

# **Diseño de Protocolos**

# Contexto

- ◆ Un protocolo se diseña para uso en un entorno distribuido
  - ▶ Muchos usuarios (ordenadores, routers, ..) deben utilizar el mismo algoritmo y el mismo lenguaje de comunicación (mensajes)
- ◆ Problema:
  - ▶ Conseguir que todos los usuarios hablen el mismo lenguaje (protocolo)
- ◆ Protocolo: norma que los usuarios deben seguir para entenderse
  - ▶ Debe ser interpretada multiples veces

# Protocolo

- ◆ Protocolo: Algoritmo distribuido que implementa una función de comunicación y da un servicio
  - ▶ Lenguaje de comunicación:
    - Mensajes: tramas con formatos predefinidos
    - Comportamiento: Reglas de intercambio de mensajes
- ◆ Técnicas de diseño de protocolos (algoritmos distribuidos)
  - ▶ Desde la concepción inicial
  - ▶ Hasta la implementación final
    - Incluyendo todo el ciclo de vida

# Ciclo de vida

Norma (ver. inicial):  
textual o formal

- Verif. y Valid.
- Analisis de prest.
- Prototipado



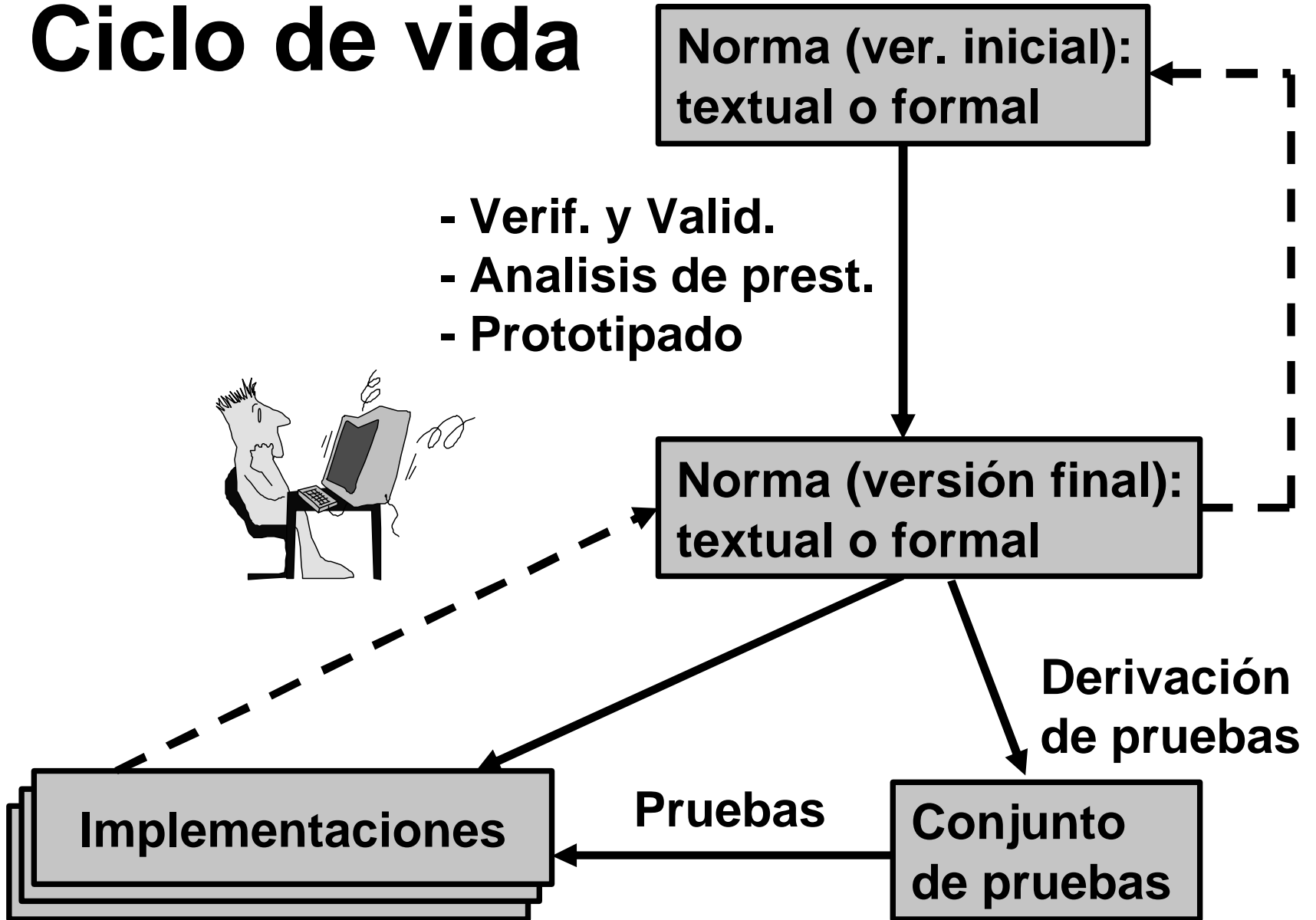
Norma (versión final):  
textual o formal

Derivación  
de pruebas

Implementaciones

Pruebas

Conjunto  
de pruebas



# Especificación: I

## ◆ Especificación de un protocolo

- ▶ Definición sin ambigüedades del algoritmo y del lenguaje de comunicación

## ◆ Las especificaciones se convierten en normas

- ▶ Legales: ISO/IEC , ITU/T, AENOR,....
  - Modelo de referencia para Inter. de Sist. Abiertos (ISO-OSI)
- ▶ De facto: normas seguidas por la industria
  - Internet STDs y RFCs
  - SUN Micro.: sus primeros productos siguen normas internet
- ▶ De empresa: para los productos de una empresa
  - IBM: SNA, DIGITAL: DECNET, ....., Microsoft, ..

# Especificación II

## ◆ Especificación textual

- ▶ Utiliza texto
  - Suele tener muchas ambigüedades, errores, ....
- ▶ Es necesario tener una implementación de referencia
- ▶ La mayoría de las especificaciones son textuales

## ◆ Especificaciones formales

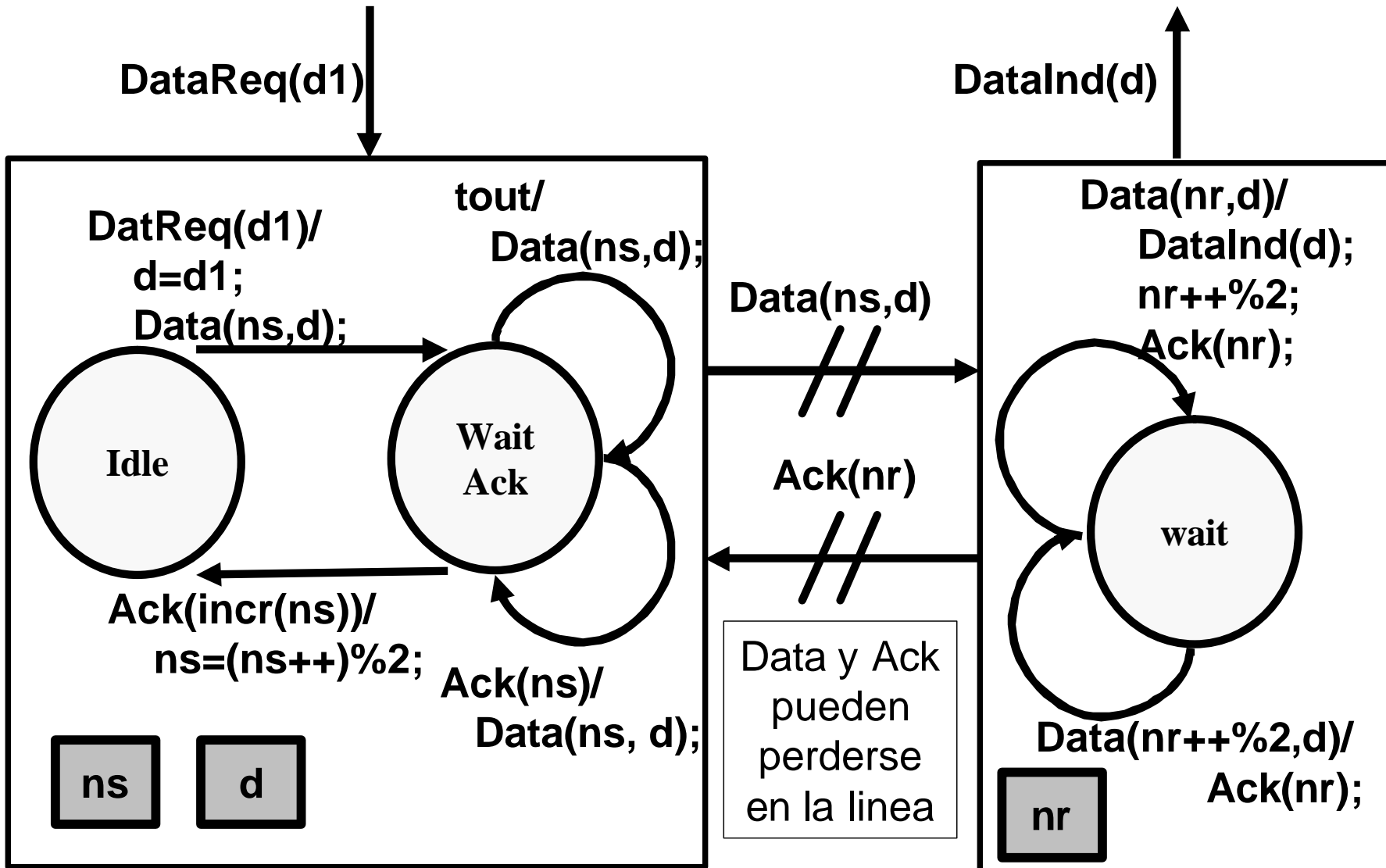
- ▶ Especificaciones precisas de carácter matemático
  - No poseen ambigüedades
    - Obligan a sobreespecificar
  - Normalmente basadas en autómatas extendidos
- ▶ Ejemplos: FAPL, SDL, LOTOS, Estelle,...

# Ejemplo de especificación textual

## ◆ Protocolo de bit alternante:

- ▶ Protocolo fiable de envío de datos
- ▶ Utiliza dos tipos de tramas: **datos** y **asentimiento**
  - Trama de datos: **tipo, numero de secuencia, datos**
  - Trama de asentimiento: **tipo, numero de secuencia**
- ▶ Comportamiento:
  - **Tramas de datos numeradas** consecutivamente
    - Trama de asentimiento asiente con el numero de secuencia de la proxima trama esperada
  - **Numero de secuencia**: entero modulo 2
    - Primera trama lleva numero de secuencia 0
  - **Tamaño de ventana 1**: solo puede haber una trama sin asentir
  - Emisor **retransmite al cabo de t seg.** si no recibe asentimiento

