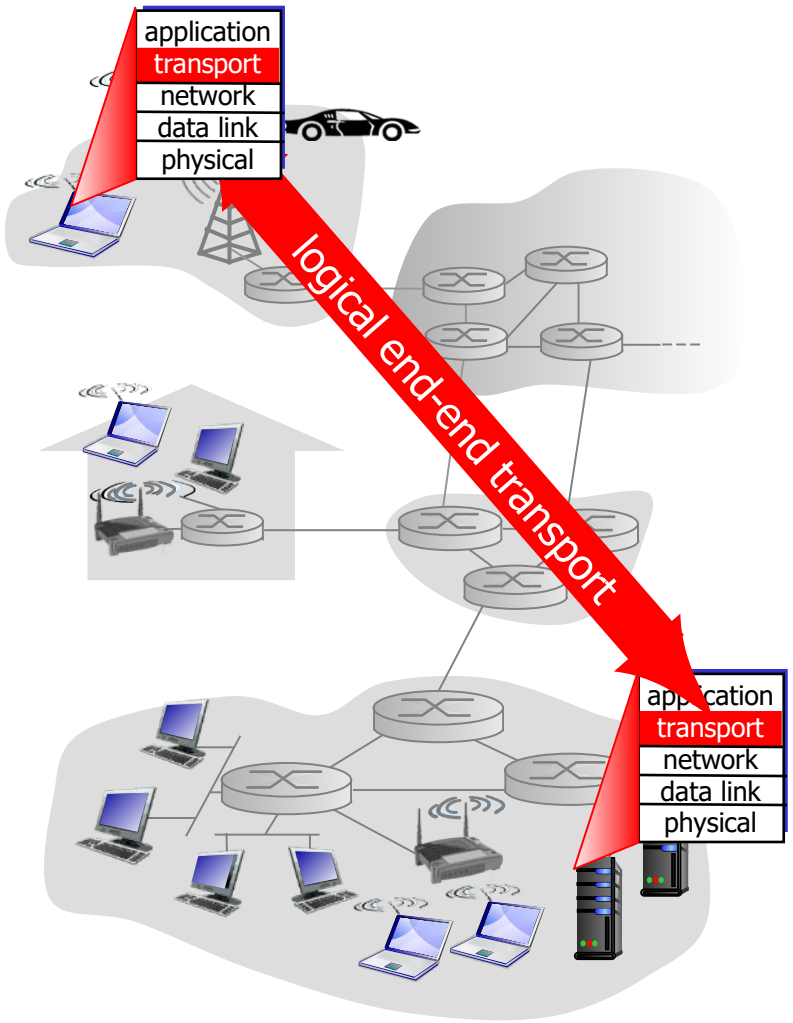
Capa de Transporte

Facultad de Ingeniería – Universidad de la República

2024

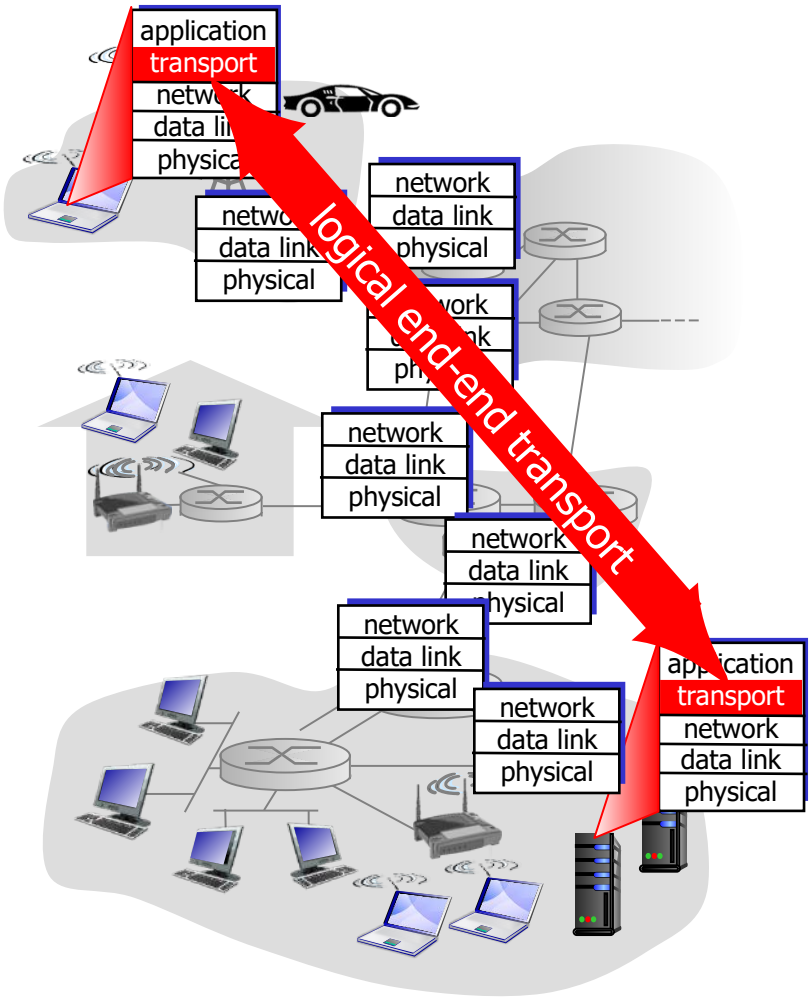
Capa de Transporte

- Proporcionar una comunicación lógica a la entidades de capa de Aplicación que residen en máquinas diferentes
- La capa de Transporte dialoga extremo a extremo (entre equipos finales):
  - Transmisor: “partir” los mensajes de capa de aplicación en segmentos, para luego entregarlos a la capa de red.
  - Receptor: re-ensamblar los segmentos en los mensajes “originales” y entregarlos a la capa de aplicación.
- Pueden existir mas de una alternativa para la elección de capa de Transporte.
- En Internet encontramos:
  - TCP (Transmission control Protocol)
  - UDP (User Datagram Protocol)



Capa de Transporte - ¿Porqué otra capa más?

- Capa de red:
  - identificador de host (ejemplo IP)
  - “encaminar” los paquetes del origen a destino
  - comunicación lógica entre equipos
  - utilizar los servicios de capa de enlace
- Capa de transporte:
  - Comunicación lógica entre procesos o aplicaciones en equipos diferentes.
  - Utiliza los servicios de capa de red.
  - Mejora o adapta los servicios de capa de red, a los ofrecidos a capa de transporte, por ejemplo servicio con garantía de entrega.
- Dimensiones de calidad de servicio en redes:
  - Garantía de entrega
  - Tasa de transferencia efectiva (throughput, descartes, priorización).
  - Requerimientos de tiempo (delay, jitter)
  - Seguridad (secreto, integridad, no repudio)



Capa de Transporte en Internet

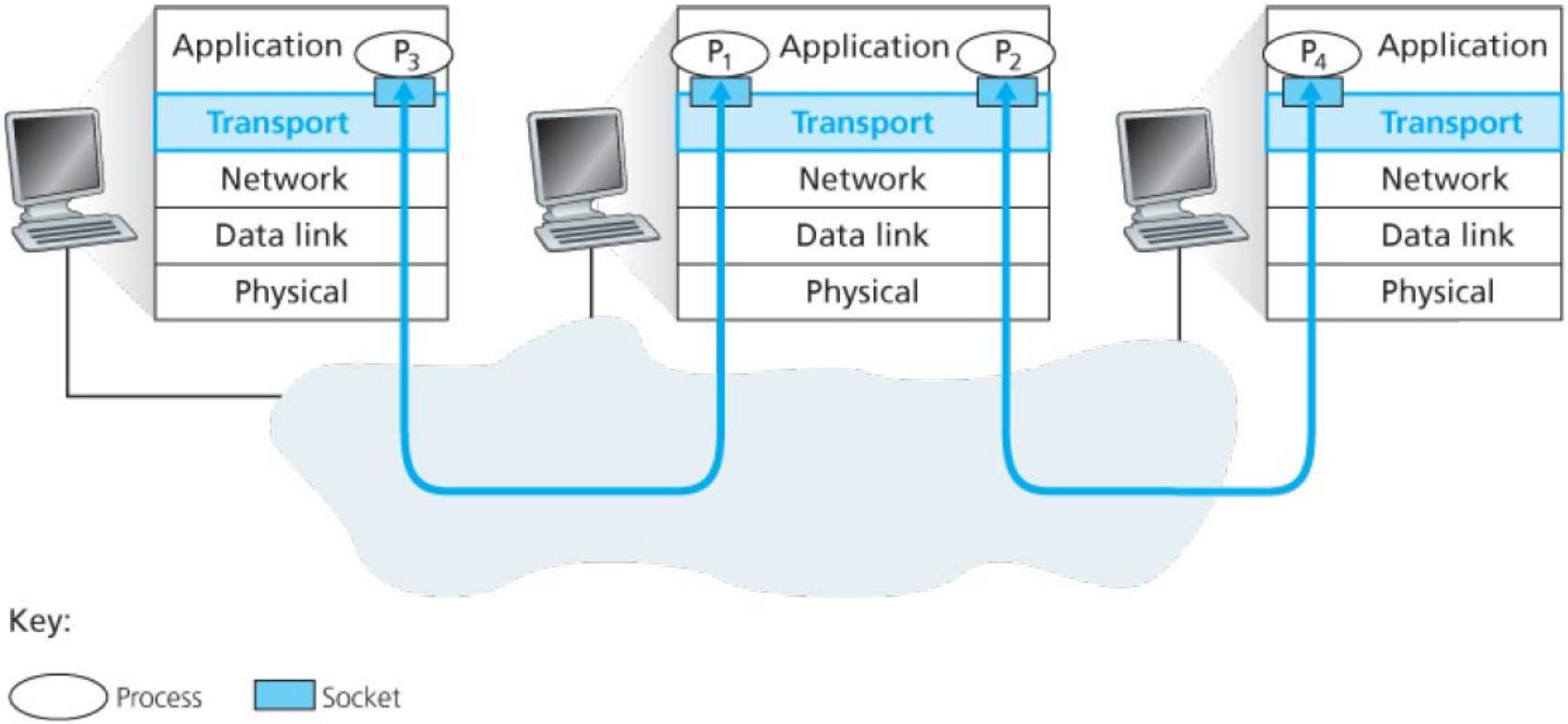
- Servicios que ofrece la capa de Transporte:
  - Orientado a Conexión y Confiable
  - No Orientado a Conexión y No confiable
- Confiable = Garantía de entrega (no significa seguro)
- Orientado a Conexión y Confiable (TCP):
  - Establecimiento y finalización de conexión
  - Entrega ordenada, sin pérdidas ni duplicados
  - Control de Flujo
  - Control de Congestión
- No orientado a conexión y No confiable (UDP):
  - No hay garantía de entrega ni de orden. Es una extensión de capa de red.
- Dimensiones que NO atiende la capa de Transporte en Internet
  - Garantías temporales (delay y jitter)
  - Garantías de ancho de banda y priorización

Funciones que debe realizar la capa de Transporte

- Para poder brindar servicios a la capa de aplicación la capa de transporte debe generalmente realizar las siguientes funciones:
  - Direccionamiento
  - Control de errores
  - Secuenciamiento
  - Control de flujo
  - Control de congestión
  - Multiplexado
  - Manejo de buffers y temporizadores

Capa de Transporte – Identificadores

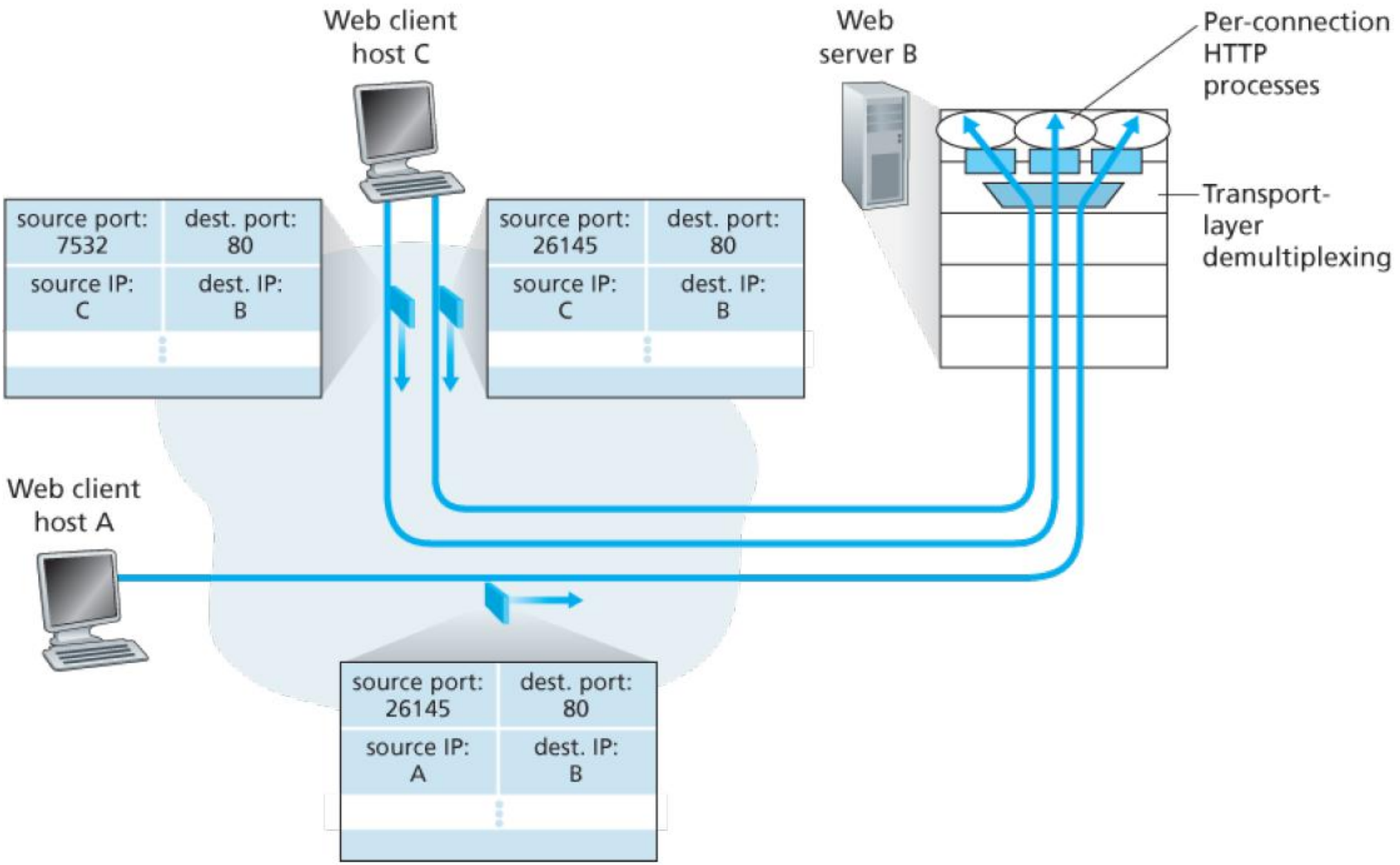
- ¿Porqué necesito un identificador en capa de Transporte?
  - Servidor: lo usual es disponer de varias aplicaciones (web y ssh).
  - Cliente: accediendo a diferentes servicios remotos
- El identificador de capa de Transporte me permite poder utilizar varias aplicaciones tanto en el cliente como en el servidor



Capa de Transporte

- TCP/UDP utilizan un identificador de 16 bits que se llama “puerto”
- Forma parte el encabezado que agregan las entidades de Transporte
- Socket: Una abstracción lógica de una conexión entre aplicaciones
    - Protocolo (TCP / UDP)
    - Identificación host origen (IP de origen)
    - Identificación host destino (IP de destino)
    - Identificación de Aplicación cliente (puerto de origen)
    - Identificación de Aplicación servidor (puerto destino)
- La capa de transporte revisa el puerto destino en el segmento, de acuerdo al puerto de destino “entrega” los datos a la aplicación correspondiente.
    - Ejemplo solicitudes http y https se las envía a la aplicación que está escuchando en esos puertos (un servidor web en este ejemplo)

Capa de Transporte



Capa de Transporte - Direccionamiento

- En general dos opciones: estática o dinámica.
- Puerto de origen: Por lo general la dirección IP de origen y el puerto de origen elije el Sistema Operativo
- Estática: "puertos bien conocidos" una aplicación servidor escucha en un puerto fijo, asignado de forma estática.
    - HTTP puerto 80
    - HTTPS puerto 443
    - SSH puerto 22
    - SMTP puerto 25
    - Etc
- Deberían registrarse, o de no hacerlo evitar colisionar.
- Dinámicos: consulta a un directorio
    - Portmaper de Unix: Puerto fijo de inicio, se negocia un nuevo puerto.
    - Nuevo Registro DNS tipo SRV
    - _xmpp-client._tcp.example.net. 86400 IN SRV 5 0 5222 server.example.net.
    - _xmpp-server._tcp.example.net. 86400 IN SRV 5 0 5269 server.example.net.

Capa de Transporte – Errores en los datos

- Un error en los datos es la alternación de los bits (“1” por “0” o “0” por “1”) en los datagramas/segmentos intercambiados.
- Para poder identificar que hubo errores necesito agregar información adicional. C.I. 1.234.567 → 1.234.567-2 (función de todos los dígitos)
- Para el manejo de errores en la transmisión, se utilizan dos estrategias:
- Detección de errores
  - Mediante algún algoritmo se detecta si los datos recibidos son los esperados
  - Si no lo son, se descartan. En caso necesario se deberá pedir retransmisión
- Corrección de errores
  - Al transmitir los datos, se les agrega suficiente redundancia para poder corregir errores
  - Puede evitar retransmisiones (a cambio de mayor redundancia, encabezados más grandes)
- En general en capa de Transporte solo se realiza detección de errores
- En capa de enlace se suelen utilizar algoritmos de detección mas complejos, en medios físicos inalámbricos como la red celular es posible utilizar corrección de errores.

Capa de Transporte – Detección de Errores

- En general se recurre a las siguientes técnicas en la capa de transporte:
  - Suma de comprobación o checksum
  - Hay otras técnicas pero no suelen ser usadas en esta capa (se verán algunas durante el estudio de la capa de enlace).
- Suma de Comprobación:
  - Se divide el mensaje en palabras de largo fijo (típicamente 2 o 4 bytes), se rellena con “ceros”.
  - Se realiza una suma de todas las palabras.
  - Se envía el resultado junto con el mensaje.
  - En el receptor, se realiza la misma operación y se verifica el resultado.
  - En caso de diferencias, se declara que hubo error y se descarta.

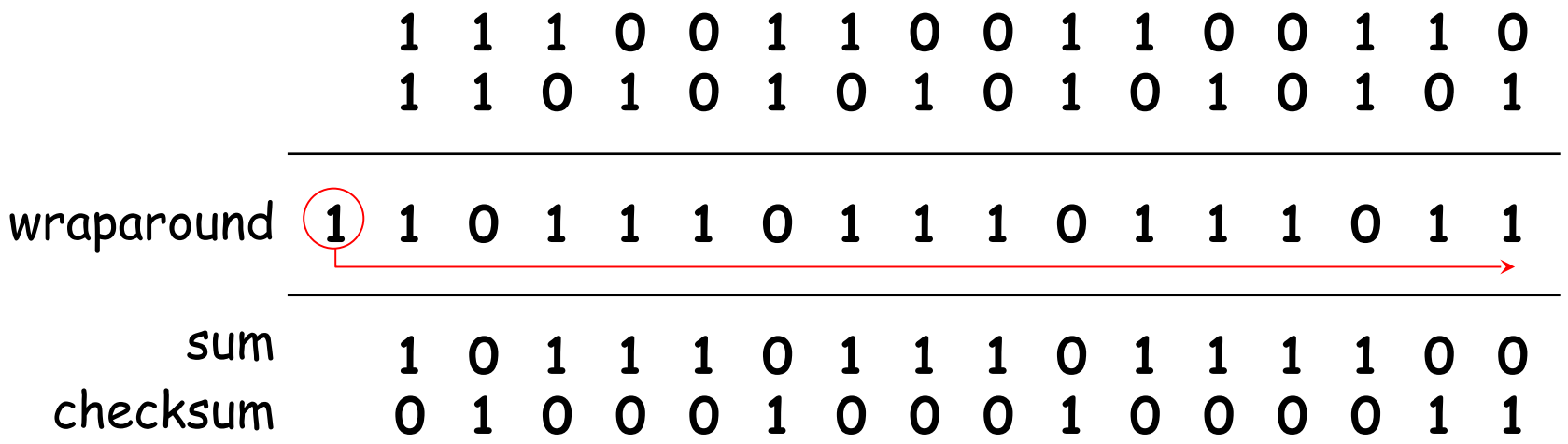
Capa de Transporte – TCP/UDP Checksum

- TCP/UDP: Simple suma de comprobación de 16 bits
- El mensaje se divide en palabras de 16 bits, y se hace la suma en complemento a 1
- Se hace el complemento a 1 (invertir 1s y 0s)
- Se hace sobre los datos, el encabezado capa 4, y un "pseudo encabezado" con información de capa 3

Capa de Transporte – TCP/UDP Checksum

- TCP/UDP: Simple suma de comprobación de 16 bits
- El mensaje se divide en palabras de 16 bits, y se hace la suma en complemento a 1
- Se hace el complemento a 1 (invertir 1s y 0s)
- Se hace sobre los datos, el encabezado capa 4, y un "pseudo encabezado" con información de capa 3

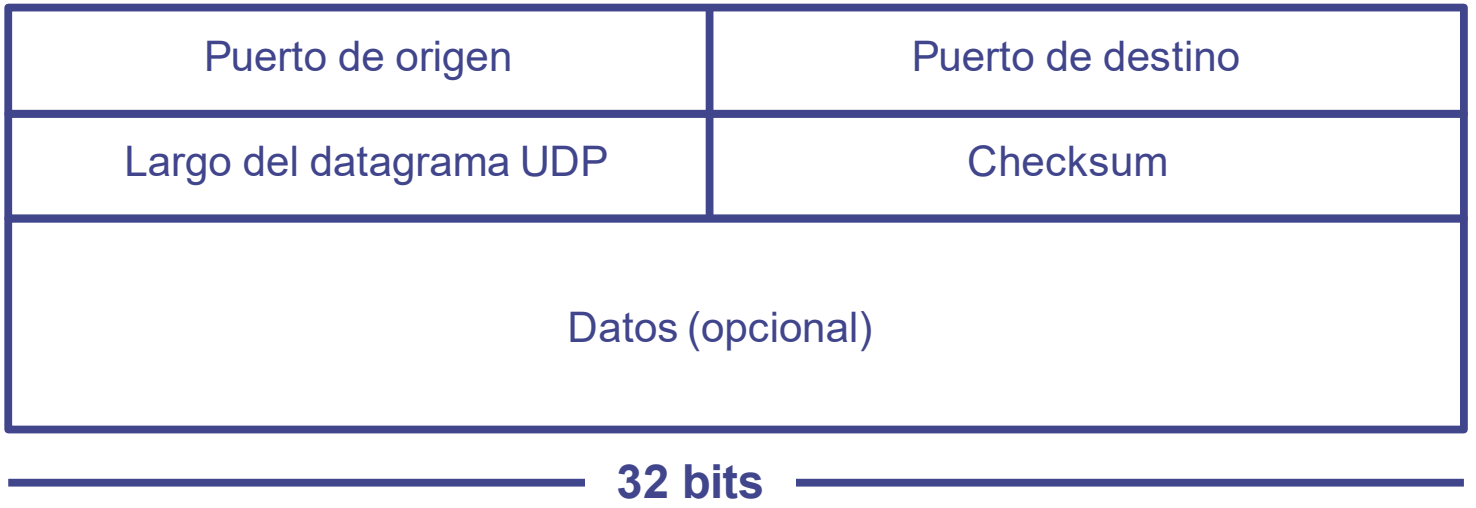
  - Ejemplo: suma de dos enteros de 16-bit



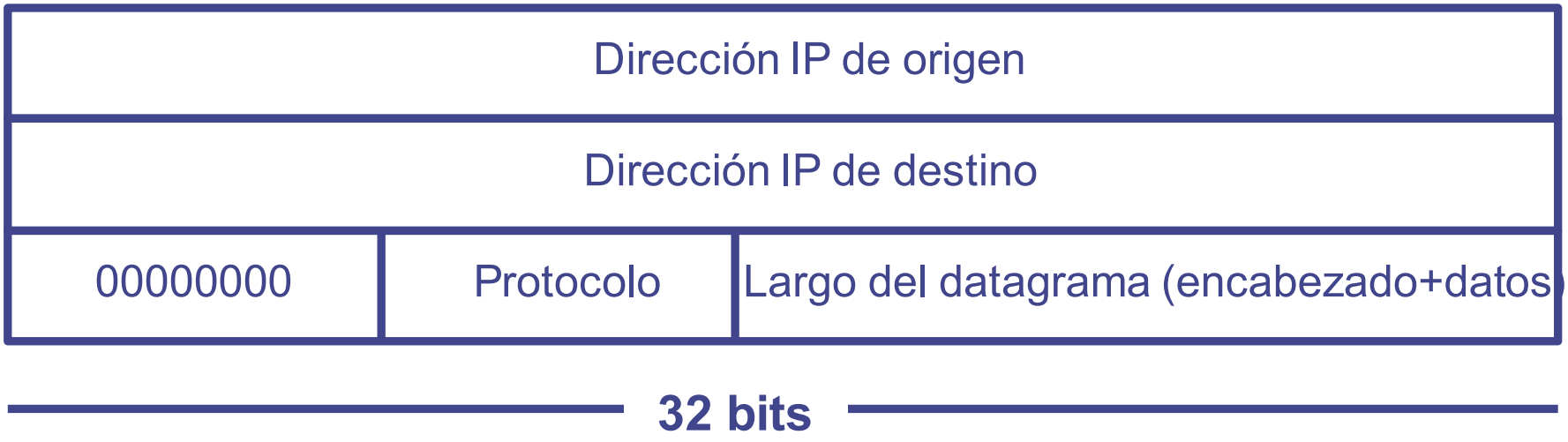
Capa de Transporte – UDP

- UDP:
  - Servicio no orientado a Conexión
  - No confiable (no hay garantía de entrega)
  - Identificar la aplicación de origen y destino.
  - “Detectar” errores
  - No tiene control de flujo (ajustarse a la capacidad del receptor)
  - No tiene control de congestión (ajustarse a la capacidad de la “red”)

Encabezado UDP:



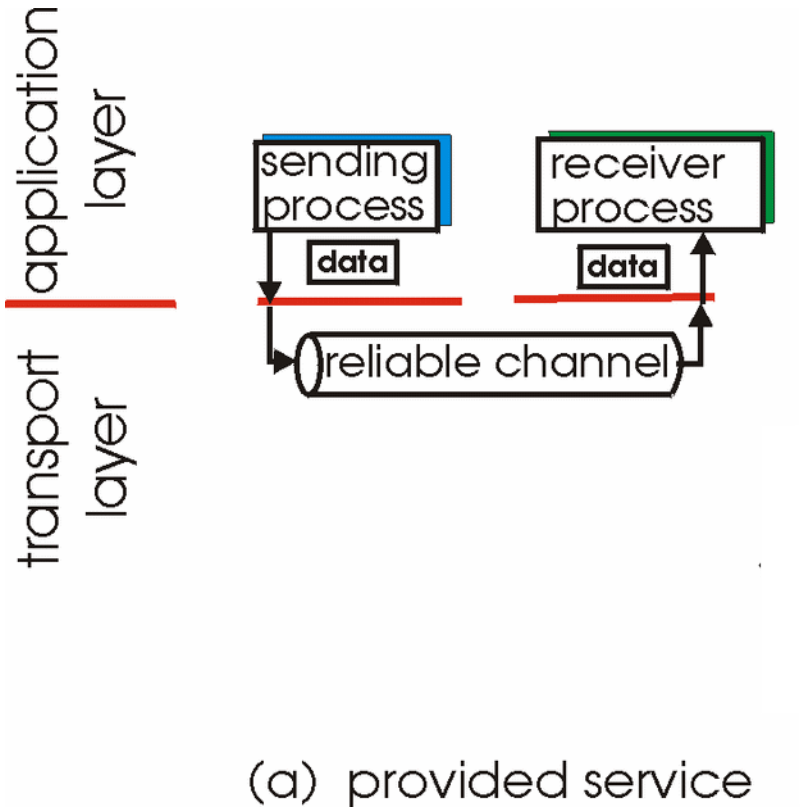
Pseudo Encabezado UDP (para checksum):



- Violación a la independencia de capas
  - Se debe recalcular en escenarios con NAT (lo veremos después)
  - Se modifica al cambiar de versión de IPv4 a IPv6 !!

