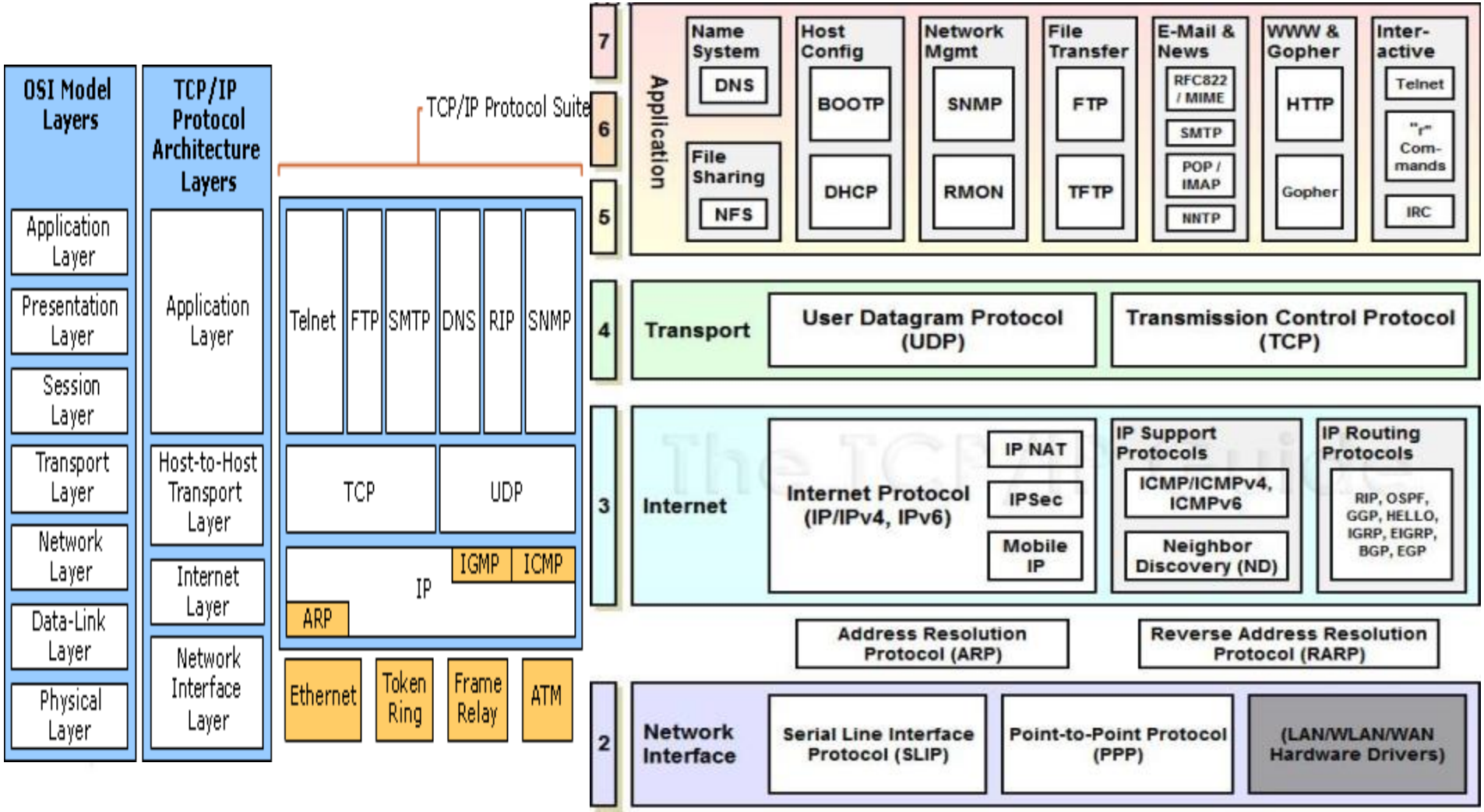


http://buildingautomationmonthly.com/tcpip-an-overview/

http://www.barrgroup.com/Embedded-Systems/How-To/Embedded-TCP-IP

1. Networking TCP/IP Stack

TCP/IP Stack Model:



http://technet.microsoft.com/en-us/library/cc786128(v=ws.10).aspx

http://talktoanit.com/c/?p=78

1. Networking TCP/IP Stack

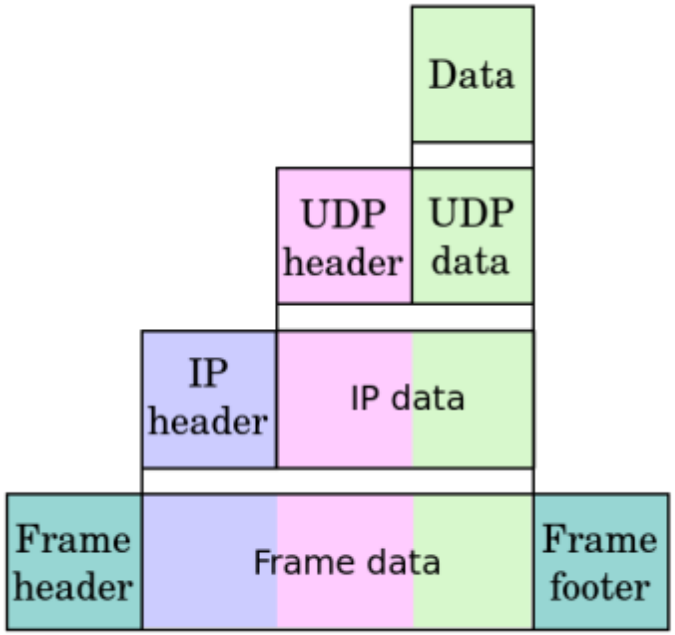
ISO/OSI Model vs. TCP/IP:

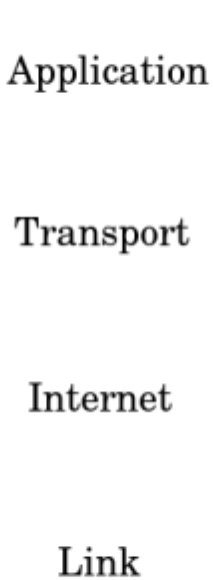
TCP/IP DoD Model			OSI Model
Application Layer (Services Layers 5,6,7) **PDU: Data**	HTTP: port **80** HTTPS/TLS/SSL: port **443** NNTP: port **119** FTP: port **21, 20** Telnet: port **23** SSH: port **22** POP3: port **110** IMAP4: port **143** SMTP: port **25**	DNS: port **53** TFTP: port **69** DHCP/BootP: port **67,68** SNMP: port **162, 161** NTP: port **123** Syslog: port **514**	Application Layer (7) **Scribe.** APIs, network services Serves the King/User Presentation Layer (6) **Translator.** Reformats, encrypts/de-crypts, compress/de-compress Session Layer (5) **Negotiator.** Establishes, manages and ends sessions.
Transport Layer (Host to Host Layer 4) **PDU: Segments**	TCP: protocol 6	UDP: protocol 17	Transport Layer (4) **Middle Manager.** Segment ID/Assembly
Internet Layer (Network Layer 3) **PDU: Packets**	IP	IP	Network Layer (3) **Mail Room Guy.** IP Addressing/Routing
Network Acces Layer 1 & 2 **PDU: Frame**	Ethernet, PPP Frame Relay MAC addresses, ARP	Ethernet, PPP Frame Relay MAC addresses, ARP	Data-Link Layer (2) **Envelope Stuffer.** Organizes bits into frames
Network Access Layer 1 & 2 **PDU: Bits or Data Stream**	Electrons, RF or Light	Electrons, RF or Light	Physical Layer (1) **The Truck.** Movement of bits.

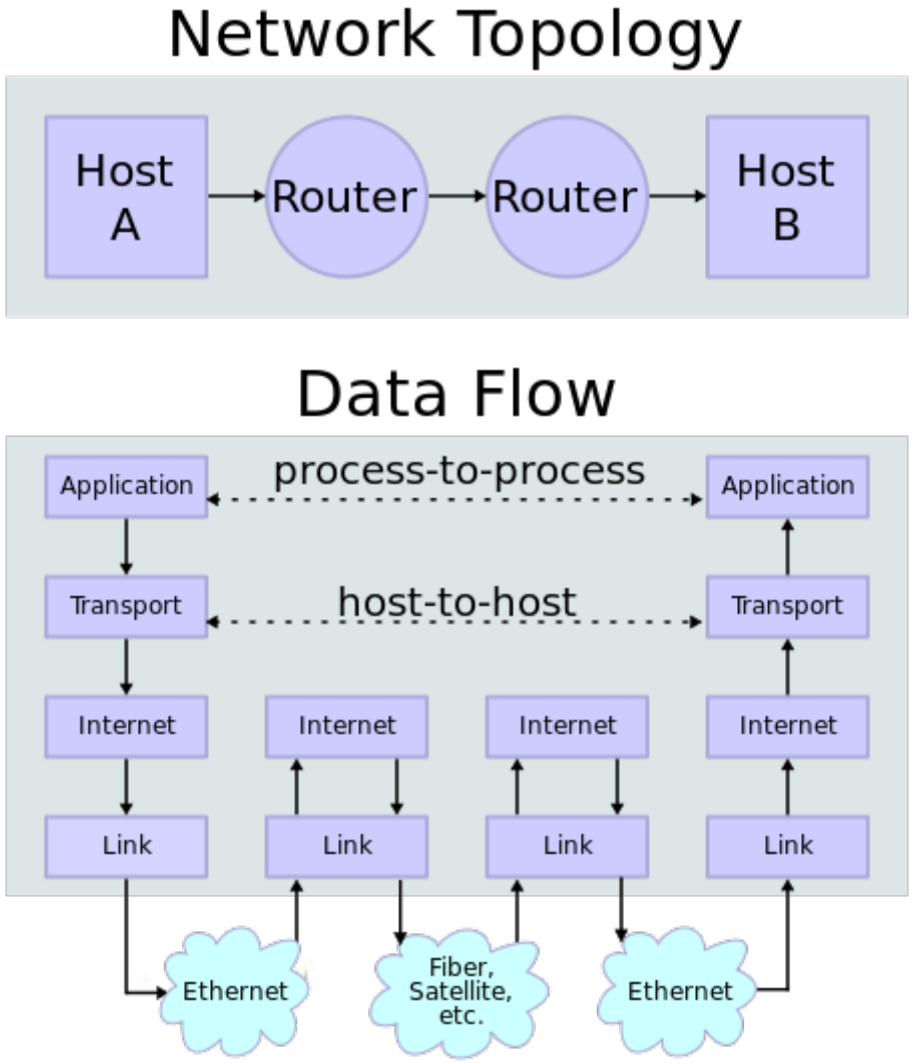
http://buildingautomationmonthly.com/tcpip-an-overview/

1. Networking TCP/IP Stack

TCP/IP Message Encapsulation:



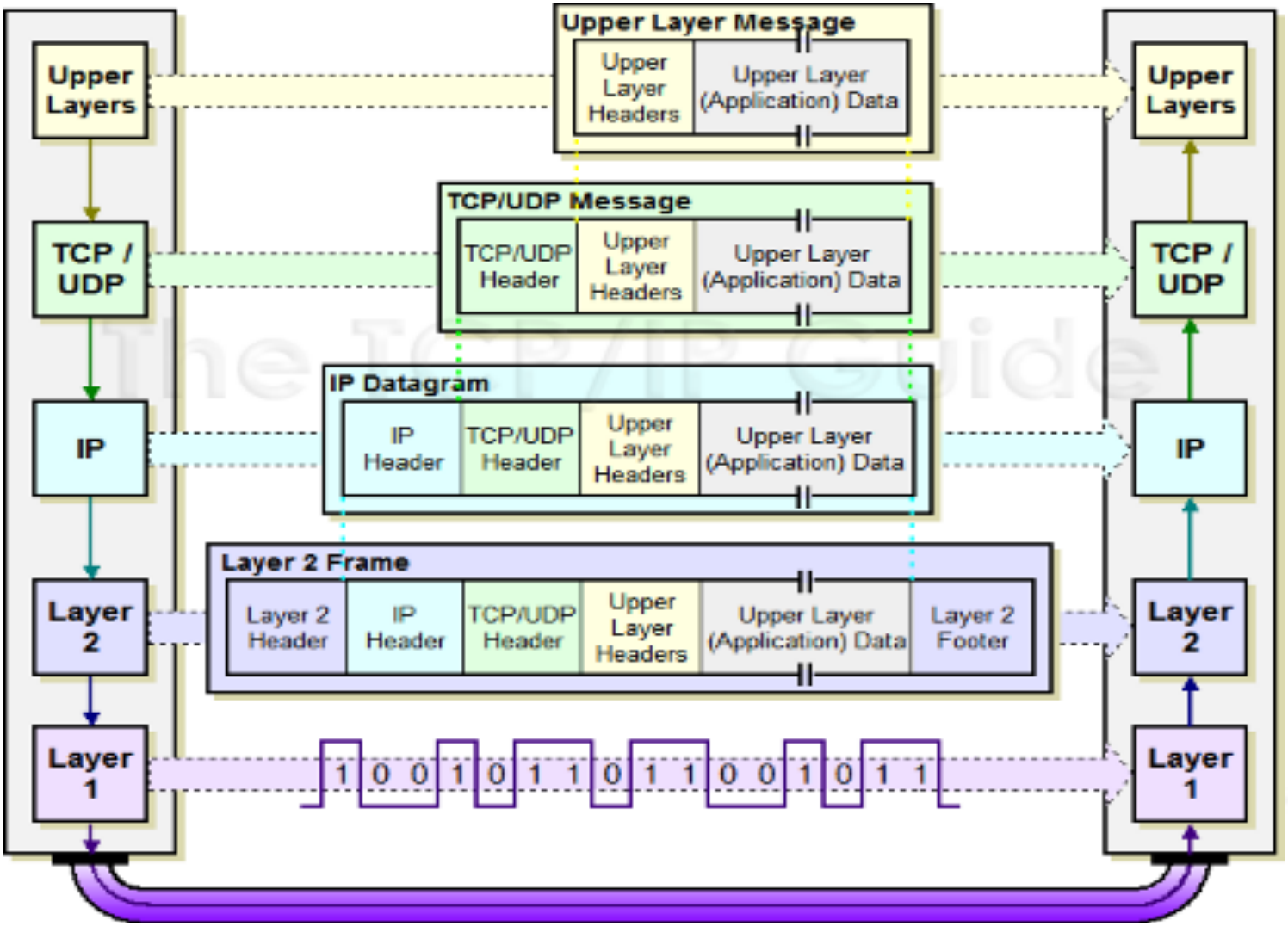




http://en.wikipedia.org/wiki/Internet_protocol_suite

1. Networking TCP/IP Stack

TCP/IP Message Flow:

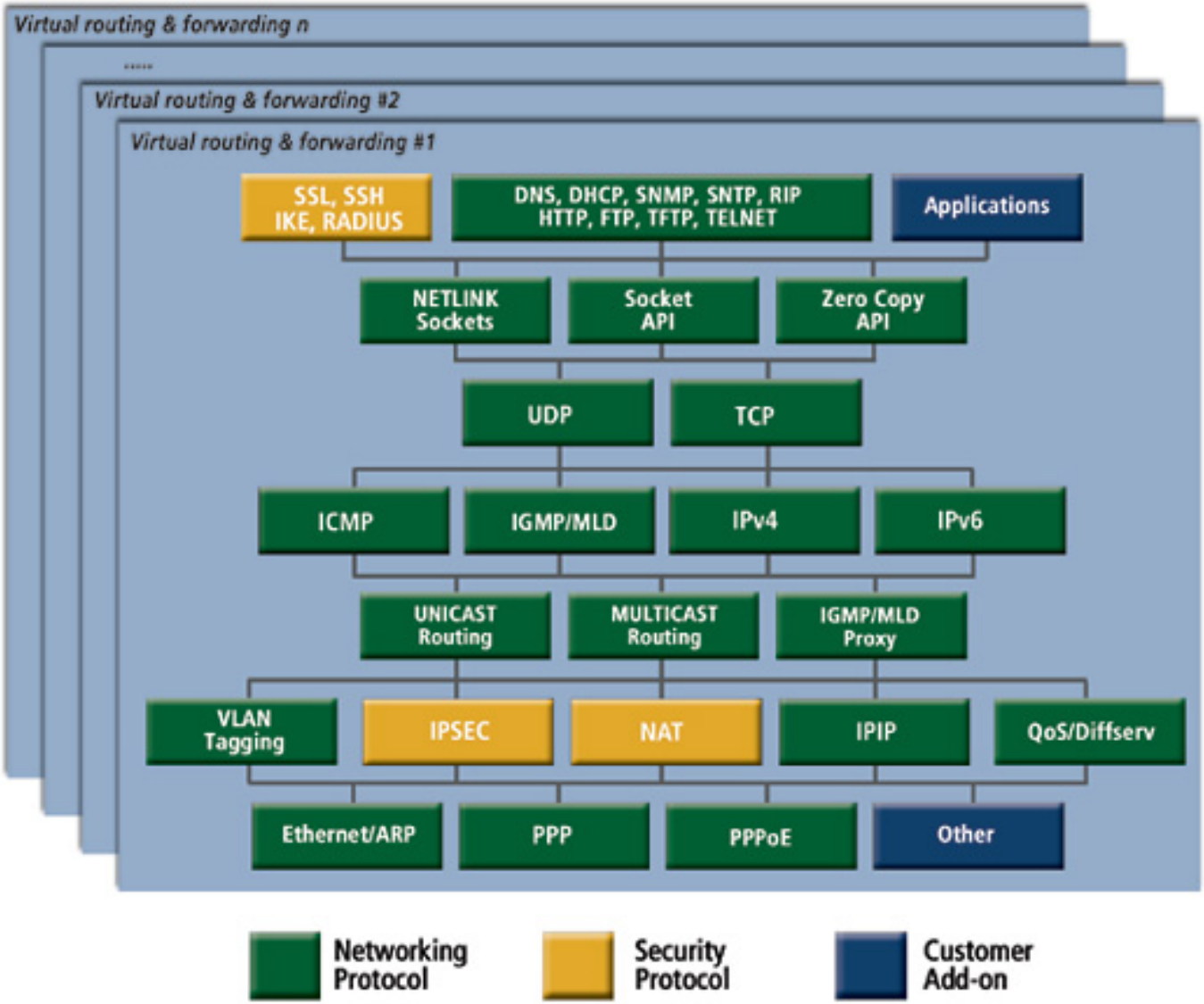


http://buildingautomationmonthly.com/tcpip-an-overview/

1. Networking TCP/IP Stack

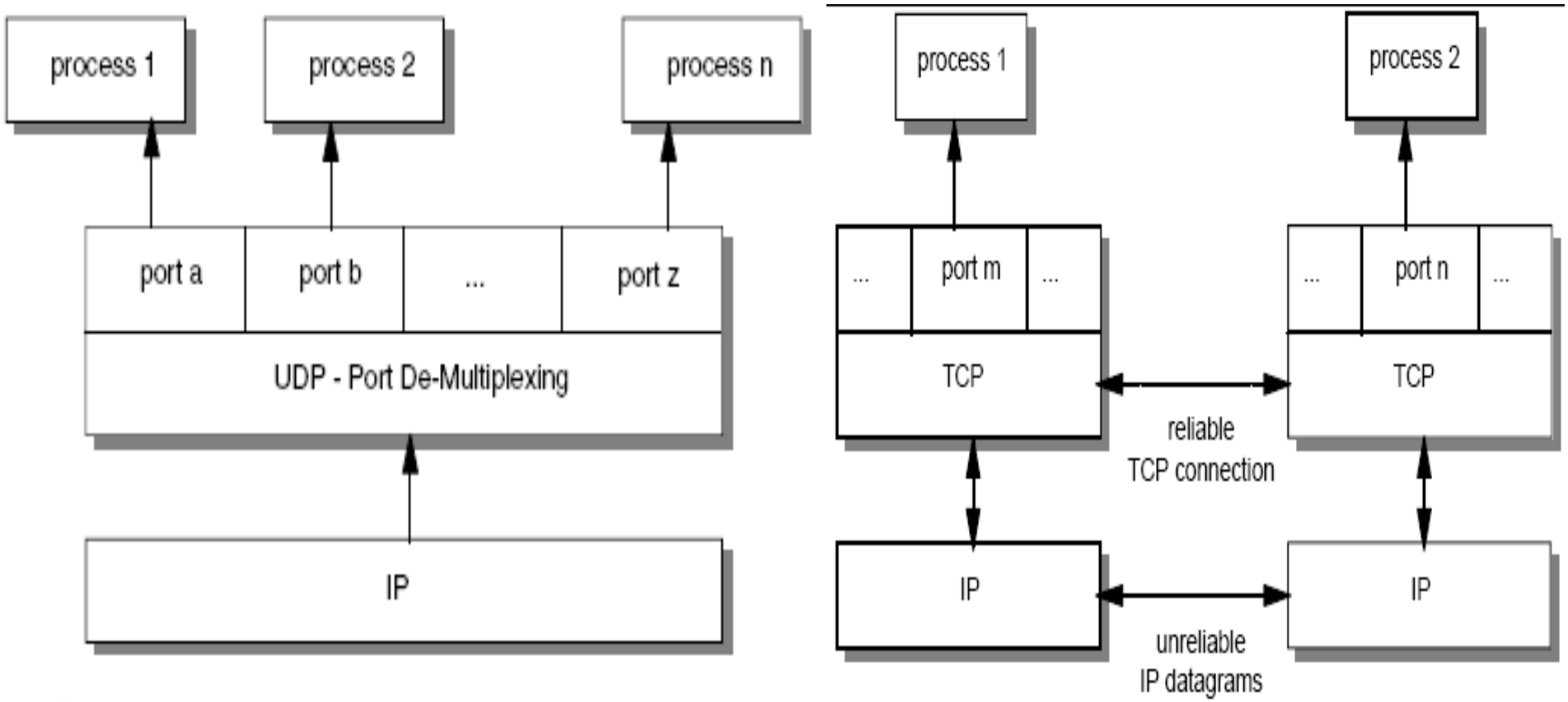
TCP/IP Virtual LAN/Networks:

http://www.ghs.com/products/comm_tcp-ip.html



1. Networking TCP/IP Stack

TCP/IP App/Port Multiplexing:

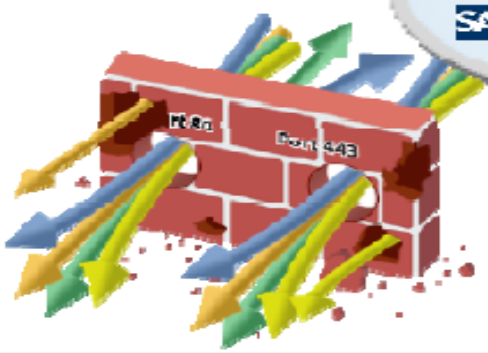


1. Networking TCP/IP Stack

What is running on port 80?