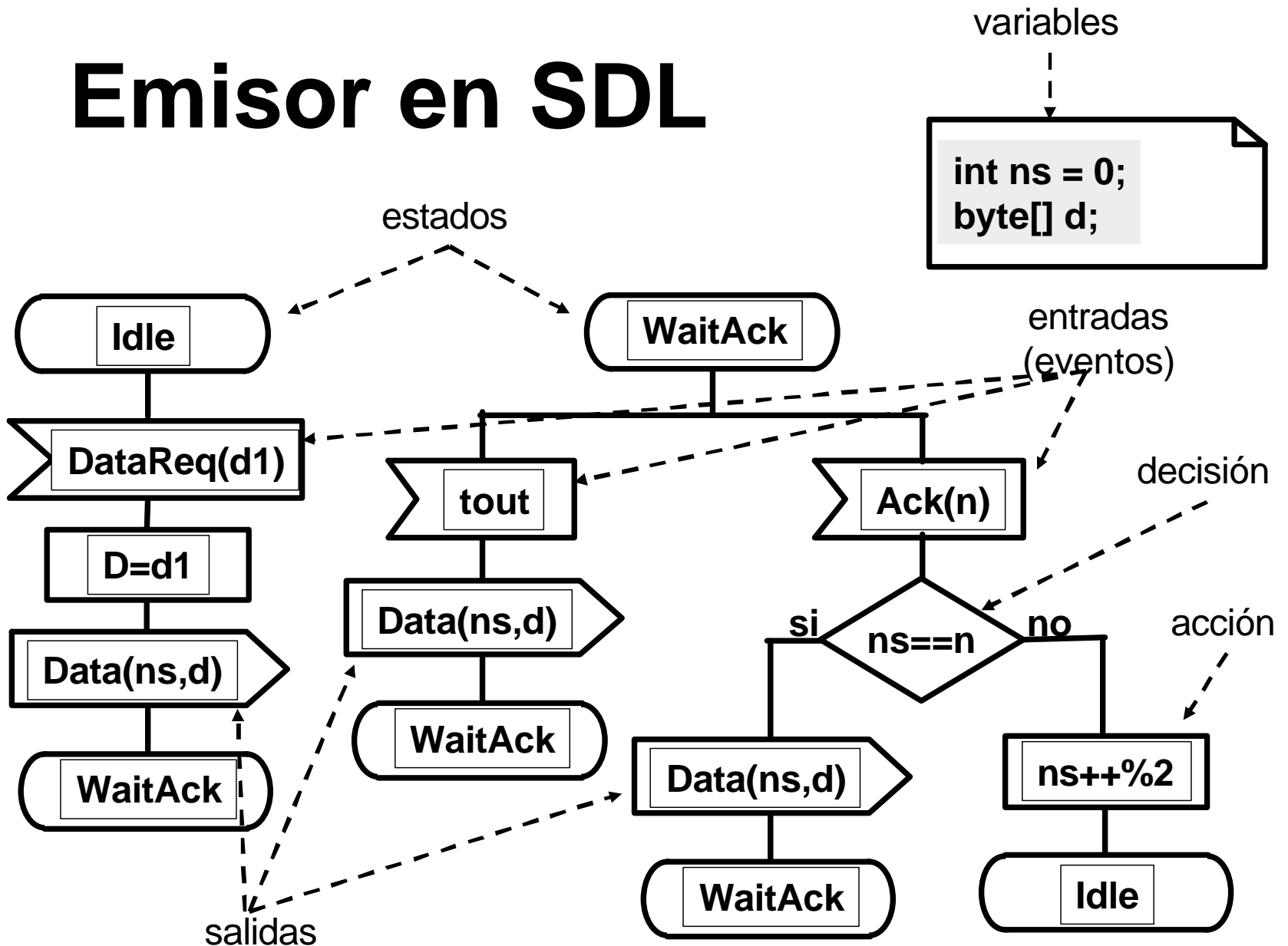
# Automata extendido



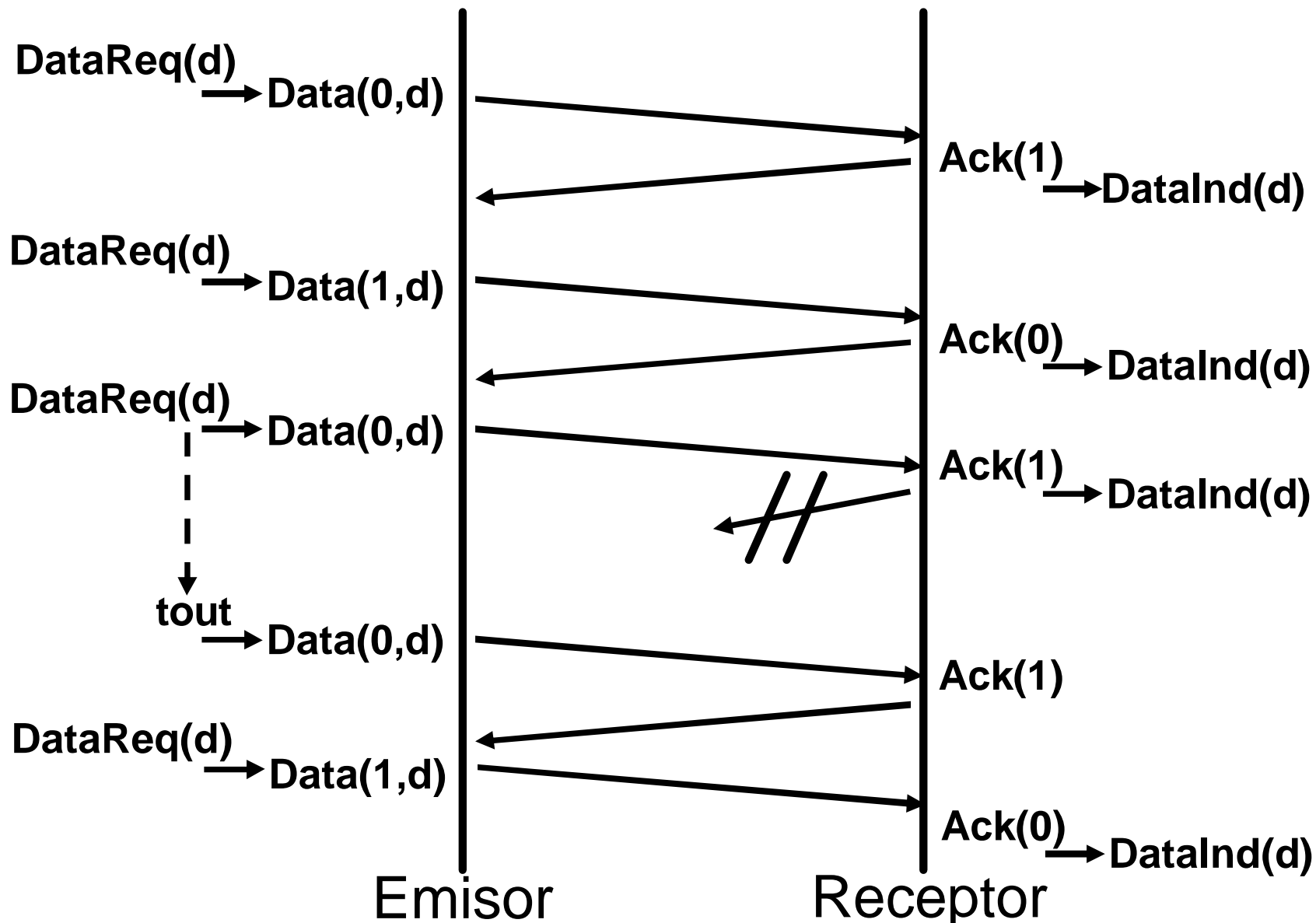
# Autómatas: comportamiento

- ◆ Estado: explícito e implícito
  - ▶ estado (explícito): representados por círculos
  - ▶ estado implícito: representado por variables
- ◆ Comportamiento: guiado por evento
  - ▶ Existe un conjunto de eventos exteriores
    - La ocurrencia de uno de estos eventos dispara transiciones
  - ▶ Los eventos se almacenan en colas de entrada a los autómatas
- ◆ Transiciones
  - ▶ Sintaxis: <evento>/<acciones>
  - ▶ Se disparan al ocurrir <evento>
    - al dispararse ejecutan las <acciones>
- ◆ Acciones
  - ▶ Envío de eventos a otros autómatas,
  - ▶ Cambio de estado o de valor de variable

# Emisor en SDL



# Comunicación: eventos



# Verificación y Validación

- ◆ **Verificación:** determinación de la corrección
  - ▶ Objetivo: determinar que el protocolo cumple el servicio
  - ▶ Obliga a especificar el servicio
- ◆ **Validación:** determinación de ausencia de propiedades indeseables
  - ▶ bloqueo, no terminación, recuperación, .....
- ◆ **En la practica se suele recurrir al prototipado**
  - ▶ Implementaciones experimentales que se analizan y ajustan mientras se ejecutan y usan
- ◆ **Resultado:** implementaciones de referencia

# Análisis de prestaciones

## ◆ Análisis de prestaciones

### ▶ Análisis de caudal y retardos

- El protocolo debe dar el servicio de forma eficiente

## ◆ Realización de modelos y análisis de prestaciones

### ▶ Se intentan analizar de forma automática sobre especificaciones formales

## ◆ Finalmente se miden sobre prototipos o implementaciones de referencia

# Implementación

## ◆ Implementación de un protocolo

- ▶ Todos los protocolos deben implementar el mismo algoritmo y hablar el mismo lenguaje
  - Sobre cualquier maquina
  - Generados con distintos compiladores y/o lenguajes

## ◆ Ideal: derivación automática a partir de especific.

- ▶ Compiladores de lenguajes de especificación
  - Ejemplo: compiladores de SDL, LOTOS, Estelle,....

## ◆ Lenguaje multiplataforma: C, Java, ..

- ▶ Permite ejecutar una implementación en cualquier ordenador.

# Pruebas

## ◆ Pruebas de conformidad

- ▶ Objetivo: determinar si un producto cumple la norma

## ◆ Realización de pruebas

- ▶ Pruebas de caja negra sobre una implementación
  - Se envían estímulos al protocolo y se analiza si sus respuestas son correctas

## ◆ El número de estados y comportamientos de un protocolo es enorme

- ▶ Es imposible analizar todo el comportamiento
  - Hay que elegir un conjunto significativo

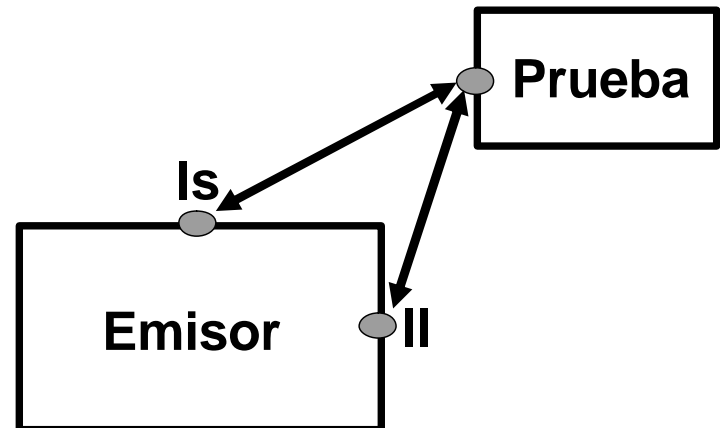
## ◆ Tema abierto de gran importancia industrial

# Secuencia de prueba: ejemplo

## ◆ Secuencia de prueba del emisor de un protocolo de bit alternante

- ▶ La secuencia identifica
  - Eventos que ocurren
  - Entradas y salidas (->)
  - Interfaz (Is e Il)
  - FINAL (evento especial)
    - indica éxito en el paso de la prueba
- ▶ Pueden existir arboles de prueba

Is:DataReq(d1)  
Il: ->Data(0,d1)  
Il:Ack(0)  
Is:DataReq(d2)  
Il: ->Data(1,d2)  
Il: ->Data(1,d2)  
Is:DataReq(d3)  
Il: ->Data(0,d3)  
**(FINAL)**



# **Ejemplo: el protocolo TFTP**

Bibliografía:

- RFC 1350
- TCP/IP Illustrated, Vol. 1, Stevens, Cap. 15

# El protocolo TFTP

## ◆ TFTP (Trivial File Transfer Protocol)

- ▶ Protocolo muy sencillo de transferencia de ficheros
  - Se usa en la carga inicial del S.O. a través de la red
    - Grabado en PROM de arranque
- ▶ Puede implementarse sobre diferentes servicios
  - normalmente se implementa sobre UDP

## ◆ Permite leer/grabar ficheros de/en el servidor

# TFTP sobre UDP unicast

## ◆ Cliente:

- ▶ Utiliza un numero de puerto (TID) aleatorio
  - Debe cambiar en cada solicitud

## ◆ Servidor: espera solicitudes en el puerto 69

- ▶ El servidor contesta por un puerto (TID) aleatorio

## ◆ Esta preparado para los tipos de error de UDP

- ▶ perdida, duplicación o desorden

# Formatos de mensajes de TFTP

**Solicitud de lectura/escritura de fichero**

2 bytes	string	1 byte	string	1 byte
codigo RRQ = 1 WRQ = 2	nombre fichero	0	modo	0