Principios de Transferencia de Datos “Confiable”

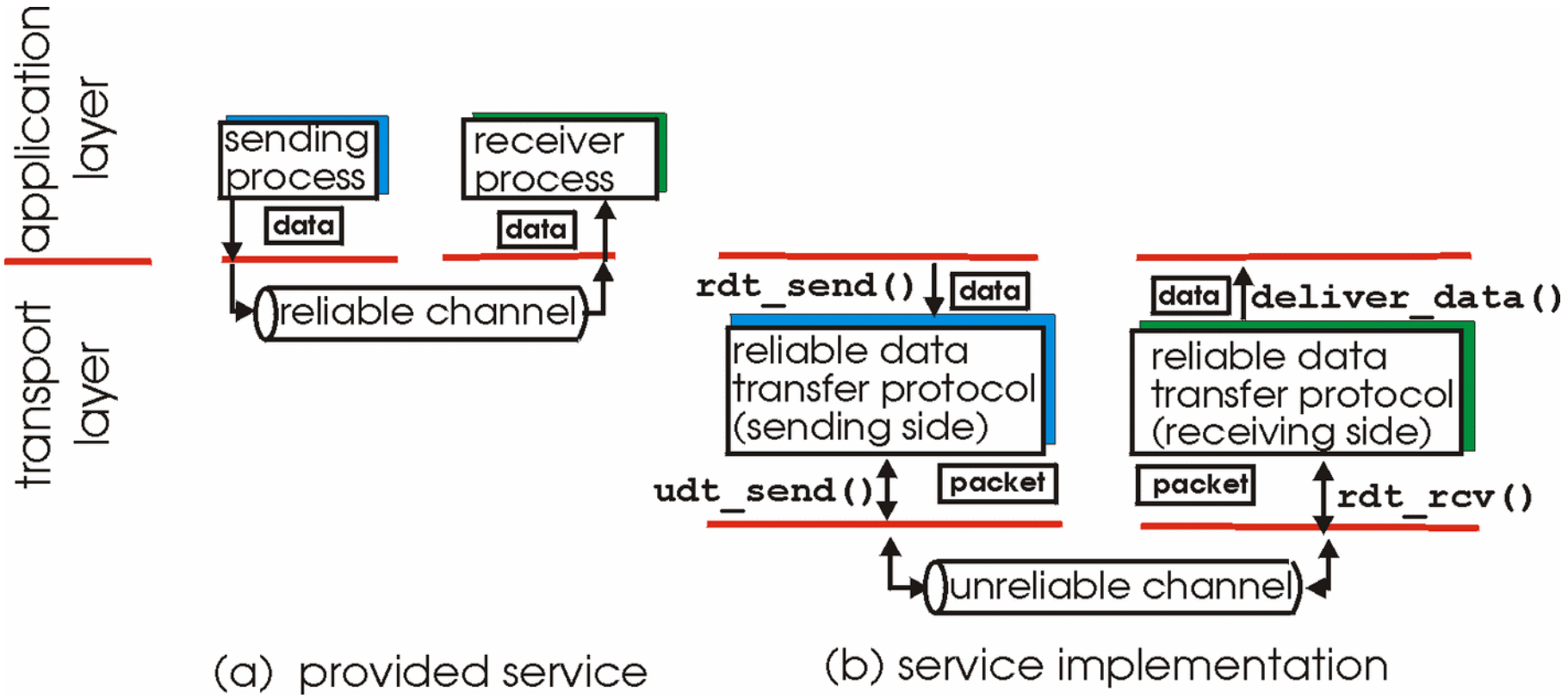
- Un canal “confiable”: aquel que permite la transferencia de datos “0” y “1” sin alterarlos.
- La abstracción del servicio “canal confiable” que ofrece la capa de transporte a la capa de aplicación debe de ser independiente de los servicios de las capas inferiores.
- Puede no ser tan simple: TCP utiliza los servicios de IP (best effort), ocurren pérdidas y los paquetes pueden llegar desordenados.



(a) provided service

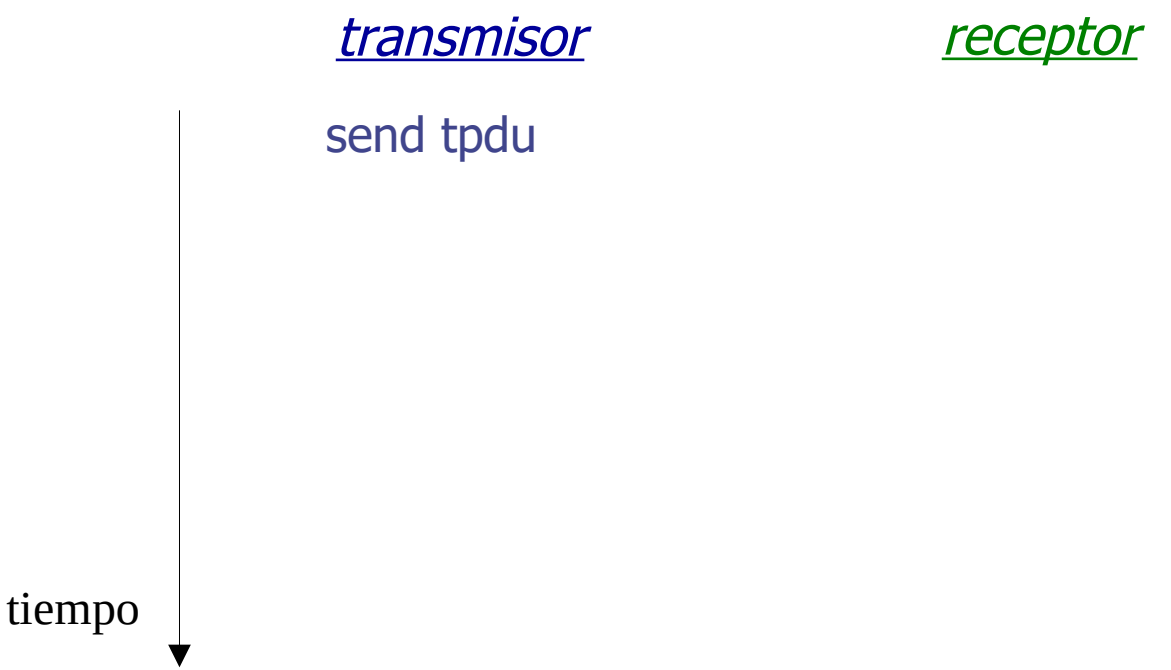
Principios de Transferencia de Datos “Confiable”

- Un canal “confiable”: aquel que permite la transferencia de datos “0” y “1” sin alterarlos.
- La abstracción del servicio “canal confiable” que ofrece la capa de transporte a la capa de aplicación debe de ser independiente de los servicios de las capas inferiores.
- Puede no ser tan simple: TCP utiliza los servicios de IP (best effort), ocurren pérdidas y los paquetes pueden llegar desordenados.



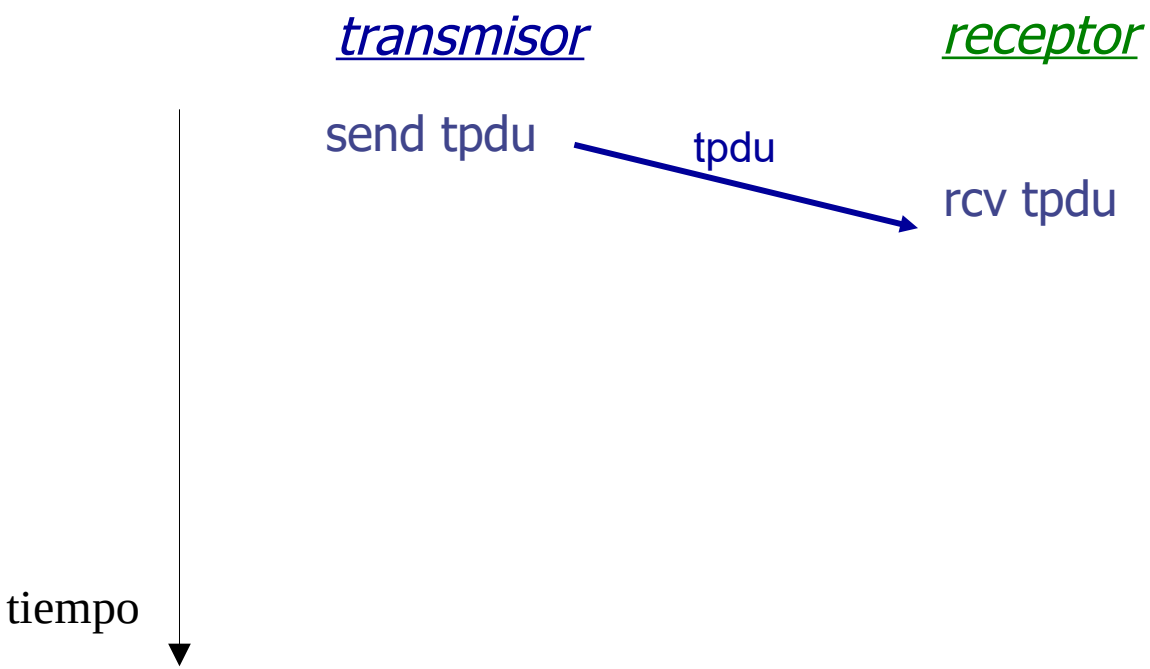
Construcción de protocolo - RDT Confiable

- RDT = Reliable Data Transfer
- Construcción Evolutiva de un protocolo comenzando de un modelo simple y luego retirando las simplificaciones.
- Usaremos diagramas de intercambio de segmentos (TPDU, Transport Protocol Data Unit)
- RDT 1.0 canal ideal sin ruido
- El transmisor tiene varias tpdu para enviar y las envía en secuencia



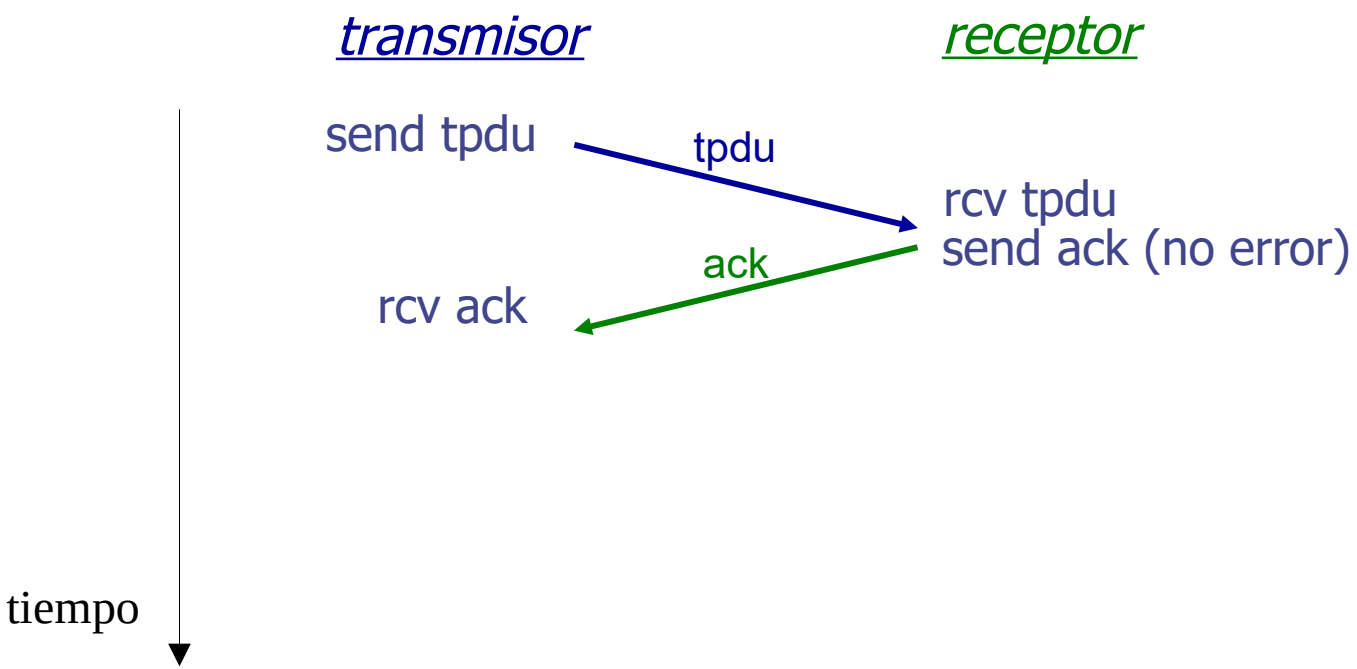
Construcción de protocolo - RDT Confiable

- RDT = Reliable Data Transfer
- Construcción Evolutiva de un protocolo comenzando de un modelo simple y luego retirando las simplificaciones.
- Usaremos diagramas de intercambio de segmentos (TPDU, Transport Protocol Data Unit)
- RDT 1.0 canal ideal sin ruido
- El transmisor tiene varias tpdu para enviar y las envía en secuencia



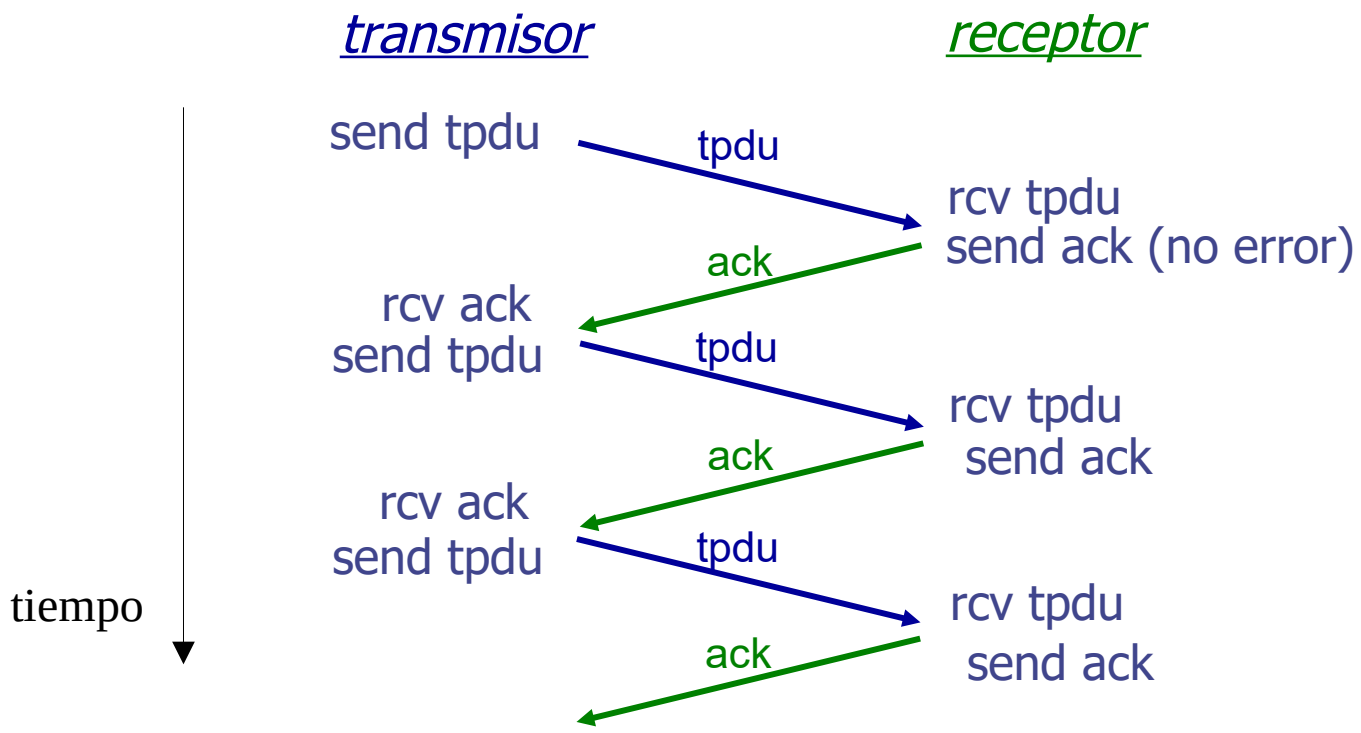
Construcción de protocolo - RDT Confiable

- RDT = Reliable Data Transfer
- Construcción Evolutiva de un protocolo comenzando de un modelo simple y luego retirando las simplificaciones.
- Usaremos diagramas de intercambio de segmentos (TPDU, Transport Protocol Data Unit)
- RDT 1.0 canal ideal sin ruido
- El transmisor tiene varias tpdu para enviar y las envía en secuencia



Construcción de protocolo - RDT Confiable

- RDT = Reliable Data Transfer
- Construcción Evolutiva de un protocolo comenzando de un modelo simple y luego retirando las simplificaciones.
- Usaremos diagramas de intercambio de segmentos (TPDU, Transport Protocol Data Unit)
- RDT 1.0 canal ideal sin ruido
- El transmisor tiene varias tpdu para enviar y las envía en secuencia



Protocolo Simple en una red con ruido - RDT 2.0

- Un Transmisor y un Receptor, solo envía datos de aplicación el transmisor.
- Errores en Capas Inferiores: Los segmentos se pueden corromper o perder totalmente.
- **Receptor:** Se agrega código detector de errores (TCP Checksum)
- **¿Transmisor?**
  - **ACK (acknowledgement):** sólo se reconocen (ACK) los segmentos que llegan bien (el receptor envía una segmento indicando que se recibió correctamente).
  - **Reloj:** Necesito de Temporizadores para recuperación de errores si no llega el reconocimiento.
- **Transport Protocol Data Unit (TPDU):** De forma genérica, a los datos de aplicación con la información de control (encabezados).

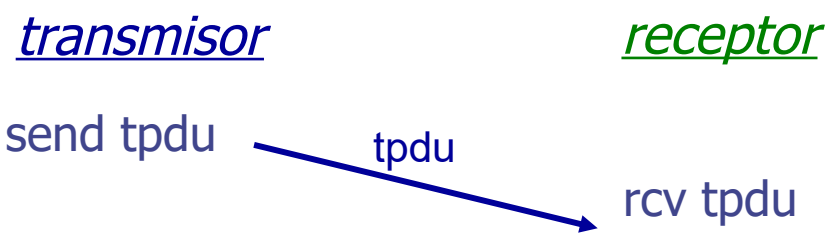
  En TCP los llamamos segmentos
- **Simplificación: TPDU de largo fijo**

Protocolo Simple en una red con ruido - RDT 2.0

(a) Sin pérdidas

(b) Con pérdidas

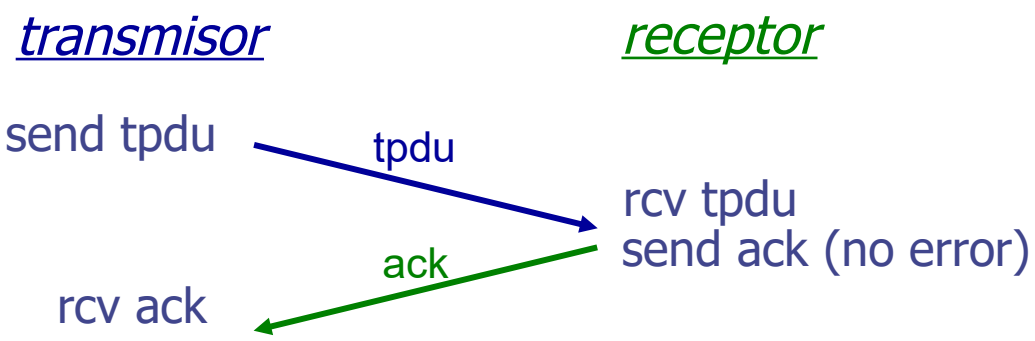
Protocolo Simple en una red con ruido - RDT 2.0



(a) Sin pérdidas

(b) Con pérdidas

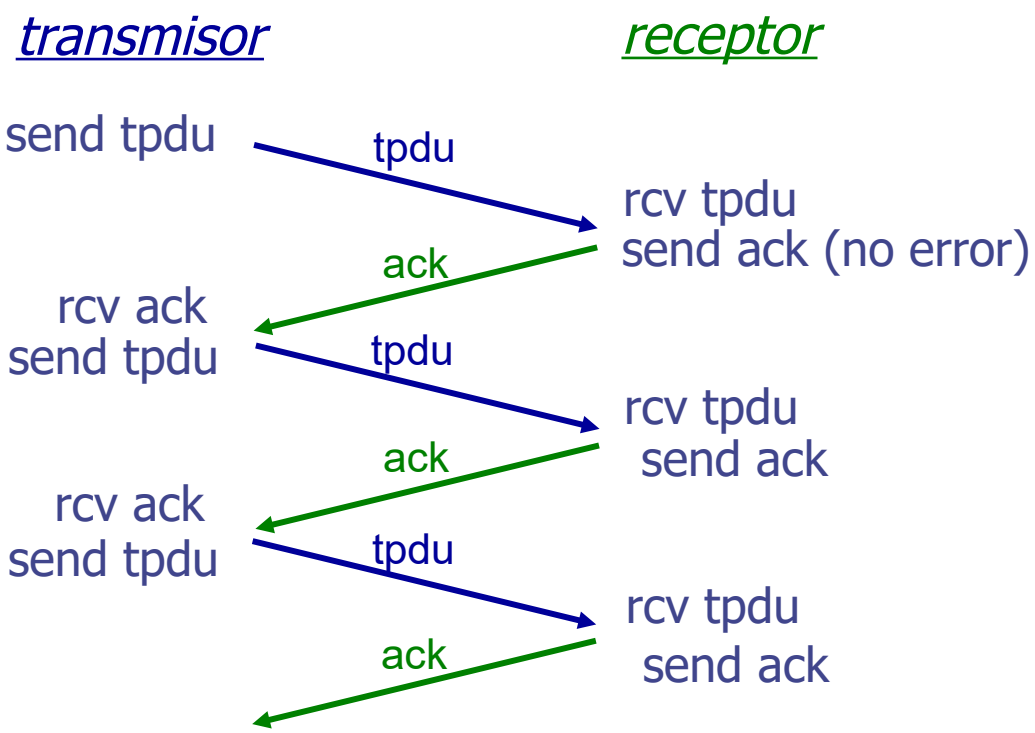
Protocolo Simple en una red con ruido - RDT 2.0