Applications Have Changed – Firewalls Have Not

- The gateway at the trust border is the right place to enforce policy control
  - Sees all traffic
  - Defines trust boundary

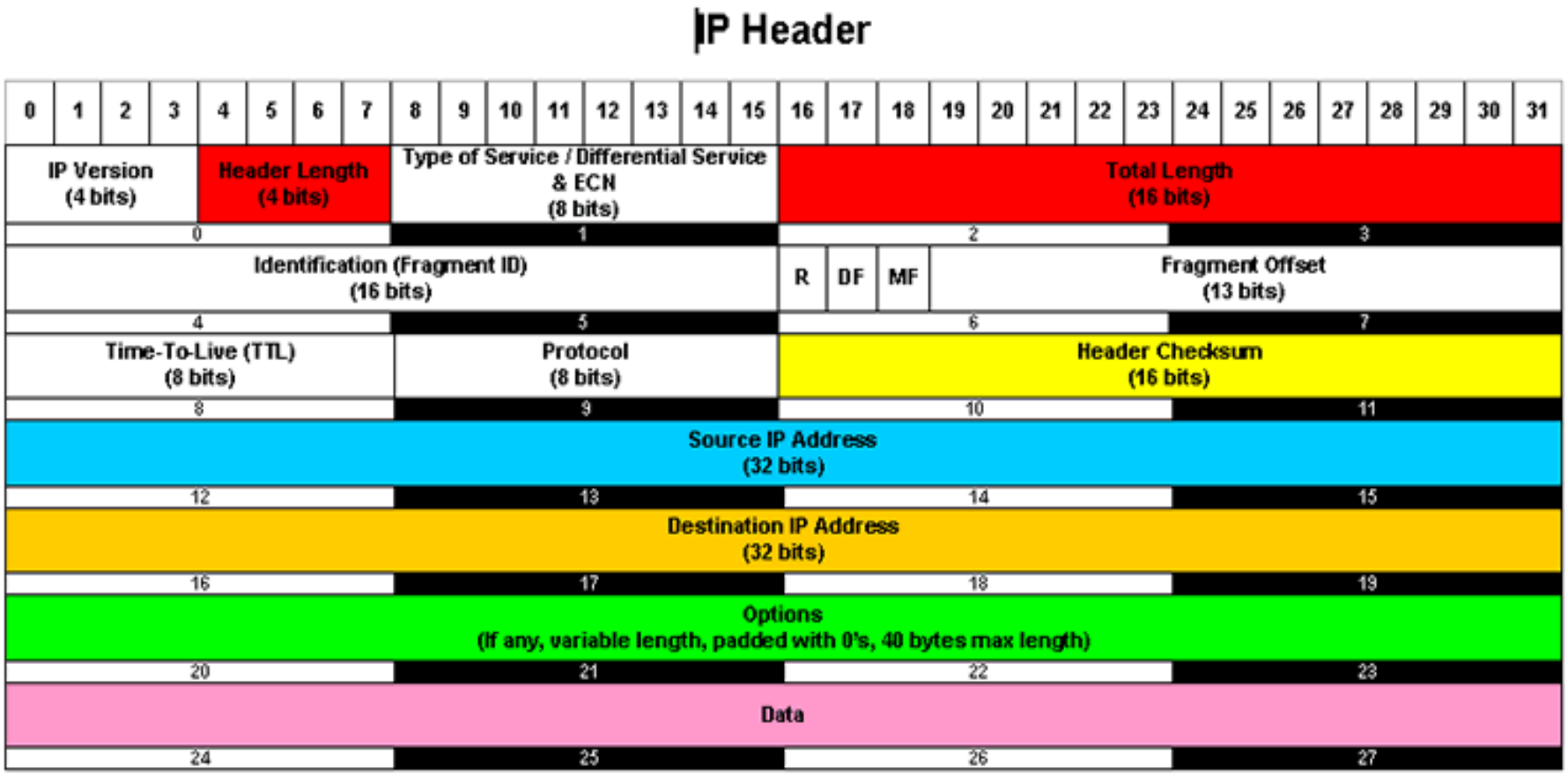


- BUT…Applications Have Changed
  - Ports ≠Applications
  - IP Addresses ≠Users
  - Packets ≠Content

Need to Restore Visibility and Control in the Firewall

1. Networking TCP/IP Stack

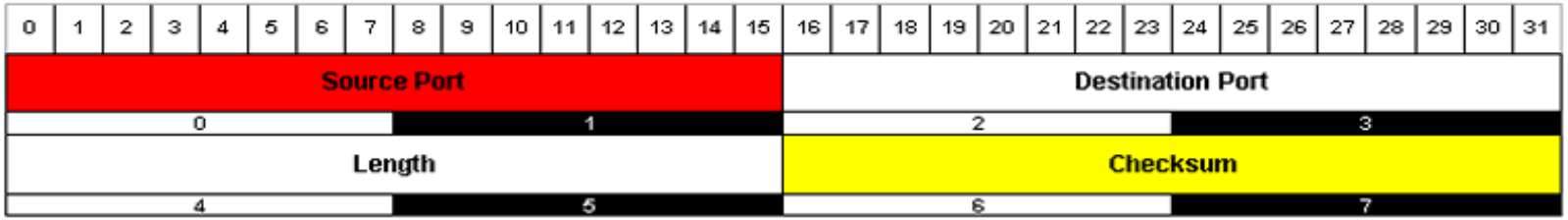
IP Header - RFC 791 – Submasking + Routing + NAT:



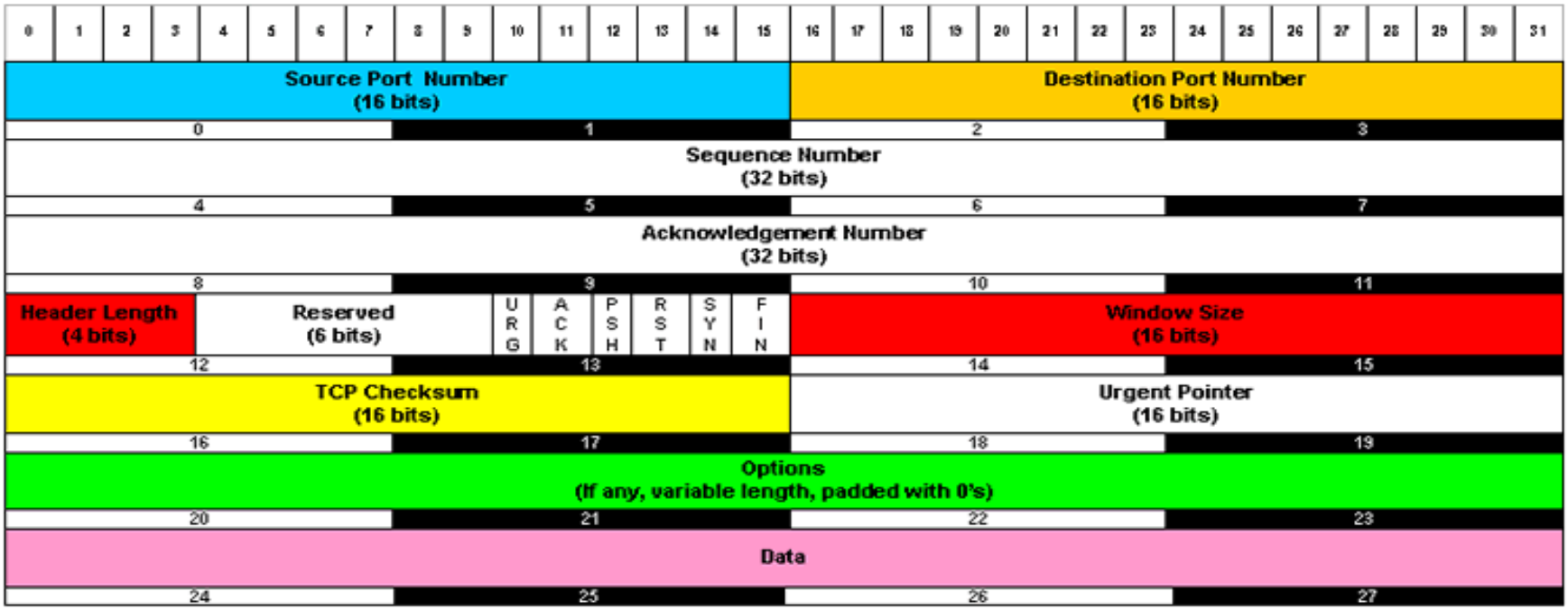
1. Networking TCP/IP Stack

UDP Header – RFC 768 – Connection-less vs. TCP Header – RFC 793 – Connection-oriented

UDP Header

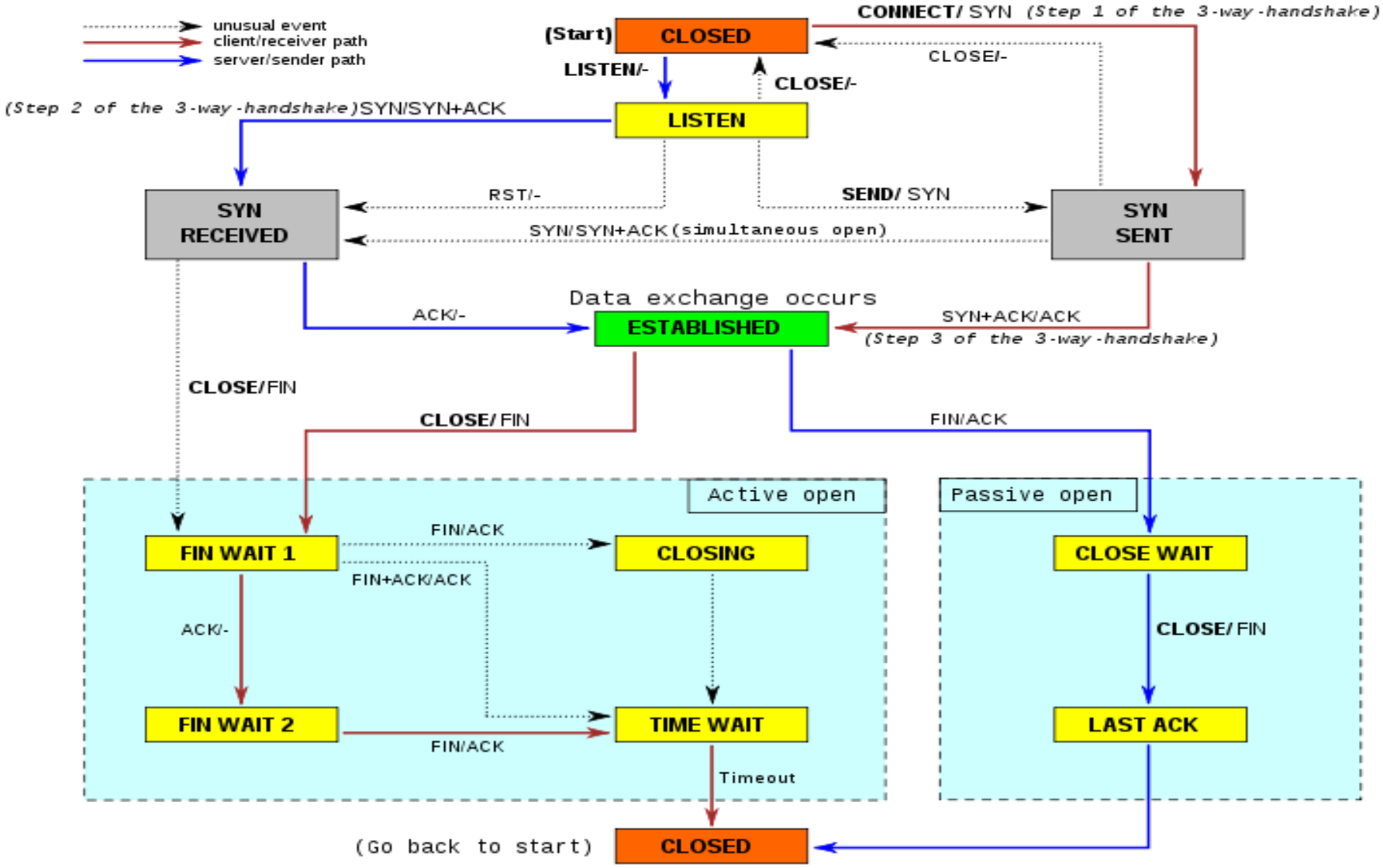


TCP Header



1. Networking TCP/IP Stack

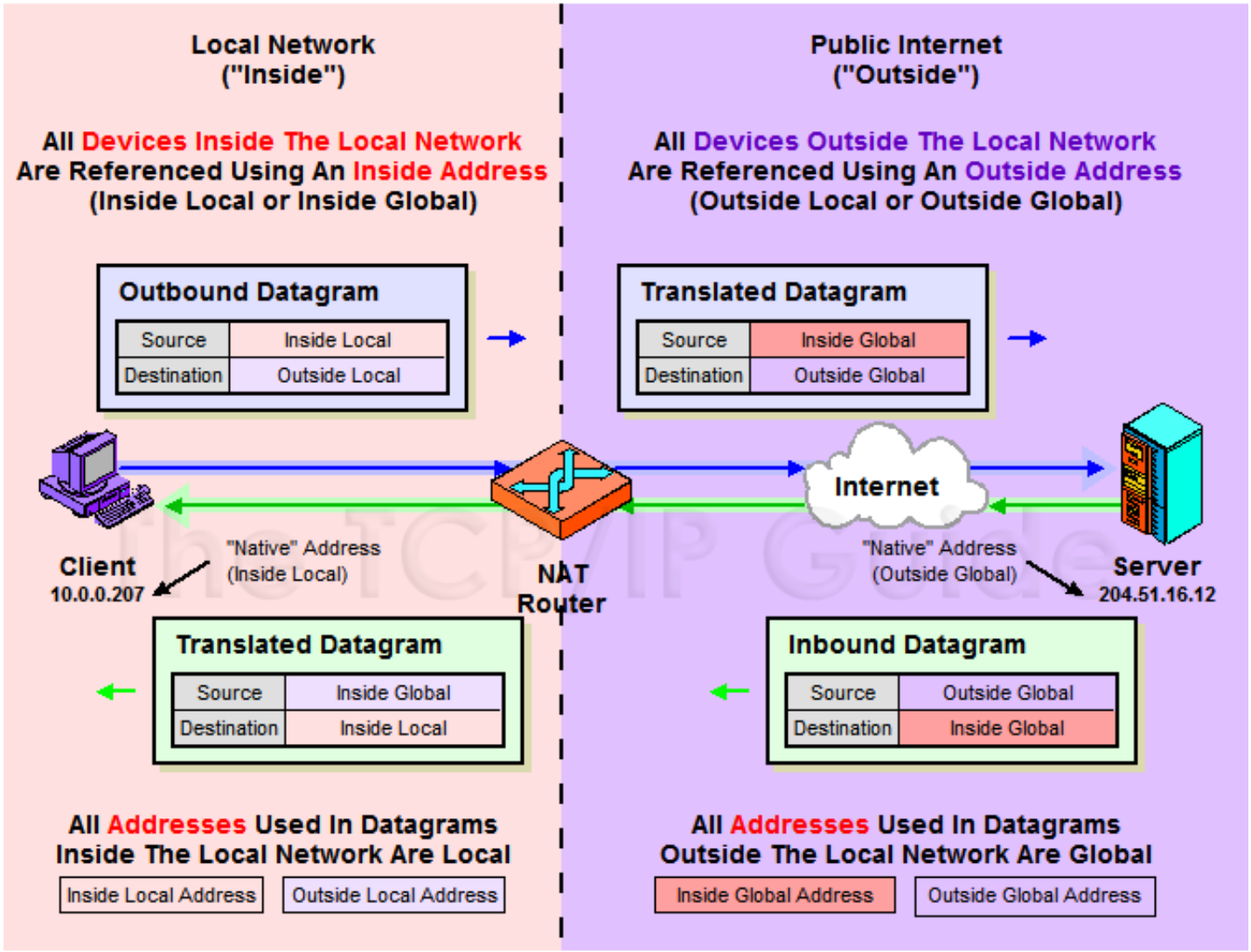
TCP State Machine – RFC 793:



1. Networking TCP/IP Stack

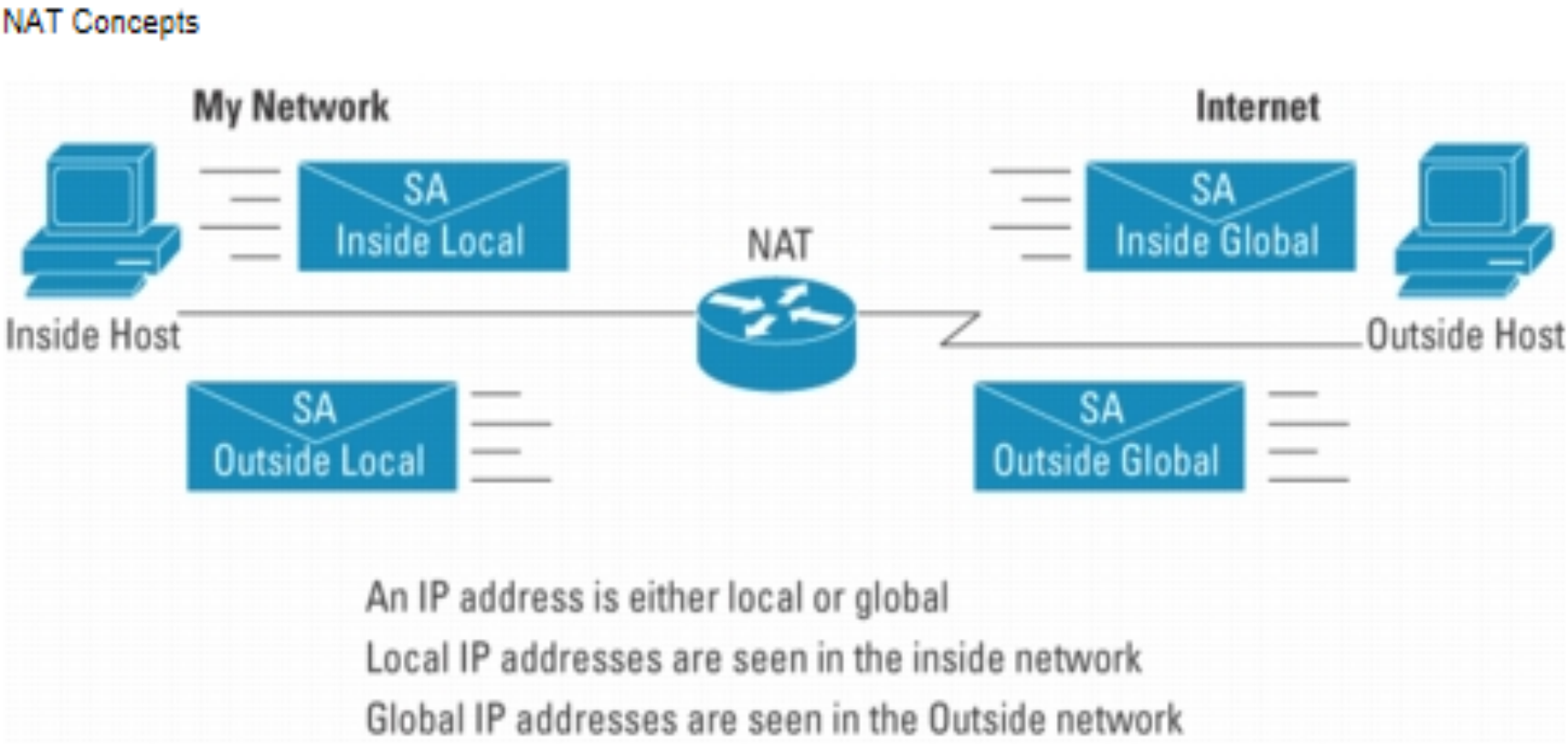
TCP/IP NAT & PAT:

http://www.tcpipguide.com/free/t_IPNATAddressTerminology-3.htm



1. Networking TCP/IP Stack

TCP/IP NAT:

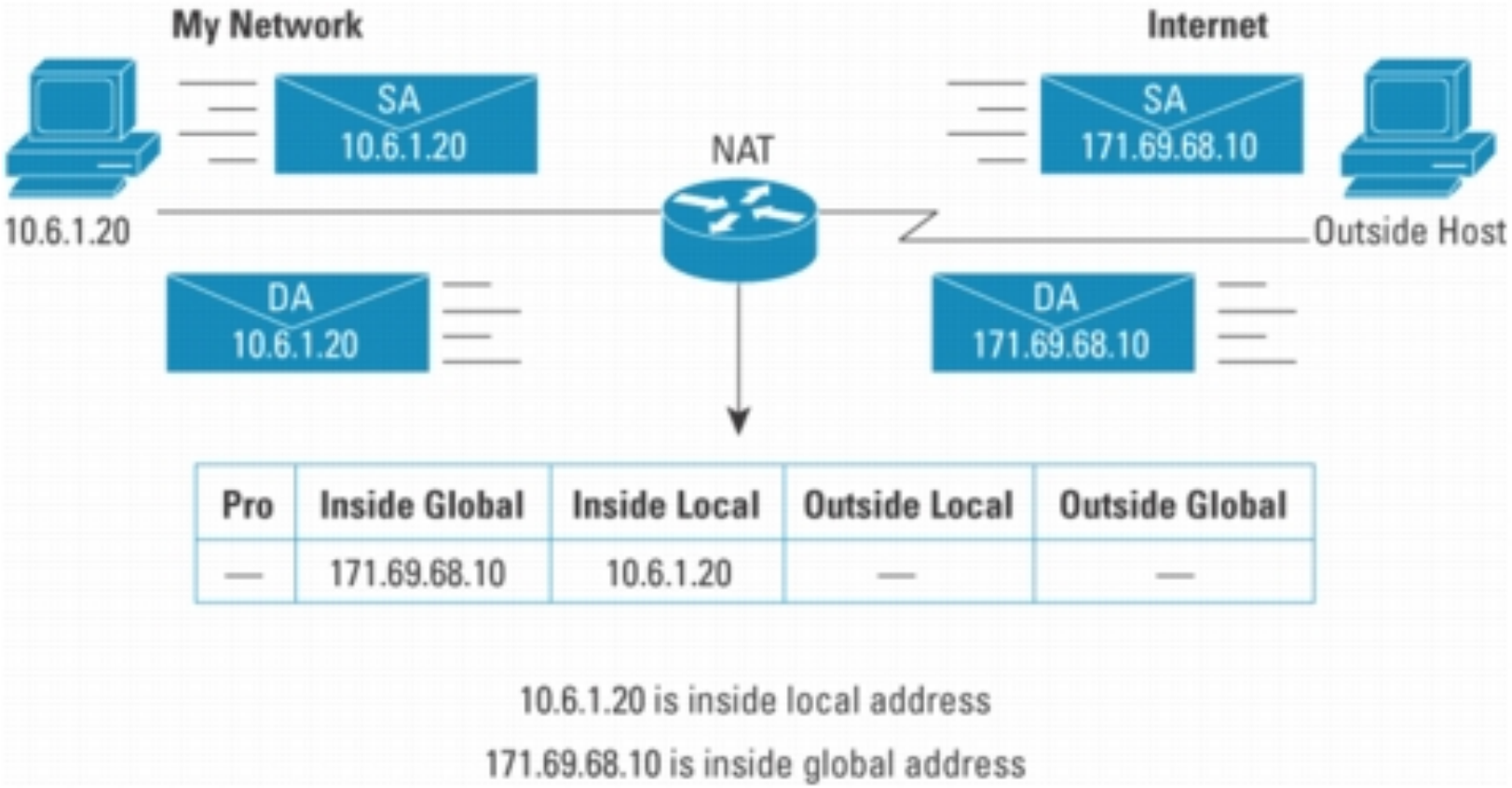


http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP NAT:

NAT Terminology « Inside Addressing»



Pro	Inside Global	Inside Local	Outside Local	Outside Global
—	171.69.68.10	10.6.1.20	—	—

10.6.1.20 is inside local address

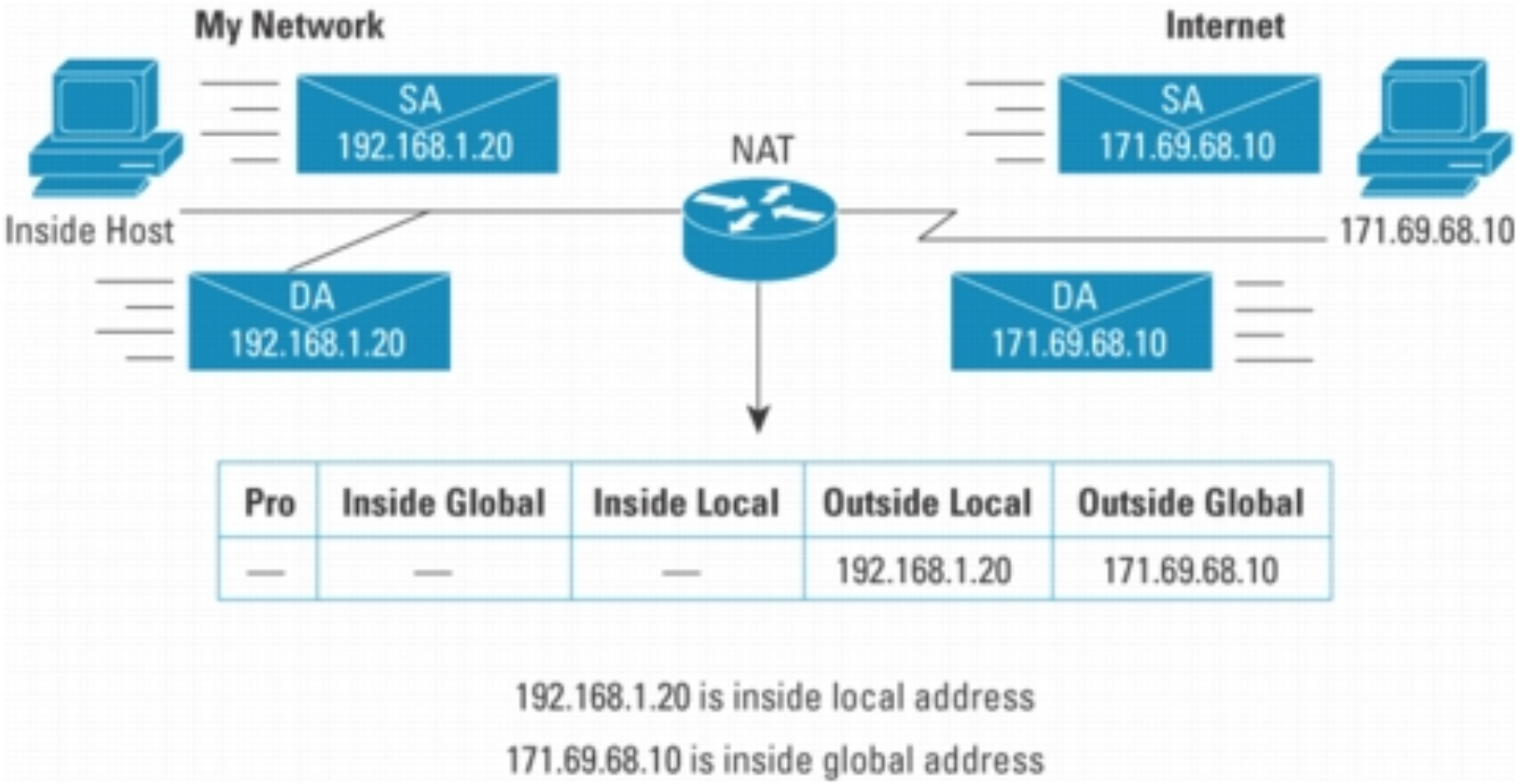
171.69.68.10 is inside global address

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP NAT:

NAT Terminology "Outside Addressing"



Pro	Inside Global	Inside Local	Outside Local	Outside Global
—	—	—	192.168.1.20	171.69.68.10

192.168.1.20 is inside local address

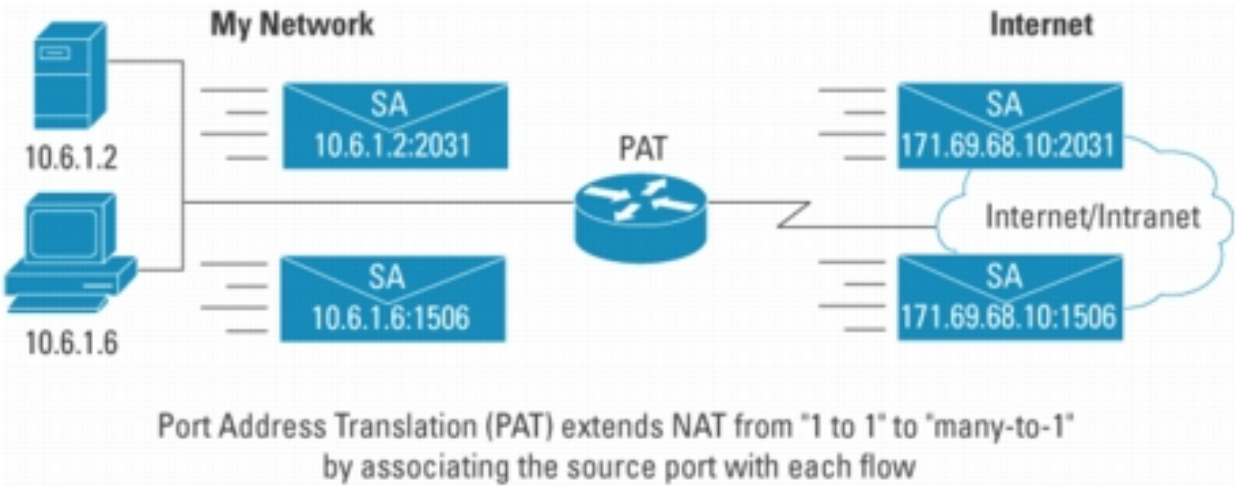
171.69.68.10 is inside global address

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1. Networking TCP/IP Stack

TCP/IP PAT:

Basic Concepts of PAT



Port Address Translation (PAT) extends NAT from "1 to 1" to "many-to-1" by associating the source port with each flow

Figure 5
Unique Source Port per Translation Entry

Pro	Inside Global	Inside Local	Outside Local	Outside Global
tcp	171.69.68.5:1405	10.6.15.2:1405	204.71.200.69:80	204.71.200.69:80

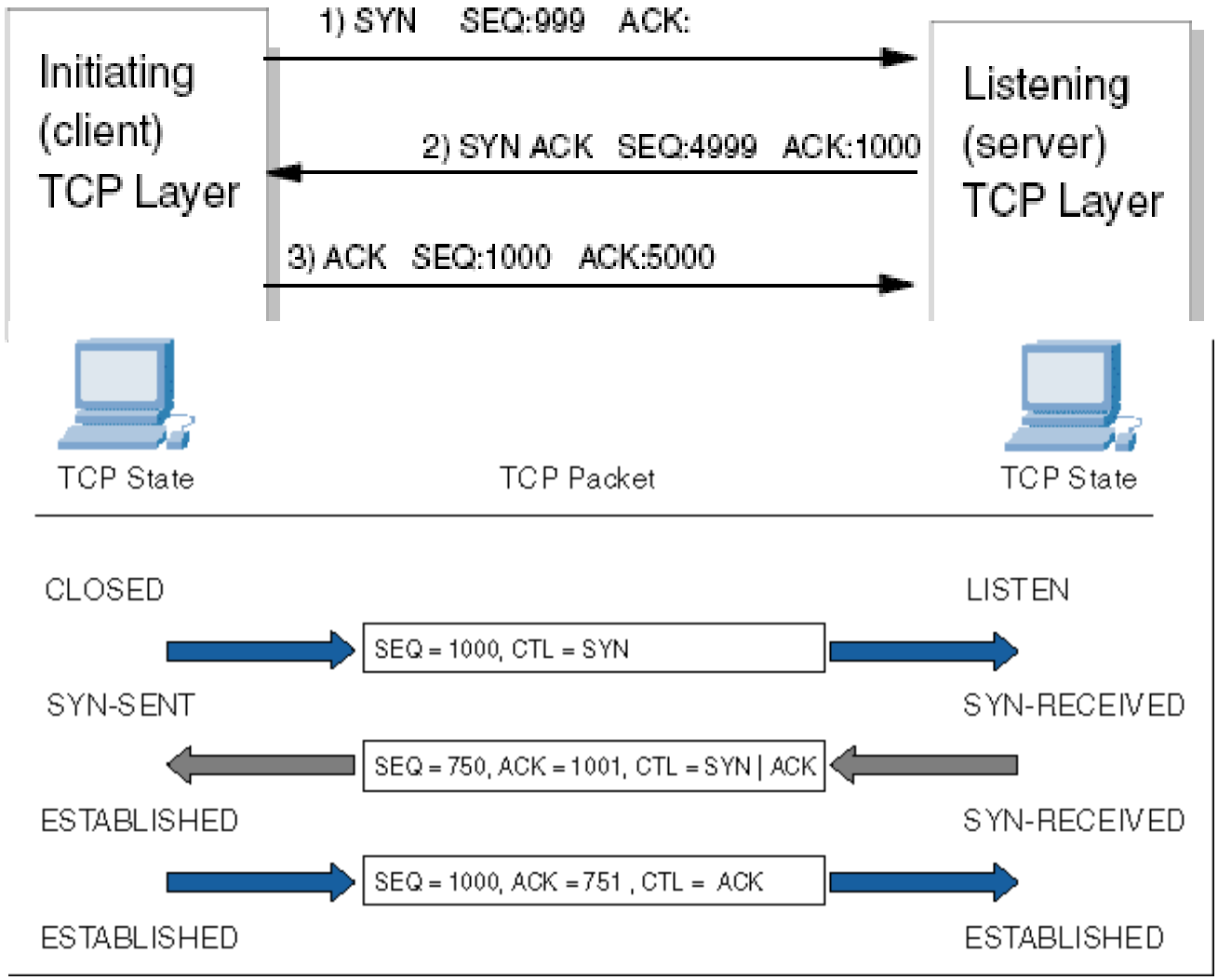
PAT (Port Address Translation) includes ports in addition to IP addresses

- Many-to-one translation
- Maps multiple IP addresses to 1 or a few IP addresses
- Unique source port number identifies each session
- Conserves registered IP addresses
- Also called NAPT in IETF documents

http://www.cisco.com/en/US/technologies/tk648/tk361/tk438/technologies_white_paper09186a0080091cb9.html

1.2 TCP/IP Networking Programming

TCP Handshake:



http://www.cisco.com/web/about/ac123/ac147/ac174/ac196/about_cisco_ipj_archive_article09186a00800c8417.html

Section Conclusion

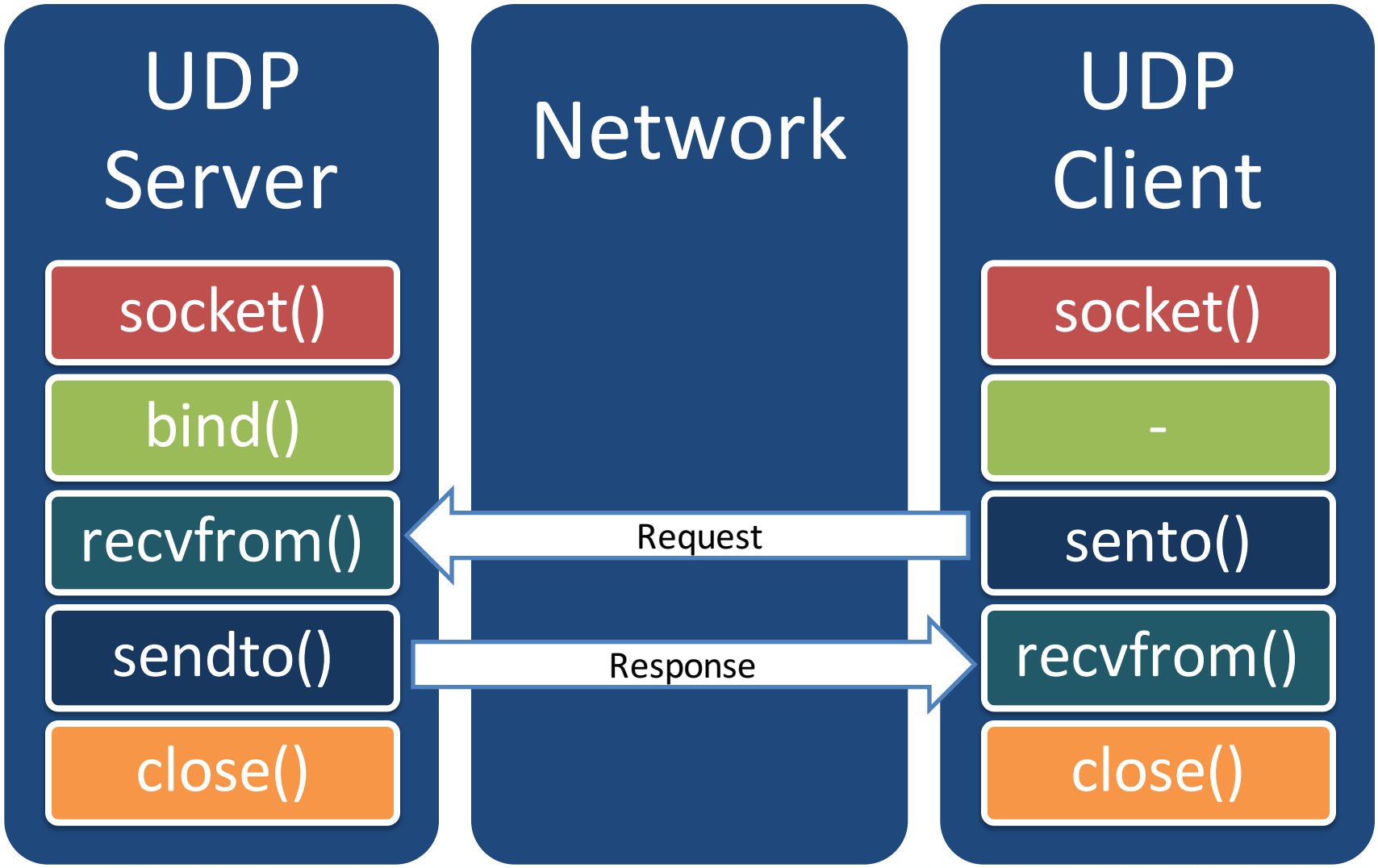
Fact: DAD needs Networking

In few **samples** it is simple to remember: UDP and TCP programming is useful for HTC … didactical samples: SNMP over UDP for monitoring, SMTP/IMAP4/POP3 over TCP for e-mail notification, FTP over TCP for file and data transfer, HTTP over TCP for web & web services, and in general TCP for RPC/RMI, CORBA, JMS, SOAP, P2P-JXTA – for distributed computing and systems.

UDP Client-Server programming, SNMP client sample, TCP Client-Server programming, SMTP client sample

Network UDP & TCP Programming

2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:



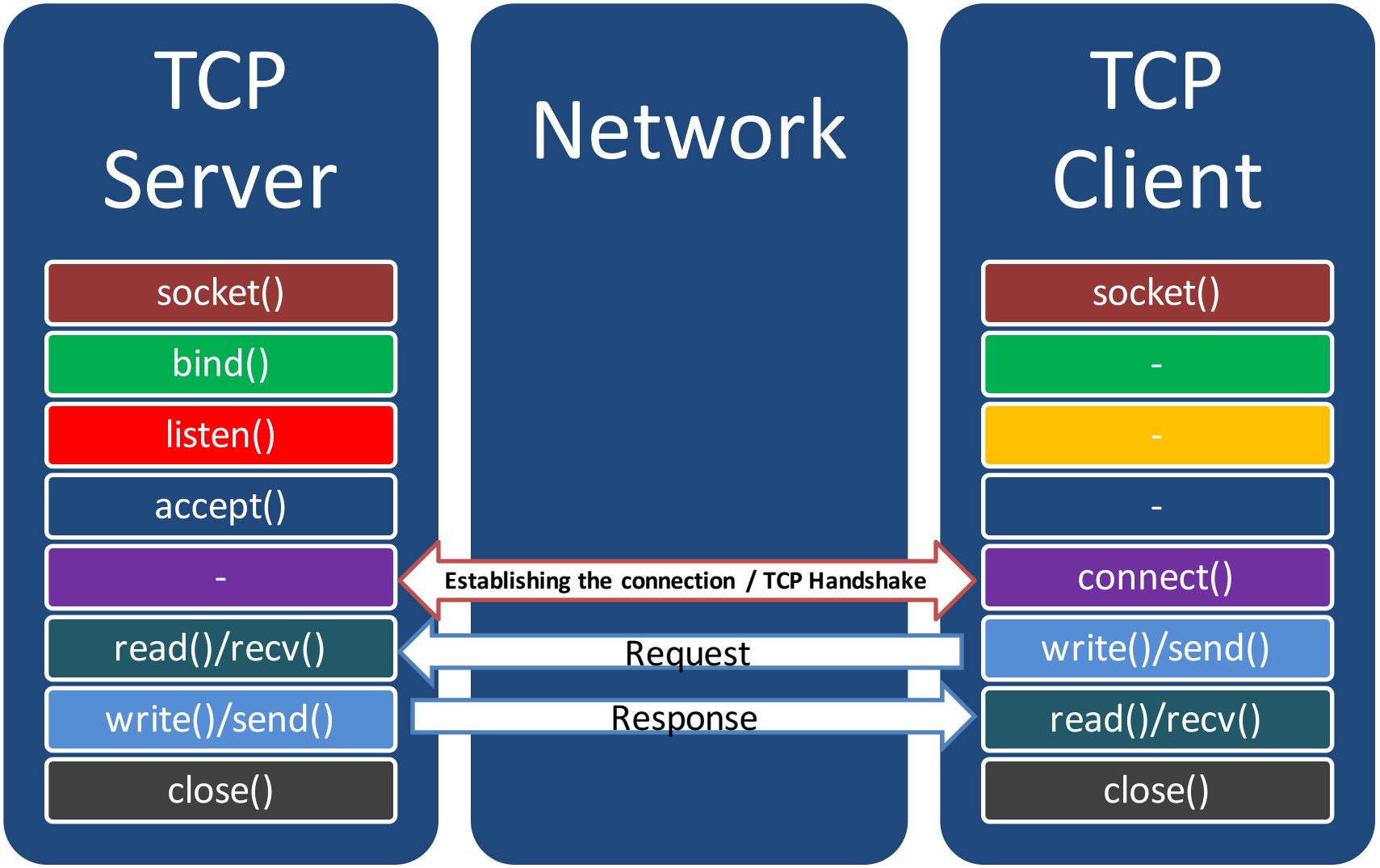
2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

```
package eu.ase.net.udp;
import java.io.*;
import java.net.*;
public class UDPServer {
    public static void main(String[] args) {
        // get a datagram socket
        DatagramSocket socket = null;
        byte[] bufResp = null;
        byte[] bufRecv = null;
        try {
         socket = new DatagramSocket(778);//it is correct because this constructor executes "bind"
         while(true) {
            bufRecv = new byte[256];
            // receive request
            DatagramPacket packet = new DatagramPacket(bufRecv, bufRecv.length);
            socket.receive(packet);

            // figure out response
            String respString = new String("OK");
            bufResp = respString.getBytes();

            // send the response to the client at "address" and "port"
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(bufResp, bufResp.length, address, port);
            socket.send(packet);
         }
        } catch(IOException ioe) {
```

2.1 TCP/IP Networking Programming – UDP Programming – Socket Primitives:

```java
package eu.ase.net.udp;
import java.io.*;
import java.net.*;
public class UDPClient {
    public static void main(String[] args) throws IOException {
        // get a datagram socket
        DatagramSocket socket = new DatagramSocket();

        // send request
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName("127.0.0.1");
        DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 778);
        socket.send(packet);

        // get response
        byte[] bufResp = new byte[256];
        packet = new DatagramPacket(bufResp, bufResp.length);
        socket.receive(packet);

        // display response
        String received = new String(packet.getData());
        System.out.print("Client de la server: " + received);

        // close socket
        socket.close();
    }
}
```

2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:



2.2 TCP/IP Networking Programming – TCP Programming – Socket Primitives:

Server

Client

```
ServerSocket serverSocket = null;
Socket clientSocket = null;

boolean listening = true;

OutputStream os = null; PrintWriter out = null;
InputStream is = null; BufferedReader in = null;
String inputLine = null, outputLine = null;


//SEVERSOCKET = SOCKET+BIND+LISTEN
serverSocket = new ServerSocket(4801);
clientSocket = serverSocket.accept();          //ACCEPT


//STABILIREA CONEXIUNII
is = clientSocket.getInputStream();
in = new BufferedReader(new InputStreamReader(is));

os = clientSocket.getOutputStream();
out = new PrintWriter(os, true);


 while ((inputLine = in.readLine()) != null) {
         System.out.println(inputLine);
         outputLine = new String("OK");
         out.println(outputLine);
         out.flush();
         if (outputLine.compareTo("La revedere!") == 0) {break;}
 }
```

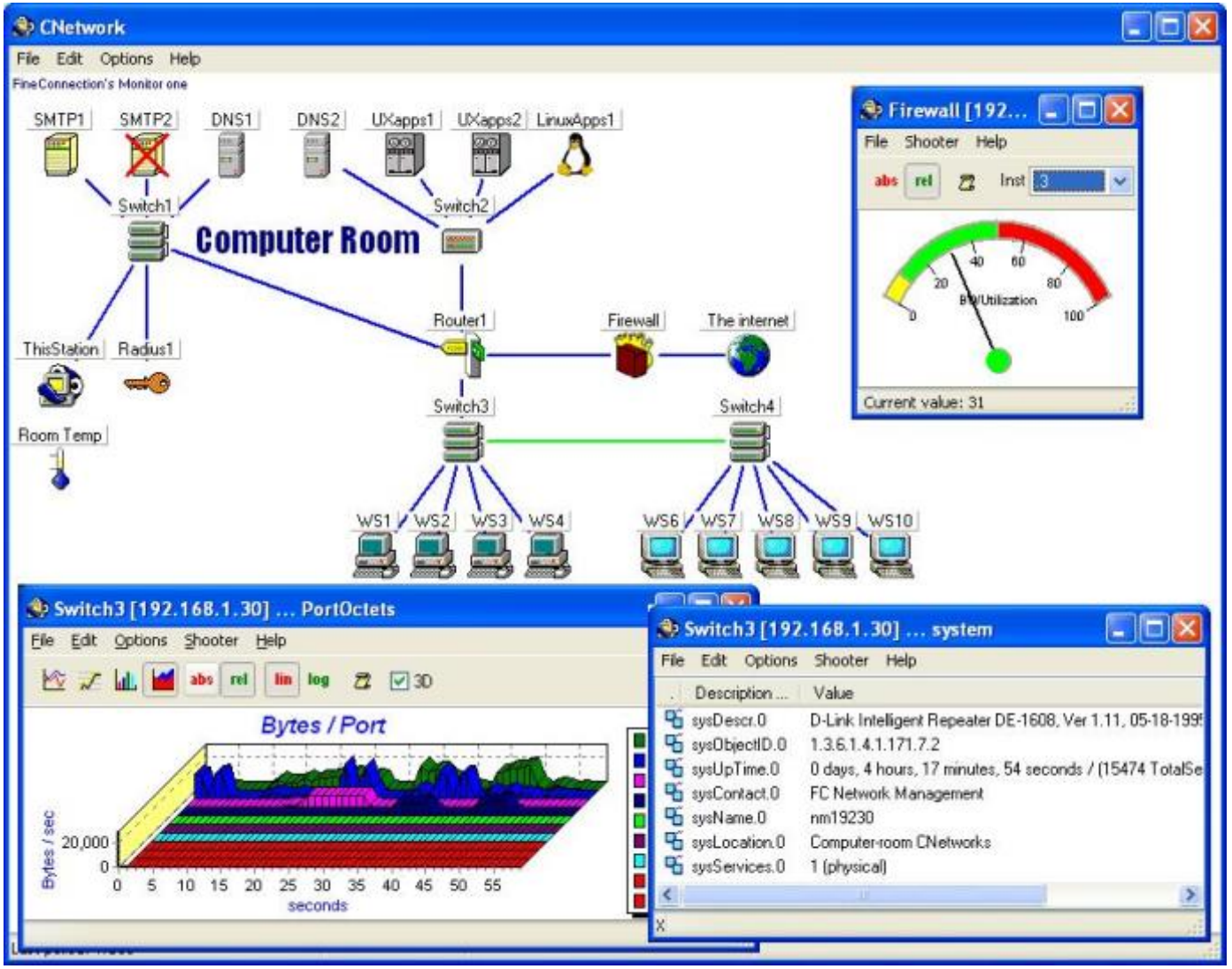
in <- outC

out -> inC

```
Socket clientSocket = null;
PrintWriter outC = null;
BufferedReader inC = null;


clientSocket = new Socket(args[0],
Integer.parseInt(args[1]));//SOCKET


//STABILIREA CONEXIUNII
//CONNECT = OUT2SERVER + INfromSERVER

//OUT2SERVER
outC = new PrintWriter(clientSocket.getOutputStream(), true);

//INfromSERVER
inC = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));


String lin = "";
outC.println("As vrea sa ma conectez.");//SEND
lin = inC.readLine();                    //RECV
System.out.println("Sever: " + lin);
```

2.3 SNMP – RFC 1157

What is the necessity for SNMP – Simple? Network Management Protocol?



http://www.filedudes.com/images/screenshots/27179.jpg

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

```
RFC1157-SNMP DEFINITIONS ::= BEGIN

    IMPORTS
        ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
            FROM RFC1155-SMI;

        -- top-level message

        Message ::=
                SEQUENCE {
                    version          -- version-1 for this RFC
                        INTEGER {
                            version-1(0)
                        },

                    community        -- community name
                        OCTET STRING,

                    data             -- e.g., PDUs if trivial
                        ANY          -- authentication is being used
                }

        -- protocol data units

        PDUs ::=
                CHOICE {
                            get-request
                                GetRequest-PDU,

                            get-next-request
                                GetNextRequest-PDU,

                            get-response
                                GetResponse-PDU,

                            set-request
                                SetRequest-PDU,

                            trap
                                Trap-PDU
                        }

        -- PDUs

        GetRequest-PDU ::=
            [0]
                IMPLICIT PDU

        GetNextRequest-PDU ::=
            [1]
                IMPLICIT PDU

        GetResponse-PDU ::=
            [2]
                IMPLICIT PDU

        SetRequest-PDU ::=
            [3]
                IMPLICIT PDU

        PDU ::=
                SEQUENCE {
                   request-id
                        INTEGER,

                    error-status      -- sometimes ignored
                        INTEGER {
                            noError(0),
                            tooBig(1),
                            noSuchName(2),
                            badValue(3),
```

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

Some of the **data types** used in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
BOOLEAN	Primitive	0x01	Boolean value: yes/no
INTEGER	Primitive	0x02	Negative and positive integers
BIT STRING	Primitive	0x03	Bit sequence
OCTET STRING	Primitive	0x04	Byte sequence (1 byte=8 bits=1octet)
NULL	Primitive	0x05	A null value
OBJECT IDENTIFIER	Primitive	0x06	An object identifier, which is a sequence of integer components that identify an object such as an algorithm or attribute type
PrintableString	Primitive	0x13	An arbitrary string of printable characters
T61String	Primitive	0x14	An arbitrary string of T.61 (eight-bit) characters
IA5String	Primitive	0x16	An arbitrary string of IA5 (ASCII) characters
UTCTime	Primitive	0x17	A "coordinated universal time" or Greenwich Mean Time (GMT) value

2.3 SNMP – RFC 1157

ASN.1 DER and BER:

Some of **structured types** defined in ASN.1 are presented in the following listing:

Data Type	Sort	Tag Number	Meaning
SEQUENCE	Constructed	0x30	An ordered collection of one or more types
SEQUENCE OF	Constructed	0x10	An ordered collection of zero or more occurrences of a given type
SET	Constructed	0x31	An unordered collection of one or more types
SET OF	Constructed	0x11	An unordered collection of zero or more occurrences of a given type
A0, A1, ...	Constructed	0xAz	Where z = 0..F in hex and it represents the z-th element in SEQUENCE data type

2.3 SNMP – RFC 1157

ASN.1 TLV:

T - tag	L - length	V - value
1-2 bytes	1-3 bytes	n bytes

TLV Object

Tag	Length	Value				
0x85	0x05	0x46	0x72	0x61	0x6E	0x6B

Tag for first names | Length of the first name | First name "Frank" in ASCII

Tag for constructed data object | **Length** of the constructed data object | **Value** of the constructed data object

Tc	Lc	Vc

T_{p1}	L_{p1}	V_{p1}	T_{p2}	L_{p2}	V_{p2}	T_{p3}	L_{p3}	V_{p3}

2.3 SNMP – RFC 1157

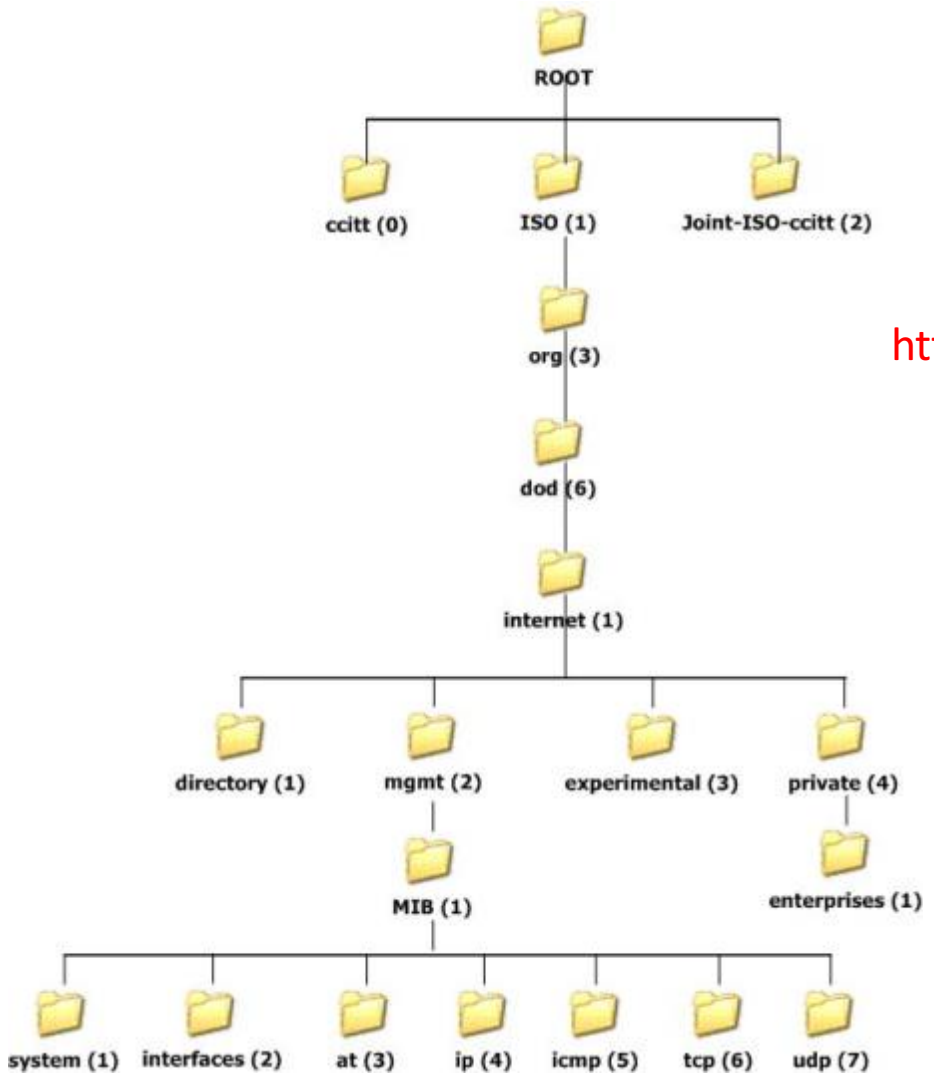
ASN.1 DER and BER:

T - tag			L - length			
b7 b6 b5 b4 b3 b2 b1 b0	Meaning	b7 b6 b5 b4 b3 b2 b1 b0	Byte 1	Byte 2	Byte 3	Meaning
0 0	Universal class	0	0 - 127	...	...	One byte is needed
0 1	Application class		0x81	128-255		Two bytes are needed
1 0	Context-specific class	The value of tag code in range 31-127	0x82	256-65535		Three bytes are needed
1 1	Private class					
... ... 0	Primitive data object					
... ... 1	Constructed data obj.					
... ... Y Y Y Y Y	Tag code (0-30)					
... ... 1 1 1 1 1	Pointer to the following byte (byte 2), which specifies the tag code					

Byte 1 | Byte 2 | Byte 3,4,5

2.3 SNMP – RFC 1157

ASN.1 OID:



http://www.oid-info.com/cgi-bin/display

2.3 SNMP – RFC 1157

ASN.1 OID:

1.2.840.113549.1.1.5
({iso(1) member-body(2) us(840) rsadsi(113549)
pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)})

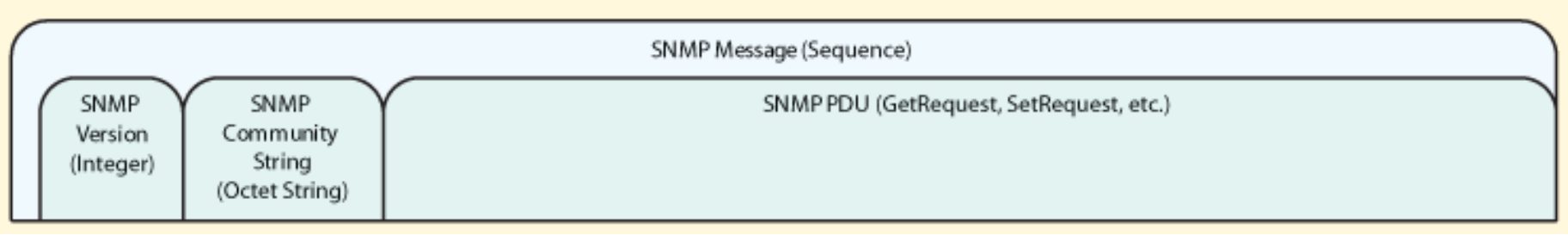
sample of OID is 1.2.840.113549.1.1.5 ({iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}) for signature obtained from SHA-1 digest on the message and RSA algorithm applied to the digest. This encoding will be at byte level: 0x06 0x09 0x2A 0x86 0x48 0x86 0xF7 0x0D 0x01 0x01 0x05. The first byte shows that here there is a OID – OBJECT IDENTIFIER field. The OID has 9 bytes length because of second byte in array. Because the first bit in length field is not set the length field has only one byte. The length field has value 9 which means the OID structure has the payload data in 9 bytes.

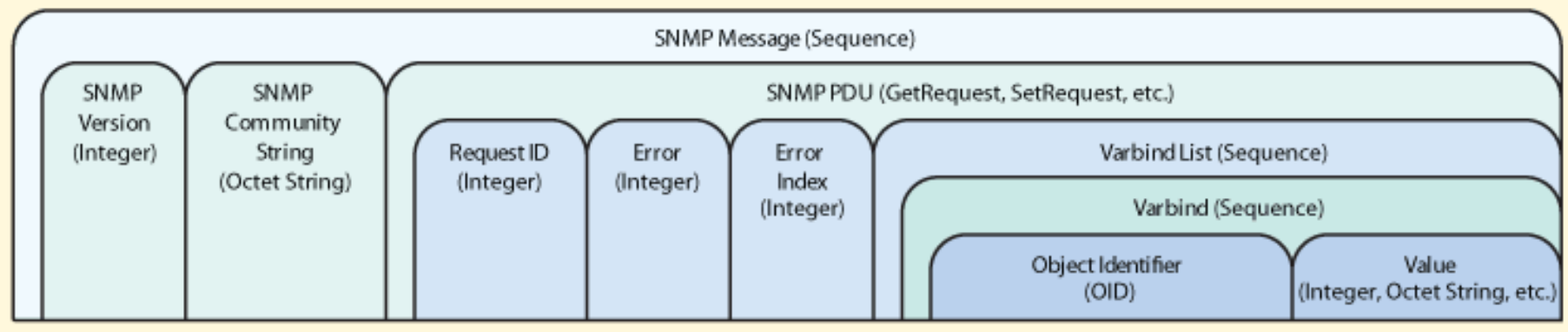
2.3 SNMP – RFC 1157

ASN.1 OID:

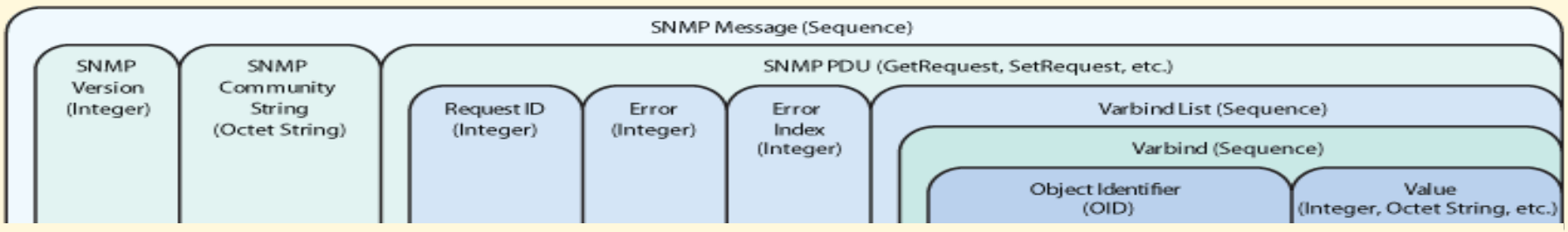
According to BER, the first two numbers of any OID (x.y) are encoded as one value using the formula (40*x)+y. The first two numbers in an OID are here 1.2. Therefore, the first two numbers of an OID are encoded as 42 or 0x2A, because (40*1)+2 = 42. After the first two numbers are encoded, the subsequent numbers in the OID are each encoded as a byte. However, a special rule is required for large numbers because one byte (eight bits) can only represent a number from 0-255. This is the case for 840 and 113549. For 840 from the OID, the first bit of the first byte should be set. The number occupies enough number of bytes till the last byte of representation is not having the first bit set. In 840 case 0x48 has the first bit NOT set. So, the formula is the last hex digit from the first byte multiplied with 2^7 = 128 (1 bit is for multiple bytes representation) and added with the value from the second byte as long as the second byte has a value less than 0x80. In 840 case, the formula is: 0x06*128 + 0x48 (form 0x86 0x48)= 768 + 72 = 840. In case of 113549 we choose the bytes 0x86 0xF7 0x0D because 0x0D is the last byte with first bit (sign bit) not set. The formula for 113549 is:
0x06*2^14 + 0x77*2^7 + 0x0d*2^0 = 6*16384 + 119*128 + 13*1 = 98304 + 15232 + 13 = 113549. For the remaining encoding {...pkcs(1) pkcs-1(1) sha1-with-rsa-signature(5)}, we will have only one byte for each number: 0x01 0x01 0x05.

2.3 SNMP – RFC 1157



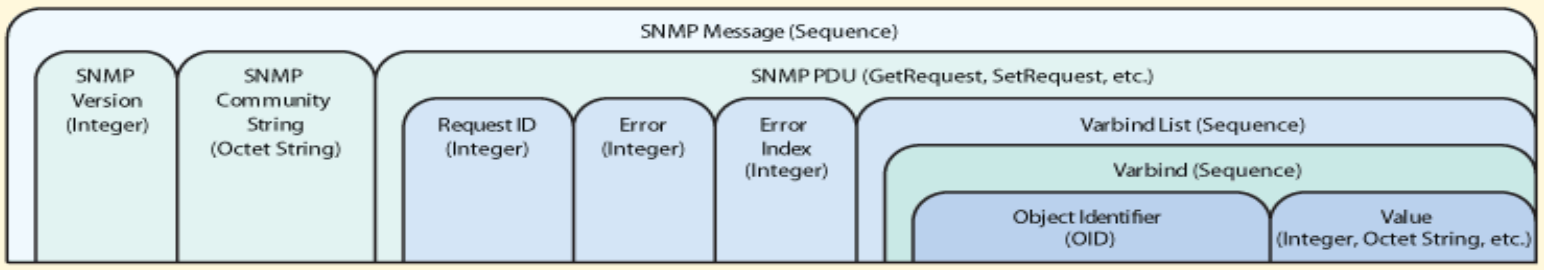


2.3 SNMP – RFC 1157



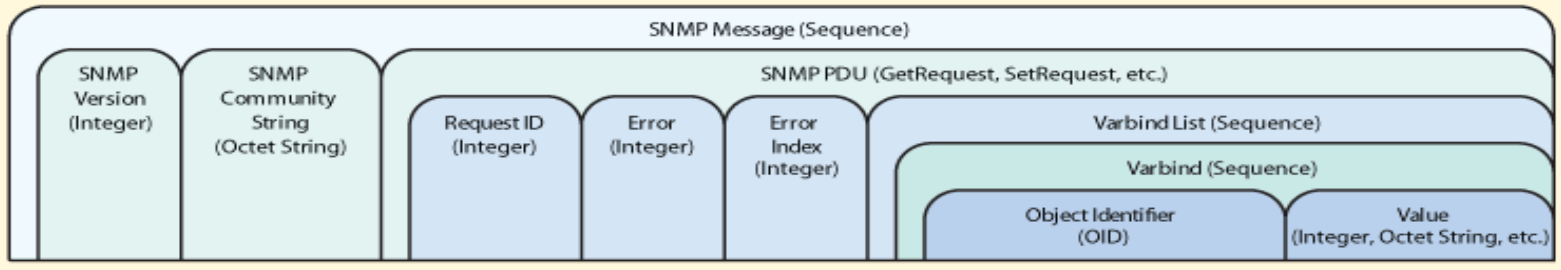
Field	Description
SNMP message	A Sequence representing the entire SNMP message consisting of the SNMP version, Community String, and SNMP PDU.
SNMP Version	An Integer that identifies the version of SNMP. SNMPv1 = 0
SNMP Community String	An Octet String that may contain a string used to add security to SNMP devices.
SNMP PDU	An SNMP PDU contains the body of the SNMP message. There are several types of PDUs. Three common PDUs are GetRequest, GetResponse, SetRequest.
Request ID	An Integer that identifies a particular SNMP request. This index is echoed back in the response from the SNMP agent, allowing the SNMP manager to match an incoming response to the appropriate request.