**Bloque de datos:**  
long. datos = 512  
long < 512 => eof

2 bytes	2 bytes	n bytes
codigo DATA = 3	numero bloque	datos

**Asentimiento:**  
n=bloque rec.

codigo ACK = 4	numero bloque
-------------------	---------------

**mensaje error:**  
0 = not defined  
1 = file not found  
2 = Access viol.  
.....

2 bytes	2 bytes	string	1 byte
codigo ERROR = 5	Codigo de error	mensaje de error	0

# Eventos del modelo

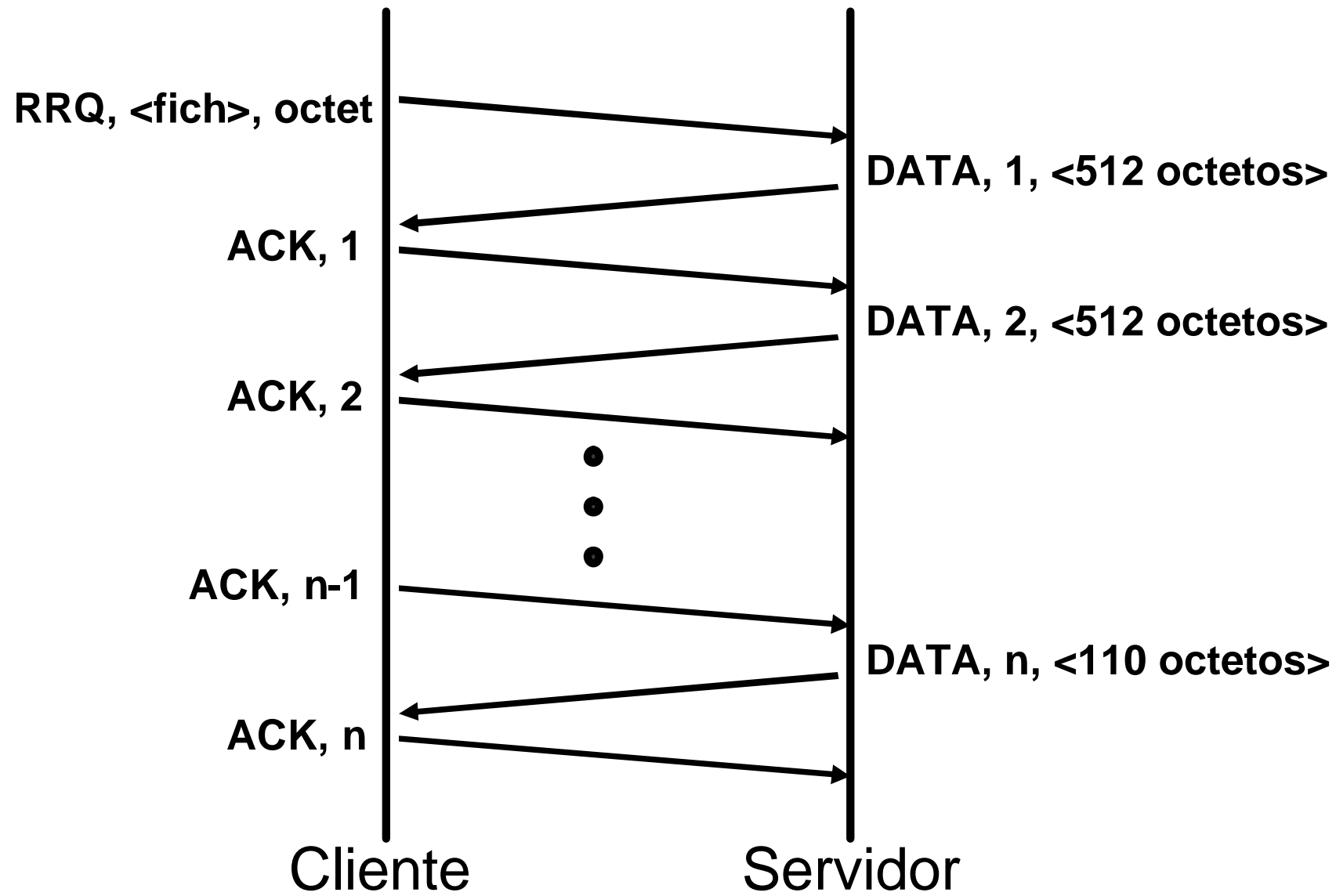
## ◆ Llegada/Envío de tramas

- ▶ Read Request: RRQ, <fich>, octet
- ▶ Write Request: WRQ, <fich>, octet
- ▶ Datos: DATA n <512 oct>
- ▶ Datos (final): DATA n <end>
- ▶ Asentimiento: ACK n

## ◆ Temporizadores

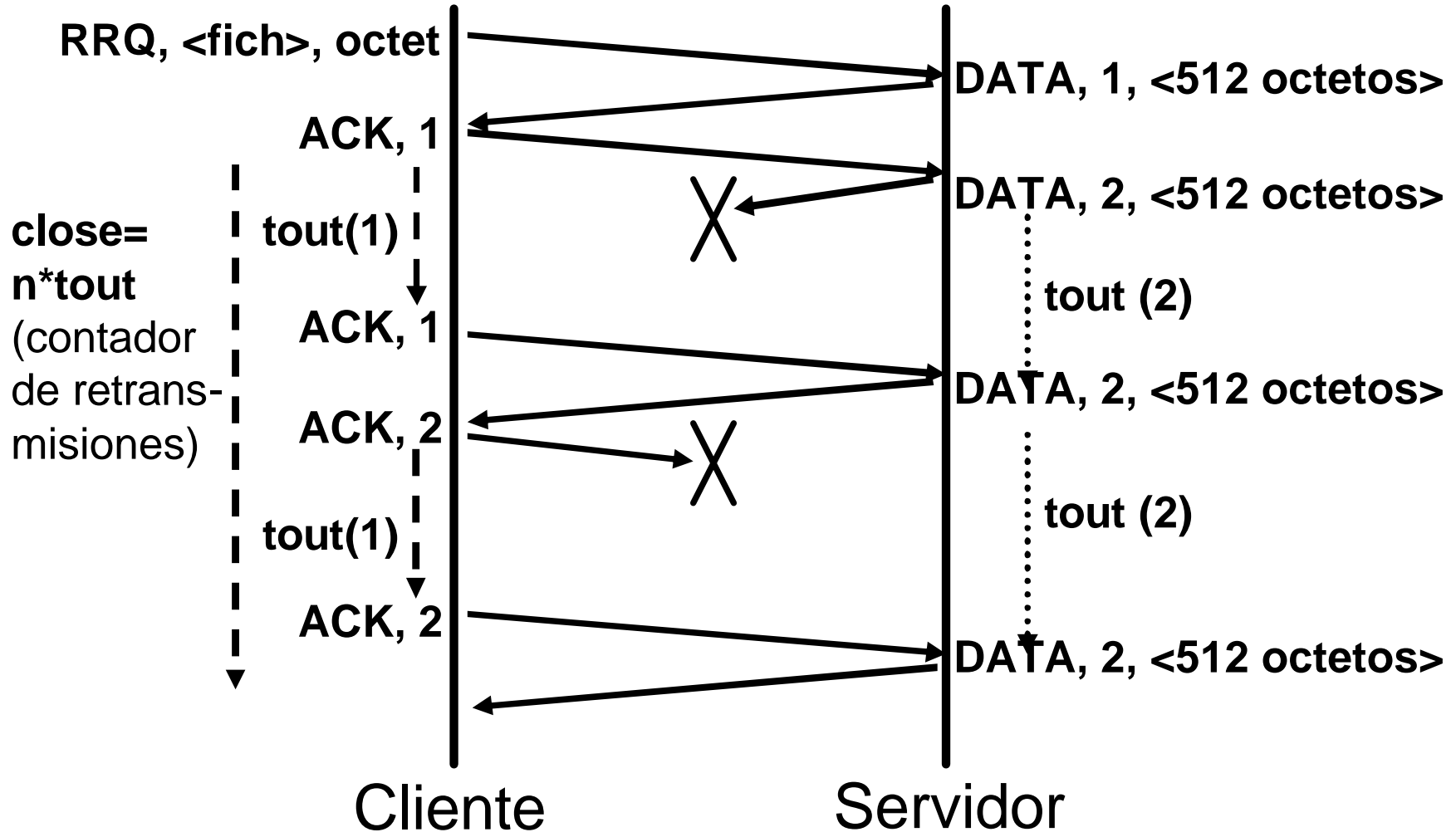
- ▶ Retransmisión: tout
  - arranque: tout(start), parada: tout(stop)
- ▶ Limite de retransmisiones: close
  - arranque: close(start), parada: close(stop)

# Lectura remota de fichero



# Lectura remota: errores

## CASO 1: perdidas



# Lectura remota: errores II

## CASO 2:

fichero inexistente

RRQ, <fich>, octet

error, 1, file not found

## CASO 3:

lectura prohibida

RRQ, <fich>, octet

error, 2, Access violation  
(no read permission)