(a) Sin pérdidas

(b) Con pérdidas

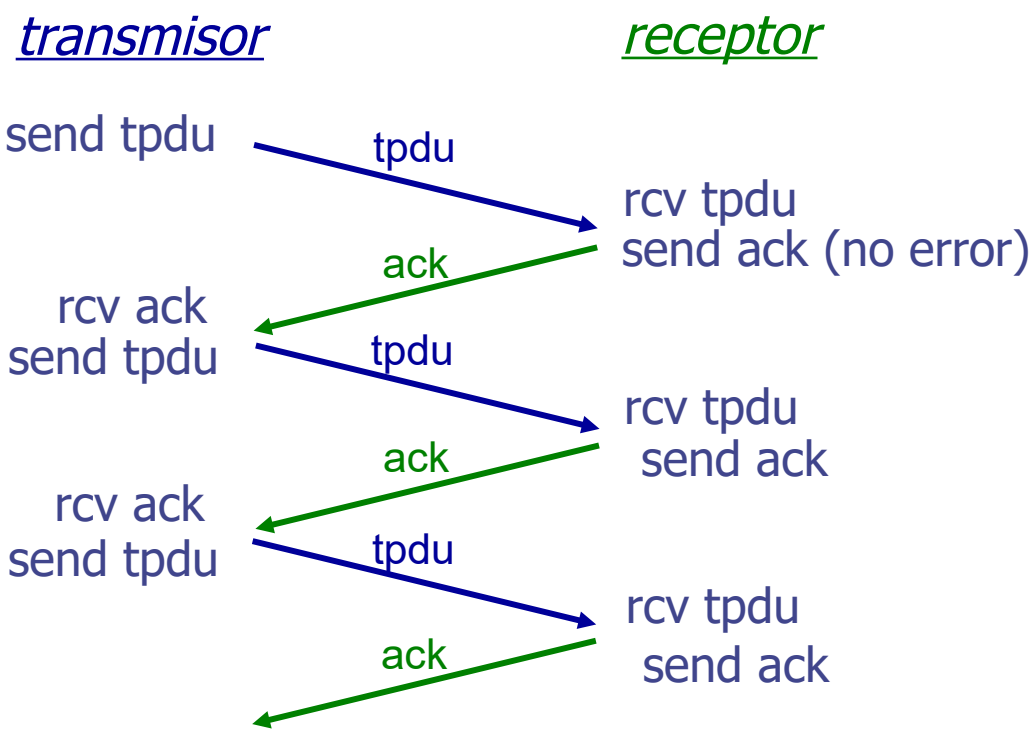
Protocolo Simple en una red con ruido - RDT 2.0



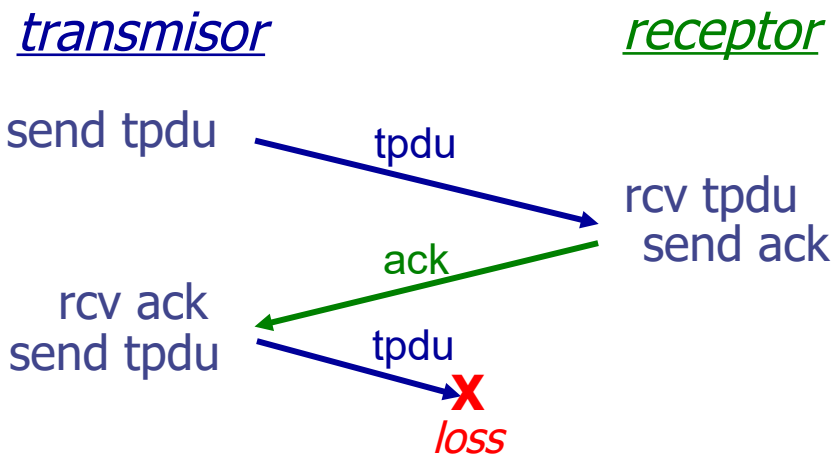
(a) Sin pérdidas

(b) Con pérdidas

Protocolo Simple en una red con ruido - RDT 2.0

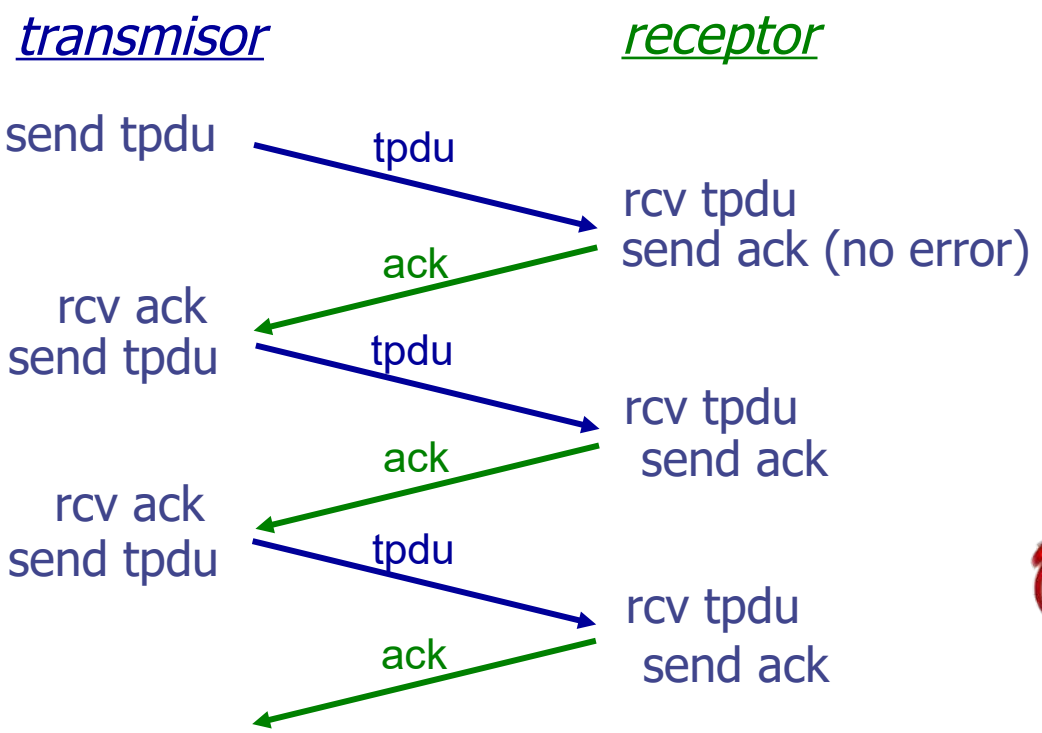


(a) Sin pérdidas

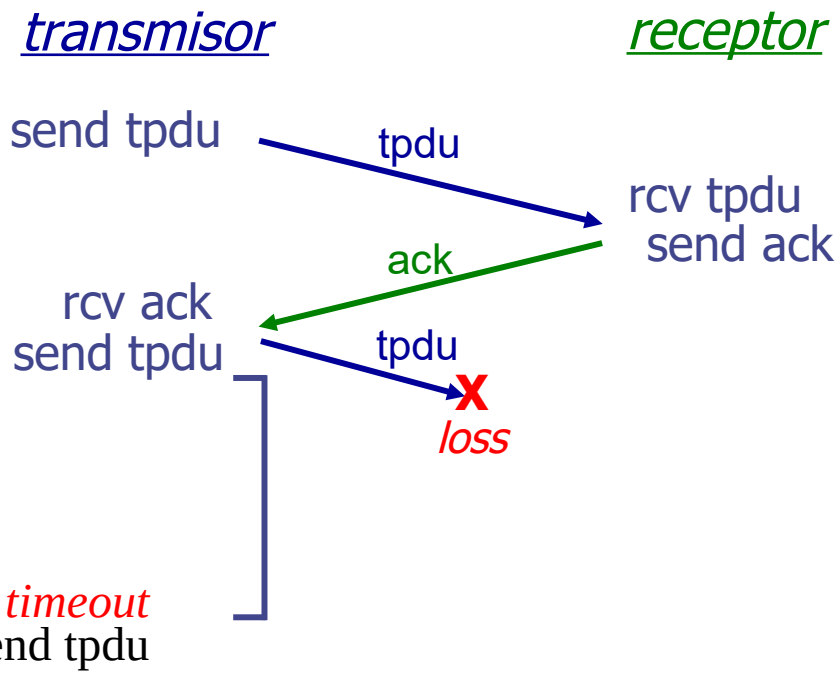


(b) Con pérdidas

Protocolo Simple en una red con ruido - RDT 2.0

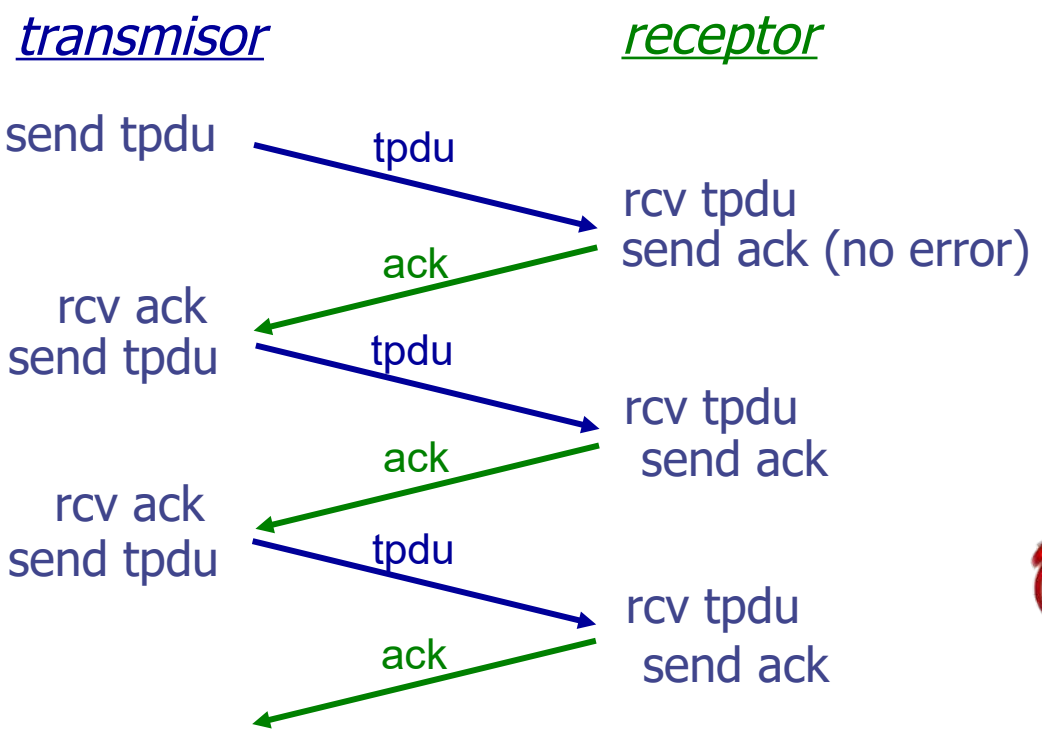


(a) Sin pérdidas

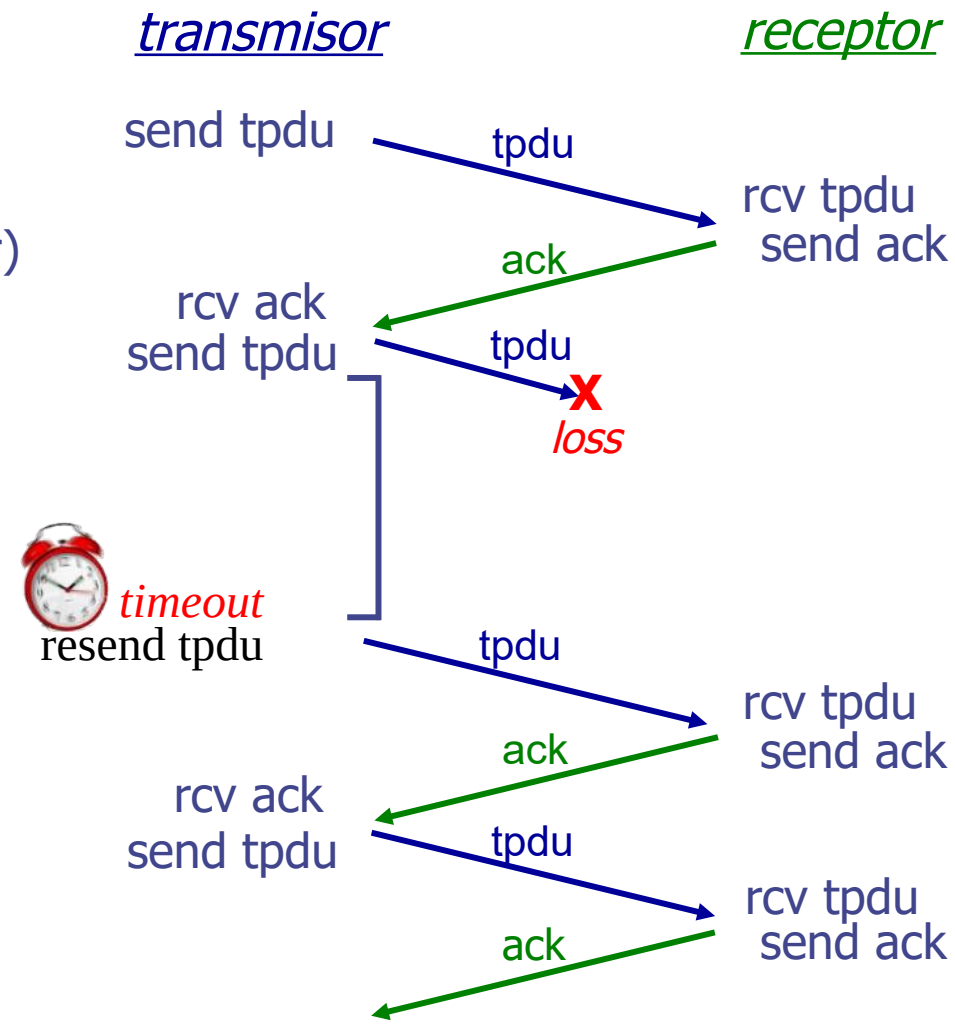


(b) Con pérdidas

Protocolo Simple en una red con ruido - RDT 2.0



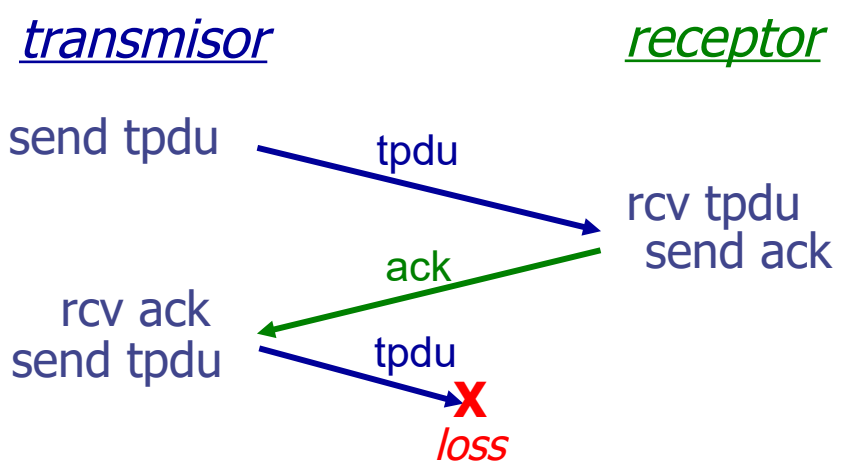
(a) Sin pérdidas



(b) Con pérdidas

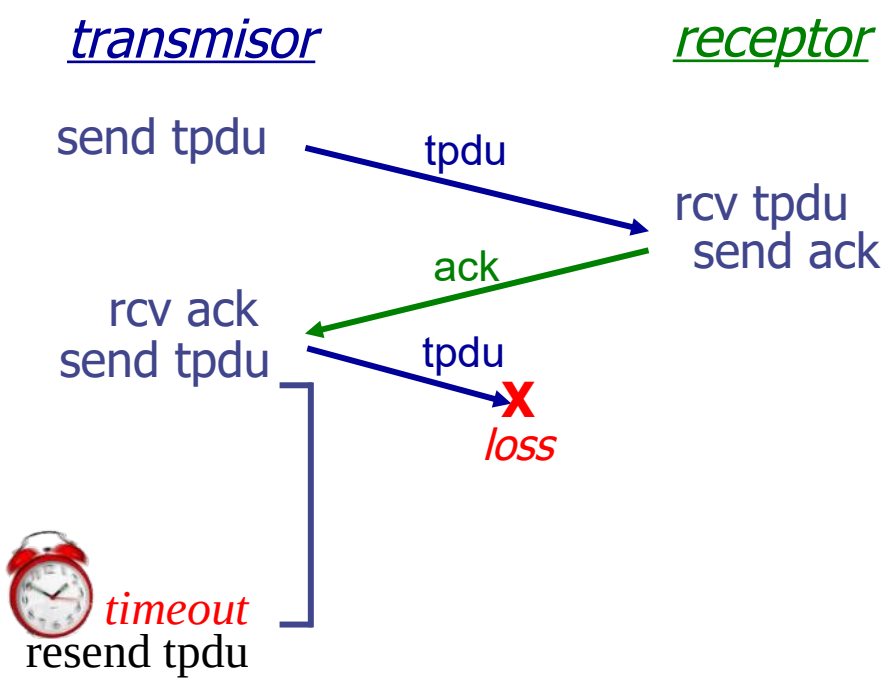
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?

¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?



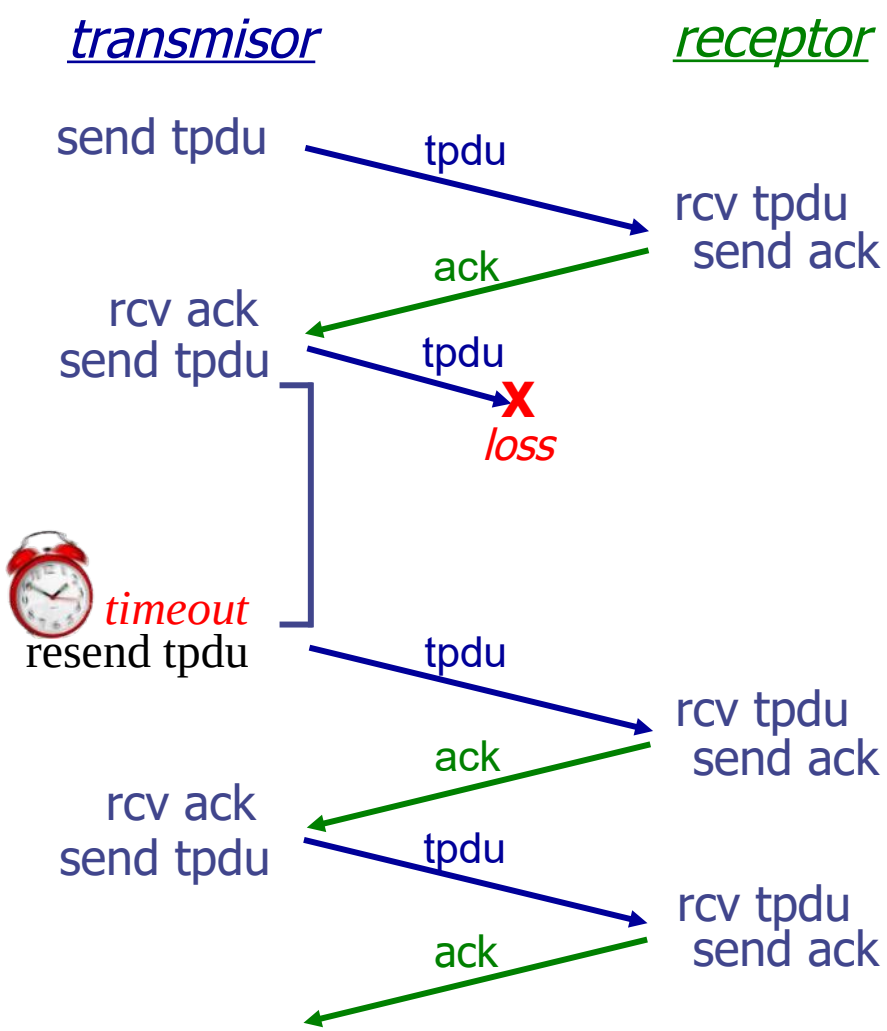
transmisor
receptor

¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?

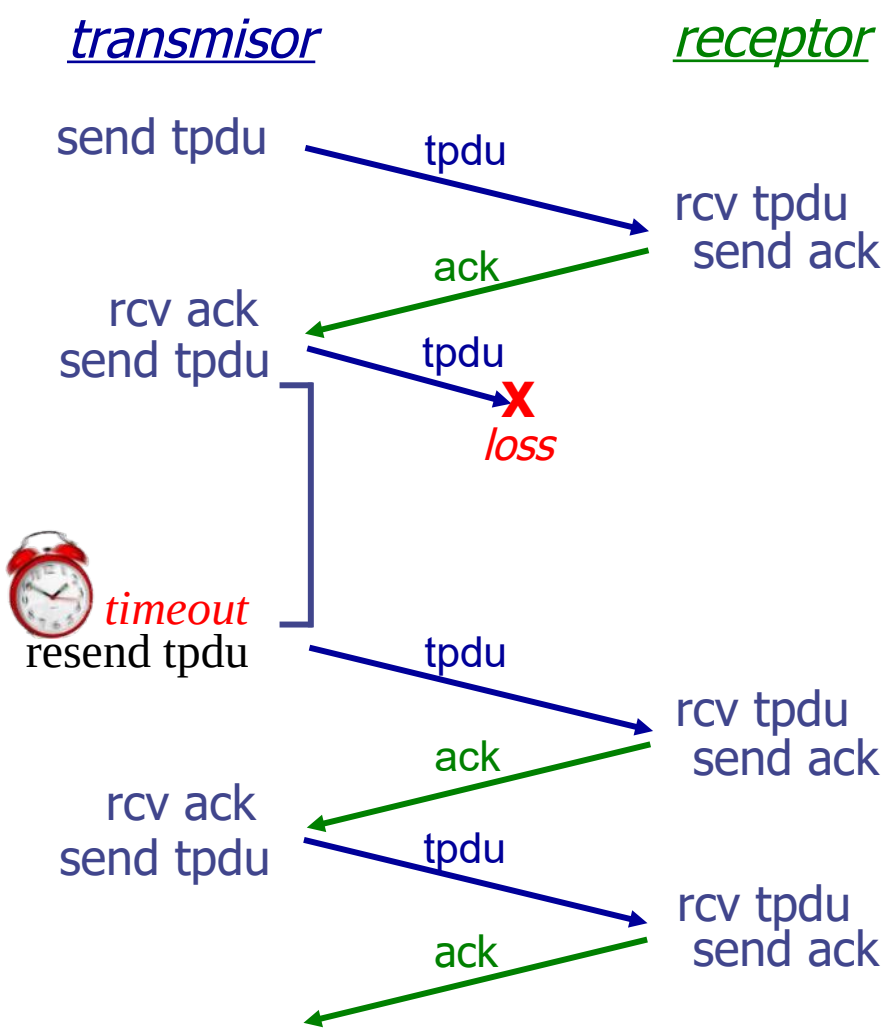


transmisor
receptor

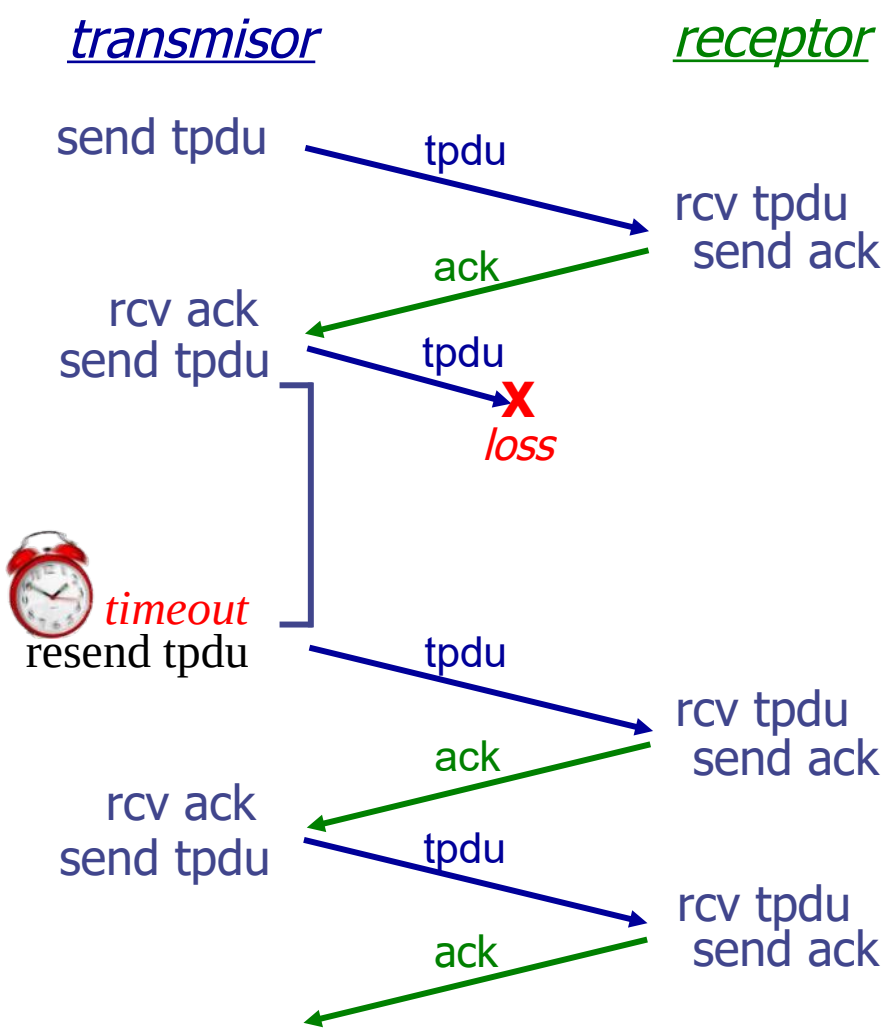
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?