2.3 SNMP – RFC 1157



Error	An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:
	• 0x00 -- No error occurred
	• 0x01 -- Response message too large to transport
	• 0x02 -- The name of the requested object was not found
	• 0x03 -- A data type in the request did not match the data type in the SNMP agent
	• 0x04 -- The SNMP manager attempted to set a read-only parameter
	• 0x05 -- General Error (some error other than the ones listed above)

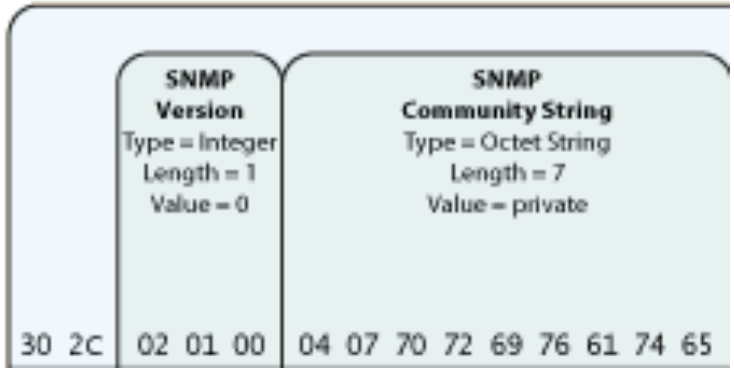
2.3 SNMP – RFC 1157

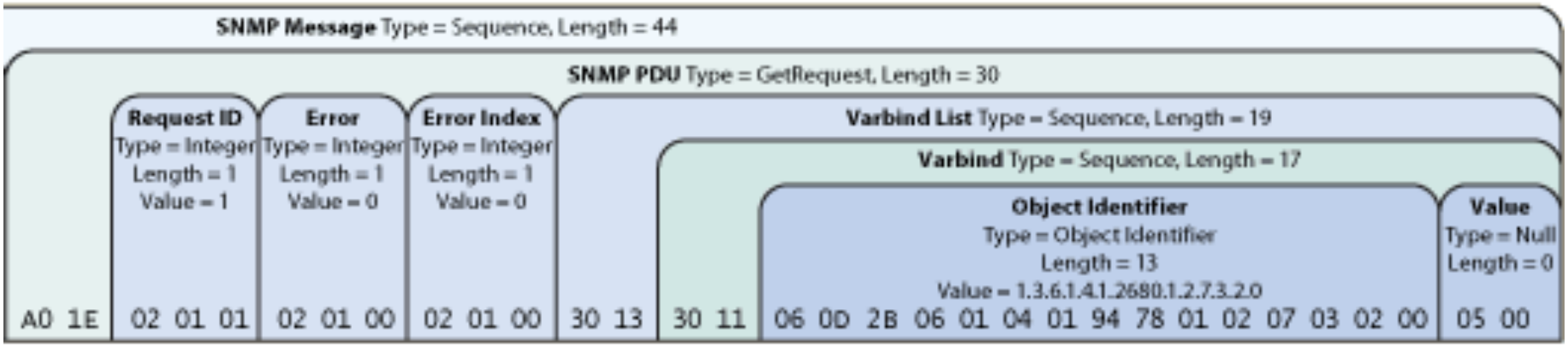


Error Index	If an Error occurs, the Error Index holds a pointer to the Object that caused the error, otherwise the Error Index is 0x00.
Varbind List	A Sequence of Varbinds.
Varbind	A Sequence of two fields, an Object ID and the value for/from that Object ID.
Object Identifier	An Object Identifier that points to a particular parameter in the SNMP agent.

Value	SetRequest PDU -- Value is applied to the specified OID of the SNMP agent.
	GetRequest PDU -- Value is a Null that acts as a placeholder for the return data.
	GetResponse PDU -- The returned Value from the specified OID of the SNMP agent.

2.3 SNMP – RFC 1157





2.4 SMTP – RFC 2821

D.1 A Typical SMTP Transaction Scenario

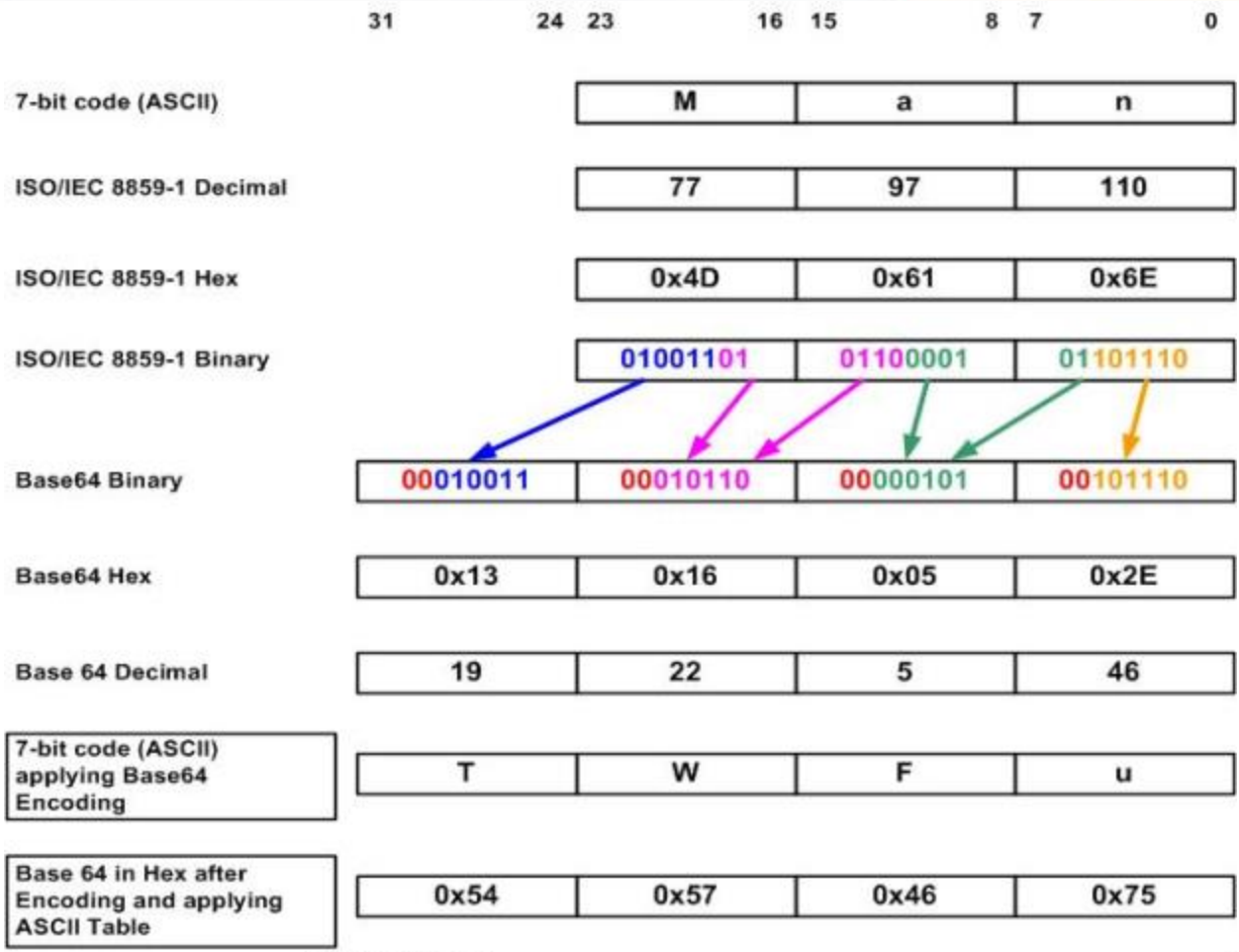
This SMTP example shows mail sent by Smith at host bar.com, to Jones, Green, and Brown at host foo.com. Here we assume that host bar.com contacts host foo.com directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host foo.com.

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>

S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

2.4 SMTP – RFC 2821

Base64 encoding is used in practice usually for transport over the network and heterogeneous environments binary code such as pictures or executable code. The techniques is very simple: to transform each 3 bytes values into 4 bytes value in order to avoid to obtain values greater then 127 per byte.

For instance, if the scope is to encode the word “Man” into Base64 encoding then it is encoded as “TWFu”. Encoded in ASCII (in ISO 8859-1, one value per byte), M, a, n are stored as the bytes 77 (0x4D), 97 (0x61), 110 (0x6E), which are 01001101, 01100001, 01101110 in base 2.



As this example illustrates, the encoding converts 3 not encoded bytes (in this case, ASCII characters) into 4 encoded ASCII characters. If the output is not multiple of 4 bytes then the '=' sign is put in order to supply the padding. Must be considered the padding with = (64 value) in case of multiple 4 bytes number.

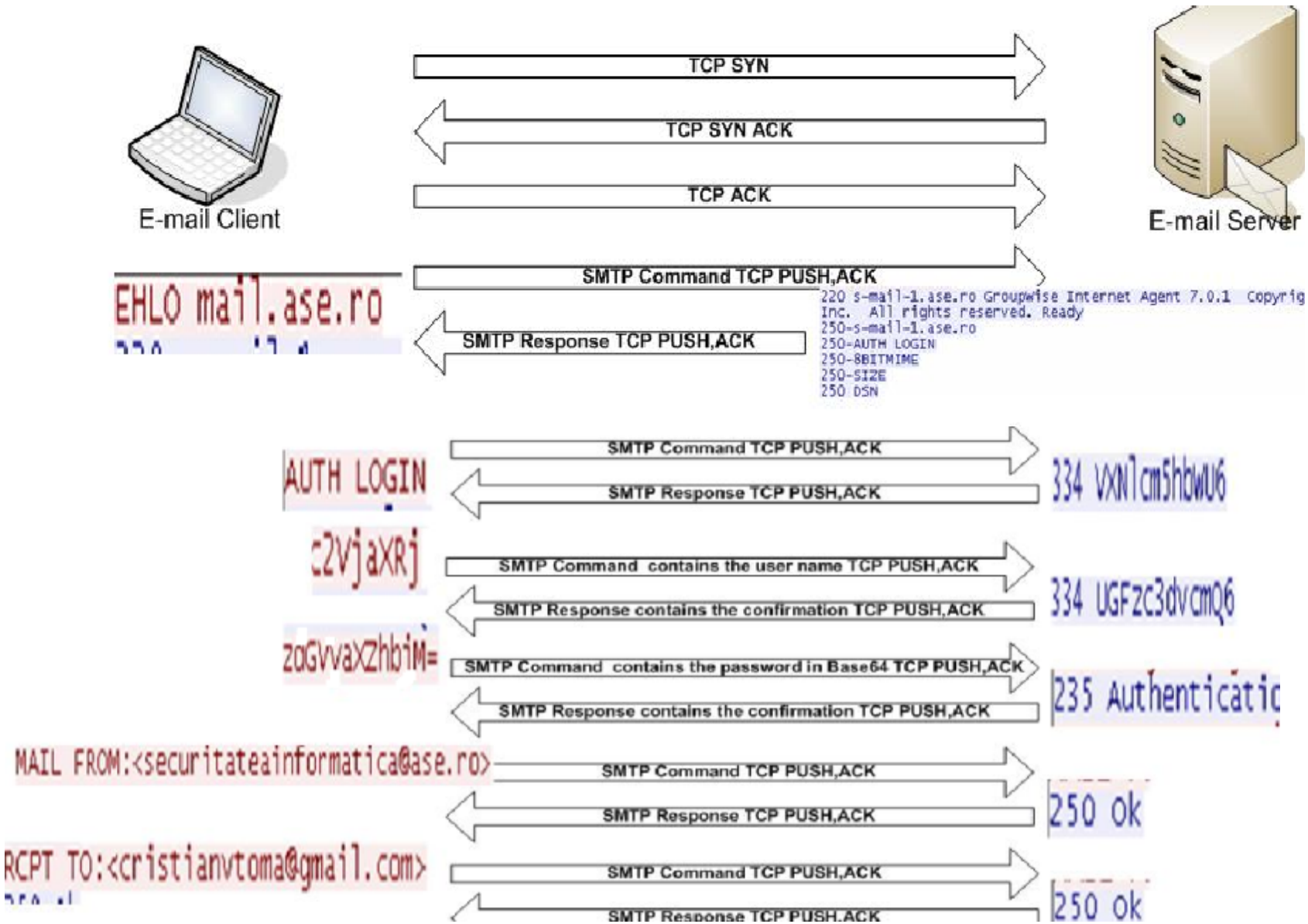
Base 64 Code

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

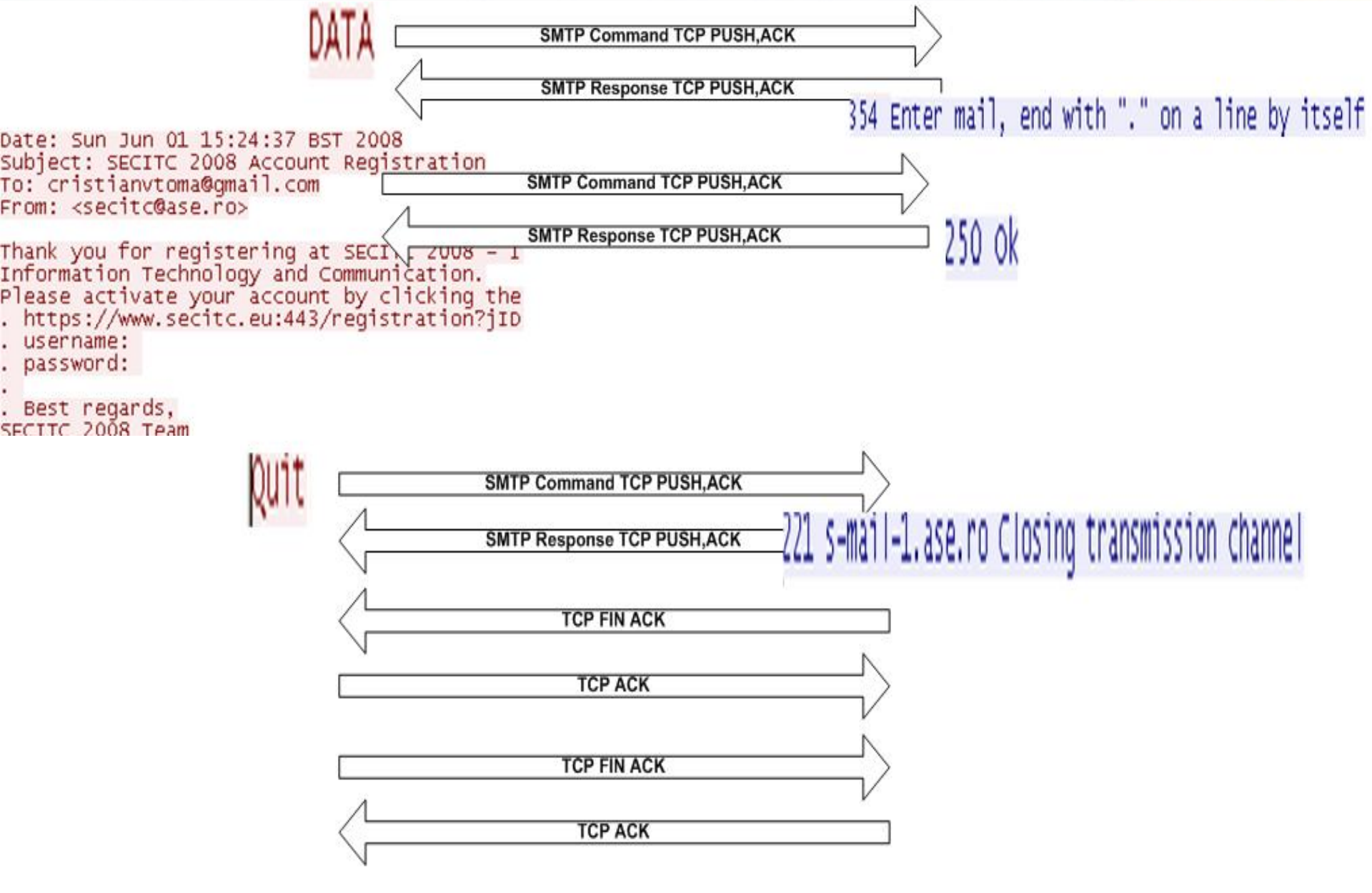
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51

0	1	2	3	4	5	6	7	8	9	+	/	=
52	53	54	55	56	57	58	59	60	61	62	63	64 (64 is used as padding)

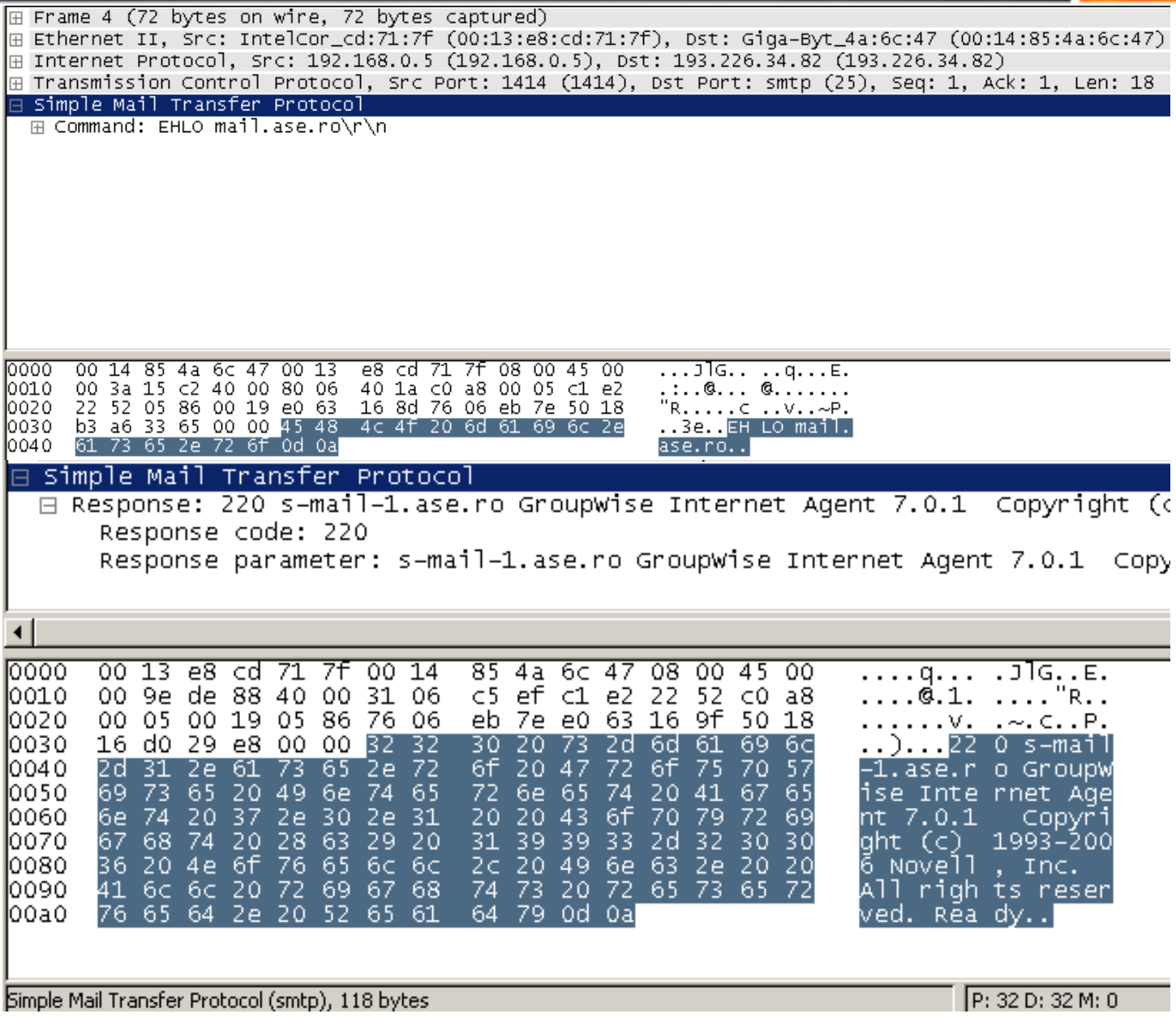
2.4 SMTP – RFC 2821



2.4 SMTP – RFC 2821



2.4 SMTP – RFC 2821



Java Network Programming
for easy sharing

Section Conclusions

Java Network Programming uses for UDP:

DatagramSocket and DatagramPacket classes on both server and client side.

Java Network Programming uses for TCP:

ServerSocket and Socket classes on server side.

Only Socket class on client side.

For both server and client, it is necessary to create byte/char Input (socket.getInputStream()) and Output (socket.getOutputStream()) streams between the Random Access Memory – RAM and the network communications channel.

As practical approach, also developed in the laboratory activities, there is a need to implement "raw" SNMP request, but third party libraries – *snmp4j.org* or product – *Naggios, Cacti, Zabbix* may be used.

As practical approach, there is a need to implement "raw" SMTP in order to send e-mails, but Sun/Oracle or third party libraries may be used – java.mail.*.

Network Programming & Java Sockets

Communicate & Exchange Ideas

Questions & Answers!

But wait…

There's More!

Java OOP + Build Automation: ANT/Maven/Gradle, Annotation, Reflection, I/O, JNI

Java RECAP

2.1.1 Java Reflection

- **Java Reflection** is an “introspective technique” that allows a computer program to examine and modify the structure and behavior (specifically the values, meta-data, properties and functions) of an object at runtime. [WIKI]

- **Java Reflection** is an advanced technique and should be used by experienced programmers that have good knowledge of Java and JVM.

- **Java Reflection** is a technique which allows different applications to do various operations that are quit impossible otherwise. It is a common approach for high-level programming languages as Java or C#.

2.1.1 Java Reflection

Samples for objects and objects arrays in **Java Reflection**:

- Operator ***instanceof***
- Displaying class methods
- Obtaining info about constructors methods
- Obtaining info about class fields
- Invoking methods by name
- Creation of new objects
- Changing the value from various field
- Using the arrays/vectors in Java Reflection context

2.1.2 Java Annotations

- Java Annotation “is the meta-tags that you will use in your code to give it some life.”