## CASO 4:

fichero corrupto

RRQ, <fich>, octet

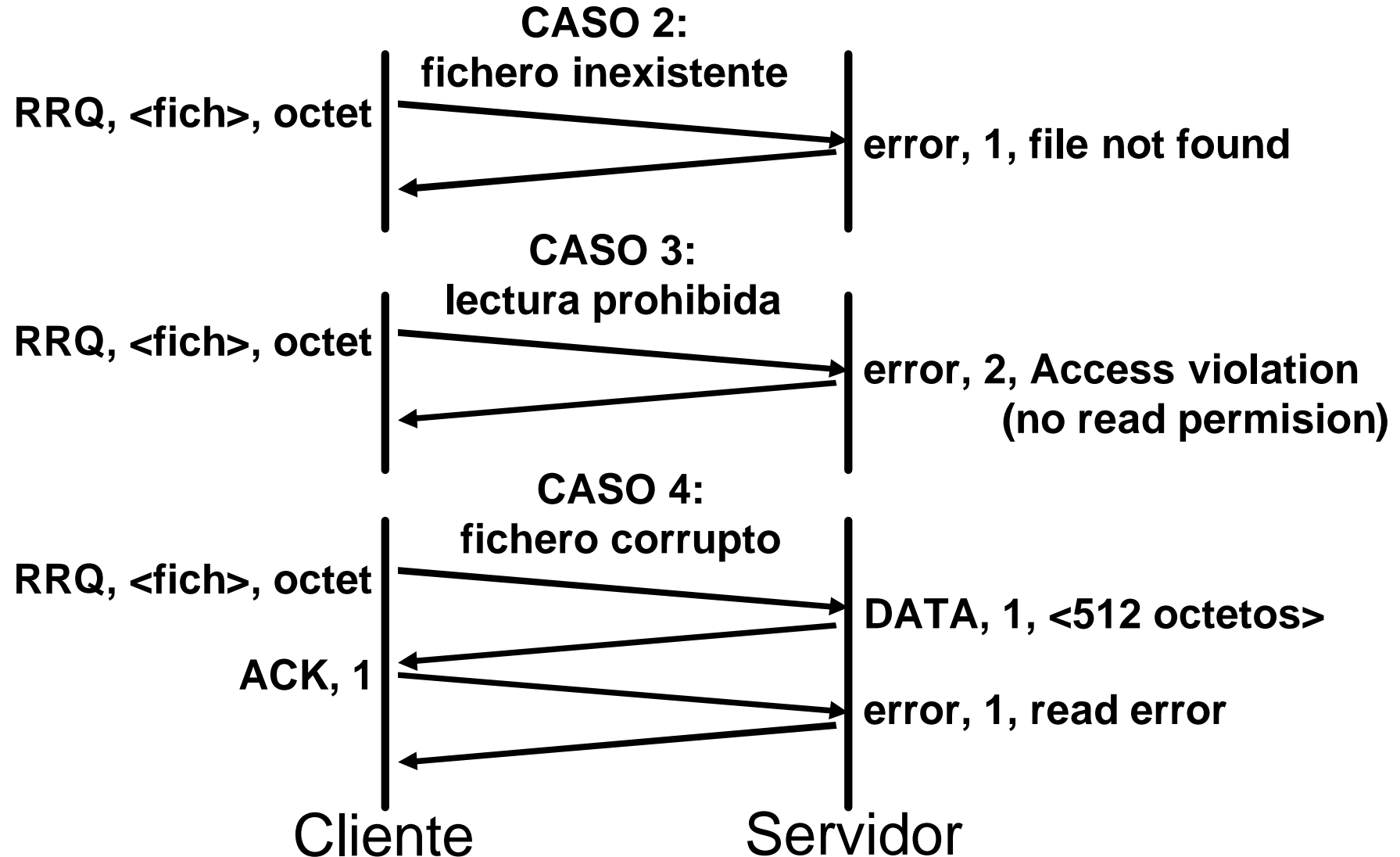
DATA, 1, <512 octetos>

ACK, 1

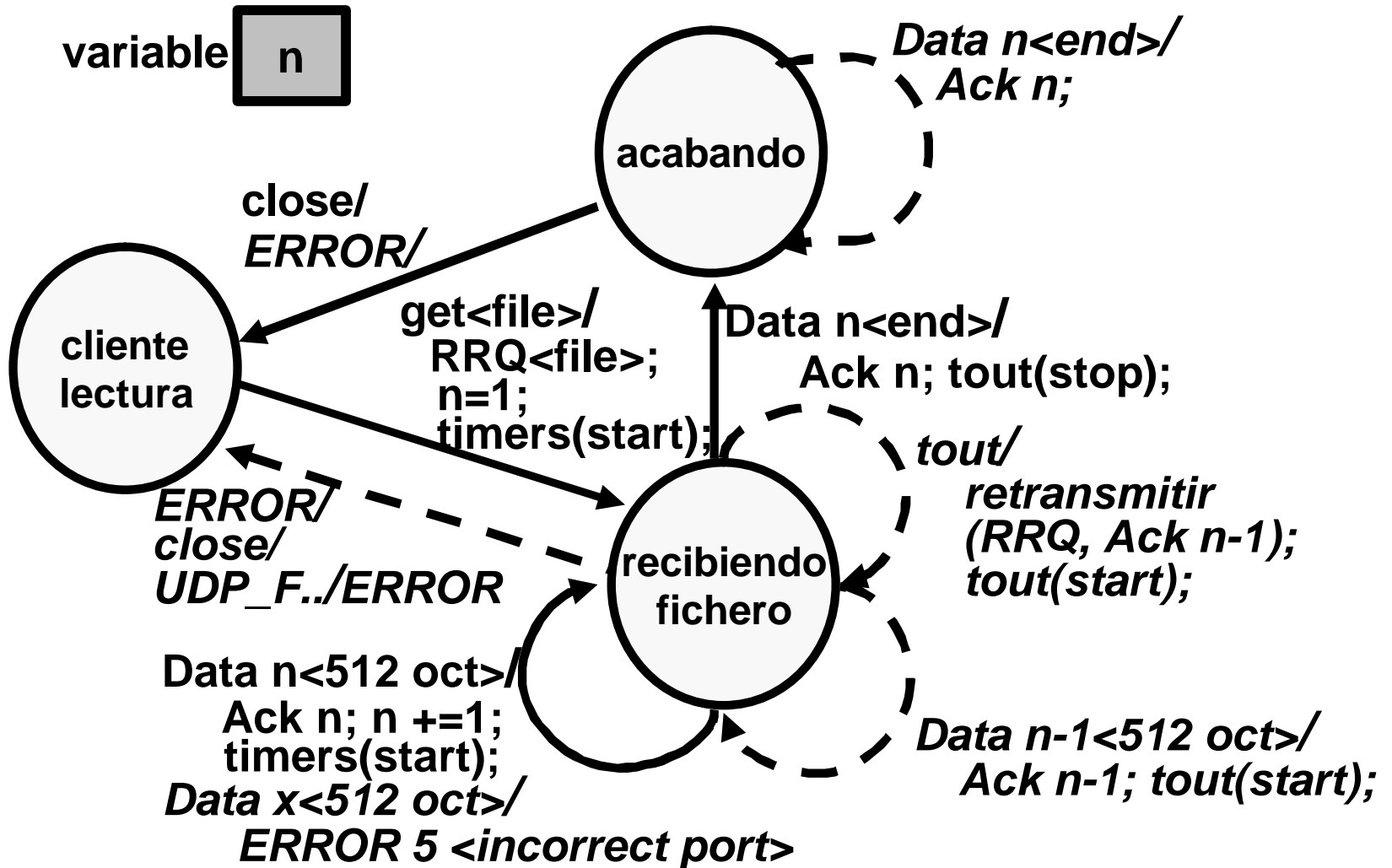
error, 1, read error

Cliente

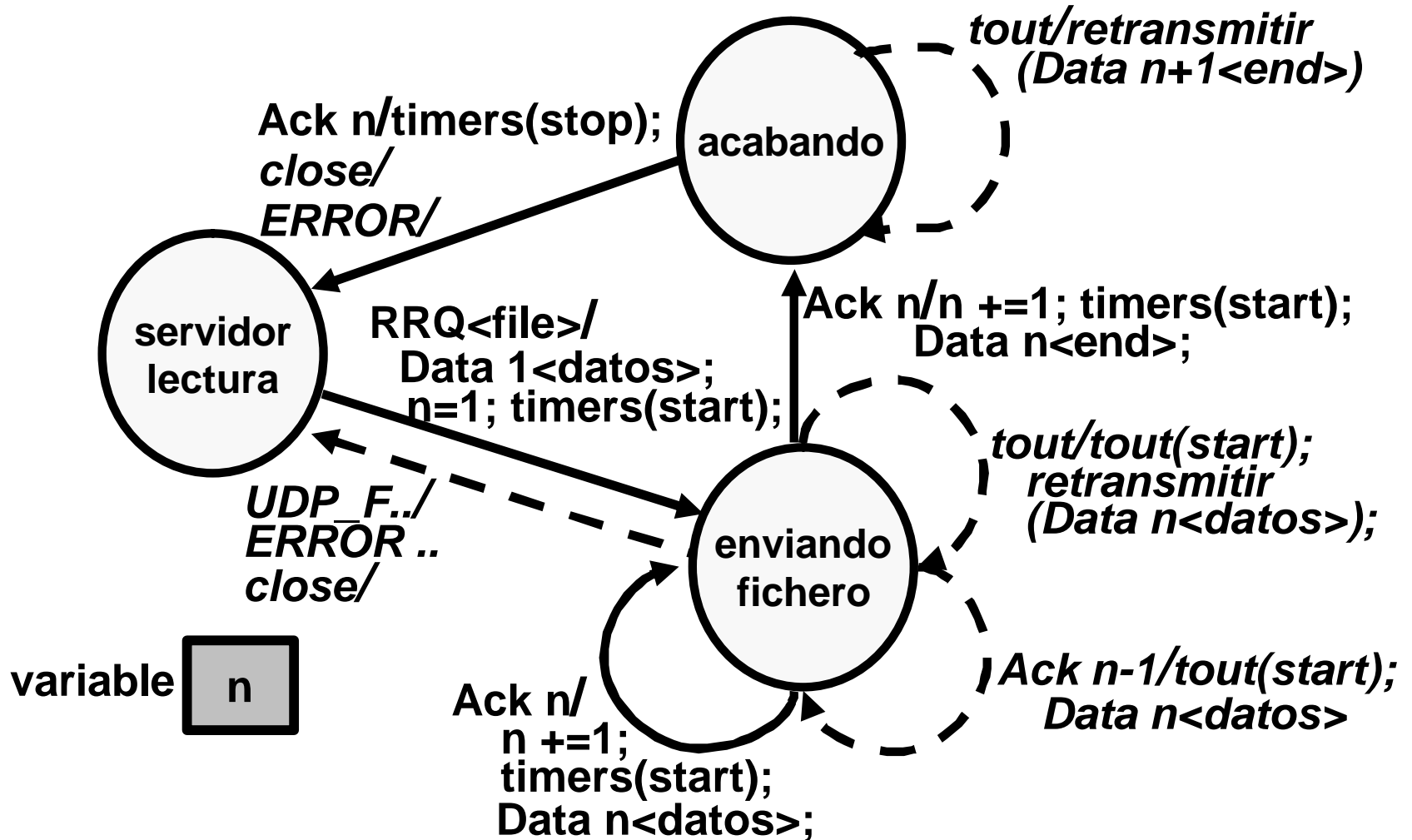
Servidor



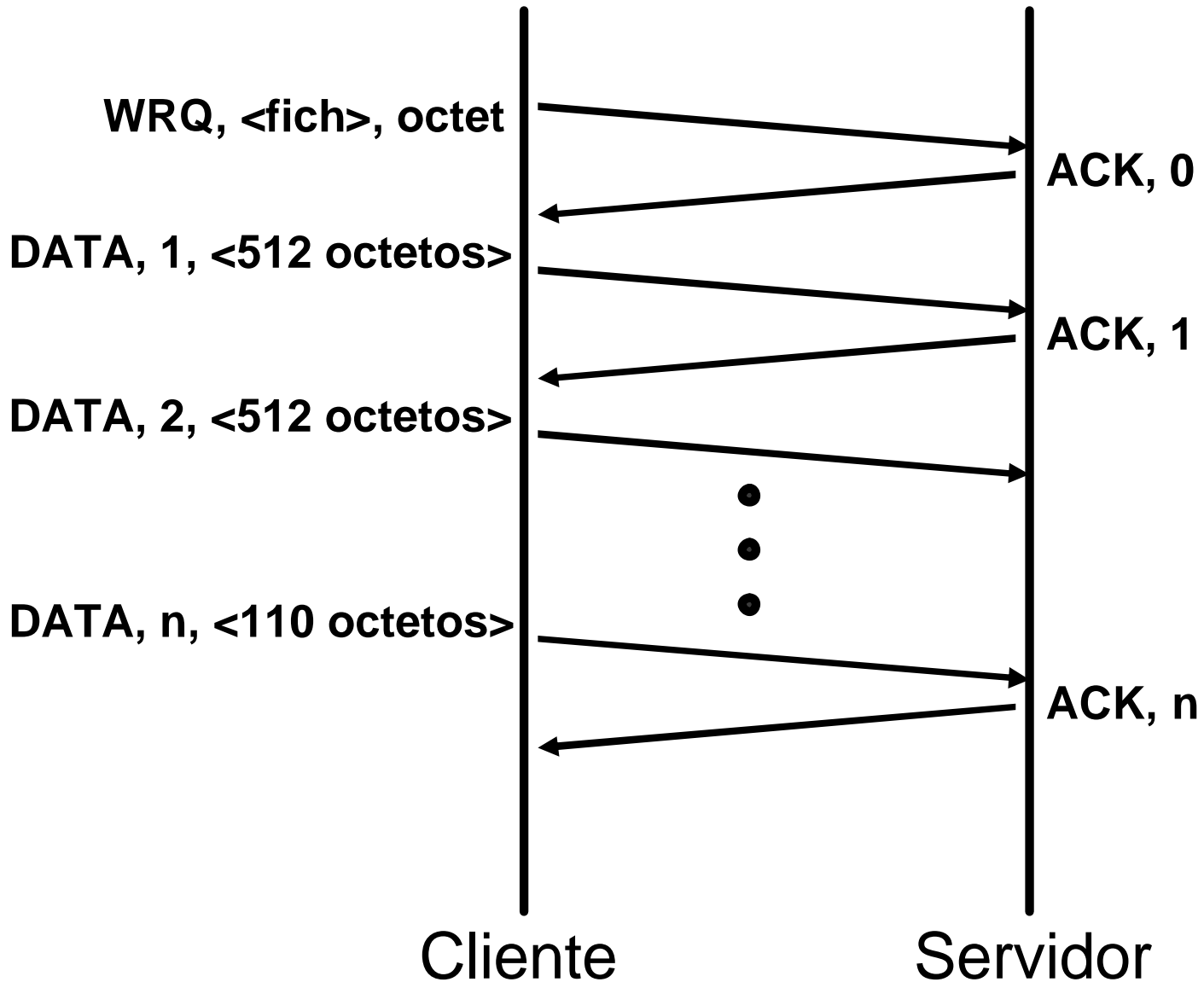
# TFTP: cliente lectura



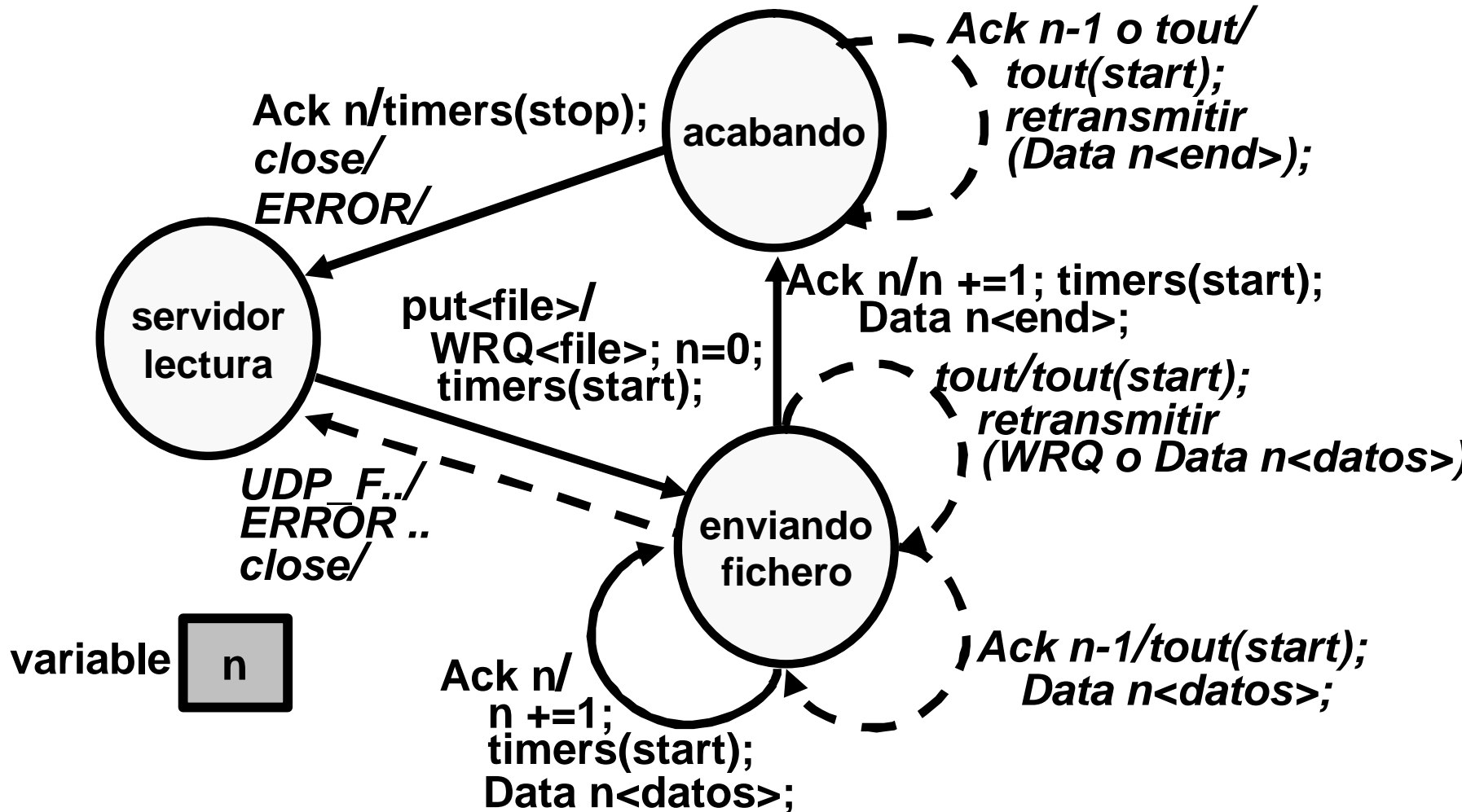