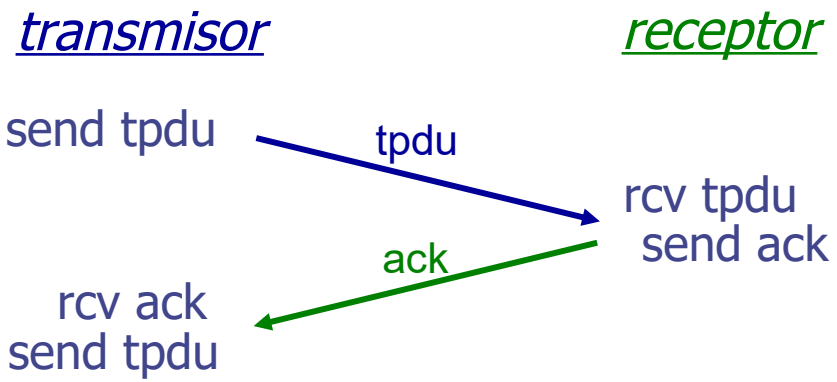


¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?

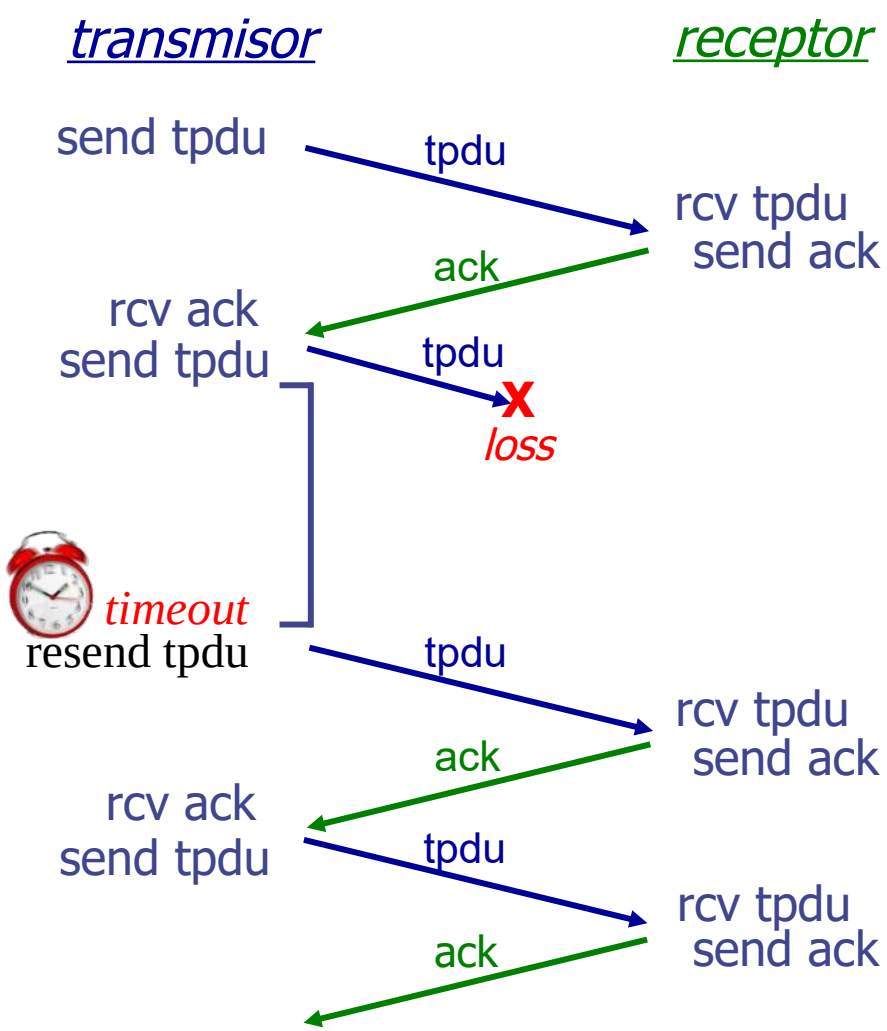


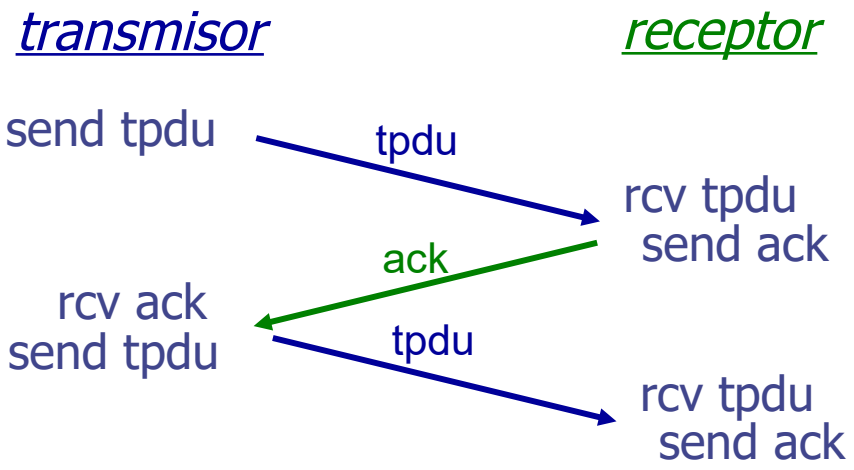
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?



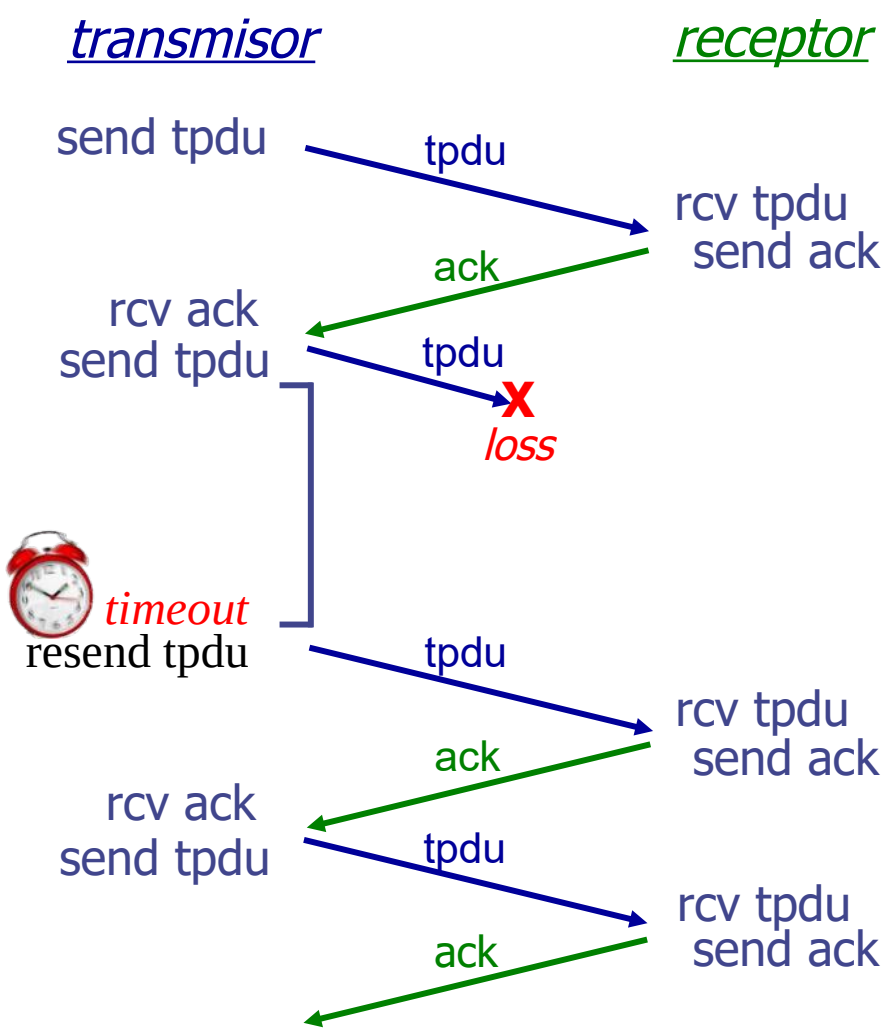


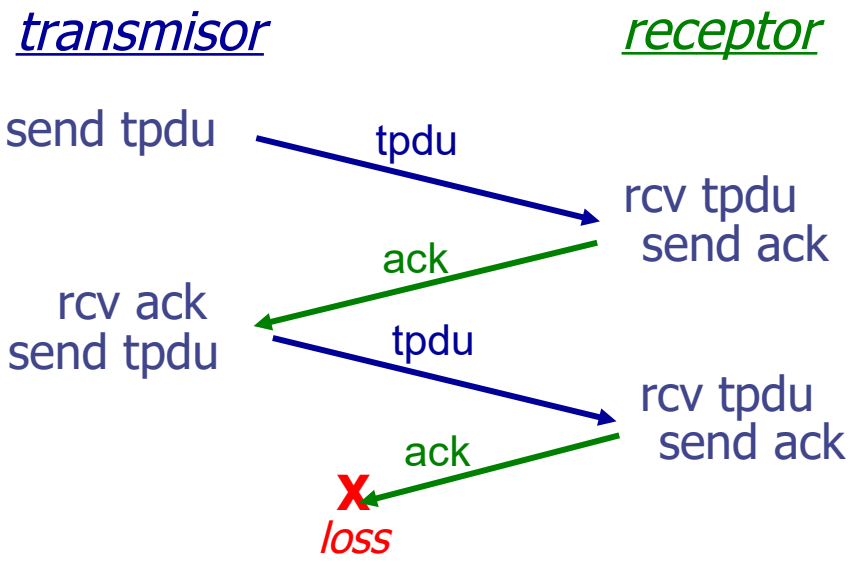
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?



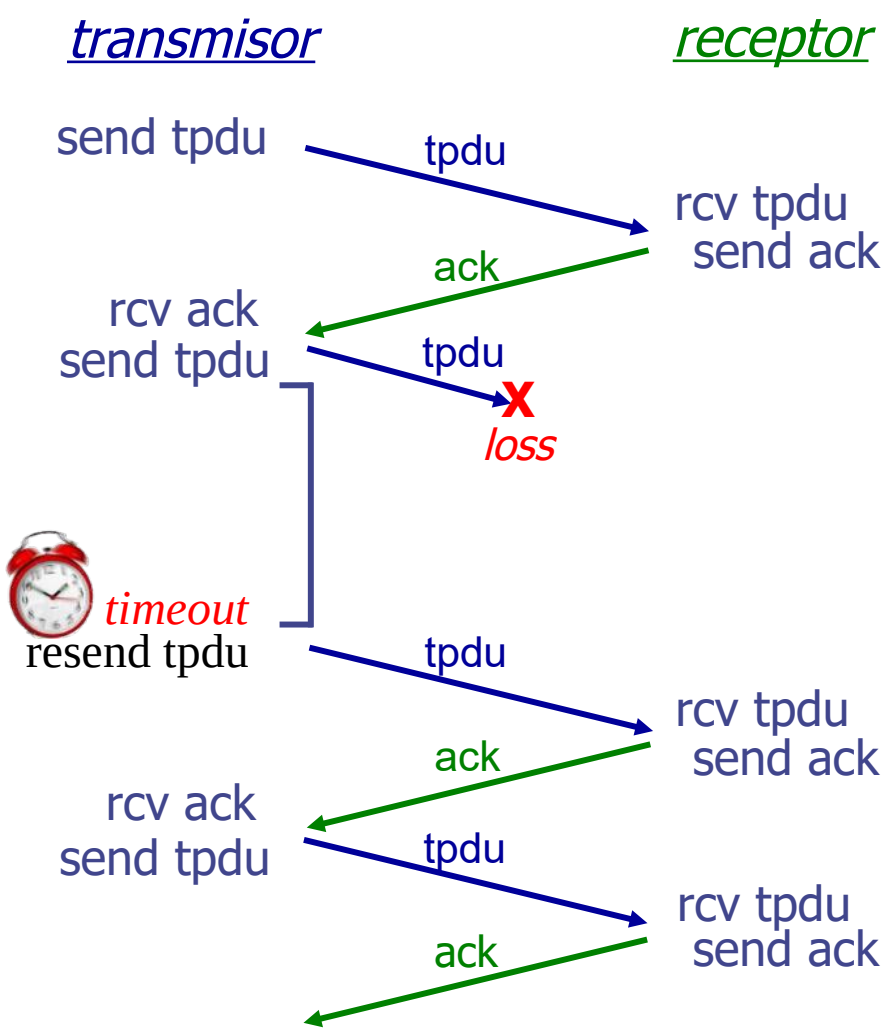


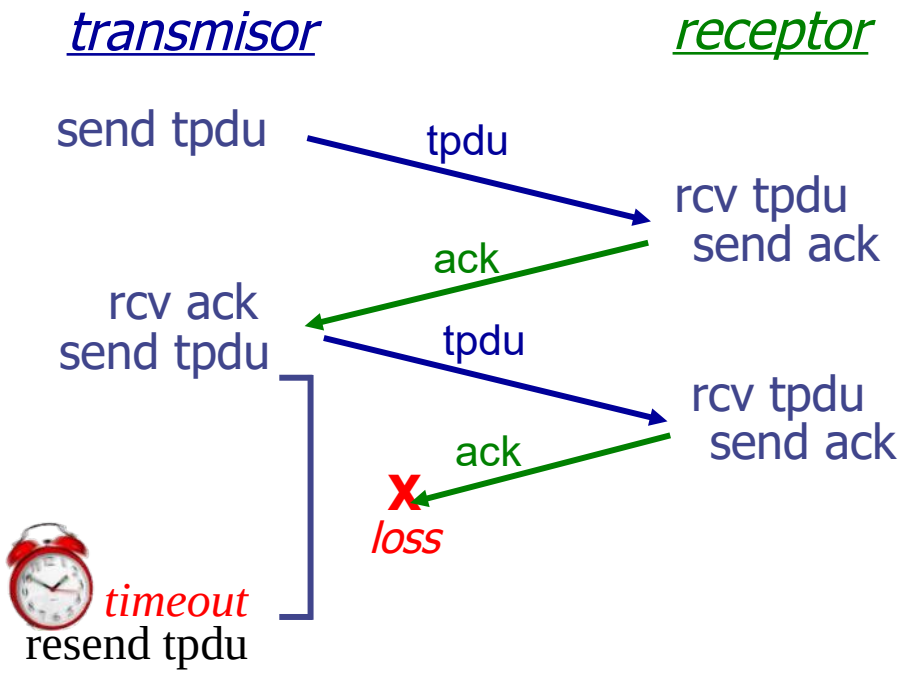
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?



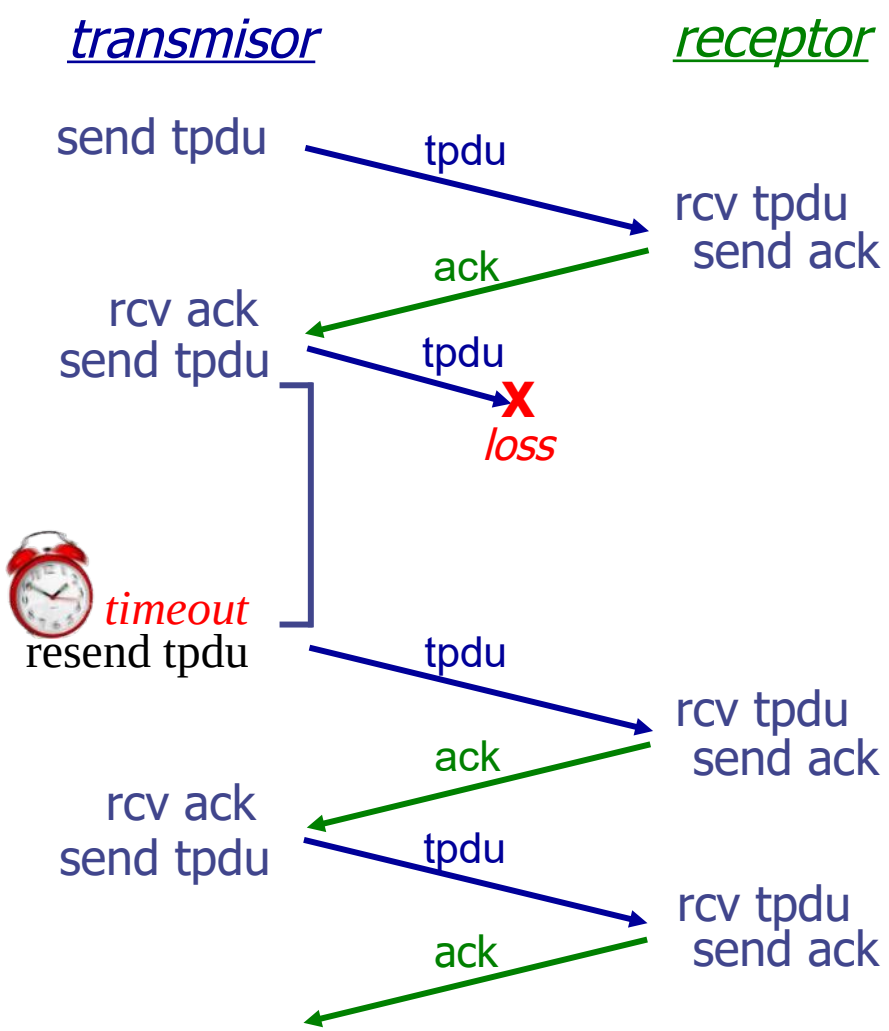


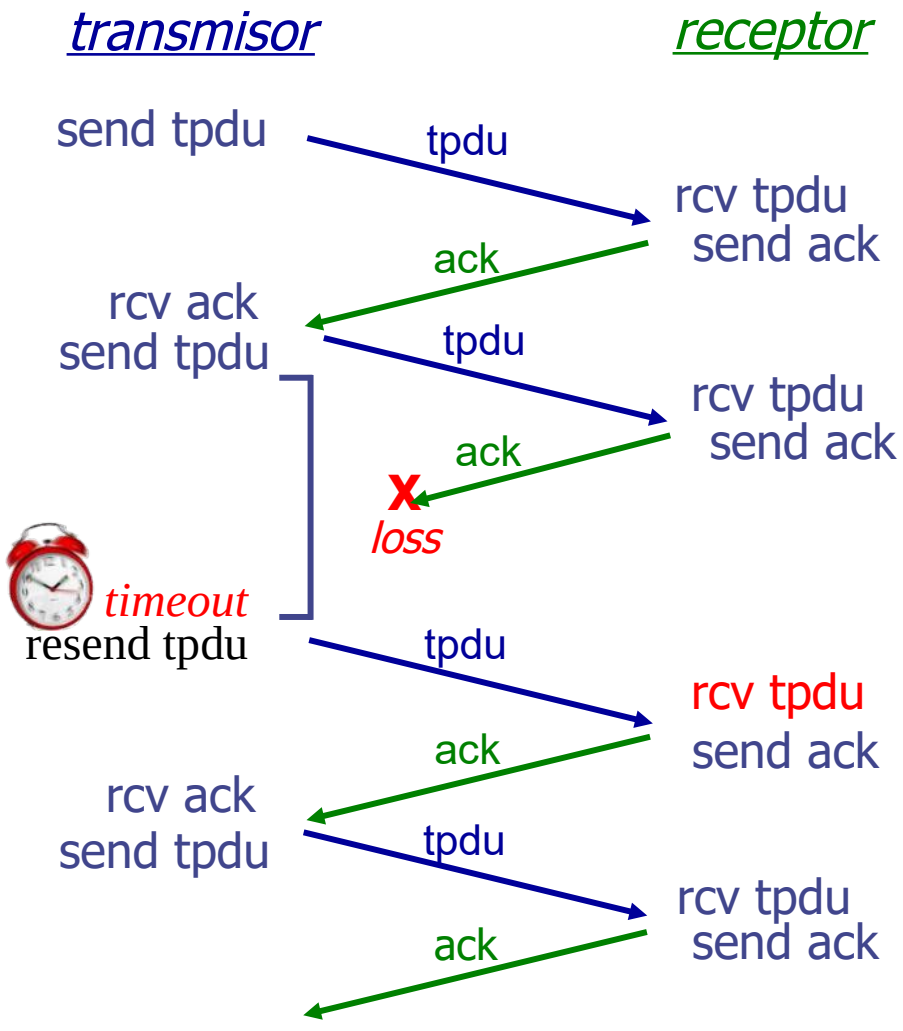
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?



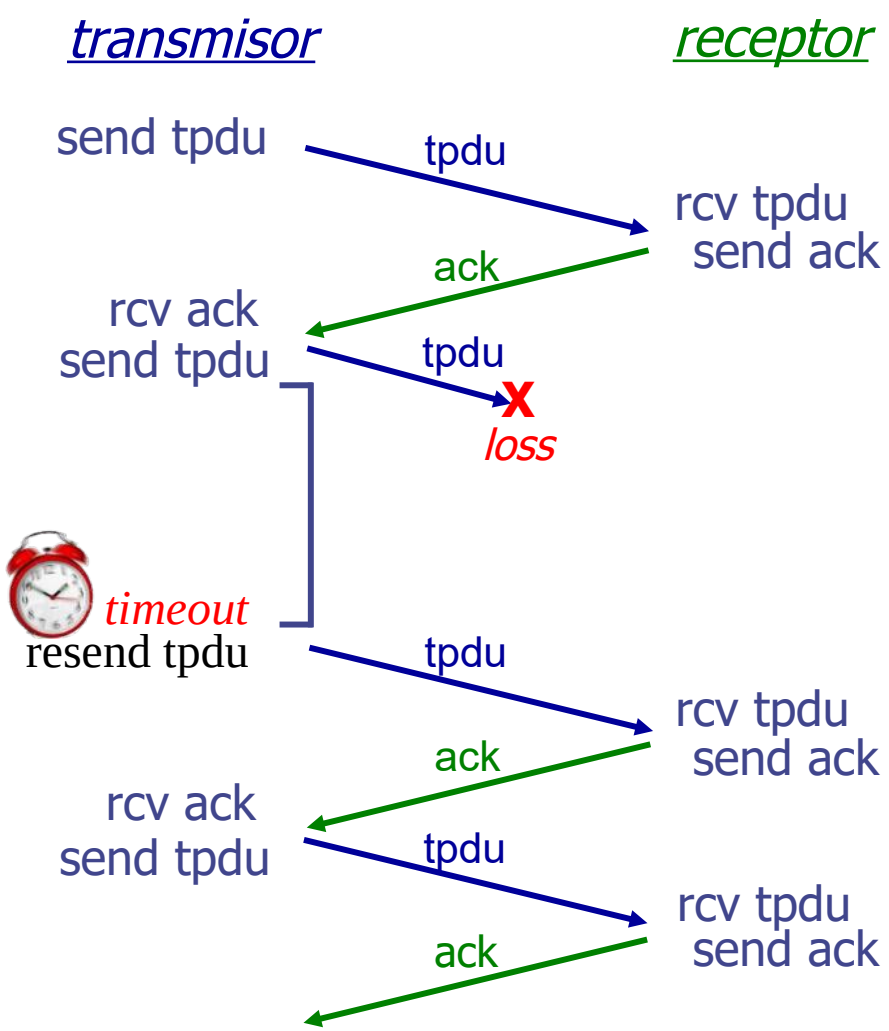


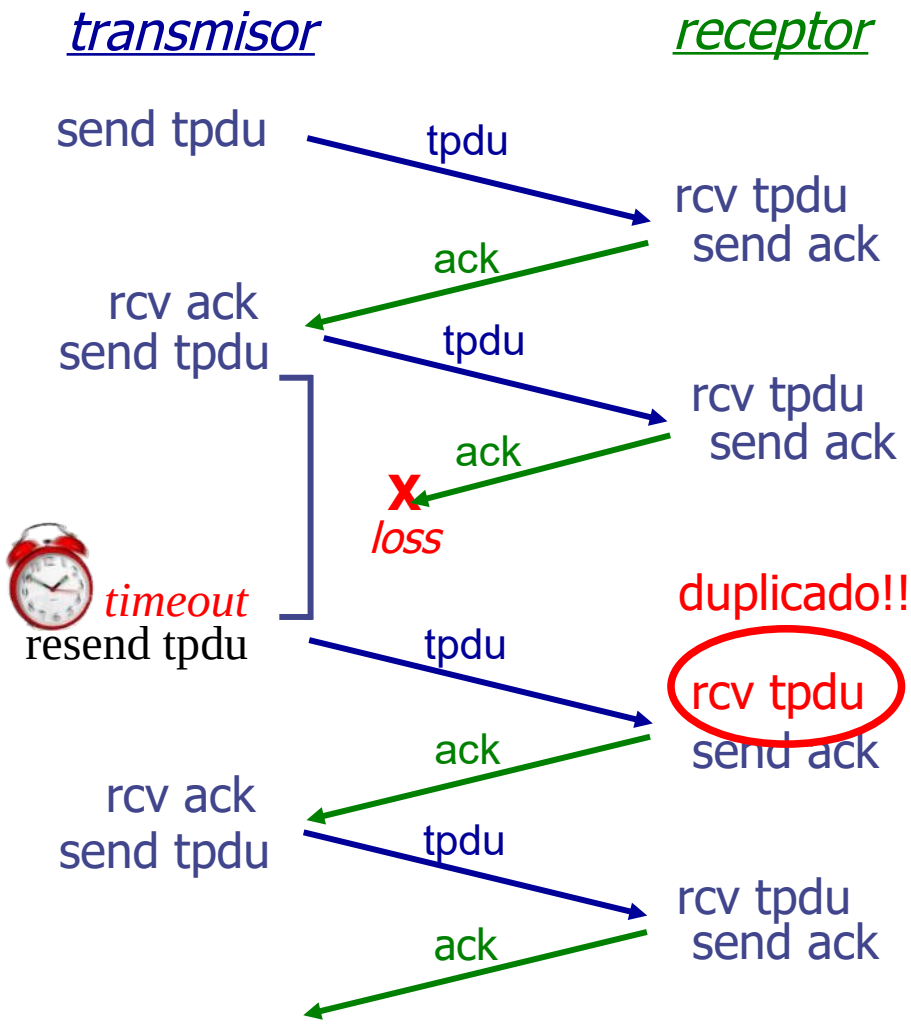
¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?





¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?





¿Pérdida de TPDU (segmento) o de reconocimiento (ACK)?

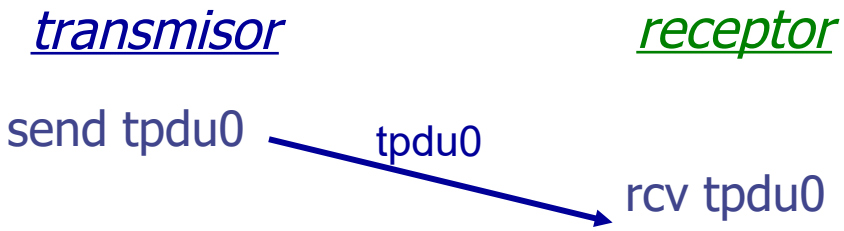
- **Problema:** se pierde ACK, receptor reenvía, y se duplica una TPDU
  - Necesidad distinguir entre una TPDU nueva de una re-enviada, surgen los de **números de secuencia.**
- **¿Cuántos números de secuencia se necesitan?**
  - En este protocolo solo 2: 0 y 1 pues hasta no recibir reconocimiento del 0 no intento mandar el 1
  - Cuando se espera la TPDU 0 se rechazan las que no sean 0, al recibirse la 0 se espera por la 1
- **Simplificación:** La “red” no re-ordena TPDU (recordar que los paquetes pueden tomar caminos distintos)

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

(a) Sin pérdida

(b) Con pérdidas

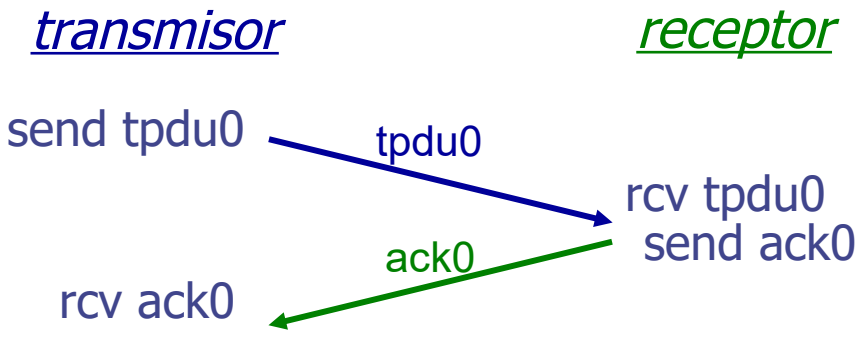
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



(a) Sin pérdida

(b) Con pérdidas

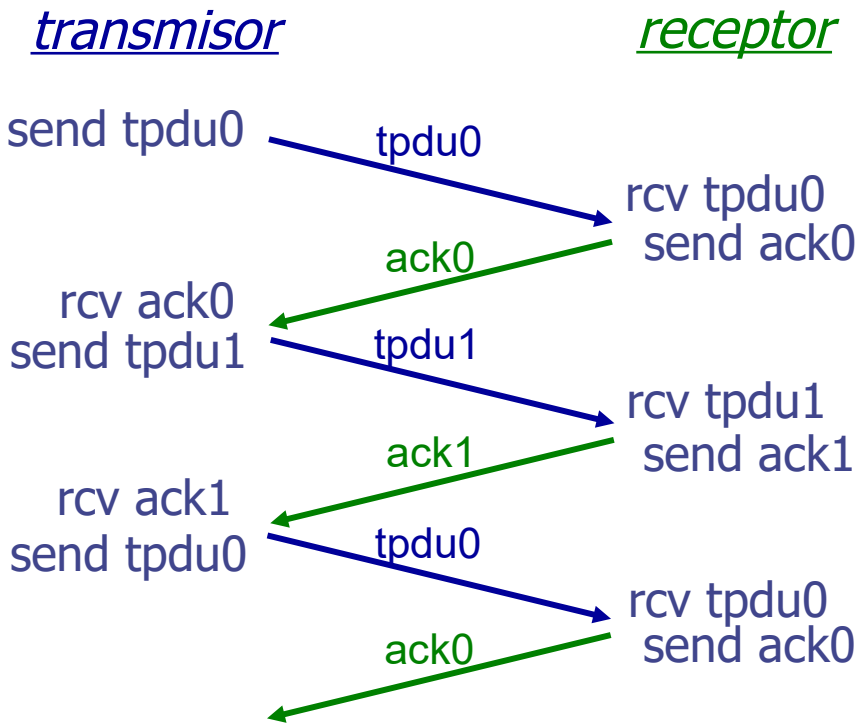
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



(a) Sin pérdida

(b) Con pérdidas

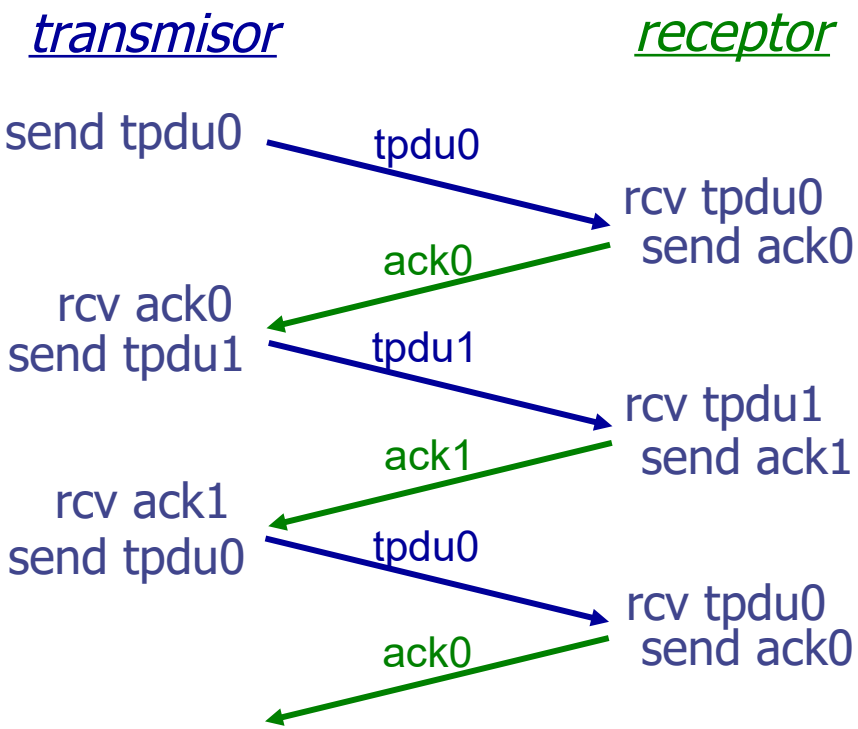
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



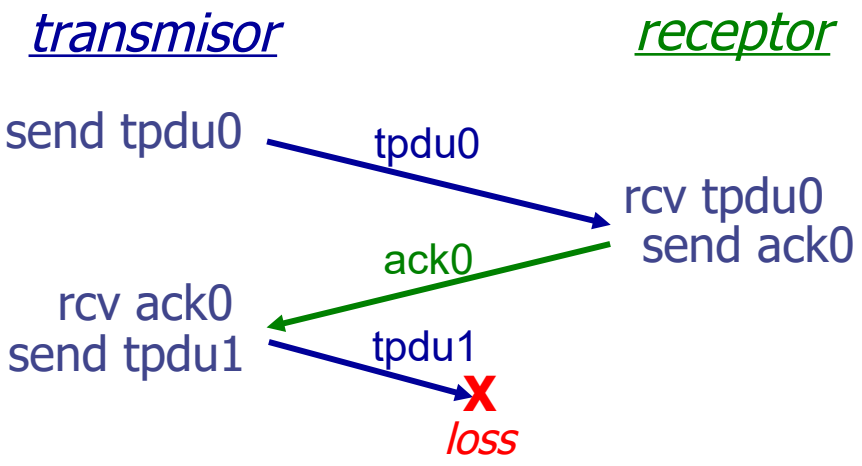
(a) Sin pérdida

(b) Con pérdidas

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

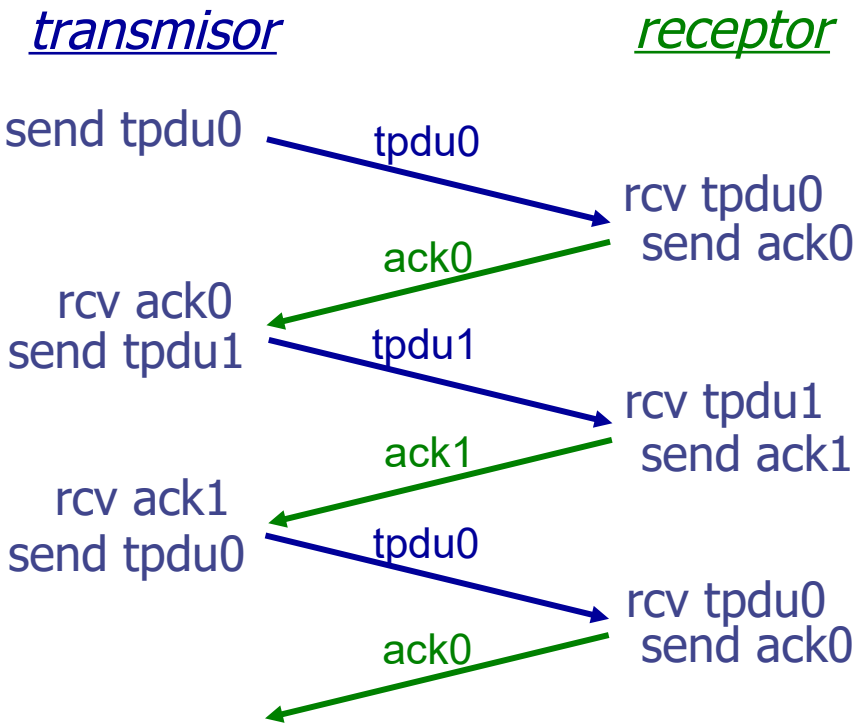


(a) Sin pérdida

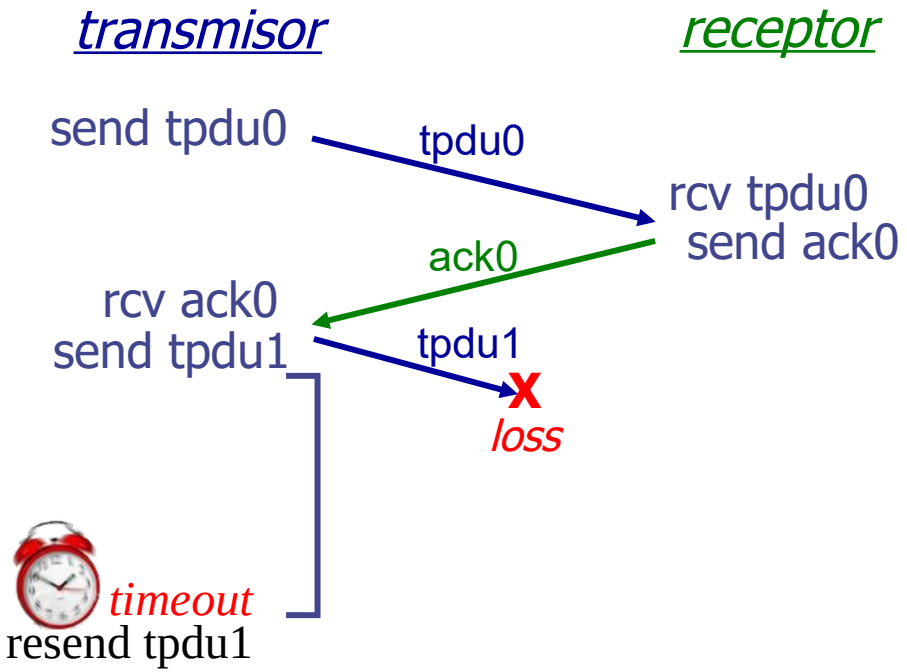


(b) Con pérdidas

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

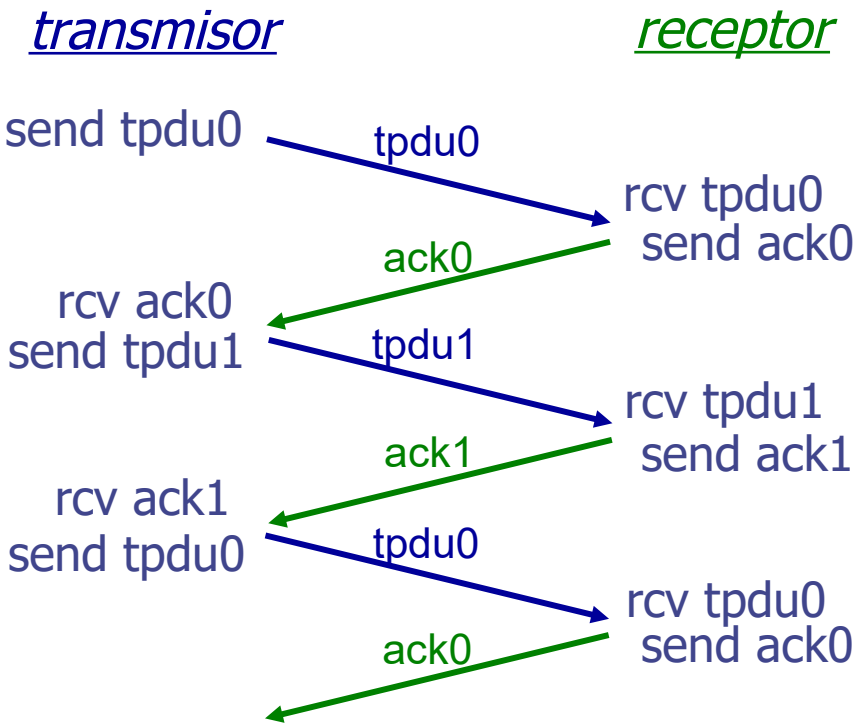


(a) Sin pérdida

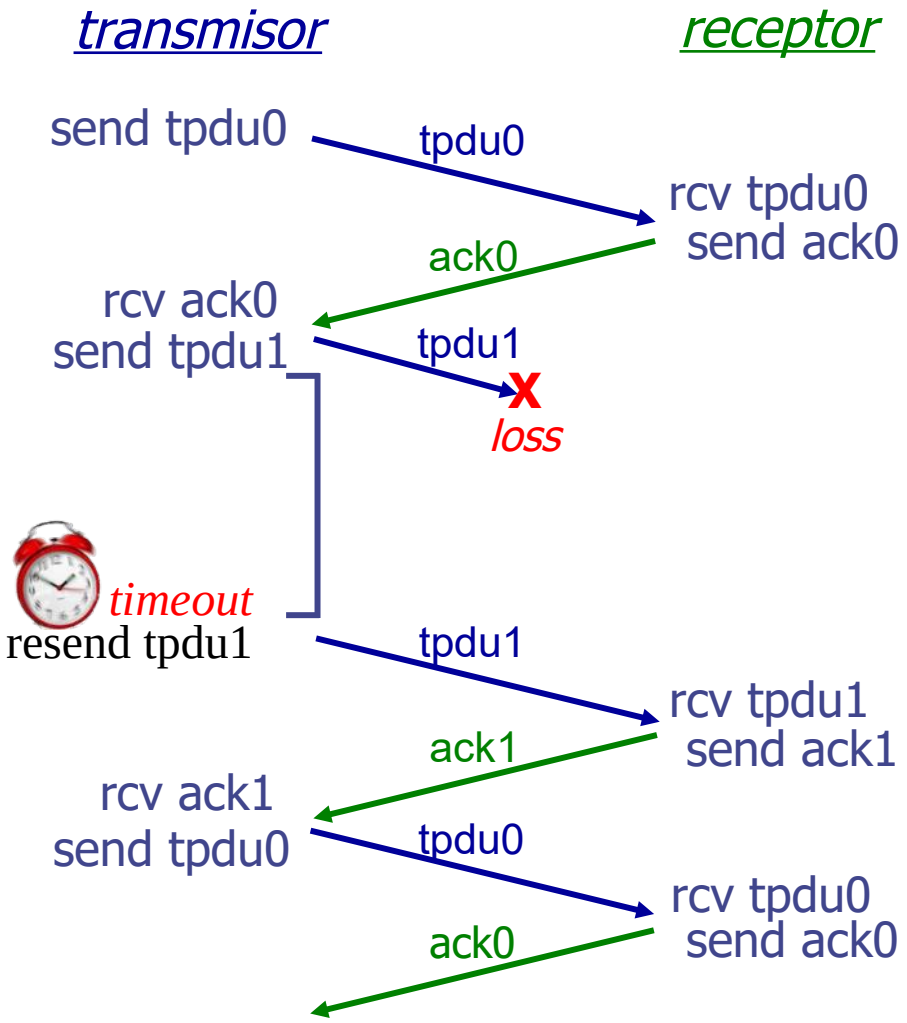


(b) Con pérdidas

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

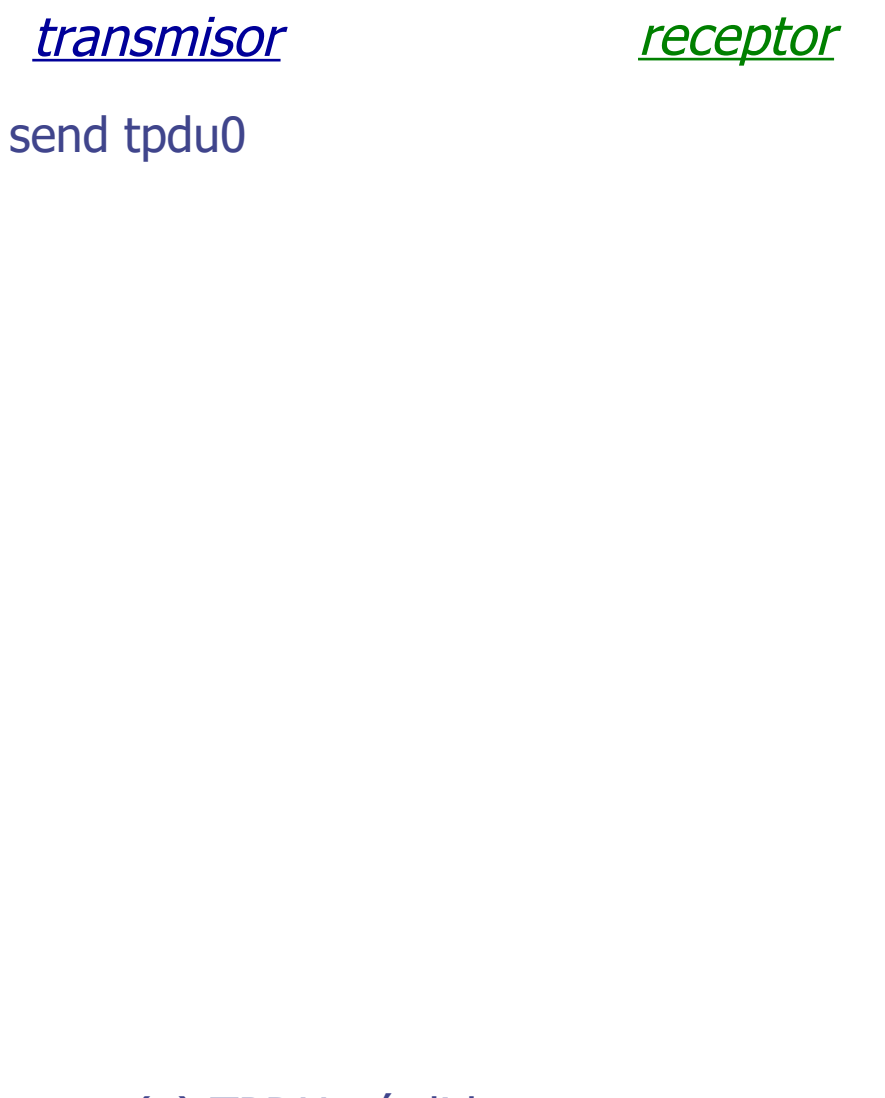


(a) Sin pérdida



(b) Con pérdidas

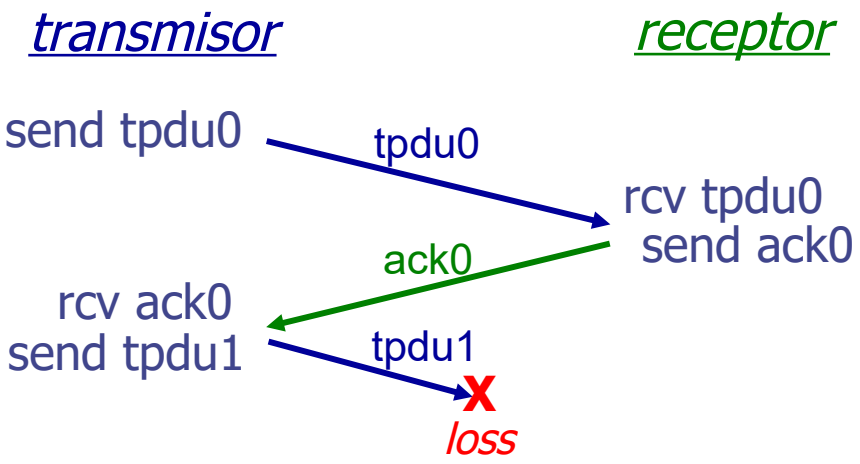
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



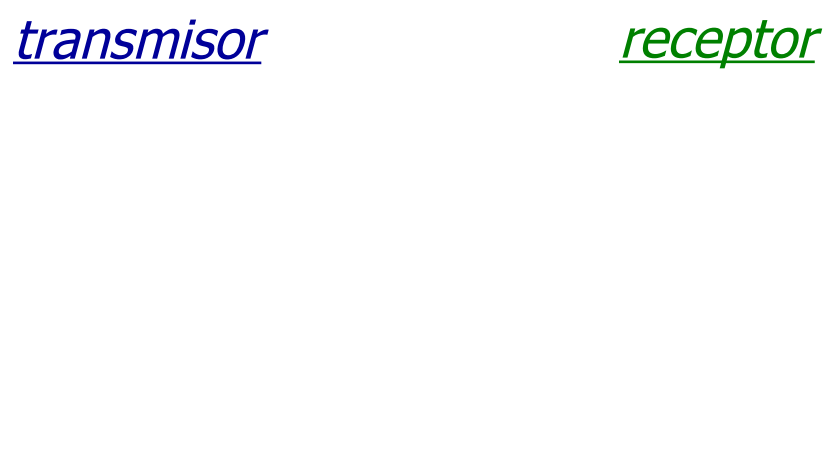
(a) TPDU pérdida

(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

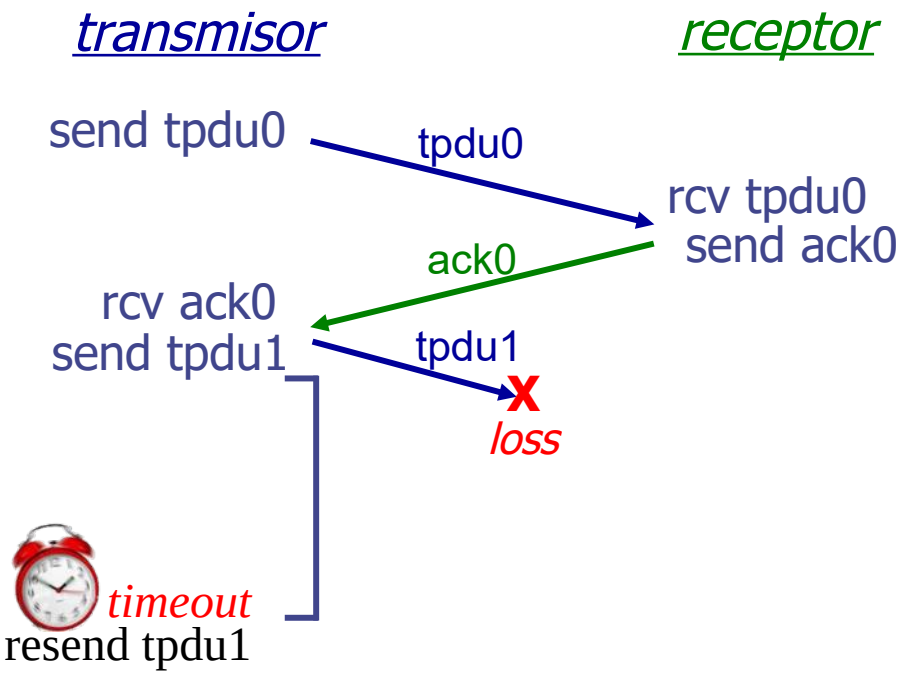


(a) TPDU pérdida



(b) ACK pérdido

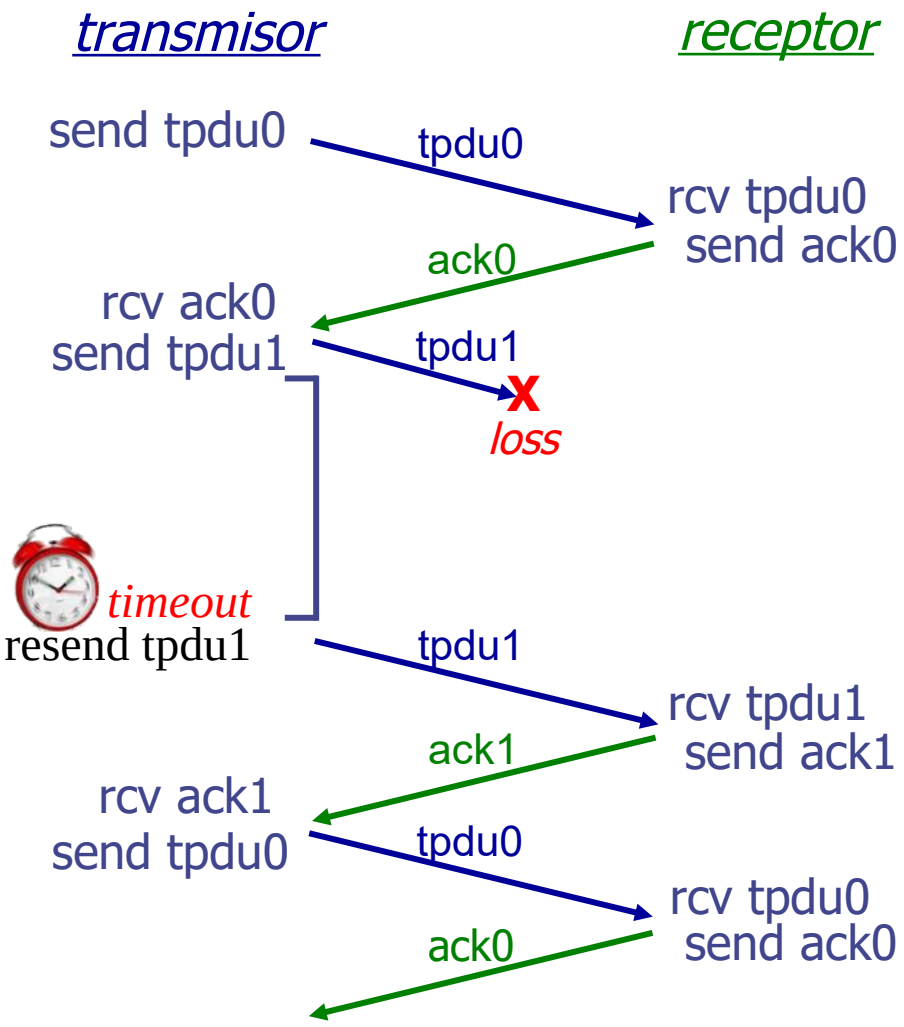
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