- There are two types: “***annotation type***” and “***annotation***”

- Define annotation – ***“annotation type”*** :

```
public @interface MyAnnotation {
   String doSomething();
}
```

- Use annotation – “***annotation***”:

```
@MyAnnotation (doSomething="What to do")
public void mymethod() { .... }
```

2.1.2 Java Annotations

Three kind of ***"annotation type"*** :

- 1. Marker – does NOT have internal elements

  Sample:

```
public @interface MyAnnotation { }
```

  Usage:

```
@MyAnnotation
 public void mymethod() { .... }
```

- 2. Single Element – has a single element represented by key=value

  Sample:

```
public @interface MyAnnotation {
    String doSomething();
}
```

  Usage:

```
@MyAnnotation ("What to do")
public void mymethod() { .... }
```

2.1.2 Java Annotations

Kind of ***"annotation type"*** :

- 3. Full-Value / Multi-Value – has multiple internal elements

  Sample:

```
public @interface MyAnnotation {
    String doSomething();
    int count;
    String date();
}
```

  Usage:

```
@MyAnnotation (doSomething="What to do", count=1, date="09-09-
  2005")
public void mymethod() { .... }
```

2.1.2 Java Annotations

Rules for defining – ***"annotation type"*** :

1. The defining of the annotation should start with '@interface' keyword.
2. The declared methods has no parameters.
3. The declared methods has no "throw exception" statements.
4. The data types of the method are:
    * primitive – byte, char, int, float, double, etc.
    * String
    * Class
    * enum
    * arrays of one of the types from above – int[], float[], etc.

In JDK 5.0 there are predefined / simple – ***"annotation"*** :

1. @Override
2. @Deprecated
3. @SupressWarnings

2.1.2 Java Annotations

Starting with JDK 5.0 there are *"meta-annotation"* that can be applied only to the *"annotation type"*:

1. Target

```
@Target(ElementType.TYPE)
@Target(ElementType.FIELD)
@Target(ElementType.METHOD)
@Target(ElementType.PARAMETER)
@Target(ElementType.CONSTRUCTOR)
@Target(ElementType.LOCAL_VARIABLE)
@Target(ElementType.ANNOTATION_TYPE)
```

2. Retention
   - @Retention(RetentionPolicy.SOURCE) – retinute la nivel cod sursa si sunt ignorate de compilator
   - @Retention(RetentionPolicy.CLASS) – retinute la nivel de compilare dar ignorate de VM la run-time
   - @Retention(RetentionPolicy.RUNTIME) – sunt retinute si utilizate doar la run-time

3. Documented – @Documented

4. Inherited – @Inherited

2.1.3 Java Library

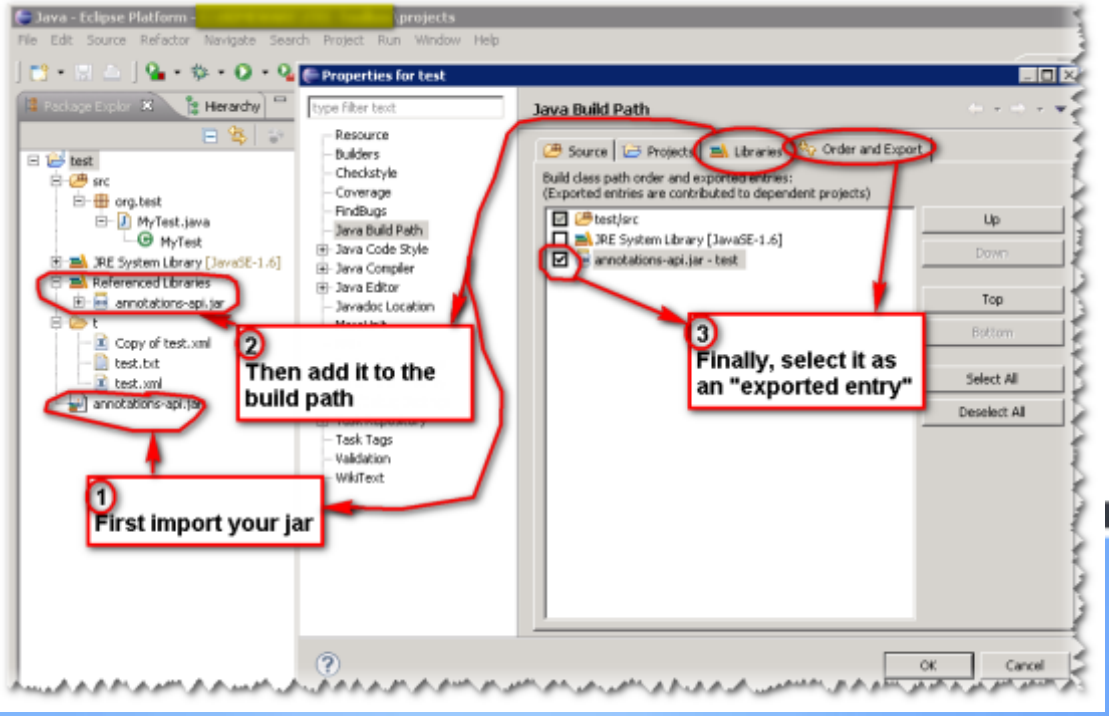
What is a **Java library**?

What are the advantages and disadvantages of Java libraries?

How can be solved in Java multiple dependencies or inclusions of the same class in the compilation phase? How were solved these problems in C/C++?

How should be created a Java library and how should be used – command line vs. IDE?

>jar -cvf archive_name.jar files_names_to_compress
>javac -classpath .:archieve_name.jar *.java
>java –classpath .:archive_name.jar file_with_main_class



2.2.1 Build Automation: Apache ANT

Optional steps to build the lectures with Gradle – all lectures are independent by any Build
Automation system (e.g. Gradle, MVN, ANT, etc.) and the labs are projects in Eclipse IDE:

Apache ANT is a Java based build tool from Apache Software Foundation. Apache Ant's build files are written in XML and they take advantage of being open standard, portable and easy to understand.

Features:

- Ant is the most complete Java build and deployment tool available.
- Ant is platform neutral and can handle platform specific properties such as file separators.
- Ant can be used to perform platform specific tasks such as modifying the modified time of a file using 'touch' command.
- Ant scripts are written using plain XML. If you are already familiar with XML, you can learn Ant pretty quickly.
- Ant is good at automating complicated repetitive tasks.
- Ant comes with a big list of predefined tasks.
- Ant provides an interface to develop custom tasks.
- Ant can be easily invoked from the command line and it can integrate with free and commercial IDEs.

2.2.1 Build Automation: Apache ANT

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="oata.HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

ant compile jar run

2.2.1 Build Automation: Apache ANT

While having a look at the buildfile, we will see some similar steps between Ant and the java-only commands:

java-only	Ant
`md build\classes`	`<mkdir dir="build/classes"/>`
`javac`	`<javac`
`  -sourcepath src`	`  srcdir="src"`
`  -d build\classes`	`  destdir="build/classes"/>`
`  src\oata\HelloWorld.java`	`<!-- automatically detected -->`
`echo Main-Class: oata.HelloWorld>mf`	`<!-- obsolete; done via manifest tag -->`
`md build\jar`	`<mkdir dir="build/jar"/>`
`jar cfm`	`<jar`
`  build\jar\HelloWorld.jar`	`  destfile="build/jar/HelloWorld.jar"`
`  mf`	
`  -C build\classes`	`  basedir="build/classes">`
`  .`	`  <manifest>`
	`  <attribute name="Main-Class" value="oata.HelloWorld"/>`
	`  </manifest>`
	`</jar>`
`java -jar build\jar\HelloWorld.jar`	`<java jar="build/jar/HelloWorld.jar" fork="true"/>`

ant compile

ant jar

ant run

2.2.2 Build Automation: Apache MVN - Maven

Optional steps to build the lectures with Gradle – all lectures are independent by any Build Automation system (e.g. Gradle, MVN, ANT, etc.) and the labs are projects in Eclipse IDE:

Apache Maven (MVN) is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Using maven we can build and manage any Java based project.

Features:

- Maven uses Convention over Configuration, which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.
- Simple project setup that follows best practices.
- Consistent usage across all projects.
- Dependency management including automatic updating.

2.2.2 Build Automation: Apache MVN - Maven

Apache Maven (MVN) Features:

- A large and growing repository of libraries.
- Extensible, with the ability to easily write plugins in Java or scripting languages.
- Instant access to new features with little or no extra configuration.
- **Model-based builds** – Maven is able to build any number of projects into predefined output types such as jar, war, metadata.
- **Coherent site of project information** – Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.
- **Release management and distribution publication** – Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
- **Backward Compatibility** – You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
- **Automatic parent versioning** – No need to specify the parent in the sub module for maintenance.
- **Parallel builds** – It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.
- **Better Error and Integrity Reporting** – Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

2.2.2 Build Automation: Apache MVN - Maven

Creating a Project

You will need somewhere for your project to reside, create a directory somewhere and start a shell in that directory. On your command line, execute the following Maven goal:

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

If you have just installed Maven, it may take a while on the first run. This is because Maven is downloading the most recent artifacts (plugin jars and other files) into your local repository. You ma
execute the command a couple of times before it succeeds. This is because the remote server may time out before your downloads are complete. Don't worry, there are ways to fix that.

You will notice that the *generate* goal created a directory with the same name given as the artifactId. Change into that directory.

```
cd my-app
```

Under this directory you will notice the following standard project structure.

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

The `src/main/java` directory contains the project source code, the `src/test/java` directory contains the test source, and the `pom.xml` file is the project's Project Object Model, or POM.

2.2.2 Build Automation: Apache MVN - Maven

The POM

The pom.xml file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want. The POM is huge and can be daunting in its complexity, but it is not necessary to understand all of the intricacies just yet to use it effectively. This project's POM is:

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3.    <modelVersion>4.0.0</modelVersion>
4.
5.    <groupId>com.mycompany.app</groupId>
6.    <artifactId>my-app</artifactId>
7.    <version>1.0-SNAPSHOT</version>
8.    <packaging>jar</packaging>
9.
10.   <name>Maven Quick Start Archetype</name>
11.   <url>http://maven.apache.org</url>
12.
13.   <dependencies>
14.     <dependency>
15.       <groupId>junit</groupId>
16.       <artifactId>junit</artifactId>
17.       <version>4.8.2</version>
18.       <scope>test</scope>
19.     </dependency>
20.   </dependencies>
21. </project>
```

```
mvn clean build
mvn package
java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App
```

2.3 Build Automation: Gradle

Optional steps to build the lectures with Gradle – all lectures are independent by any Build Automation system (e.g. Gradle, MVN, ANT, etc.) and the labs are projects in Eclipse IDE:

Gradle is an advanced general purpose **build management system** based on Groovy.

Features:

- It supports automatic download and configuration of project dependencies (libraries).
- It supports Maven and Ivy (ANT) repositories for retrieving project dependencies, allowing reuse of the artifacts of existing build systems.
- It supports multi-project and multi-artifact builds.

(Optional full Gradle presentation @ISM – Cyber-Security Master – www.ism.ase.ro, by Marius Popa @ Oracle)

https://gradle.org/install/
https://plugins.gradle.org/
https://github.com/jabedhasan21/java-hello-world-with-gradle/blob/master/README.md
https://www.tutorialspoint.com/gradle/index.htm

2.3.1. Gradle Intro and Basics

Projects and tasks in **Gradle**:

- A Gradle build consists of one or more projects.
- A ***project*** using Gradle describes its build via a ***build.gradle*** file. ***build.gradle*** file is located in the root folder of the project. ***build.gradle*** file is based on a *Domain Specific Language* (DSL) written in Groovy.
- ***build.gradle*** build file defines a project and its tasks.
- A ***task*** represents a piece of work which a build performs.

```
task hello {
  doLast {
    println 'Hello Gradle'
    println 'Project name: ' + rootProject.name
  }
}
```

```
mepopa@MEPOPA-RO /d/Work/P006_IoT_GW/GradleTraining
$ gradle project
:projects

------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'GradleTraining'
No sub-projects

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :tasks

BUILD SUCCESSFUL

Total time: 3.676 secs
```

```
mepopa@MEPOPA-RO /d/Work/P006_IoT_GW/GradleTraining
$ gradle hello
:hello
Hello Gradle
Project name: GradleTraining

BUILD SUCCESSFUL

Total time: 3.162 secs
```

2.3.2. Installing and configuring Gradle

- ***Requirement***: JDK (Java Development Kit) installation.
- ***Installing***: Download the latest release of Gradle from http://gradle.org/gradle-download/ | https://gradle.org/install/ for the usage on the command line. Add the /bin folder to PATH environment variable.
- ***Gradle daemon***: avoid starting the Java virtual machine for every build. Usage: org.gradle.daemon=true in gradle.properties in the ${HOME}/.gradle; Executing gradle with the --daemon parameter on the command line or gradle --stop to stop it.
- Set specific JVM options: GRADLE_OPTS environment variable. Eg. export GRADLE_OPTS=-Xmx1024m defines 1 GB as maximum heap size.

2.3.3. Gradle plug-ins

- Plug-ins extend Gradle core functionality. Extension to Gradle adding some preconfigured tasks.
- Gradle ships with a number of plug-ins (java), and custom plug-ins can be developed.
- Adding a plug-in in ***build.gradle*** file: apply plugin: 'pluginname' (Eg. apply plugin: 'java').
- Registry for Gradle plug-ins: https://plugins.gradle.org/.

2.3.3. Gradle plug-ins

build.gradle

```
task hello {
  doLast {
    println 'Hello Gradle'
    println 'Project name: ' + rootProject.name
  }
}
```

```
mepopa@MEPOPA-RO /d/Work/P006_IoT_GW/GradleTraining
$ gradle tasks
:tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
----------
components - Displays the components produced by root project 'GradleTraining'. [incubating]
dependencies - Displays all dependencies declared in root project 'GradleTraining'.
dependencyInsight - Displays the insight into a specific dependency in root project 'GradleTraining'.
help - Displays a help message.
model - Displays the configuration model of root project 'GradleTraining'. [incubating]
projects - Displays the sub-projects of root project 'GradleTraining'.
properties - Displays the properties of root project 'GradleTraining'.
tasks - Displays the tasks runnable from root project 'GradleTraining'.

Other tasks
-----------
hello

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL

Total time: 3.418 secs
```

build.gradle

```
apply plugin: 'java'
task hello {
  doLast {
    println 'Hello Gradle'
    println 'Project name: ' + rootProject.name
  }
}
```

```
mepopa@MEPOPA-RO /d/Work/P006_IoT_GW/GradleTraining
$ gradle tasks
:tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles classes 'main'.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles classes 'test'.

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Documentation tasks
-------------------
javadoc - Generates Javadoc API documentation for the main source code.
```

```
Help tasks
----------
components - Displays the components produced by root project 'GradleTraining'. [incubating]
dependencies - Displays all dependencies declared in root project 'GradleTraining'.
dependencyInsight - Displays the insight into a specific dependency in root project 'GradleTraining'.
help - Displays a help message.
model - Displays the configuration model of root project 'GradleTraining'. [incubating]
projects - Displays the sub-projects of root project 'GradleTraining'.
properties - Displays the properties of root project 'GradleTraining'.
tasks - Displays the tasks runnable from root project 'GradleTraining'.

Verification tasks
------------------
check - Runs all checks.
clean - Deletes the build directory.
test - Runs the unit tests.

Other tasks
-----------
hello

Rules
-----
Pattern: clean<TaskName>: Cleans the output files of a task.
Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.
Pattern: upload<ConfigurationName>: Assembles and uploads the artifacts belonging to a configuration.

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL

Total time: 5.03 secs
```

2.3.4. Gradle dependency management

- Managing the classpath of Gradle projects: adding JAR files, directories or other projects to the build path of the application.
- Automatic download of Java library dependencies, specifying the dependency in Gradle build file.
- A Java library is identified by Gradle via its project's groupId:artifactId:version (also known as GAV in Maven).
- Adding a dependency: new entry in the dependency section in ***build.gradle*** file.

```
repositories {
    jcenter()
}

dependencies {
    compile fileTree(dir: 'libs',
include: ['*.jar'])
    testCompile 'junit:junit:4.12'
}
```

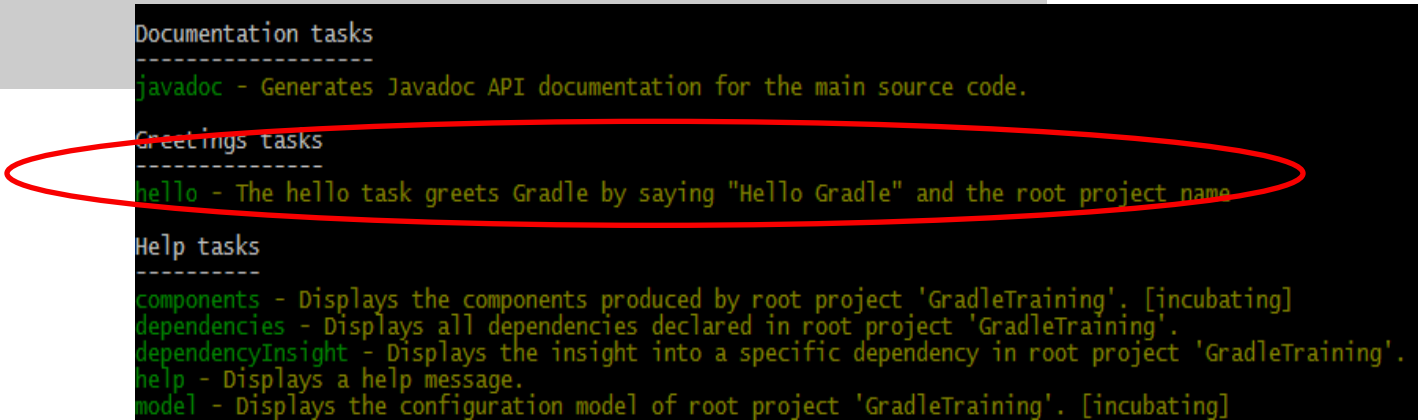
```
mepopa@MEPOPA-RO /d/Work/P006_IoT_GW/GradleTraining
$ gradle clean build
:clean UP-TO-DATE
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar
:assemble
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build

BUILD SUCCESSFUL

Total time: 4.357 secs
```

2.3.5. Gradle tasks

- Default Gradle tasks: tasks for introspection of Gradle itself (eg. task tasks).

- Custom Gradle tasks: tasks created by developers (eg. task hello). Running the gradle tasks task, the hello task will be listed under ***Other tasks***. Tasks without a group are considered as private tasks. Groups can be applied with the group property and a description can be applied by using the description property .

```
task hello {

    group 'greetings'
    description 'The hello task greets Gradle by
saying "Hello Gradle" and the root project name'

    doLast {
        println 'Hello Gradle'
        println 'Project name: ' + rootProject.name
    }
}
```



2.3.6. Building Java projects

- Java plug-in: provides tasks to compile Java source code, run unit tests, create Javadoc and create a JAR file.
- Default project layout:

  - Java source code: src/main/java.

  - Java tests: src/test/java.

- Different project structure: sourceSets.

```
sourceSets {
  main {
     java {
        srcDir 'src'
        }
     }
  test {
     java {
        srcDir 'test'
        }
     }
}
```

- Start the build: gradle build for HelloWorld.java.

```
mepopa@MEPOPA-RS /d/Work/P006_IoT_GW/GradleTraining
$ gradle build
I am not invoked, but I always get printed
:compileJava
:processResources UP-TO-DATE
:classes
:jar
:assemble
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test UP-TO-DATE
:check UP-TO-DATE
:build

BUILD SUCCESSFUL

Total time: 5.434 secs
```

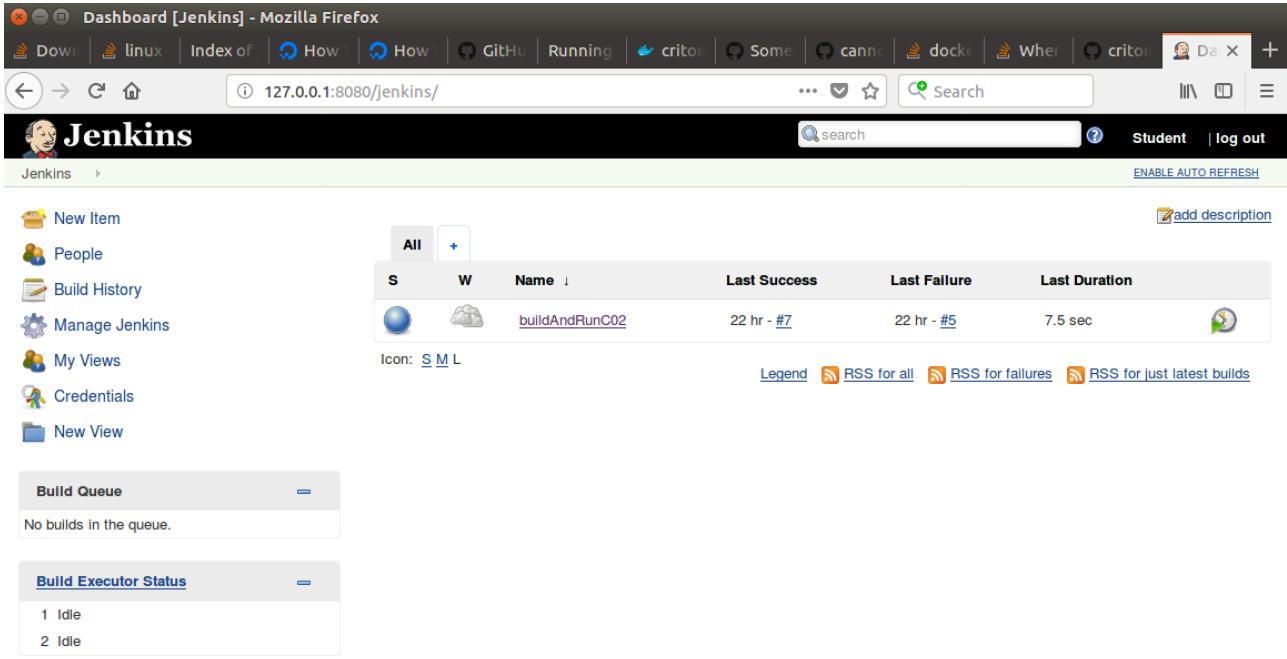
2.4 Jenkins/Hudson – DevOps Build Automation