# TFTP: servidor lectura



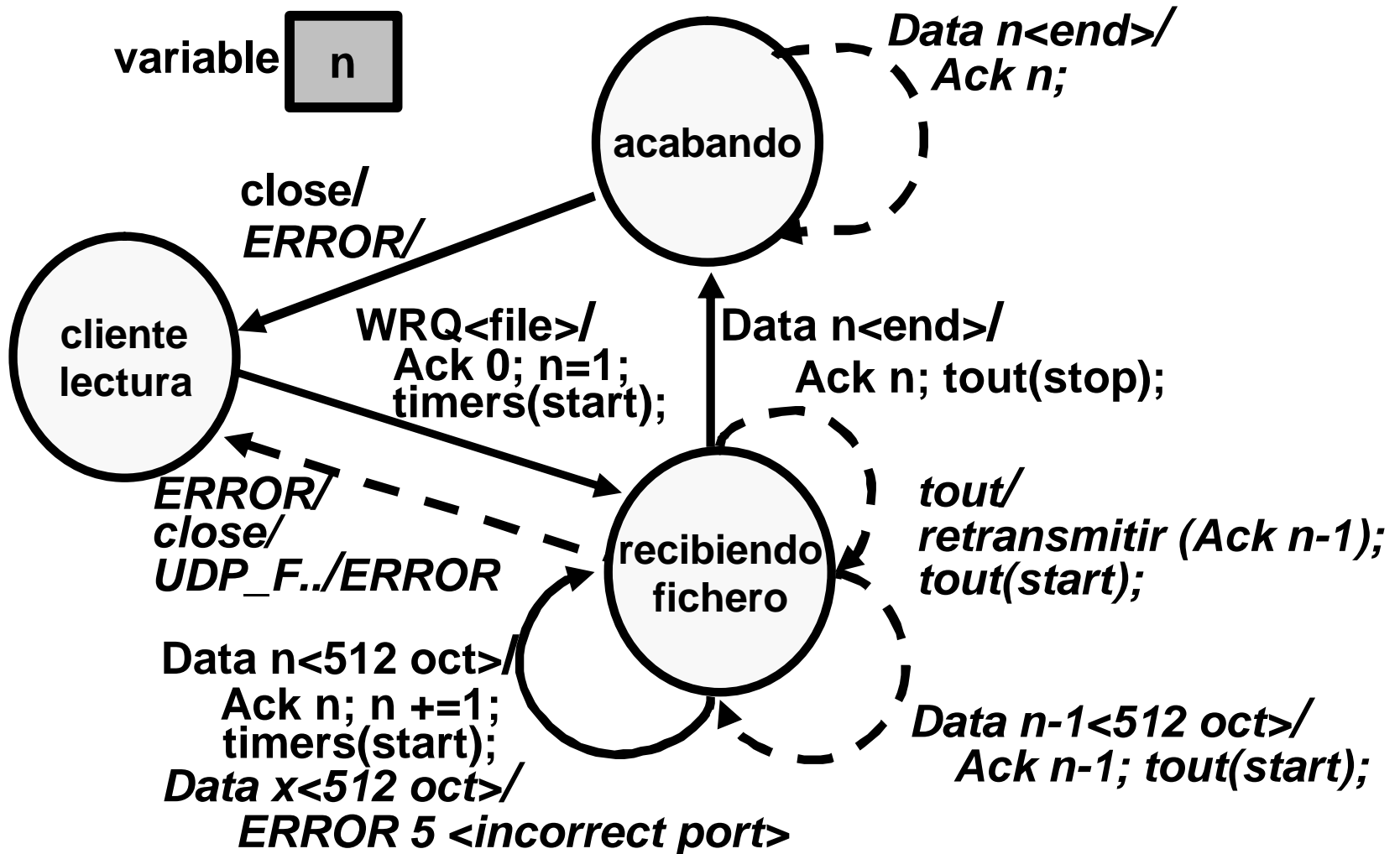
# Escritura remota de fichero



# TFTP: cliente escritura



# TFTP: servidor escritura



# Implementación: características

## ◆ Invocación:

- ▶ >java tftp get <file> <host>

## ◆ Arquitectura

- ▶ Herencia, visibilidad y uso de objetos
  - Relación entre objetos y sus funciones
- ▶ Flujo de datos, intercambio de información entre obj.