(a) TPDU pérdida

(b) ACK pérdido

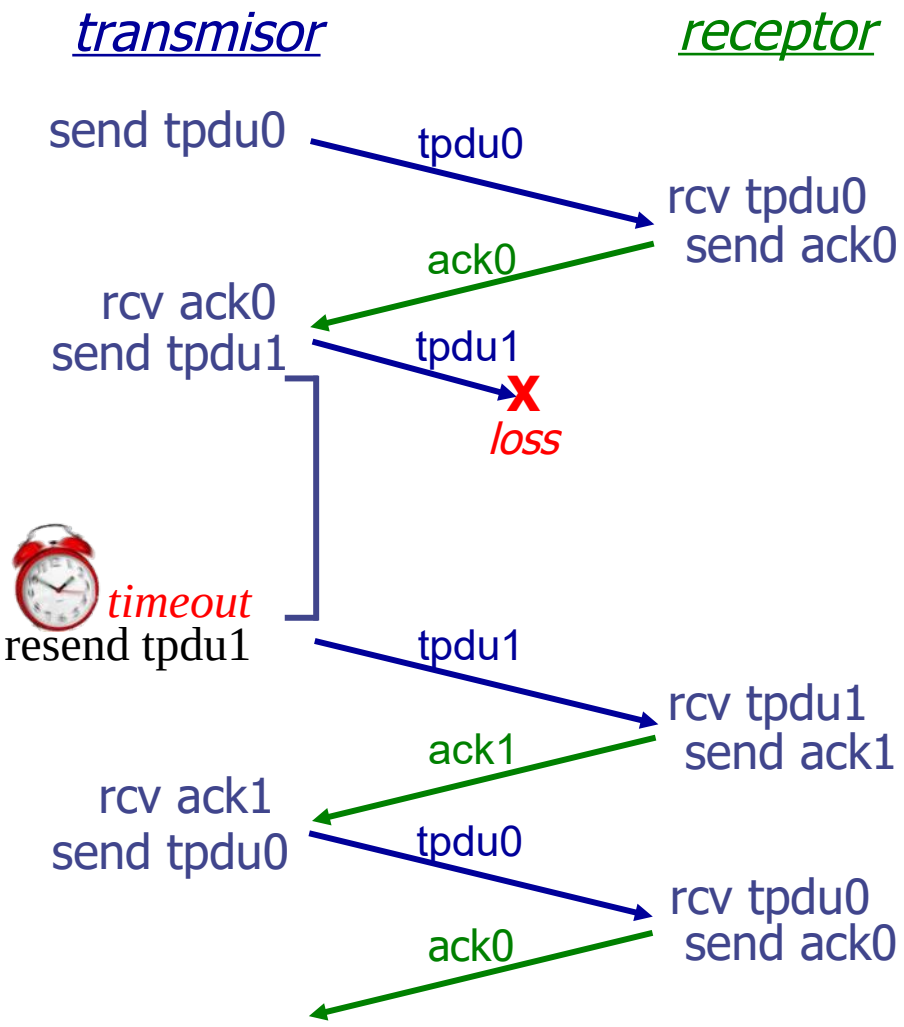
Protocolo Simple RDT 2.1 – Número de secuencia n = 1