```
export JAVA_HOME=/opt/software/java/jdks/jdk1.8.0_161
export CLASSPATH=.:$JAVA_HOME/jre/lib
export CATALINA_HOME=/opt/software/apache-tomcat-9.0.4
export
PATH=.:$JAVA_HOME/bin:$CATALINA_HOME/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/usr/games
```

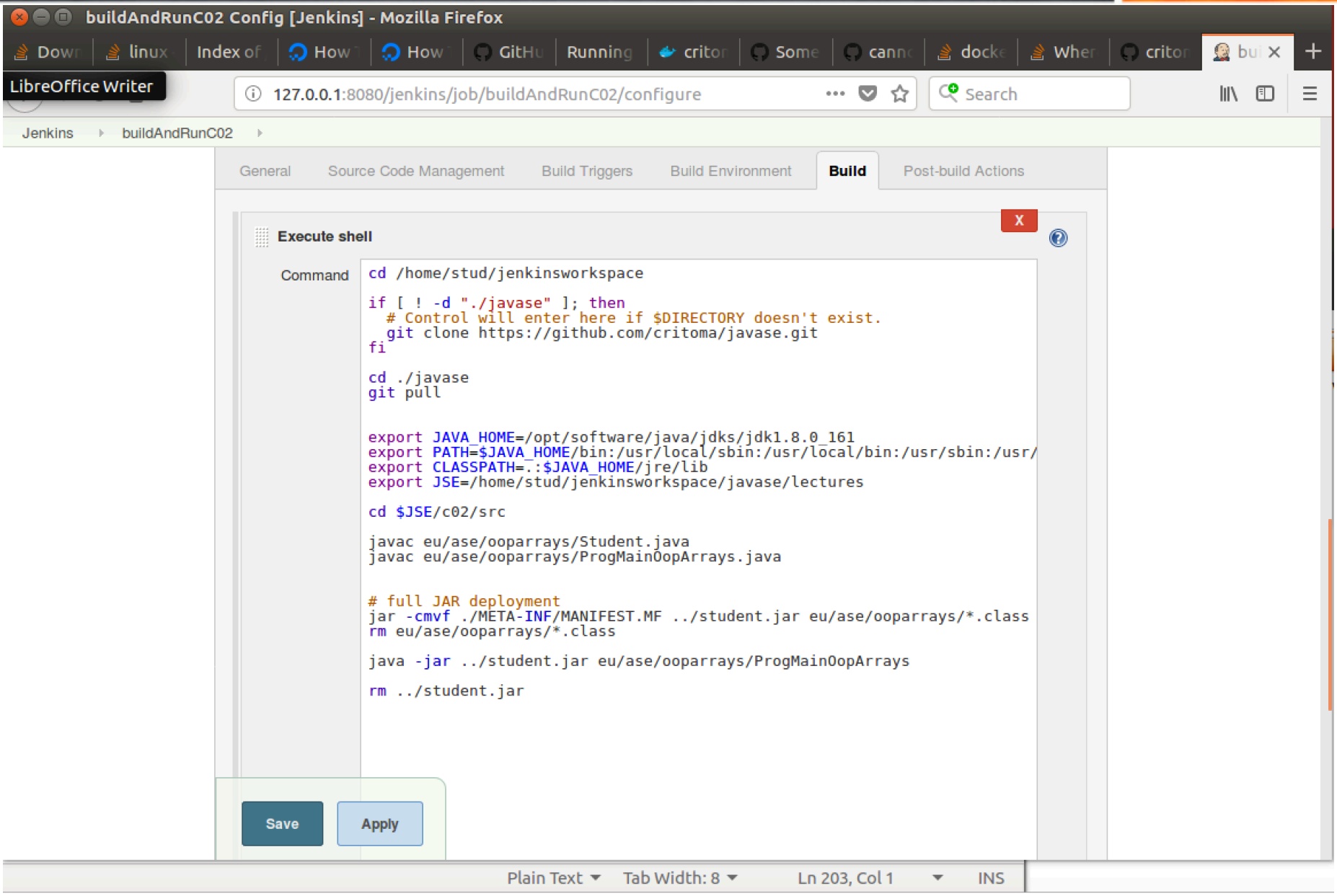
```
cd $CATALINA_HOME
bin/startup.sh
```

http://127.0.0.1:8080
http://127.0.0.1:8080/jenkins/

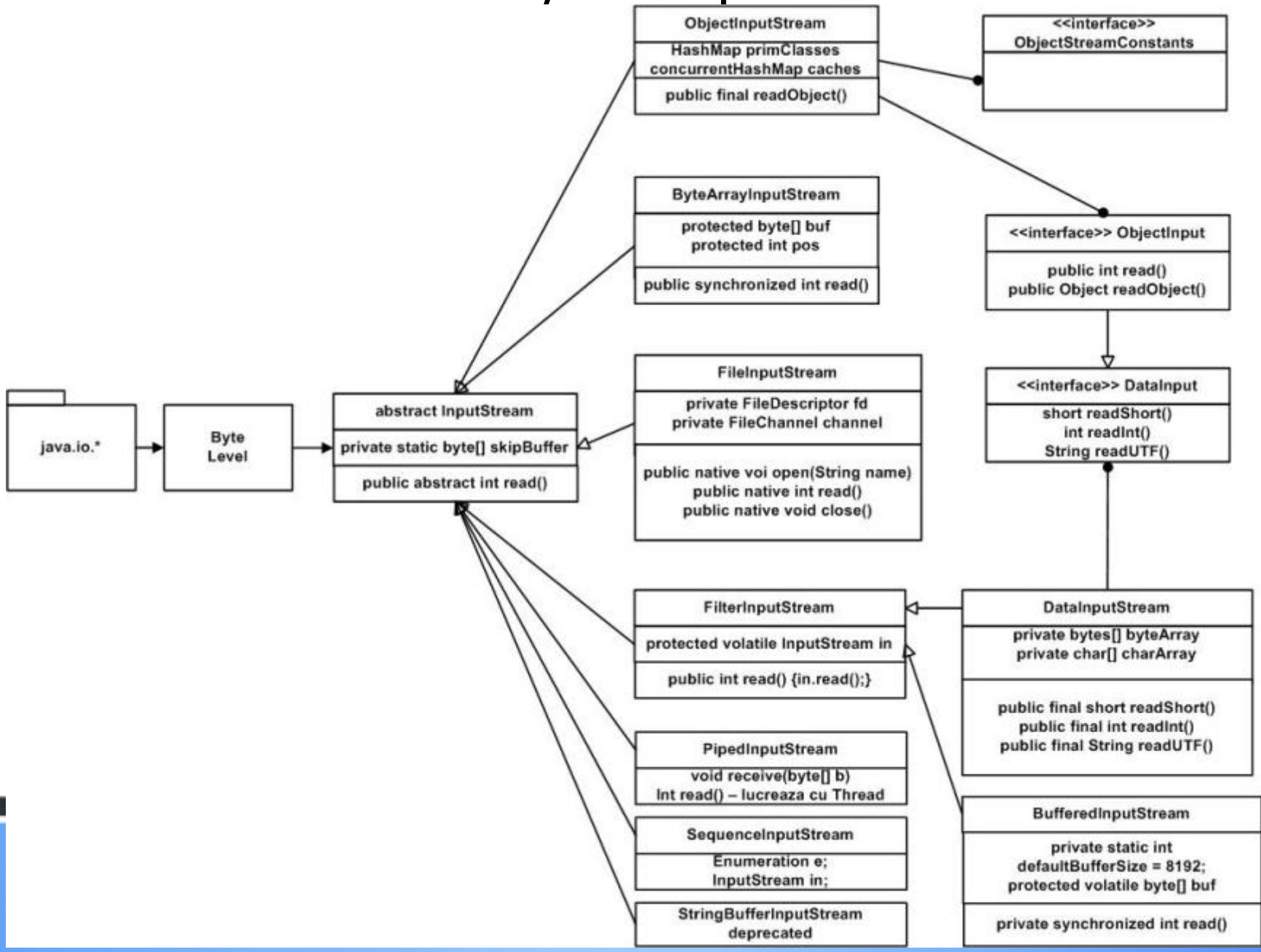
```
bin/shutdown.sh
```



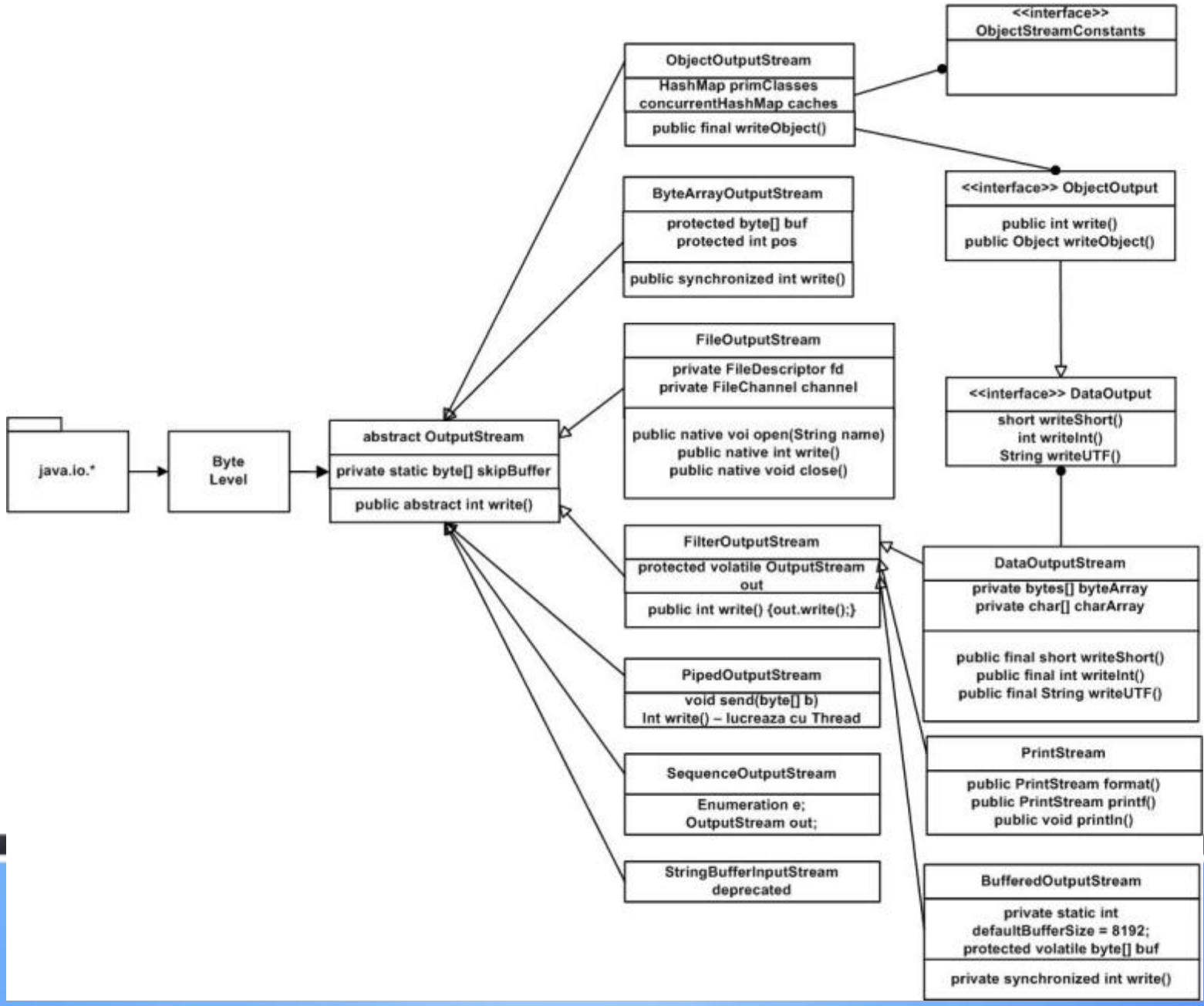
2.4 Jenkins/Hudson – DevOps Build Automation



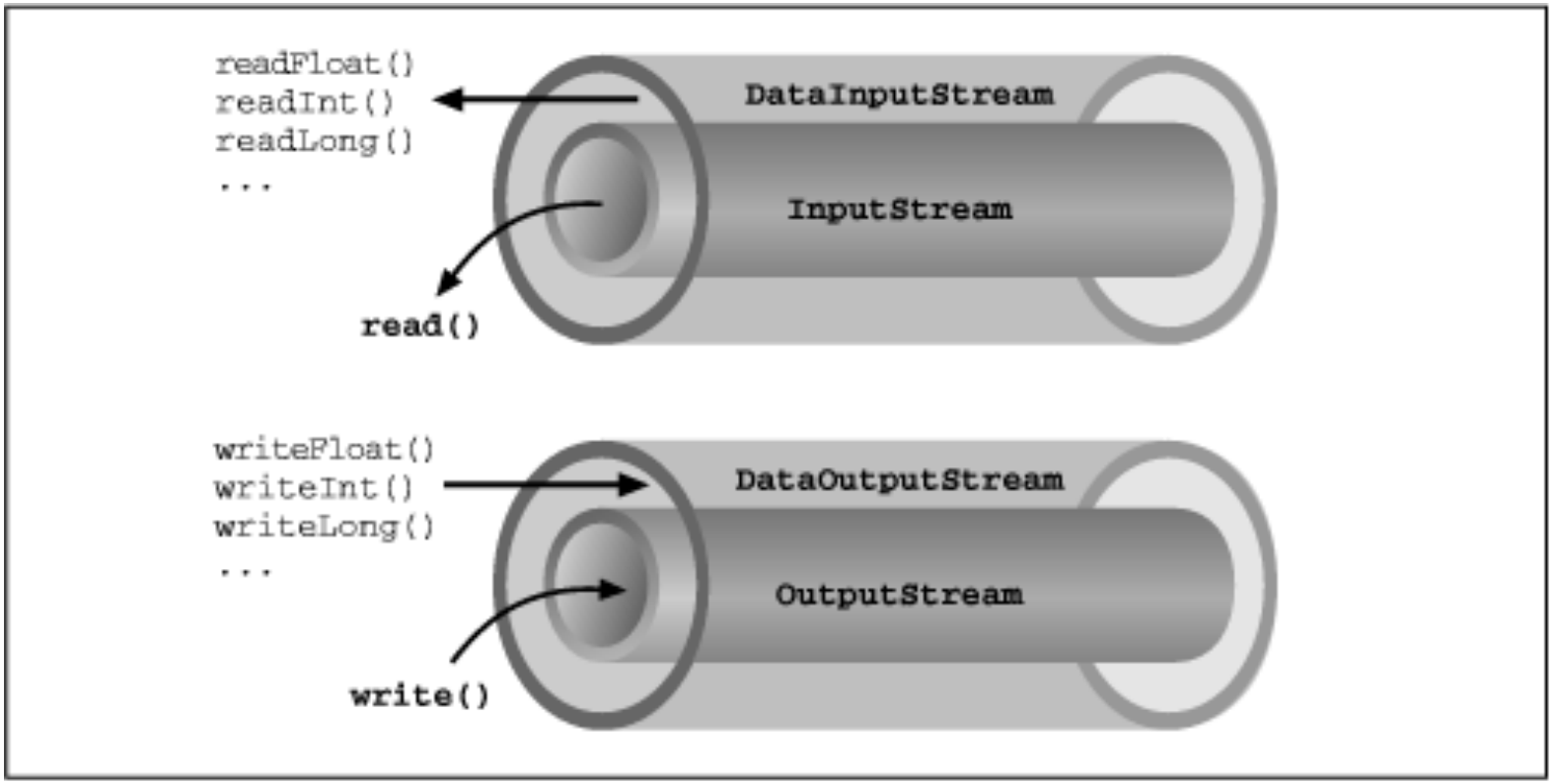
2.5 Java I/O – Input Stream



2.5 Java I/O – Output Stream

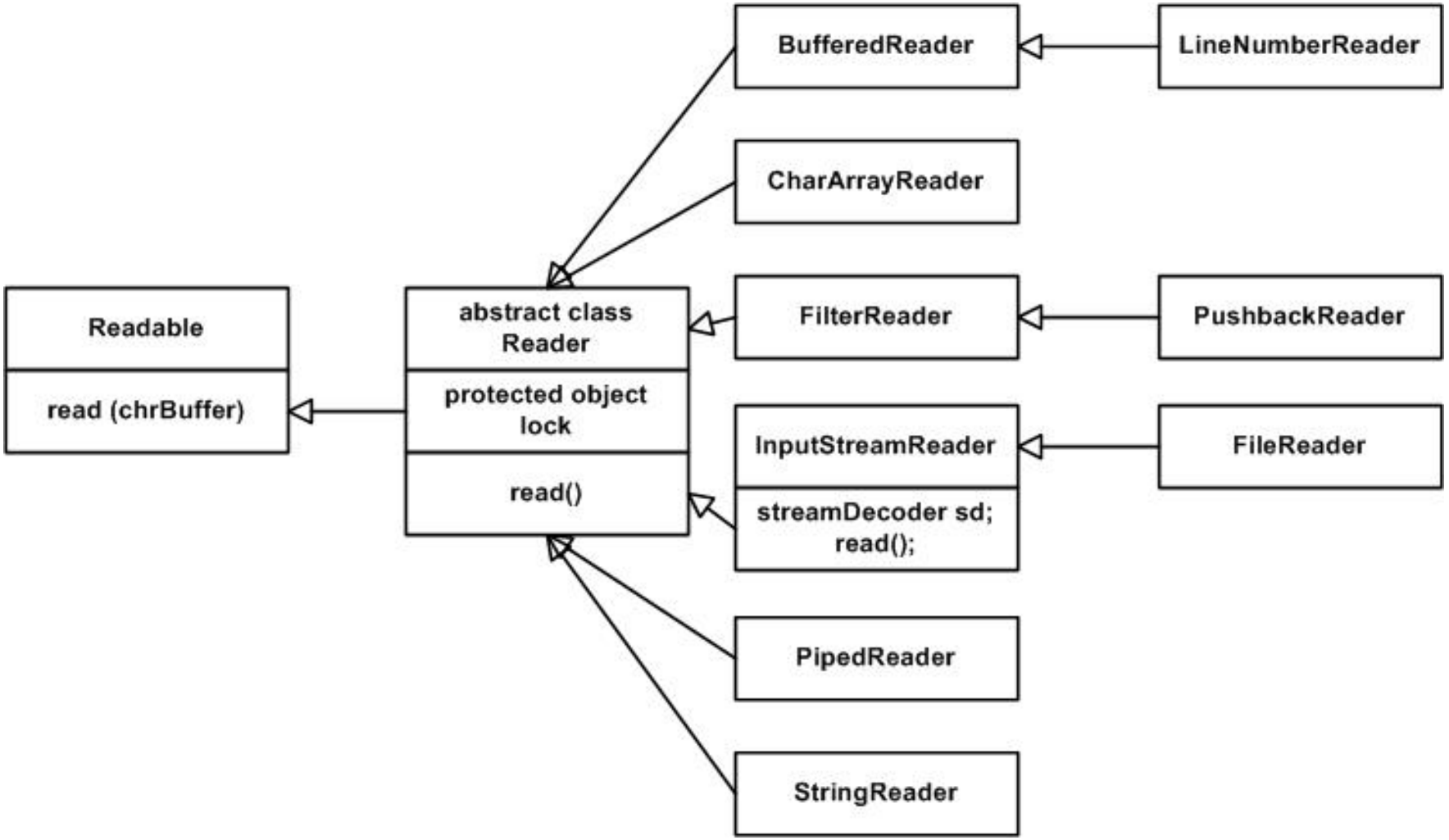


2.5 Java I/O – Streams Encapsulation

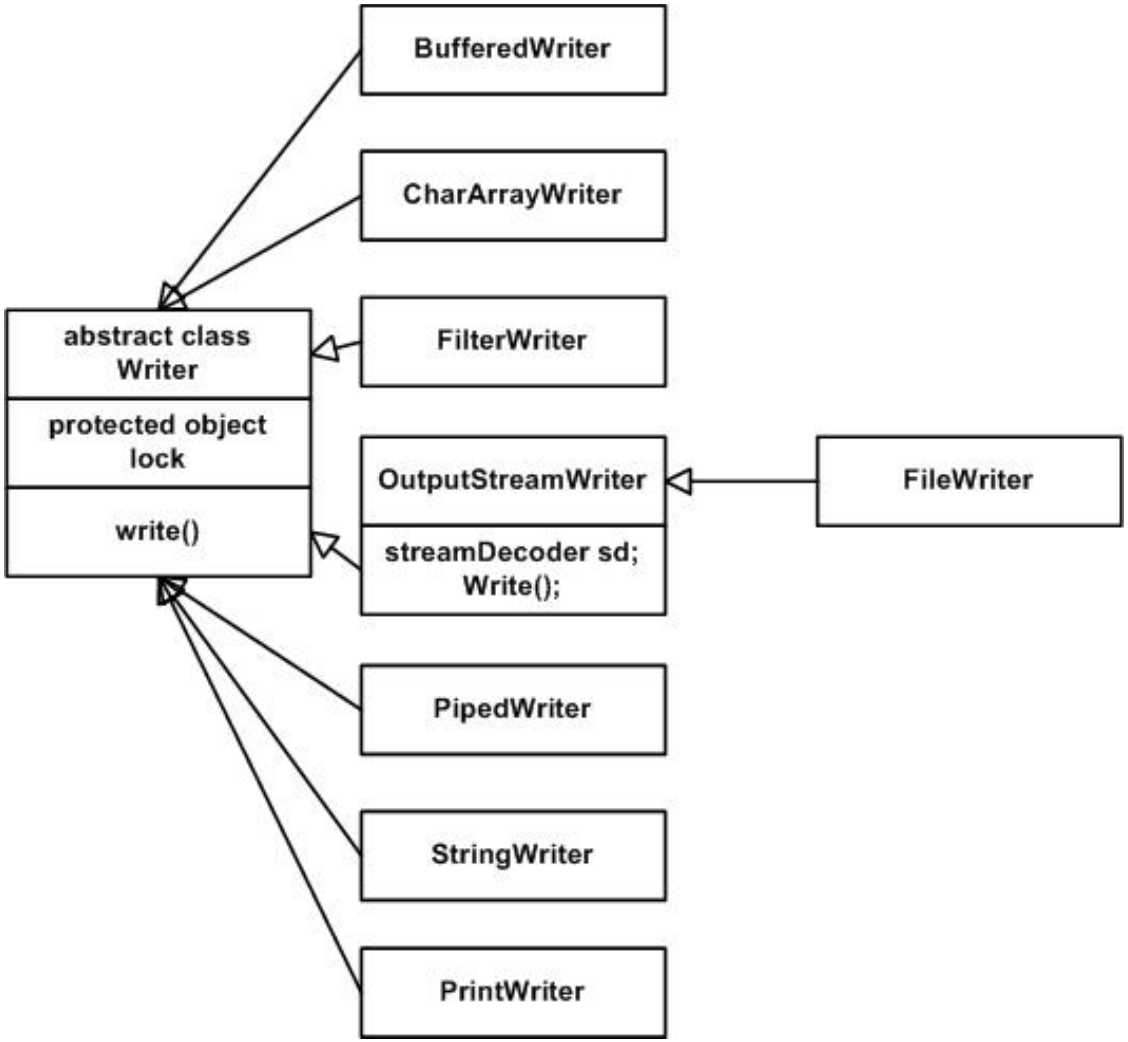


http://doc.sumy.ua/prog/java/exp/ch10_01.htm

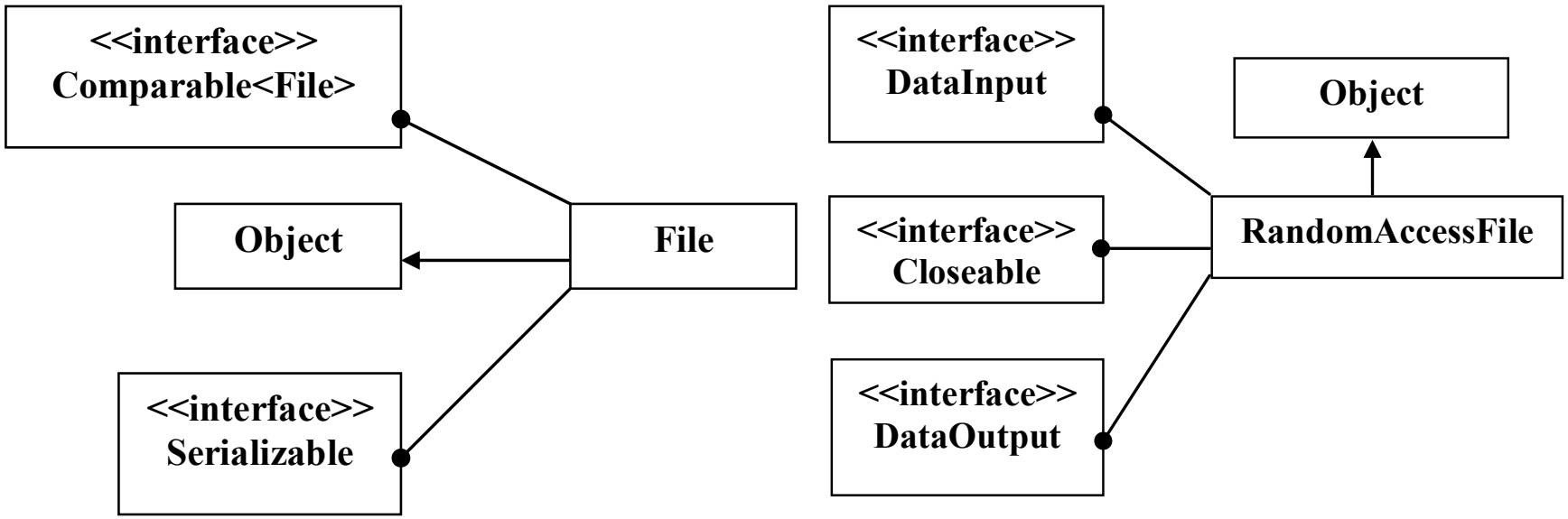
2.5 Java I/O – char level reading



2.5 Java I/O – char level writing

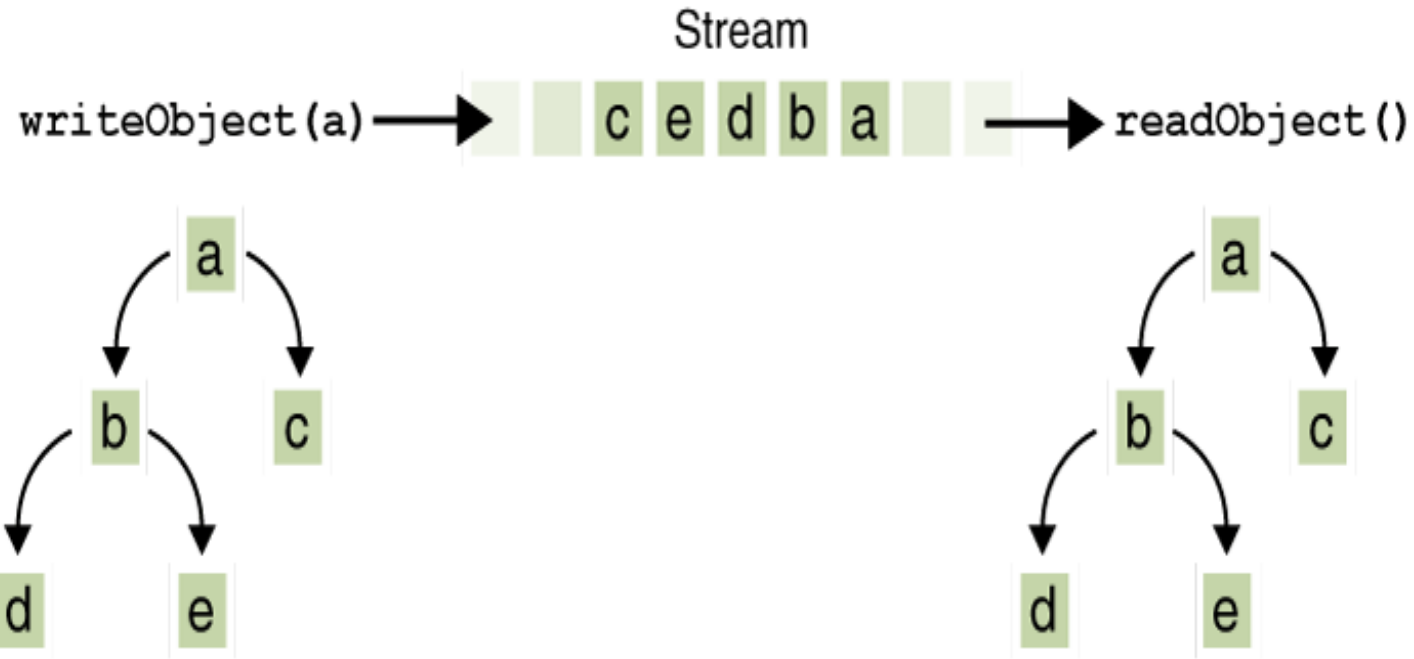


2.5 Java I/O – File Access



2.5 Java I/O – Serialization

This is demonstrated in the following figure, where `writeObject` is invoked to write a single object named **a**. This object contains references to objects **b** and **c**, while **b** contains references to **d** and **e**. Invoking `writeobject(a)` writes not just **a**, but all the objects necessary to reconstitute **a**, so the other four objects in this web are written also. When **a** is read back by `readObject`, the other four objects are read back as well, and all the original object references are preserved.



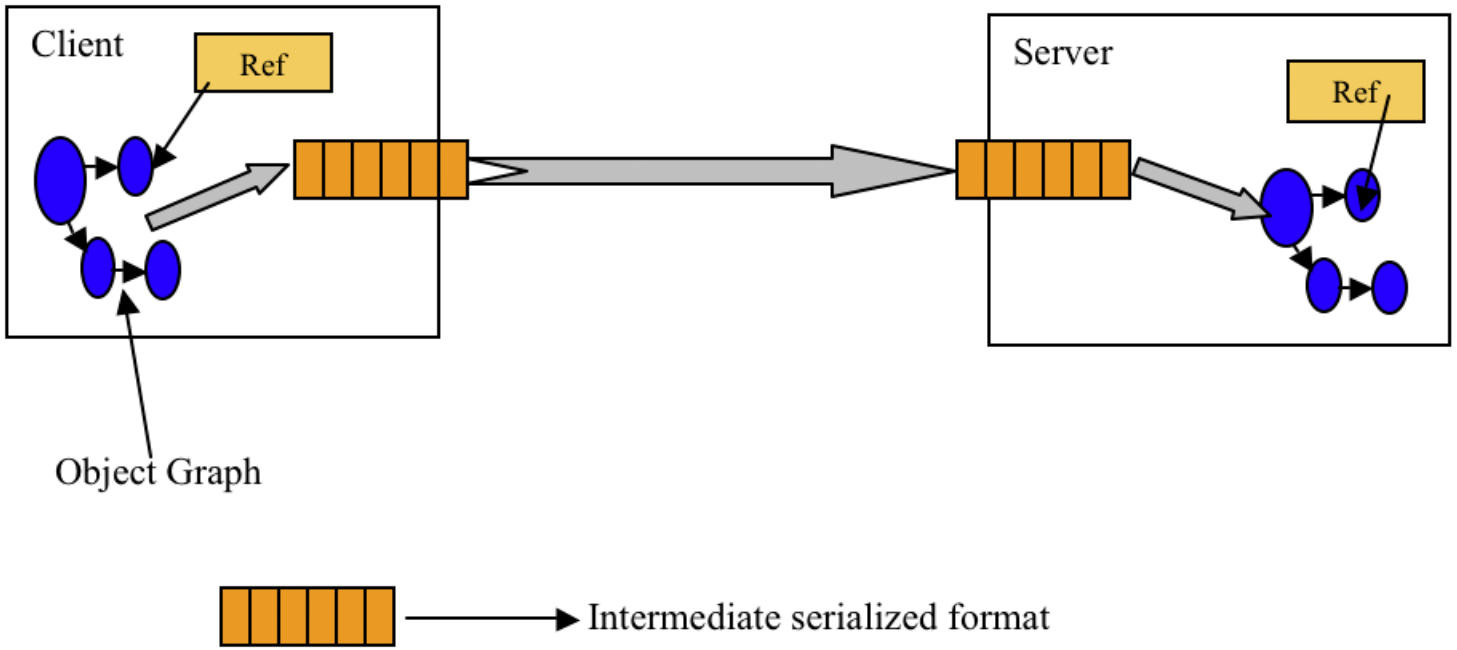
I/O of multiple referred-to objects

2.5 Java I/O – Serialization

What is going to be saved and restored by serialization in Java?

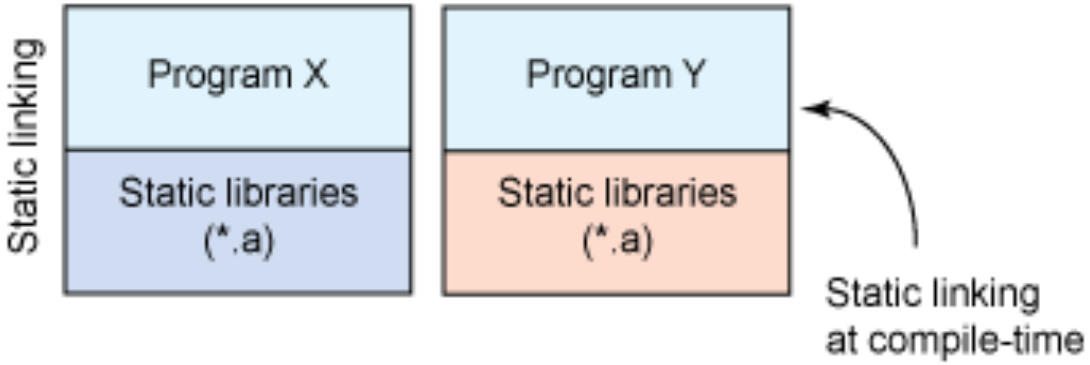
- **Non-static fields? Static fields?**
- **Transient fields?**
- **Private and public fields and/or methods?**
- **Prototype / signature of the methods and / or the implementation of the methods?**

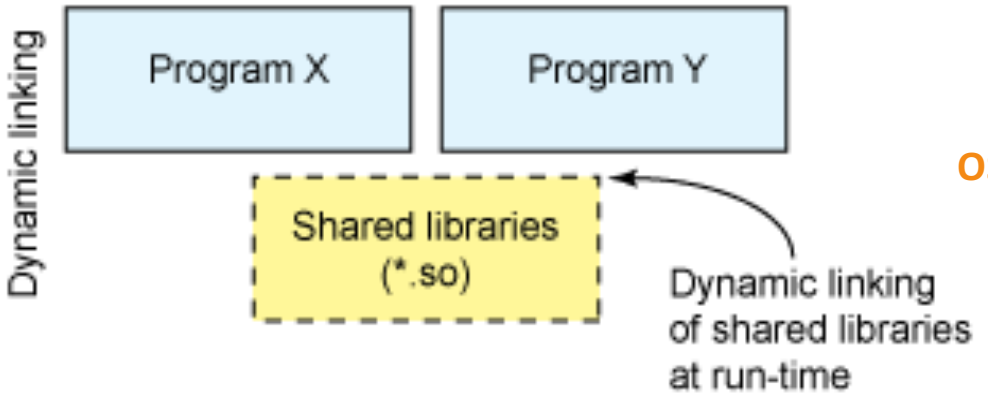
http://www.javaworld.com/community/node/2915



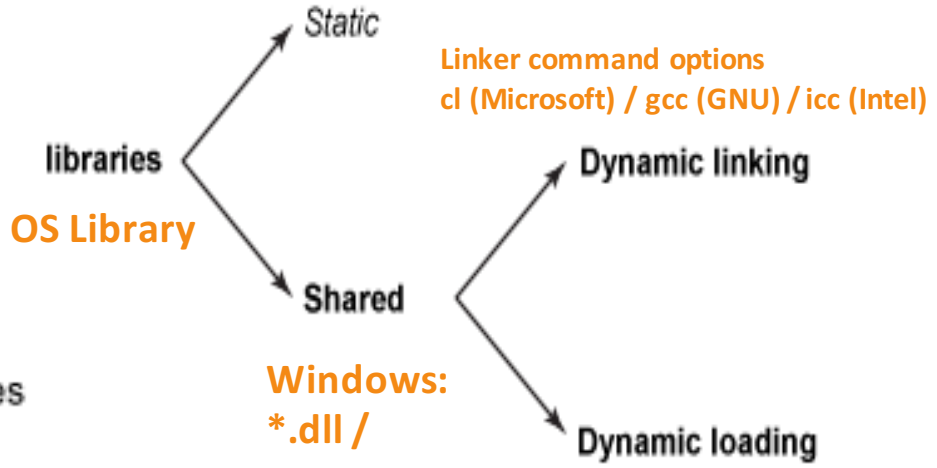
2.6 JNI – Linux vs. Win libraries

http://www.ibm.com/developerworks/linux/library/l-dynamic-libraries/





Windows: *.lib / Linux: *.a



Linker command options
cl (Microsoft) / gcc (GNU) / icc (Intel)

OS Library

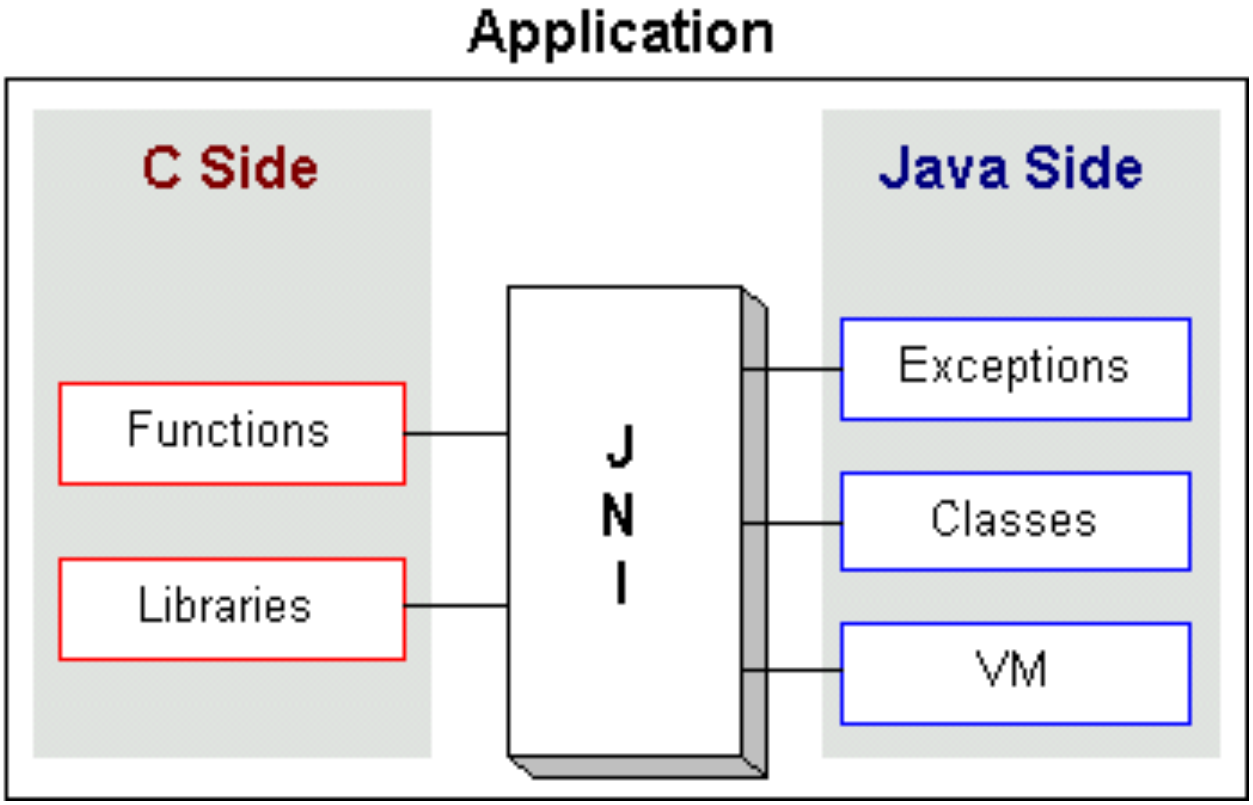
Windows: *.dll /

Linux: *.so

@ Run-time