## ◆ Concurrencia

- ▶ Secciones críticas
- ▶ Interbloqueos

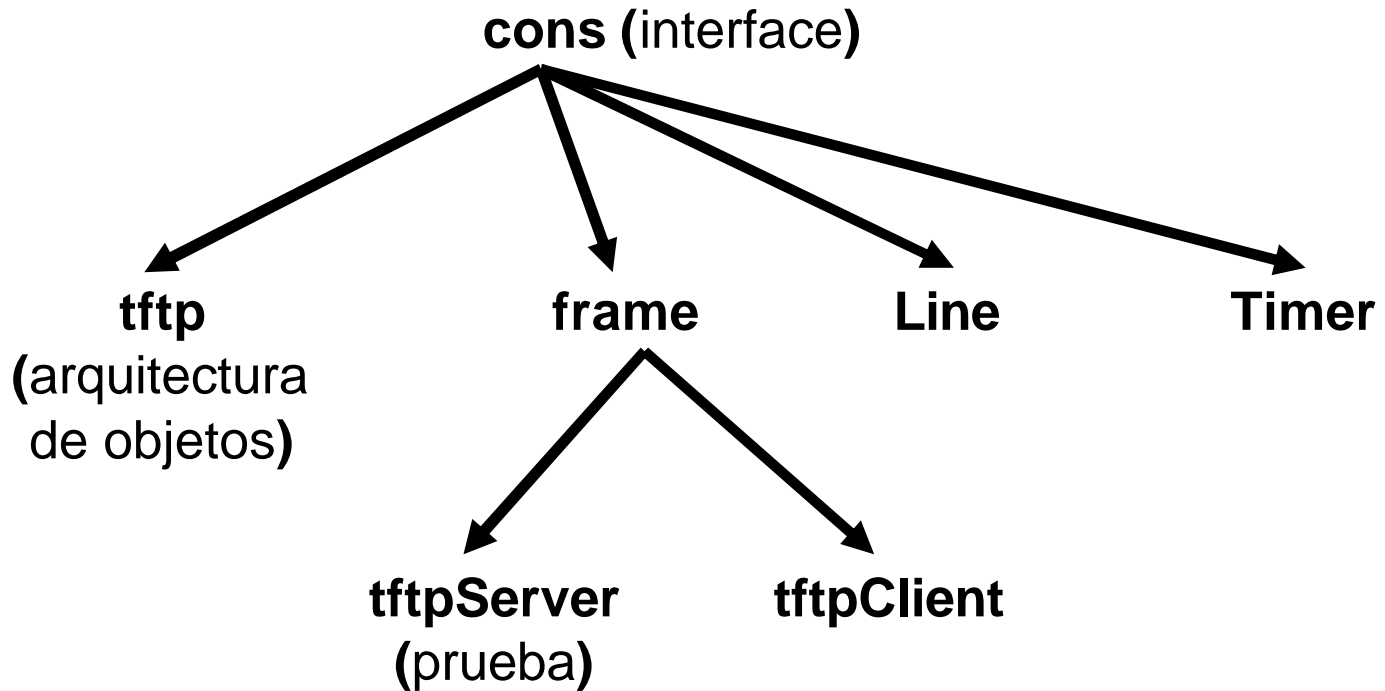
## ◆ Gestión de memoria

- ▶ minimizar copias, pasar referencias

## ◆ Pruebas

- ▶ Incluir procesos y puntos de prueba y de traza

# Arquitectura: herencia de objetos



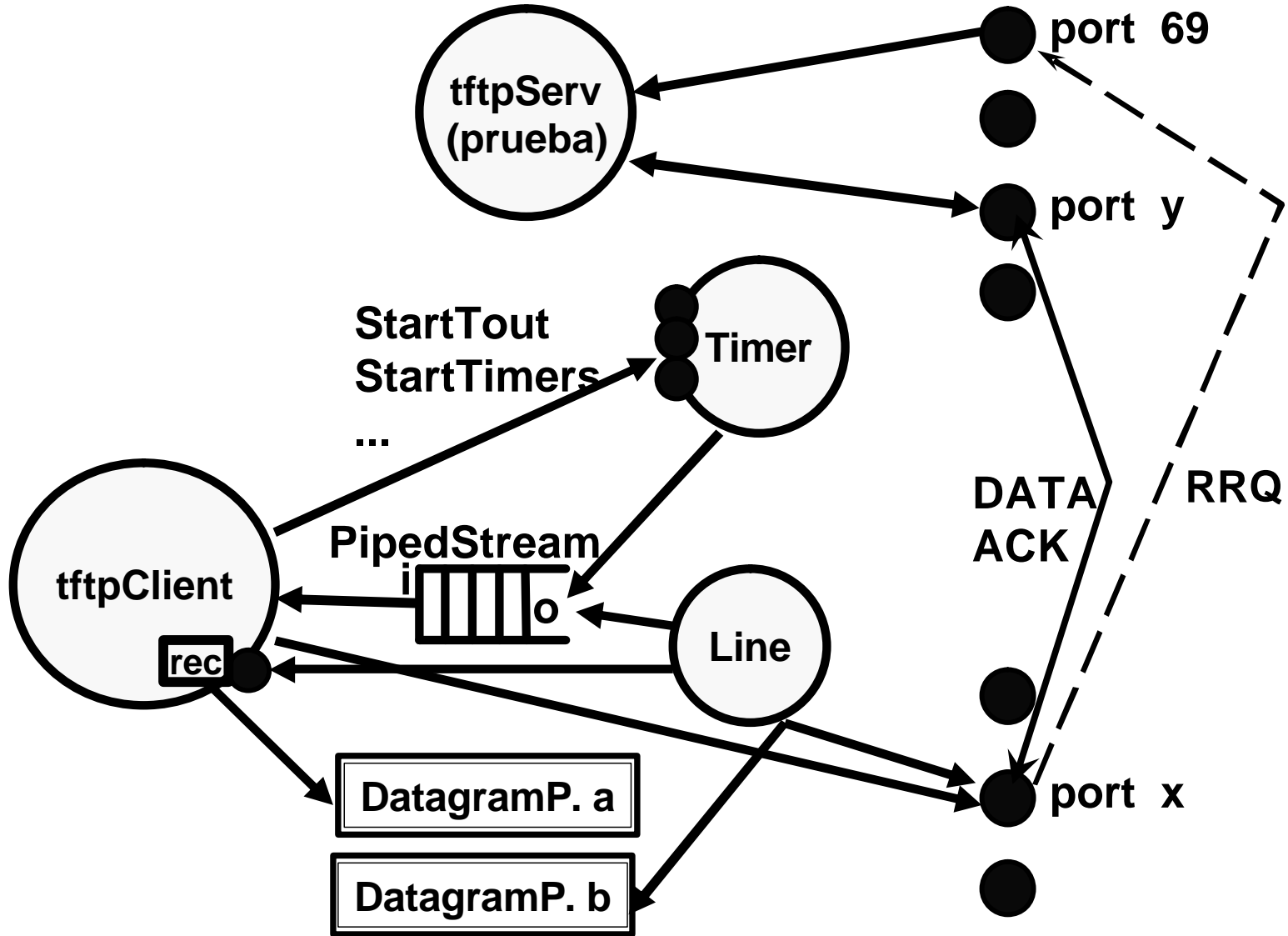
# cons: constantes

```
import java.io.*;
import java.net.*;

interface cons {
    static final int tout=0, close=1, frame=2;
                                     // Protocol messages
    static final int espera=0, recibiendo=1, acabando=2;
                                     // Protocol states
    static final int RRQ=1, WRQ=2, DATA=3, ACK=4, ERROR=5;
                                     // Protocol frame codes
    static final int ServerPort=69;
}

```

# tftp: procesos y objetos



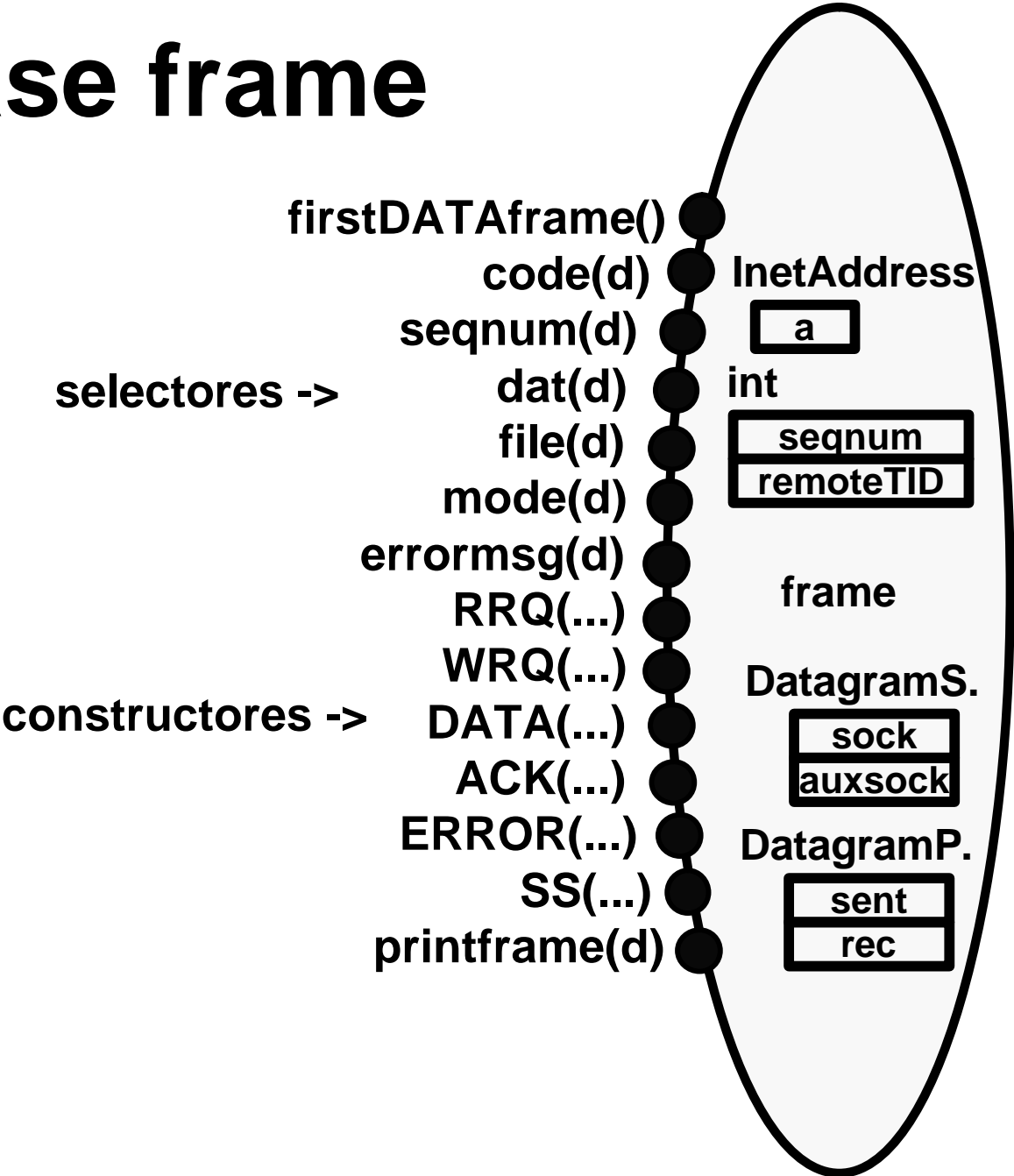
# tftp: arquitectura de objetos

```
public class tftp {
    public static void main (String args[]) throws IOException {
        if (args.length!= 3){throw(new RuntimeEx. ("Syntax: .."));
        } else if (args[0].equals("get")){

            // the objects created reflect directly the architecture
            // of the figure of the previous page

            PipedOutputStream o = new PipedOutputStream();
            PipedInputStream i = new PipedInputStream(o);
            DatagramSocket sock = new DatagramSocket();
            tftpServ ts = new tftpServ(); // Test server
            Timer ti = new Timer(o); // Timer manager
            tftpClient tf = new tftpClient(args[1], args[2], i, ti, sock);
            Line l = new Line(tf, o, sock); // Line Manager
        } else {System.out.println("unknown com .....");}
    }
}
```

# Clase frame



# Formatos de mensajes de TFTP

**Solicitud de lectura/escritura de fichero**

2 bytes	string	1 byte	string	1 byte
codigo RRQ = 1 WRQ = 2	nombre fichero	0	modo	0

**Bloque de datos:**  
long. datos = 512  
long < 512 => eof

2 bytes	2 bytes	n bytes
codigo DATA = 3	numero bloque	datos

**Asentimiento:**  
n=bloque rec.

codigo ACK = 4	numero bloque
-------------------	---------------

**mensaje error:**  
0 = not defined  
1 = file not found  
2 = Access viol.  
.....

2 bytes	2 bytes	string	1 byte
codigo ERROR = 5	Codigo de error	mensaje de error	0

# Tramas: variables y otros

---

```
class frame implements cons {  
  
    InetAddress a;           // variable for address construction  
    int seqnum = 0;          // sending sequence number  
    int remoteTID;         // server port number  
    DatagramSocket sock;  // main socket  
    DatagramSocket auxsock; // auxiliary socket for tftpServ  
    DatagramPacket sent, rec; // sending and receiving buffers  
  
    boolean firstDATAframe ()  
        {return((code(rec)==DATA)&&(remoteTID==ServerPort));}  
        // predicate for detecting first frame send  
        //      condition: data frame with remote port 69  
  
    String ch(int b) { byte[] bb = {(byte) b}; return new String(bb); }  
    // convert 1 byte numeric code to an equivalent String format
```

# Frames: selectors

```
int code(DatagramPacket d)
    { return (((d.getData())[0])<< 8)|((d.getData())[1]); }
// returns the code indicating the type of frame
```

```
int seqnum(DatagramPacket d)
    { return (((d.getData())[2])<< 8)|((d.getData())[3]);}
// returns the sequence number of a DATA or ACK frame
```

```
byte[] dat(DatagramPacket d) {
    byte[] b = new byte[(d.getLength() - 4)];
    System.arraycopy(d.getData(), 4, b, 0, (d.getLength() - 4));
    return b;
} // returns the data, as a byte array, of a data frame
```

```
String file(DatagramPacket d) {
    int i; byte[] b = d.getData();
    for (i = 2; b[i] != 0; i++){;}; // detection of string delimiter
    return new String(b, 2, (i-2)); // ascii to String conversion
} // returns the filename, as a String, of a RRQ or WRQ frame
```

# Tramas: selectores

```
String mode(DatagramPacket d) {  
    int del, i;  
    byte[] b = d.getData(); // get frame as byte array  
    for (i = 2; b[i]!=0; i++){;}; // detection of first string delimiter  
    del=i;  
    for (i = del+1; b[i]!=0; i++){;}; // second string delimiter  
    return new String(b, (del+1), (i-(del+1))); // ascii to String  
} // returns the mode, as a String, of a RRQ or WRQ frame
```

```
String errmsg (DatagramPacket d) {  
    int i; byte[] b = d.getData(); for (i = 4; b[i]!=0; i++){;};  
    return new String(b, 4, (i-4));  
} // returns the error msg, as a String, of an ERROR frame
```