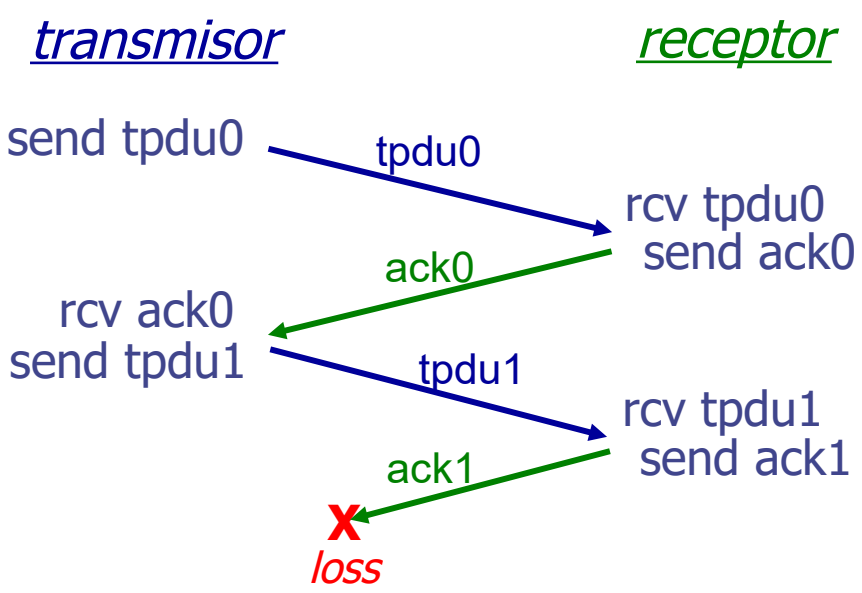
(a) TPDU pérdida

(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

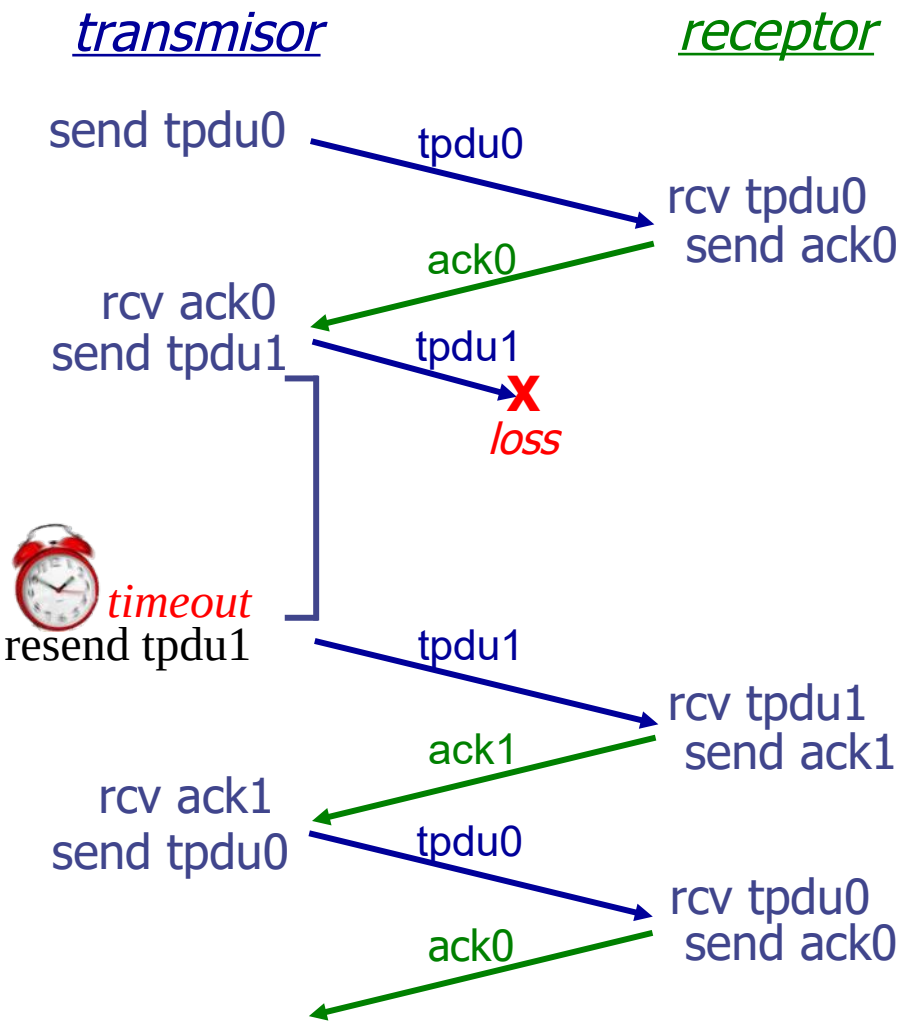


(a) TPDU pérdida

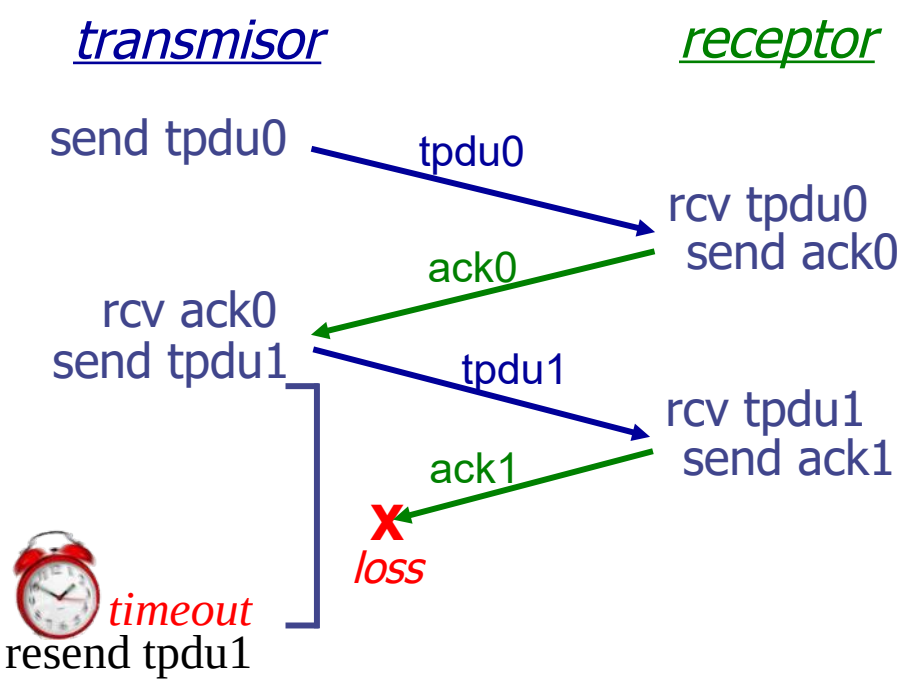


(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

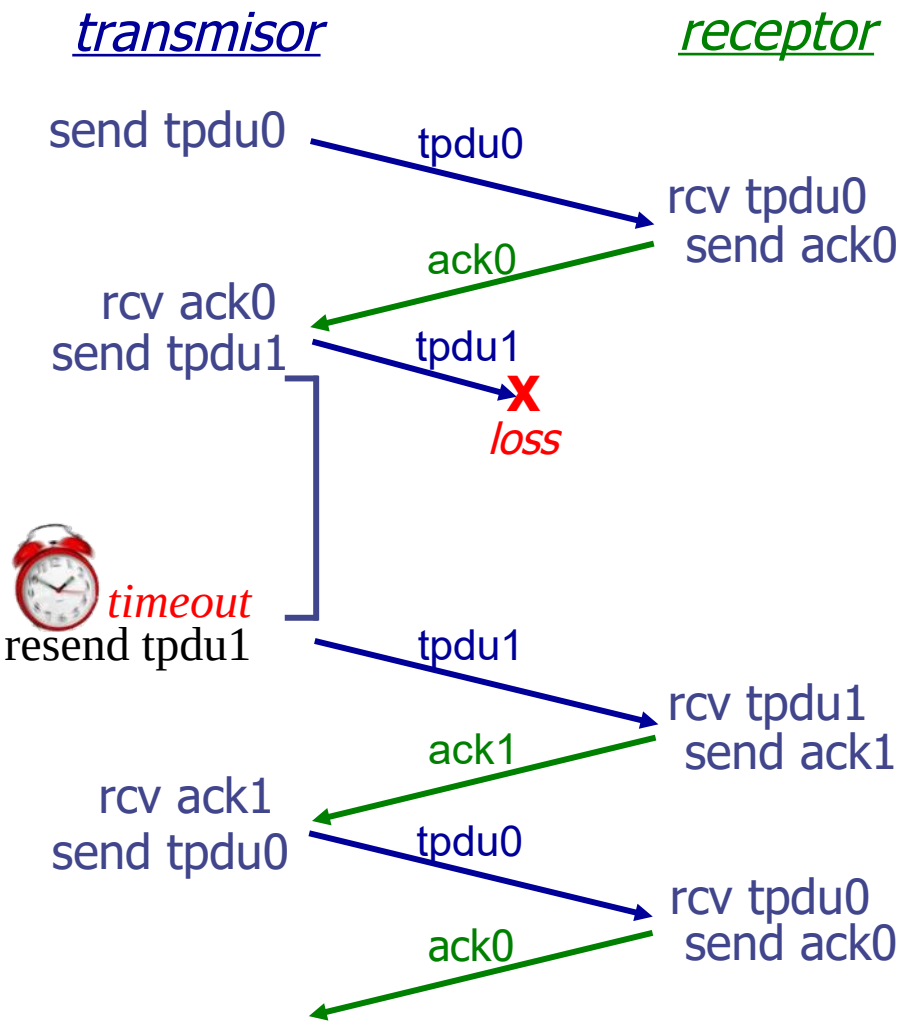


(a) TPDU pérdida

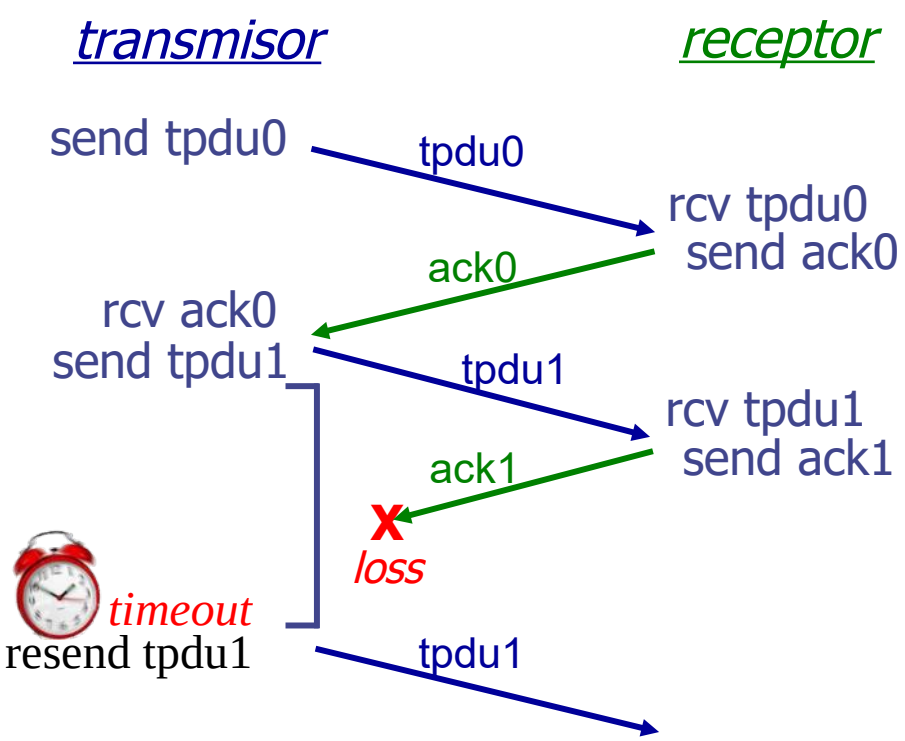


(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

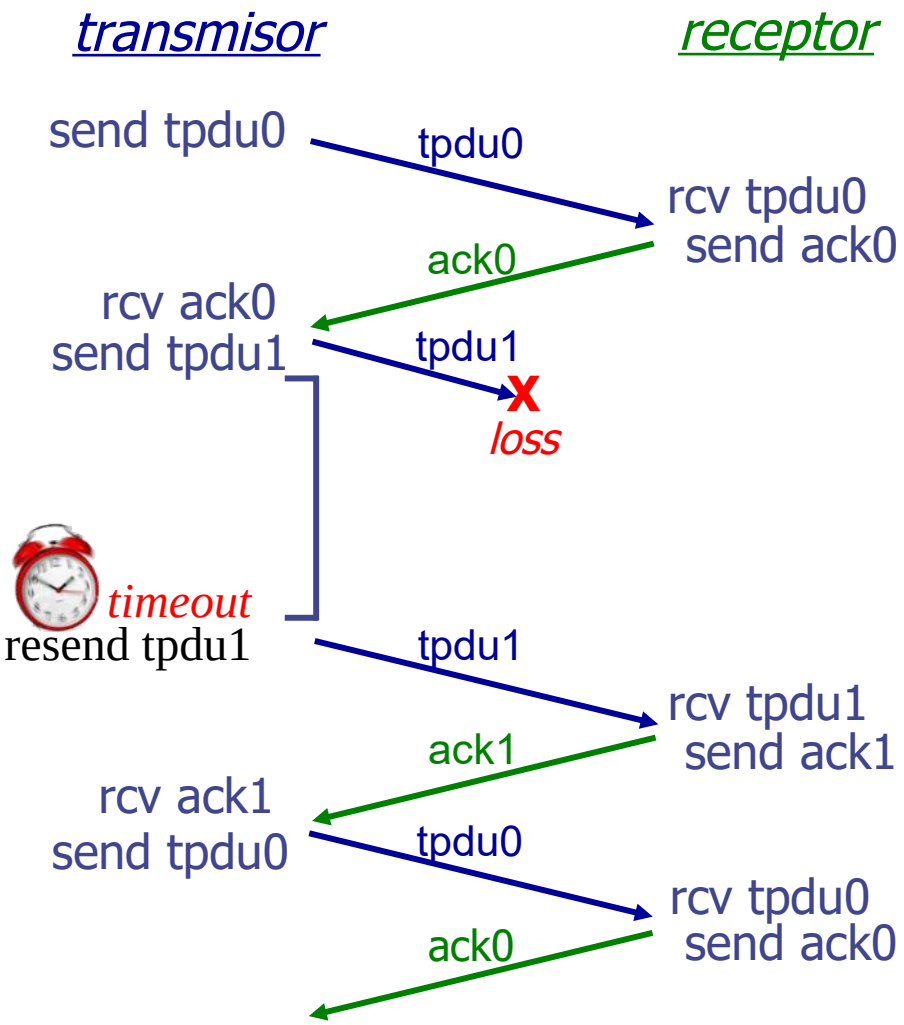


(a) TPDU pérdida

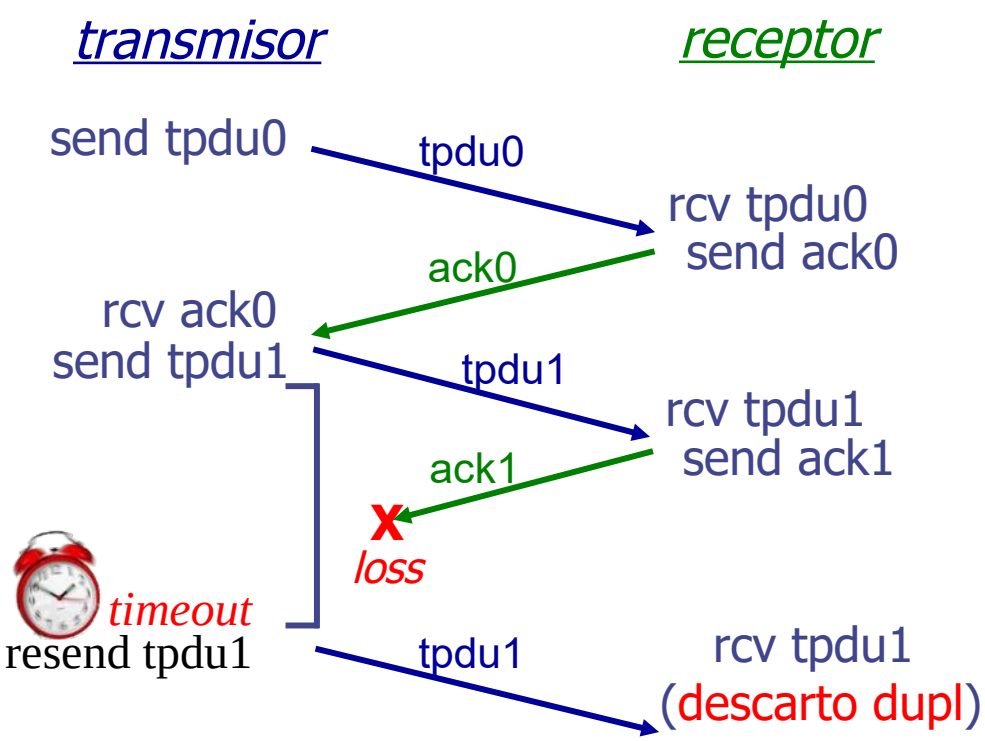


(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1

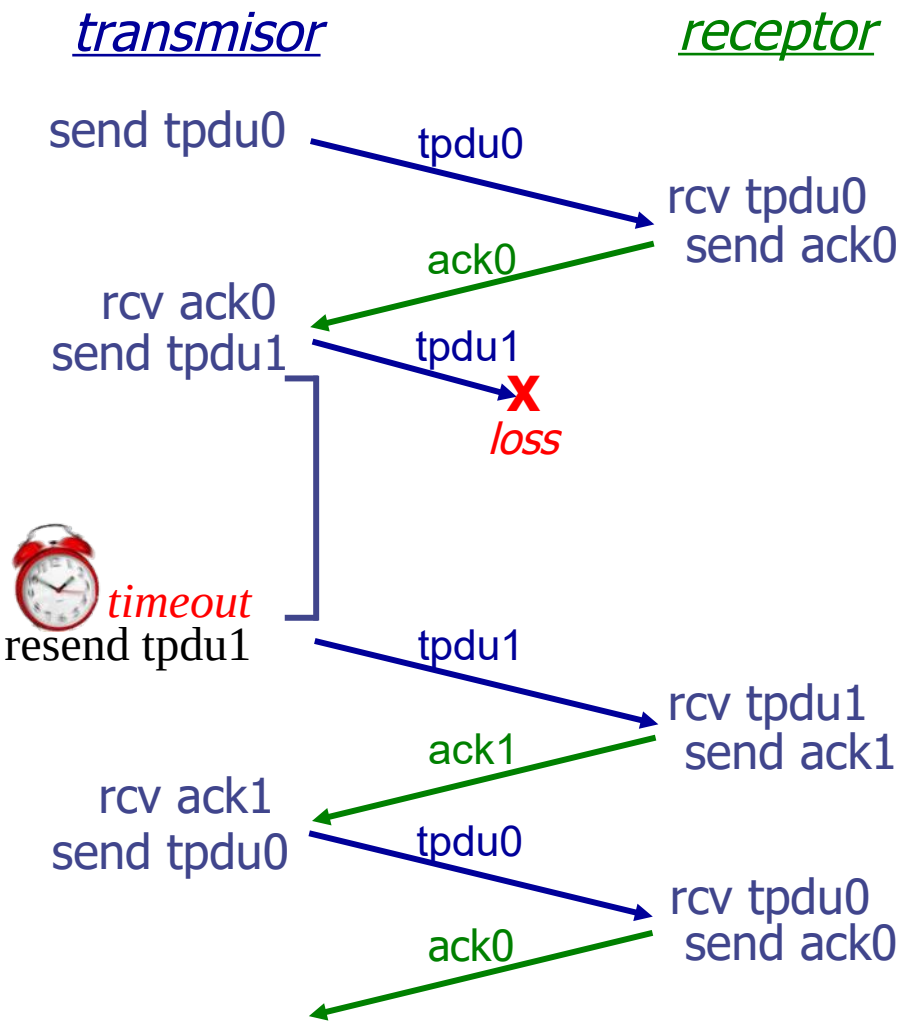


(a) TPDU pérdida

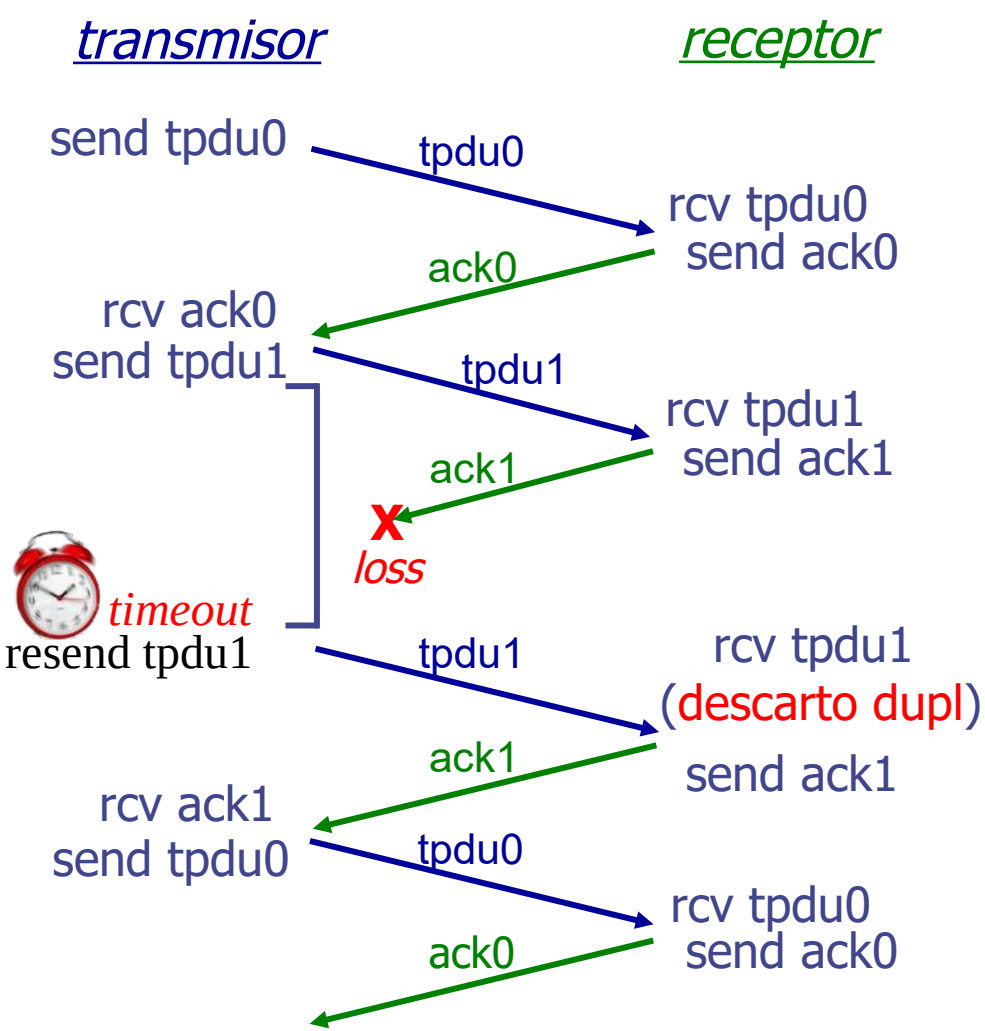


(b) ACK pérdido

Protocolo Simple RDT 2.1 – Número de secuencia n = 1



(a) TPDU pérdida



(b) ACK pérdido

Protocolo Simple - Eficiencia

- A este tipo de protocolos se les llama protocolos de “stop and wait” o “stop and go”.
- Ejemplo:
  - enlace de 1 Gbps link (R, rate, velocidad de línea)
  - RTT (round trip time = tiempo de ida y vuelta) 30 ms
  - Largo de trama 8000 bit trama (~”segmento”)

$$D_{serailización} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

$U_{transmisor}$: *utilización* – fracción de tiempo que en transmisor está transmitiendo

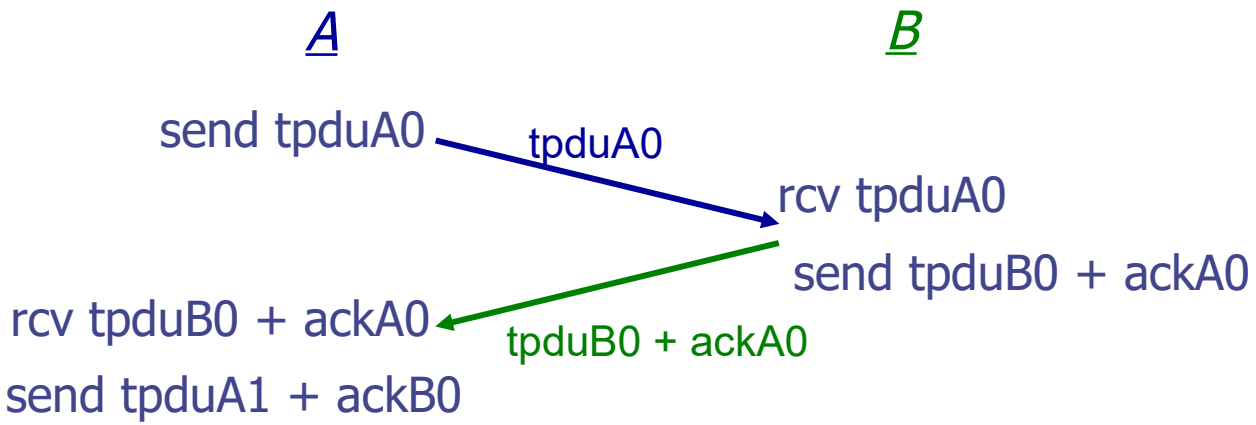
$$U_{transmisor} = \frac{L/R}{RTT+L/R} = \frac{0.008}{30 + 0.008} = 0{,}00027\ (0{,}027\ \%)$$

$$B_{Efectiva} = \frac{L}{RTT+L/R} = \frac{8000\ bits\ x1000}{(30 + 0.008)} = 266.596\ bps$$

- **Eficiencia:** sensible a “RTT x R”

Protocolo Simple – Mejoras de Eficiencia

- Es habitual el uso de la red para transmitir datos en ambas direcciones (números de Sec y ACK)
- Como el campo ACK es parte del encabezado, puedo enviar información en la misma TPDU que lleva el ACK
- **Esto se llama piggybacking**
- Problema: Si no tengo datos, ¿espero? ¿Cuánto?
  - POCO: debo mandar una TPDU solo con ACK
  - MUCHO: el transmisor retransmite porque expira un temporizador



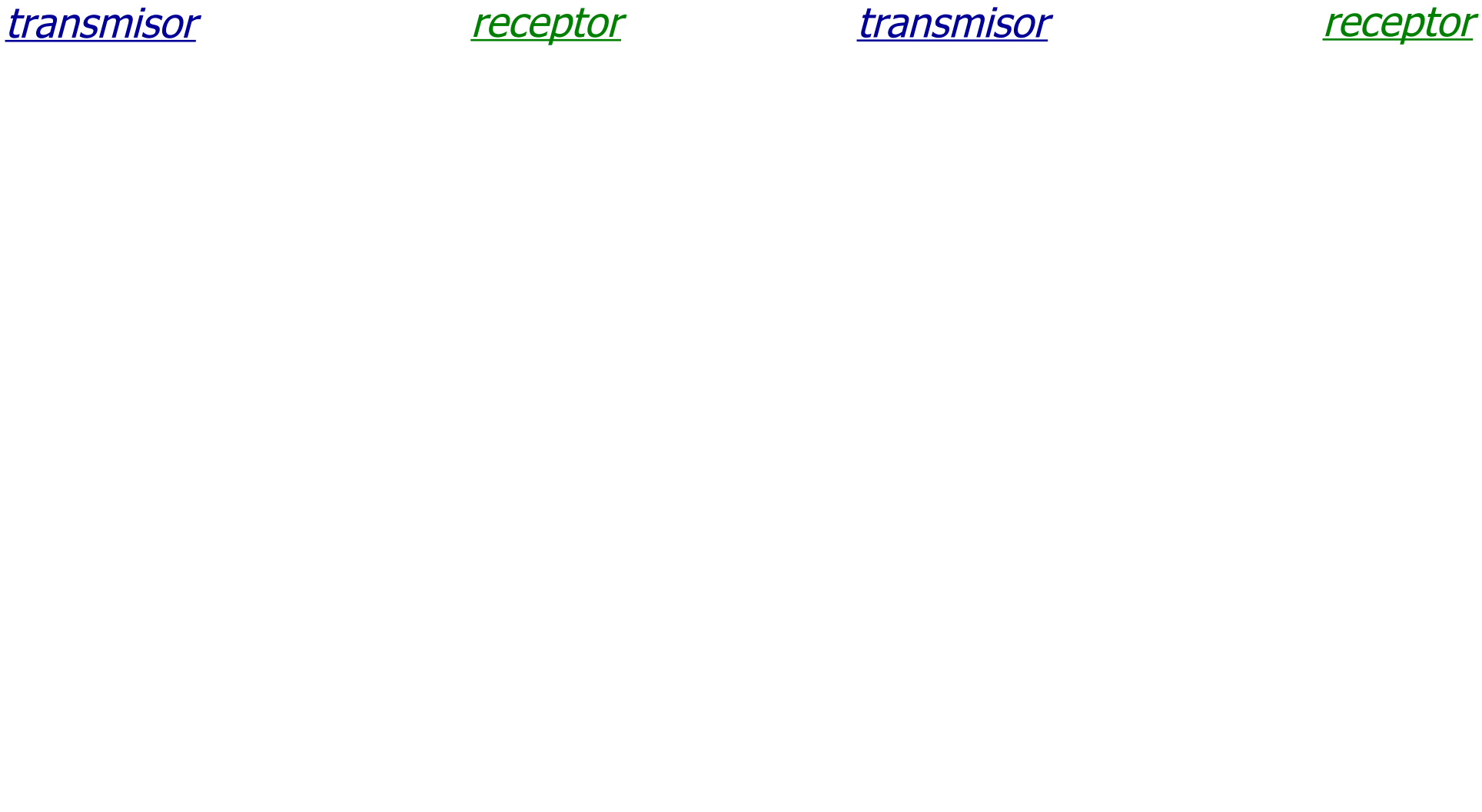
Protocolo Simple – Mejoras de Eficiencia

- **Pipeline o Tubería:** ¿Me cambia la eficiencia que existan varias TPDU en camino al receptor sin ser reconocidas?
- Por ejemplo 1000 TPDUs

$$U_{transmisor} = \frac{1000 \times L/R}{RTT + L/R} = \frac{8}{30 + 0.008} = \quad 0{,}27\ (27\ \%)$$

- Enviar varias TPDU incrementa la complejidad de nuestro protocolo, debemos incrementar la cantidad de bits destinados para números de secuencia.
- Si tenemos n bits para el N° de secuencia:
- Números de secuencia de 0 a 2^n-1
  - Lo llamaremos de 0 a MAX_SEQ