```
/* Open the shared object */
dl_handle = dlopen( lib, RTLD_LAZY );
if (!dl_handle) {
  printf( "!!! %s\n", dlerror() );
  return;
}
```

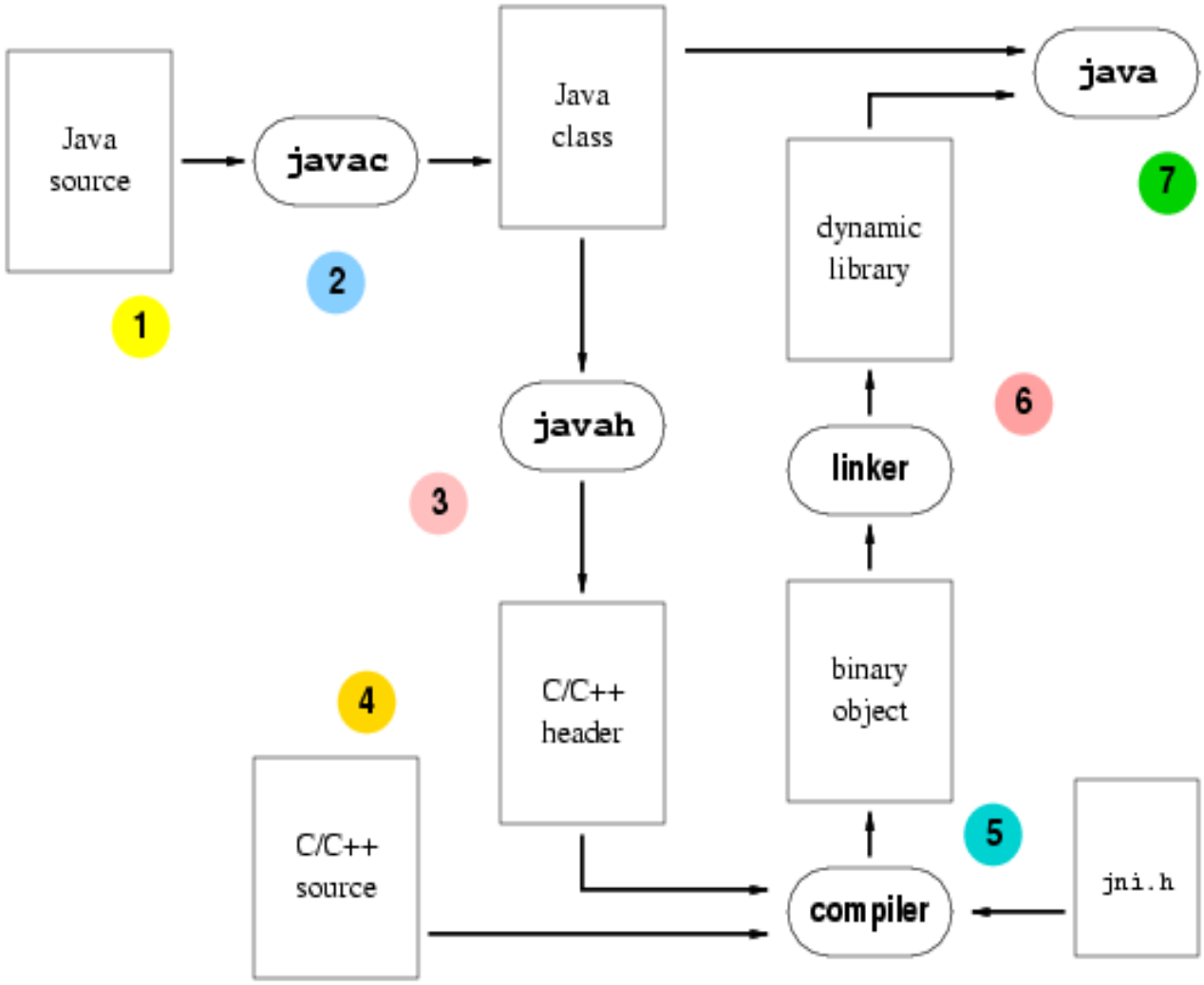
2.6 Java Native Interface



http://www.iam.ubc.ca/guides/javatut99/native1.1/concepts/index.html

2.6 Java Native Interface

http://cs.fit.edu/~ryan/java/language/jni.html



1. Create Java source code with native methods

native *return type* method (*arguments*);

2. Compile Java source code and obtain the class files
3. Generate C/C++ headers for the native methods; javah gets the info it needs from the class files
4. Write the C/C++ source code for the native method using the function prototype from the generated include file and the typedefs from include/jni.h
5. Compile the C/C++ with the right header files
6. Use the linker to create a dynamic library file
7. Execute a Java program that loads the dynamic library

static {
System.loadLibrary("*dynamic libary*"); }

Thanks!

DAD – Distributed Application Development
End of Lecture 2 – Summary of Java SE & Network Programming – section 2