# Tramas: constructores

```
DatagramPacket RRQ(String file, String type) throws IOException. {  
    byte[] b =(ch(0)+ch(RRQ)+file +ch(0)+type+ch(0)).getBytes();  
    return new DatagramPacket(b, b.length, a, remoteTID);  
} // construct a new Datagram packet with RRQ frame
```

```
DatagramPacket WRQ(String file, String type) throws IOException. {  
    byte[] b =(ch(0)+ch(WRQ)+file+ch(0)+type+ch(0)).getBytes();  
    return new DatagramPacket(b, b.length, a, remoteTID);  
} // construct a new Datagram packet with WRQ frame
```

```
DatagramPacket ERROR(int code, String msg) throws IOException. {  
    byte[] b = ((ch(0) + ch(ERROR) + ch(0) + ch(code) +  
                msg + ch(0)).getBytes());  
    return new DatagramPacket(b, b.length, a, remoteTID);  
} // construct a new Datagram packet with ERROR frame
```

# Frames: Constructors II

```
DatagramPacket DATA (byte[] b, int l) throws IOException {  
    b[0] = 0;  
    b[1] = DATA;  
    b[2] = (byte) (seqnum >> 8);  
    b[3] = (byte) seqnum;  
    return new DatagramPacket(b, l, a, remoteTID);  
} // construct a Datagram packet with DATA frame  
    // sequence number inserted from "seqnum" variable  
    // port and address from "a" and "remoteTID" variables  
    // length of array "b" = 4 + length of data
```

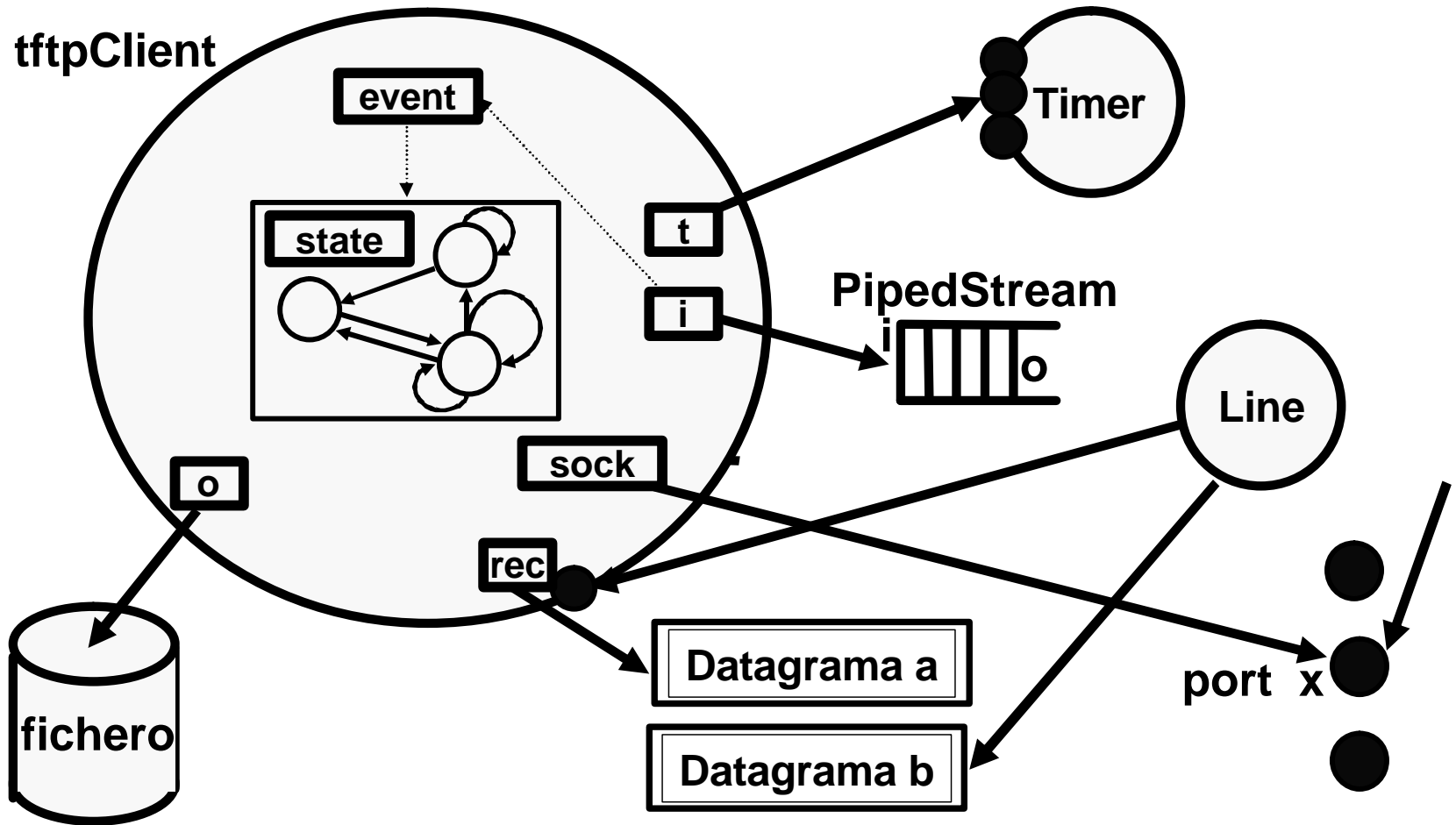
```
DatagramPacket ACK() throws IOException {  
    byte[] b = {0, ACK, (byte)(seqnum >> 8), (byte)seqnum};  
    return new DatagramPacket(b, b.length, a, remoteTID);  
} // construct a new Datagram packet with ACK frame
```

```
void SS (DatagramSocket s, DatagramPacket p) throws IOE.{  
    s.send(sent = p); printframe(p);  
} // send Datagram, store in sent for retransmission and print it
```

# Tramas: trazas

```
void printframe (DatagramPacket dp) {
    byte[] b = dp.getData();
    if (code(dp) == RRQ)
        {System.out.println("RRQ" + " file=" + file(dp) + .....);}
    if (code(dp) == WRQ)
        {System.out.println("WRQ" + " file=" + file(dp) + .....);}
    if (code(dp) == DATA)
        {System.out.println("DATA" + " seqnum=" + .....);}
    if (code(dp) == ACK)
        {System.out.println("ACK" + " seqnum=" + .....);}
    if (code(dp) == ERROR)
        {System.out.println("ERROR" + " error code=" + .....);}
}
```

# tftpClient: el protocolo



# Cliente: cabecera

```
class tftpClient extends frame implements Runnable {  
    String file, host;           // parametros de activación  
    FileOutputStream o; PipedInputStream i;  
    Timer t;                       // referencia al gestor de timers  
    int event, state = espera; // variable de evento y estado
```

```
tftpClient (String f, String h, PipedInputStream inp, Timer ti,  
                                                    DatagramSocket s) {  
    file = f; host = h; i = inp; t = ti; sock = s;  
    try {  
        o = new FileOutputStream(file);  
        a = InetAddress.getByName(host);  
        Thread tc = new Thread(this); tc.start();  
    } catch (UnknownHostException e) {System.err.println("Get: ...");}  
    } catch (IOException e) {System.err.println("Get: " + e);}  
}
```

# Cliente: automata

```
public void run () {
  try {
    remoteTID = ServerSocket; // primera transición
    .....
    while (state != espera) { // bucle de espera de eventos
      event = i.read();
      switch (event) {
        case frame: ..... // procesa la trama
        case close: ..... // retorna a estado inicial
        case tout: ..... // retransmite
        default:
          break;
      }
    }
    o.close();
  } catch ( ..... excepciones )
}
```

# Cliente: transiciones I

```
remoteTID = ServerPort;           // primera transición
SS(sock, (sent=RRQ(file, "netascii")));
t.startTimers();  state = recibiendo;

while (state != espera) {         // bucle espera de eventos
    event = i.read();             // espera evento
    switch (event) {
        case frame: .....
        case close:
            state = espera;
            break;
        case tout:
            if (state == recibiendo) { SS(sock, sent); t.startTout(); }
            break;                // reenvío de trama
        default:
    }
}
```

# Ciente: transiciones II

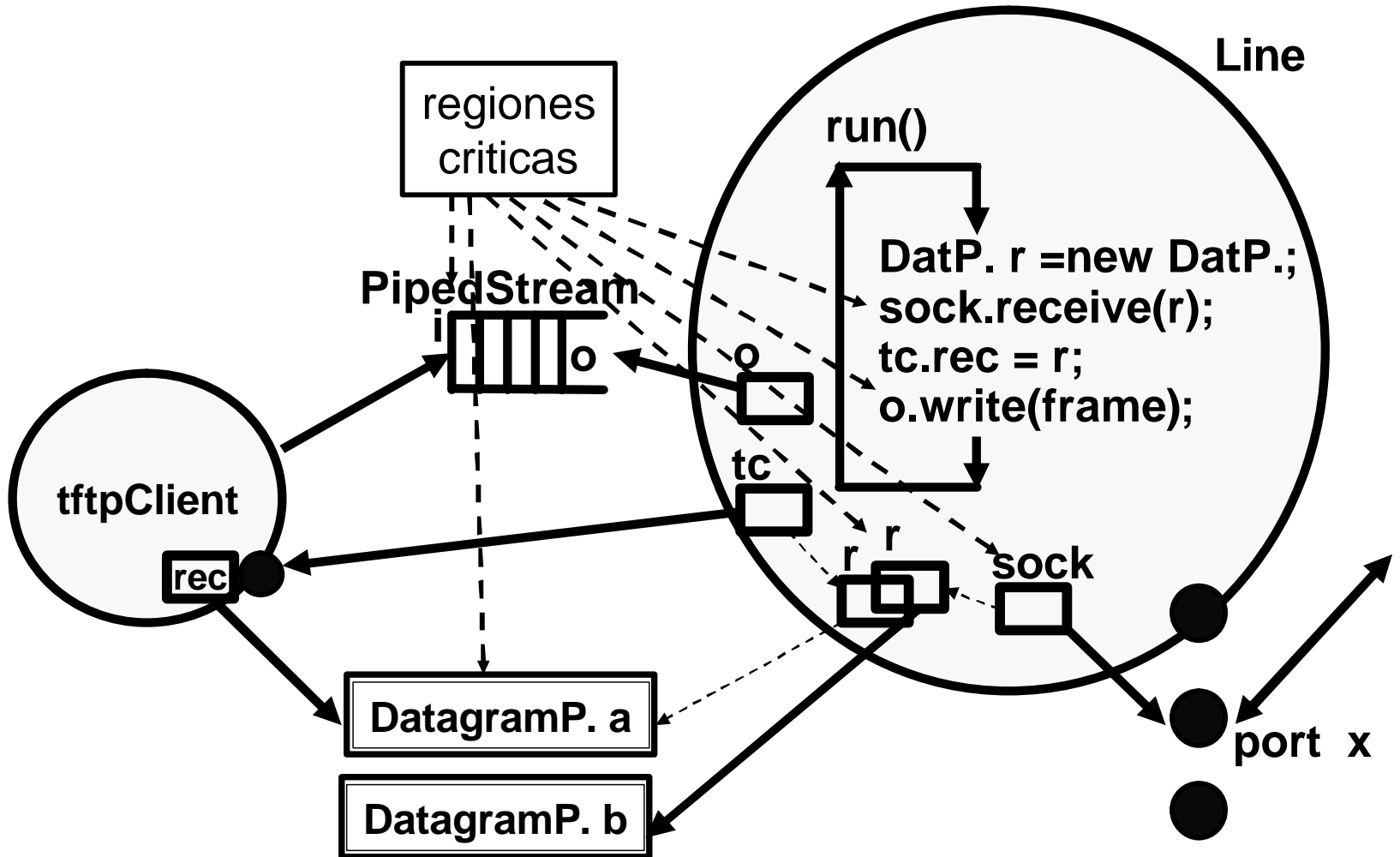
---

case frame:

```
if (firstDATAframe()){remoteTID=rec.getPort();}
                                // guarda nuevo TID remoto
if (rec.getPort() != remoteTID)
    {SS(sock, ERROR(0, "wrong TID")); break;}
                                // trama no llega del servidor
if (code(rec)==DATA){
    if ((seqnum(rec)) != seqnum) // trama de datos nueva
        {seqnum=seqnum(rec); t.startTimers(); o.write(dat(rec));}
    SS(sock, (sent=ACK()));      // envía (re)asentimiento
    if (state == recibiendo) {t.startTout();}
    if (rec.getLength() < 516) {state =acabando; t.stopTout();}
}
                                // última trama de datos
if (code(rec)==ERROR) {state = espera; t.stopTimers();}
break;
```

---

# Line

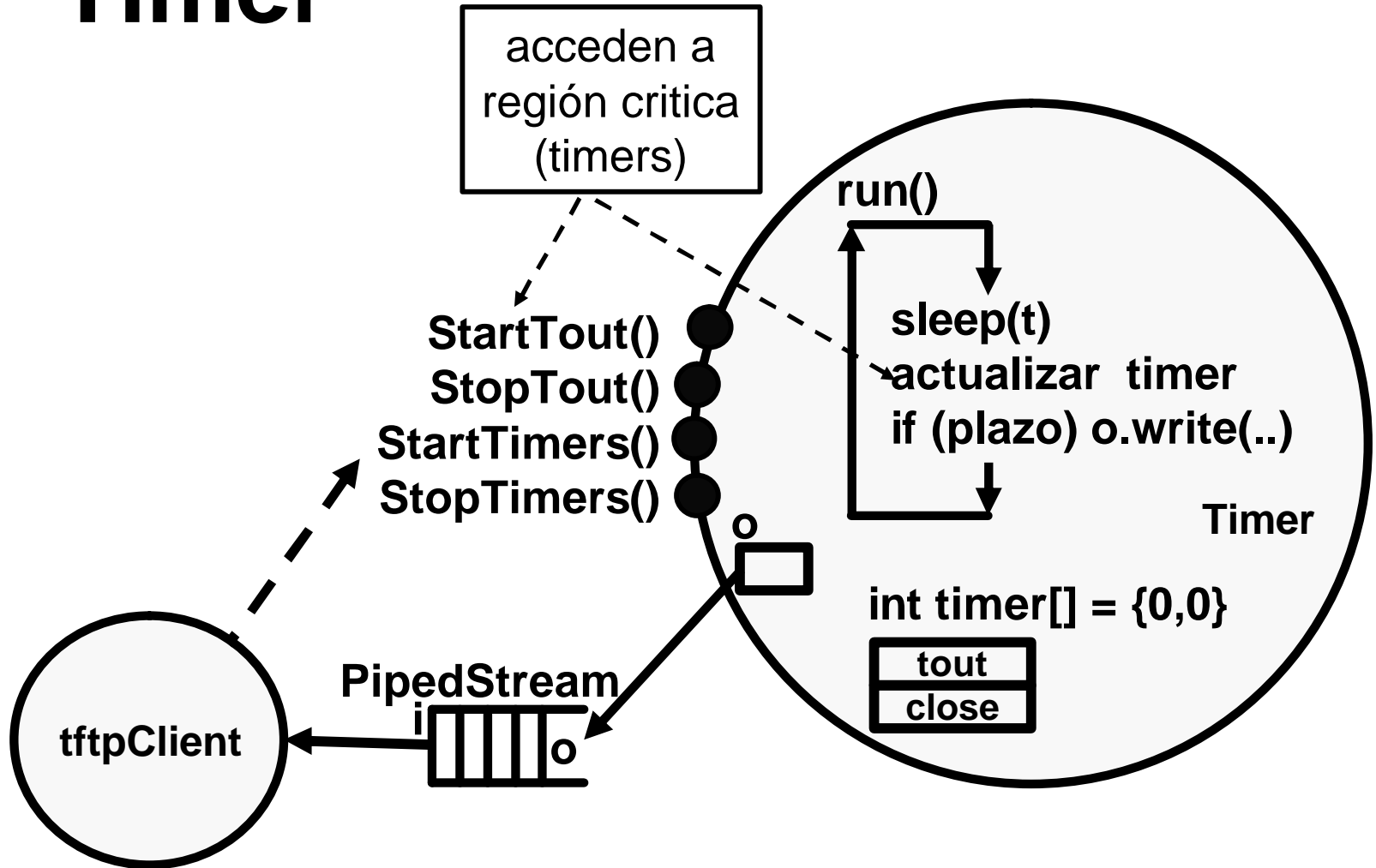


# Line

```
class Line implements Runnable, cons {
    tftpClient tc; DatagramSocket sock; PipedOutputStream o;

    Line (tftpClient tf, PipedOutputStream ot, DatagramSocket s)
        {tc=tf; o=ot; sock=s; Thread t = new Thread(this); t.start();}
    public void run () {
        while (true) {
            try {
                DatagramPacket r =new DatagramPacket(new byte[0],0);
                sock.receive(r);          // espera llegada de datgrama
                tc.rec = r; // deja datagrama recibido en buf. de tftpClient
                o.write((byte)frame); o.flush(); // envia evento "frame"
            } catch (IOException e) {System.out.println("Line: " + e);}
        }
    }
}
```

# Timer



# Timer I

---

```
class Timer implements Runnable, cons {  
    int[] timer = {0,0} ; // array con contadores de los dos timers  
    PipedOutputStream o; // stream para envío de timeouts
```

```
Timer (PipedOutputStream ot)
```

```
{ o = ot; Thread t = new Thread(this); t.start(); }
```

```
synchronized void startTout () { timer[tout] = plazo; }
```

```
synchronized void stopTout () { timer[tout] = 0; }
```

```
synchronized void startTimers () // arranca los dos timers  
    { timer[tout] = plazo; timer[close] = plazo*fin; }
```

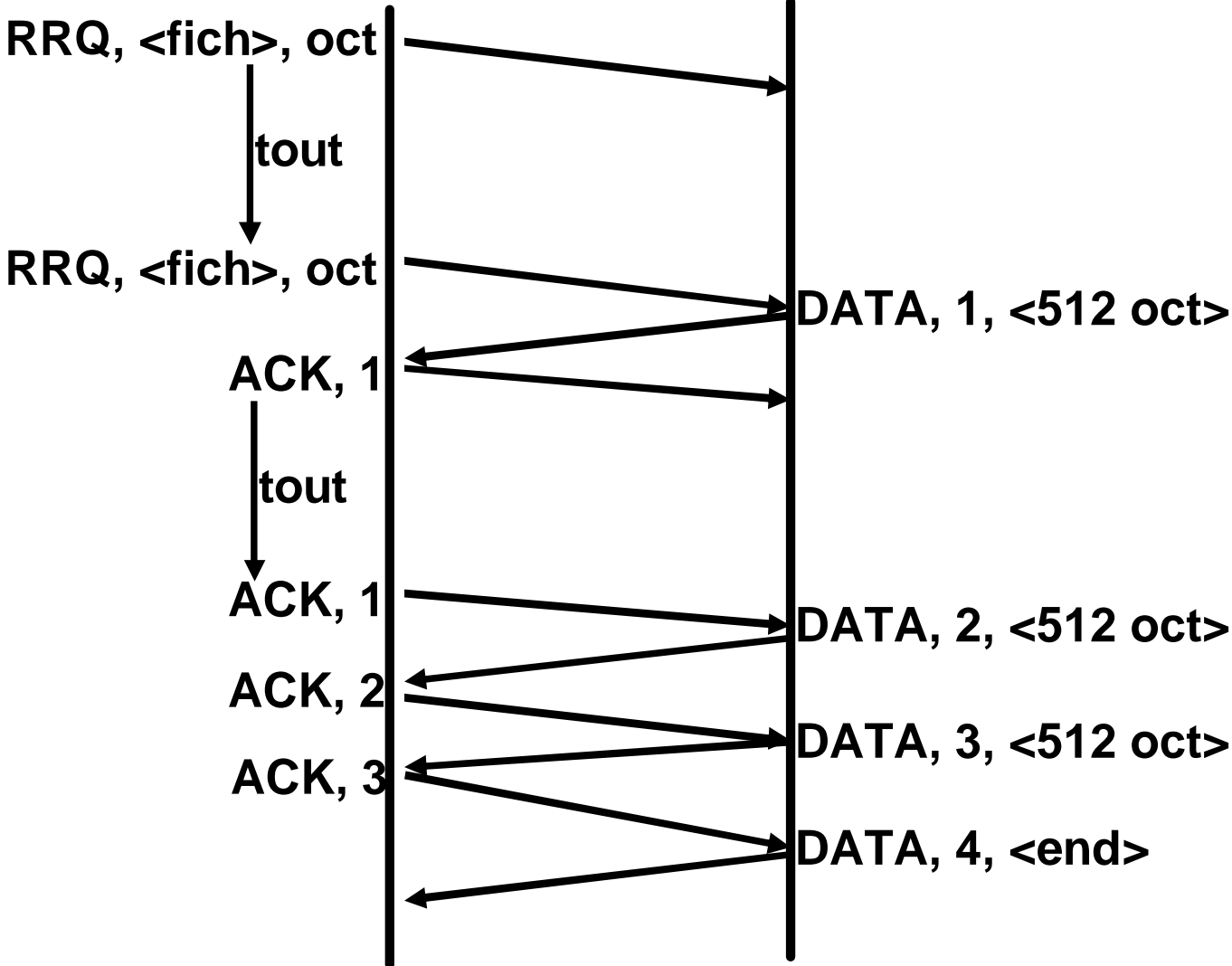
```
synchronized void stopTimers () // para los dos timers  
    { timer[tout] = 0; timer[close] = 0; }
```

---

# Timer II

```
public void run () {  
try {  
    while (true) {  
        Thread.sleep (1000);    // espera 1 segundo  
        synchronized (this) {  
            for (int i=0; i < timer.length; i++) { // todos los timers  
                if (timer[i] > 0) if (--timer[i] == 0) { // tic en timer[i]  
                    o.write((byte)i); o.flush();    // envía timeout  
                }  
            }  
        }  
    }  
} catch (IOException e) {System.out.println("Timer: " + e);  
} catch (InterruptedException. e) {System.out.println("Tim ...");}  
}
```

# Prueba: Lectura remota de fichero



# Programa de pruebas

```
class tftpServ extends Thread implements Runnable {
    FileInputStream i;
    tftpServ() {Thread ts = new Thread(this); ts.start();}

    int sendDATA () throws IOException {
        int len; byte b[] = new byte[516];
        len = i.read(b, 4, 512);
        SS(auxsock, DATA(b, (len+4)));
        this.seqnum++;
        return len;
    } // envía tramas de datos del fichero "prueba.txt"

    public void run() { S
        try {
            ..... // secuencias de pruebas
        } catch (SomeException e) {System.err.println(".....");
        }
    }
}
```

# Secuencia de prueba

```
seqnum = 1;           // inicializar numero de secuencia
r = new DatagramPacket(new byte[516], 516);
sock = new DatagramSocket(ServerPort); // socket para RRQ
auxsock = new DatagramSocket(); // socket para resto de com.
i = new FileInputStream("prueba.txt"); // tamaño: 3,5 bloques

sock.receive(r);      // espera primer RRQ
remoteTID=r.getPort(); a=r.getAddress();
sock.receive(r);      // espera siguiente retransmision de RRQ
l = sendData();       // envia datos
auxsock.receive(r);   // espera ACK ( no envia DATA)
auxsock.receive(r);   // espera siguiente retransmision de ACK
l = sendData();       // envia datos
auxsock.receive(r);   // espera ACK
l = sendData();       // envia datos
auxsock.receive(r);   // espera ACK
l = sendData();       // envia datos
```

# Ejercicio Protocolo-1

## ◆ Realizar un servidor de lectura de TFTP

- ▶ Debe ser un servidor concurrente
  - Espera peticiones que atiende en thread independiente
- ▶ La arquitectura de objetos debe reutilizar la máxima cantidad de código del cliente
  - Se recomienda reutilizar frame y Timer
    - El servidor debe ser un objeto derivado de la clase frame
  - Se recomienda modificar tftpClient y Line

## ◆ Recomendación para prueba servidor TFTP:

- ▶ proceso independiente (en otra ventana de ejecución)
- ▶ lee ficheros solamente del directorio de trabajo