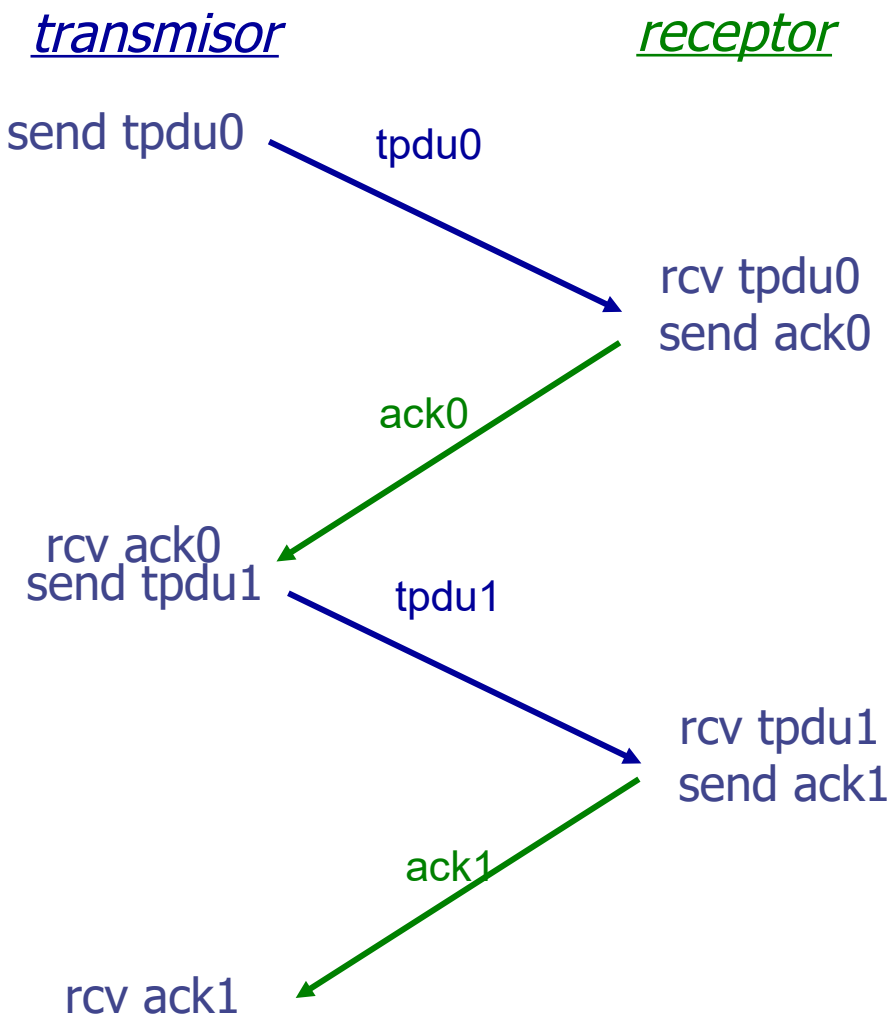
Mejora de eficiencia



(a) Stop and wait

(b) Envio y verifico más tarde

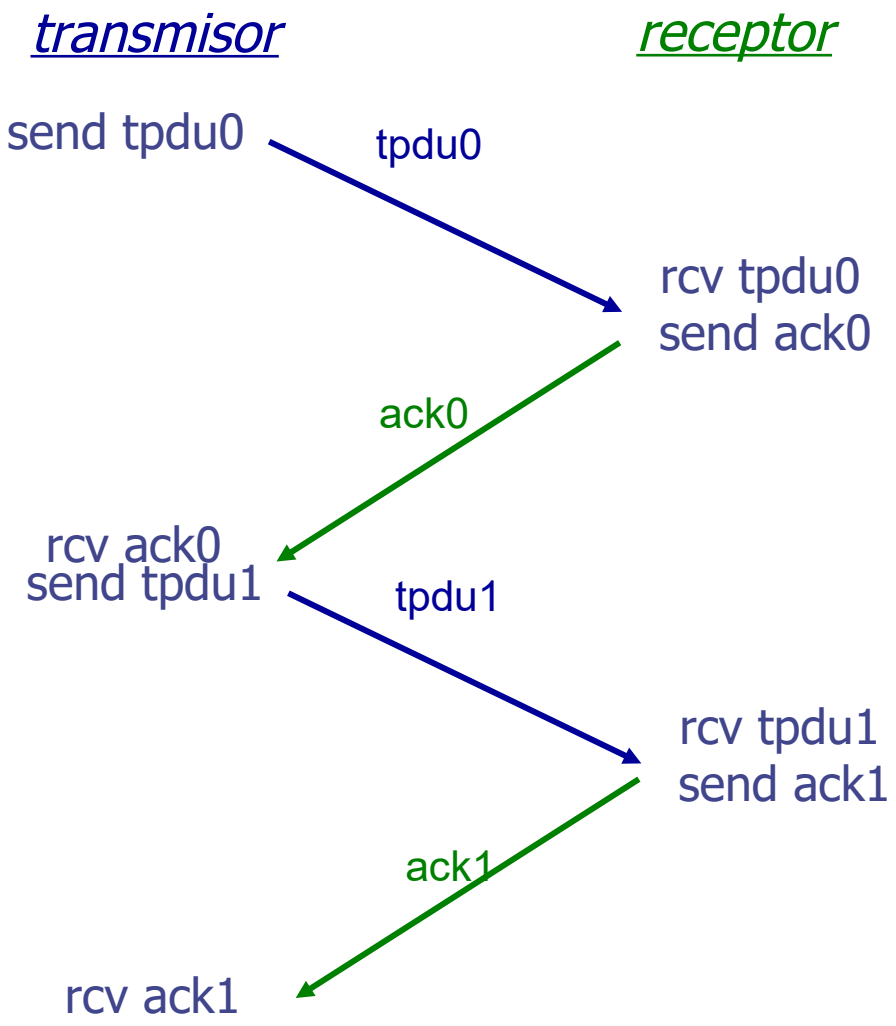
Mejora de eficiencia



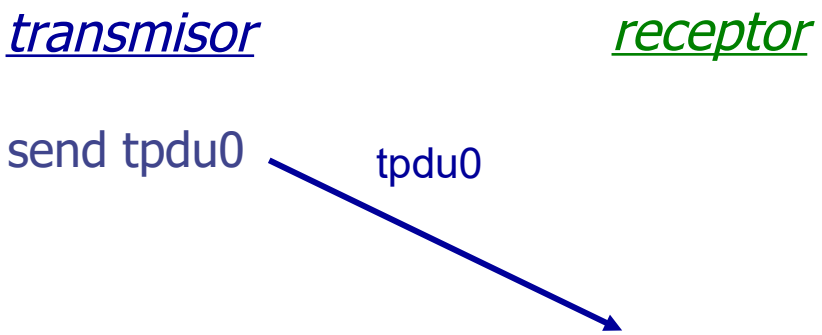
(a) Stop and wait

(b) Envio y verifico más tarde

Mejora de eficiencia

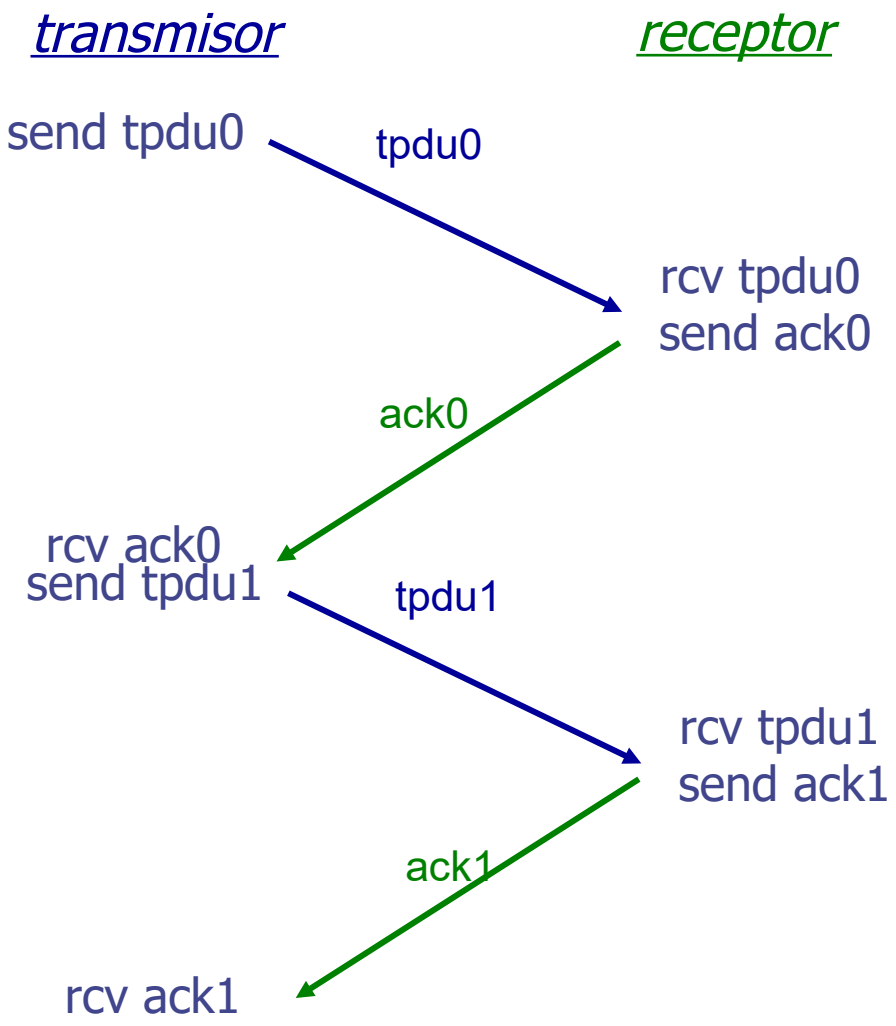


(a) Stop and wait

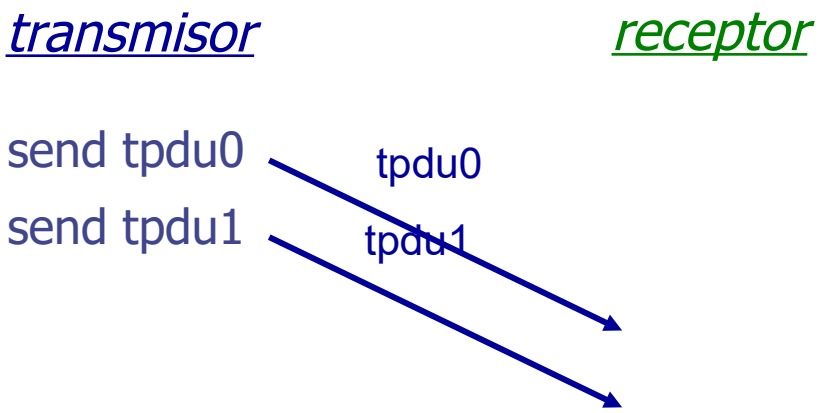


(b) Envio y verifico más tarde

Mejora de eficiencia

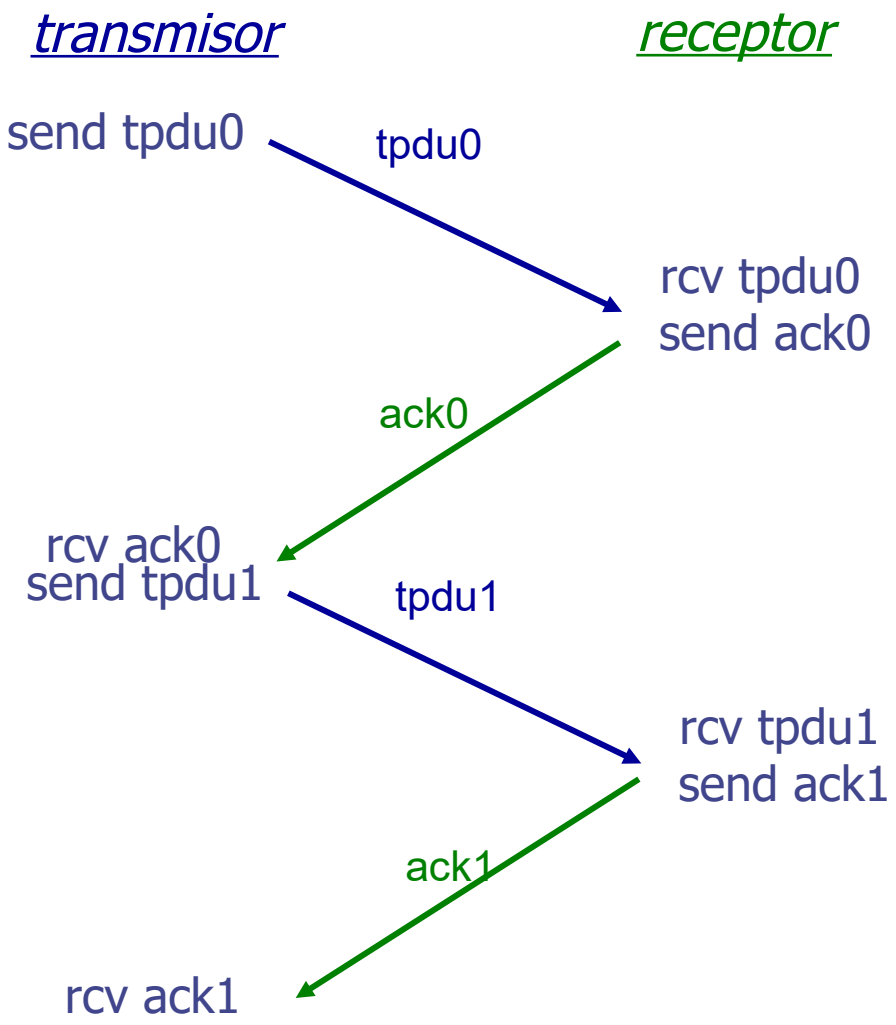


(a) Stop and wait

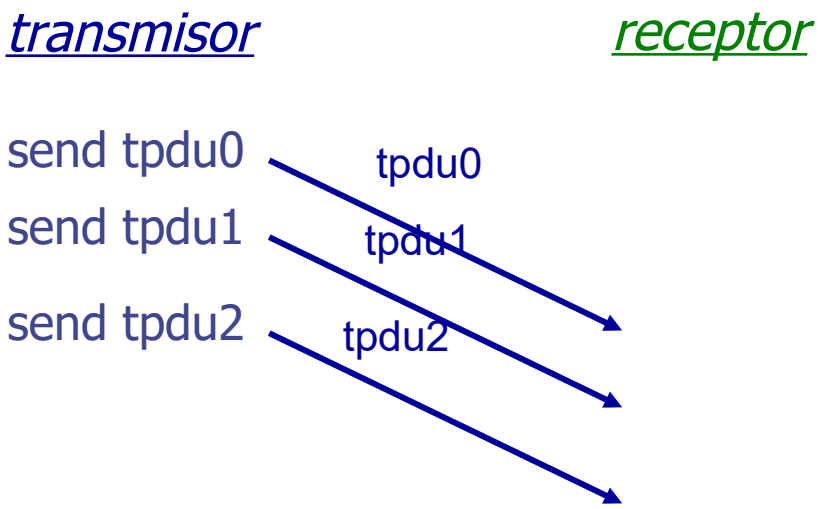


(b) Envio y verifico más tarde

Mejora de eficiencia

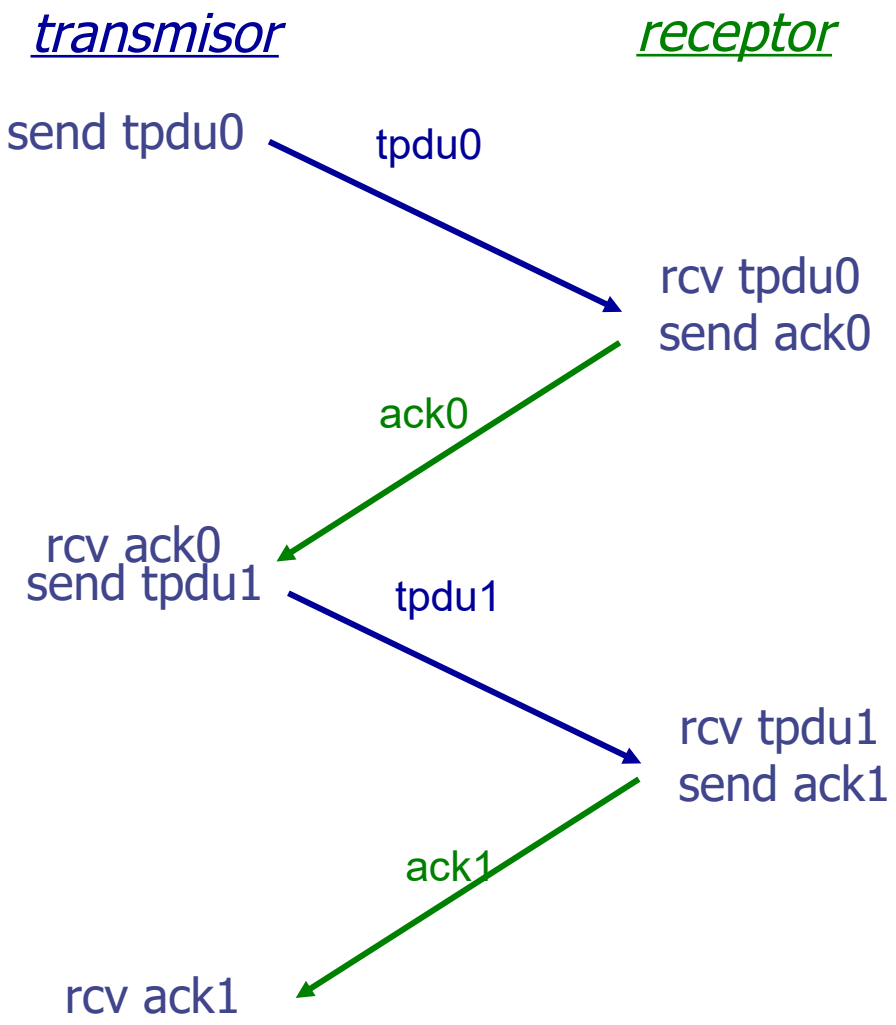


(a) Stop and wait

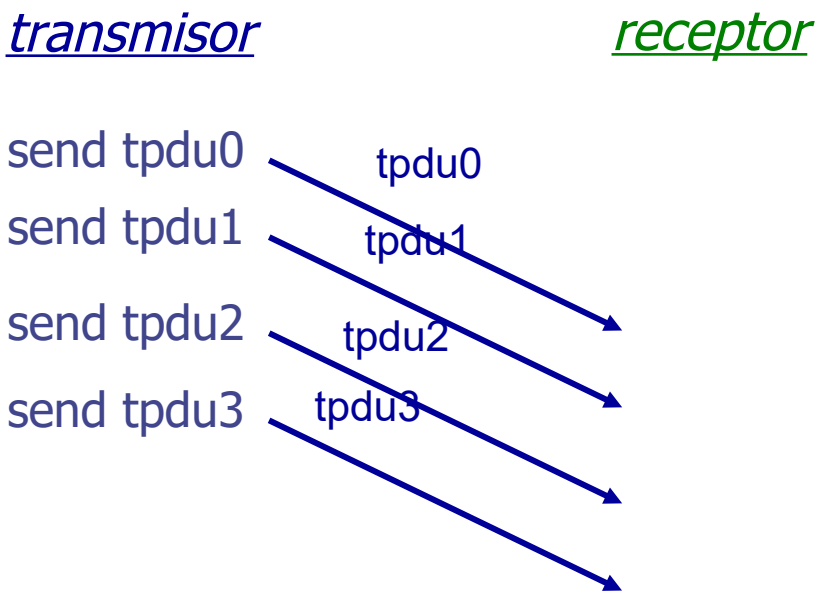


(b) Envio y verifico más tarde

Mejora de eficiencia

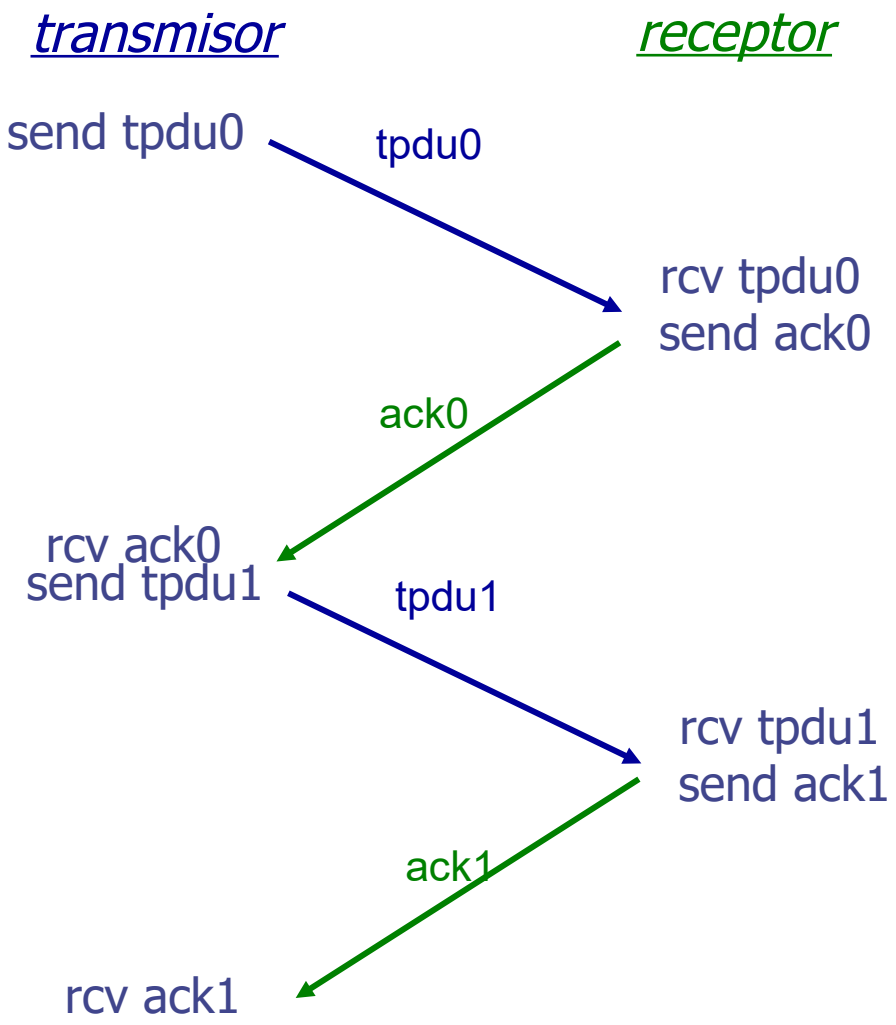


(a) Stop and wait

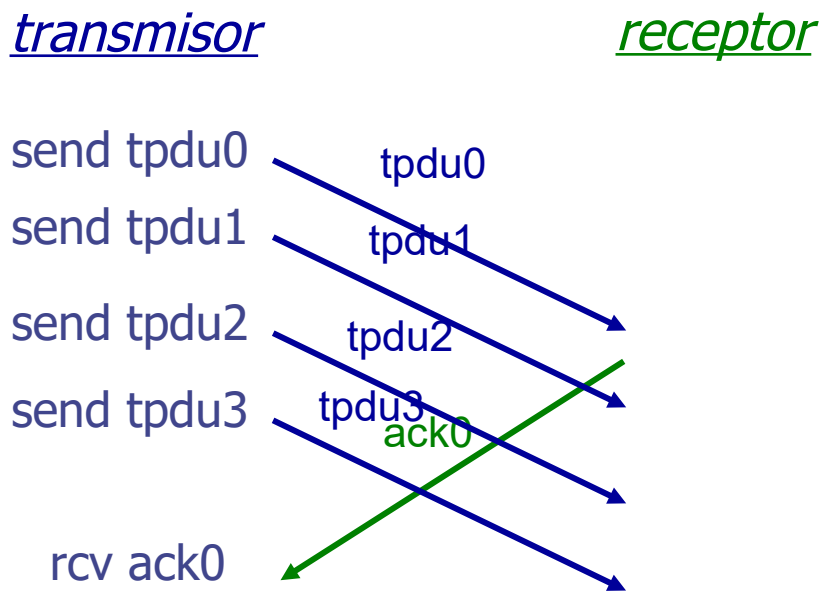


(b) Envio y verifico más tarde

Mejora de eficiencia

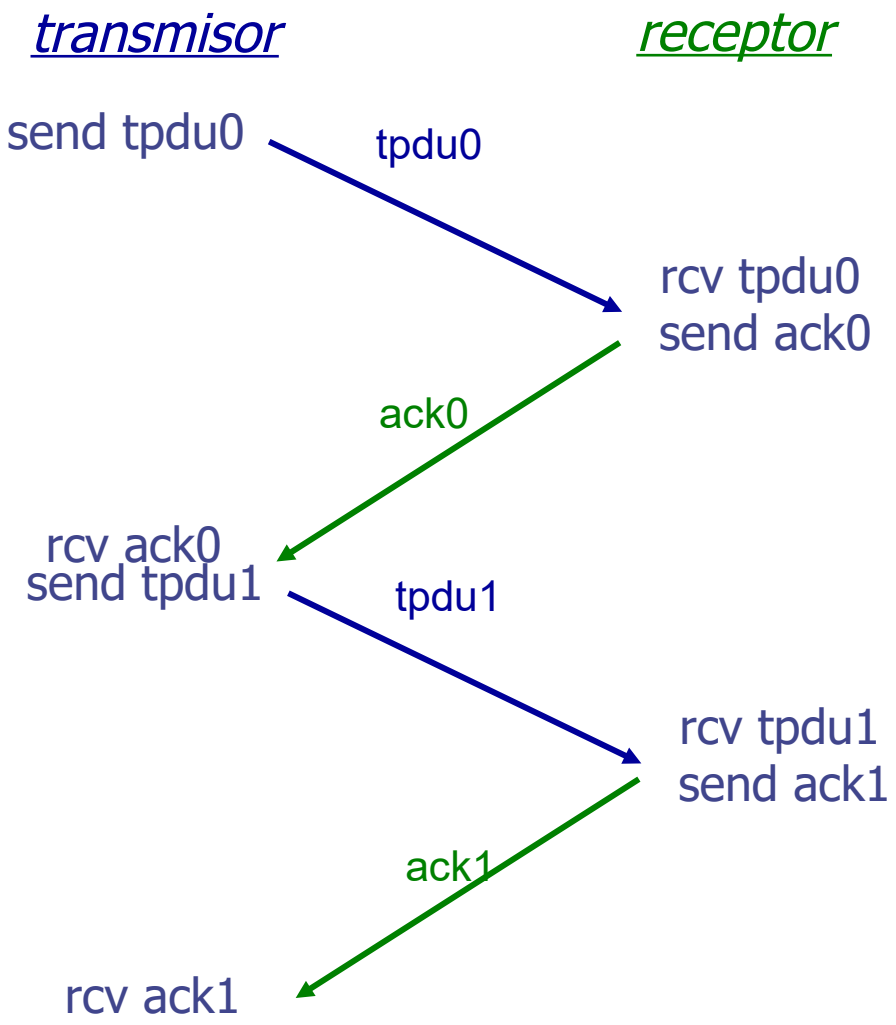


(a) Stop and wait

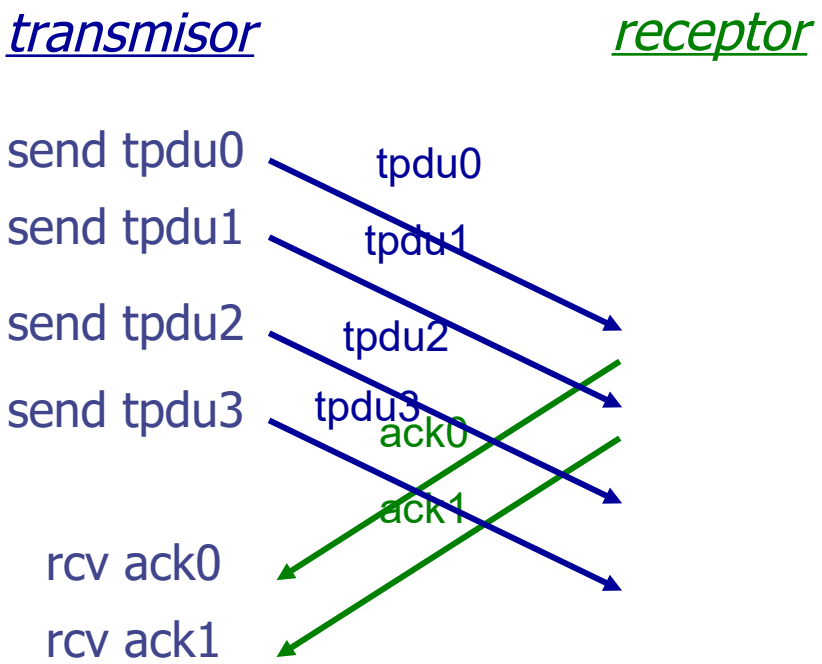


(b) Envio y verifico más tarde

Mejora de eficiencia

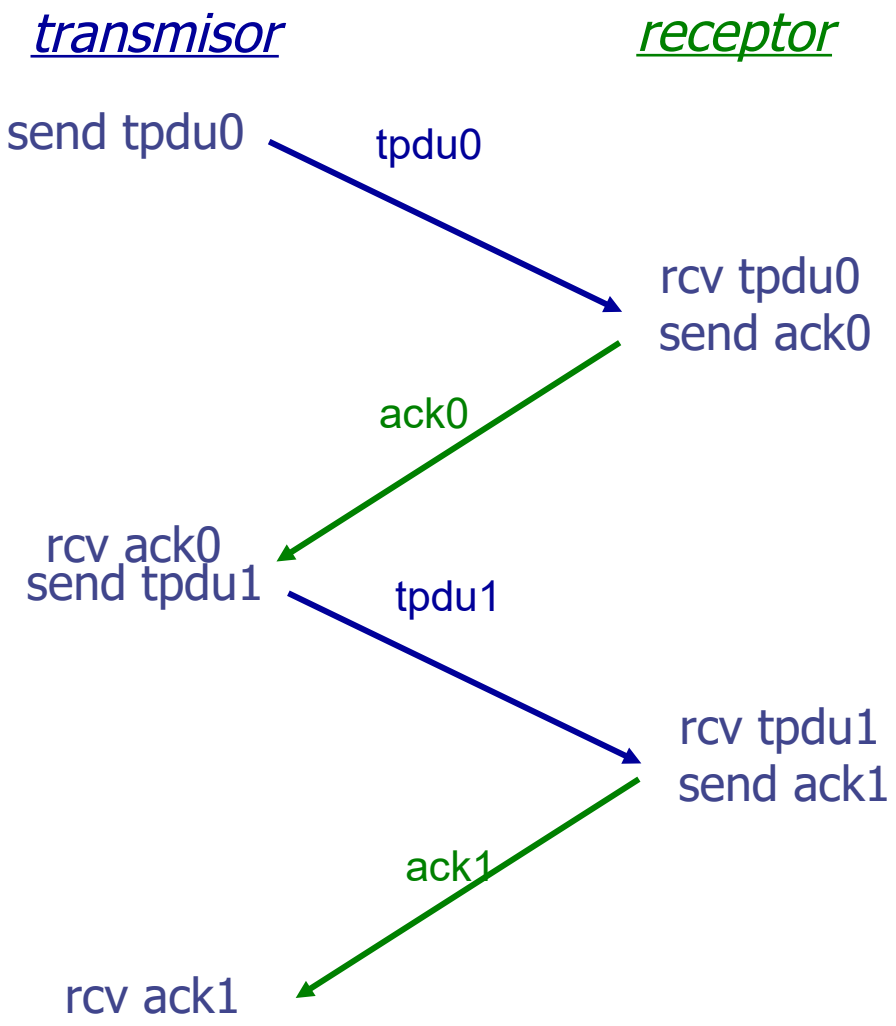


(a) Stop and wait

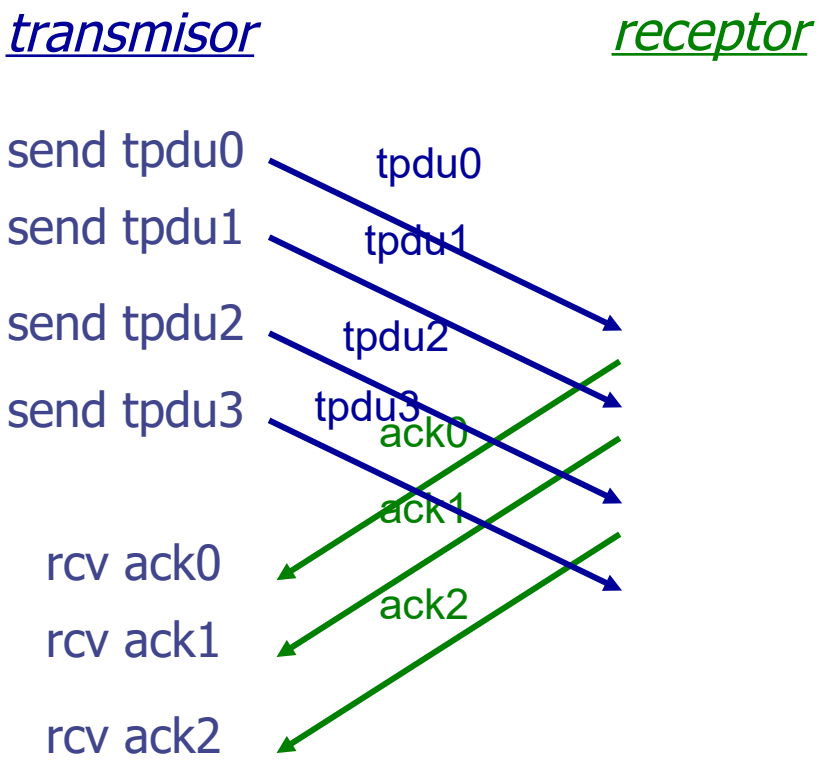


(b) Envio y verifico más tarde

Mejora de eficiencia

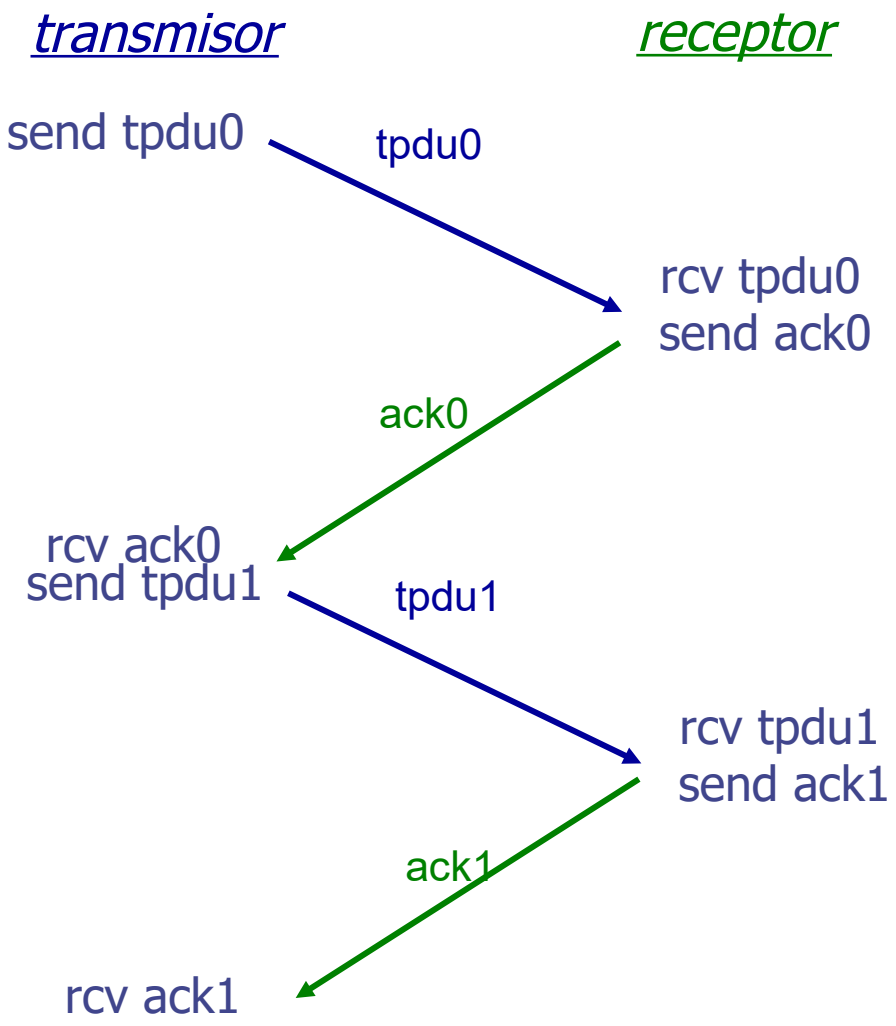


(a) Stop and wait

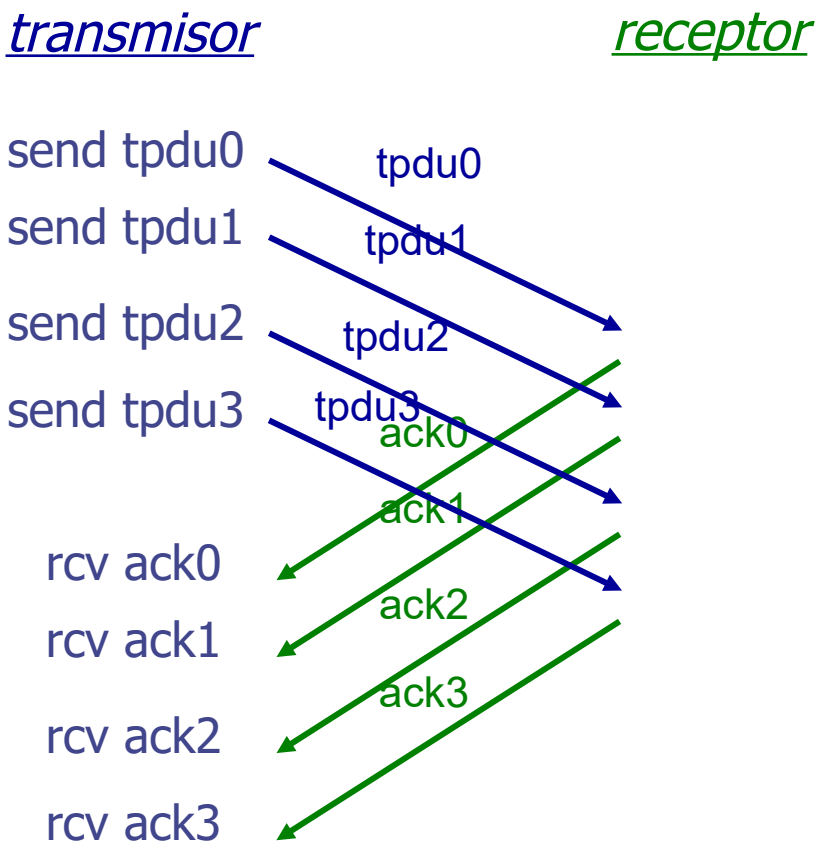


(b) Envio y verifico más tarde

Mejora de eficiencia



(a) Stop and wait



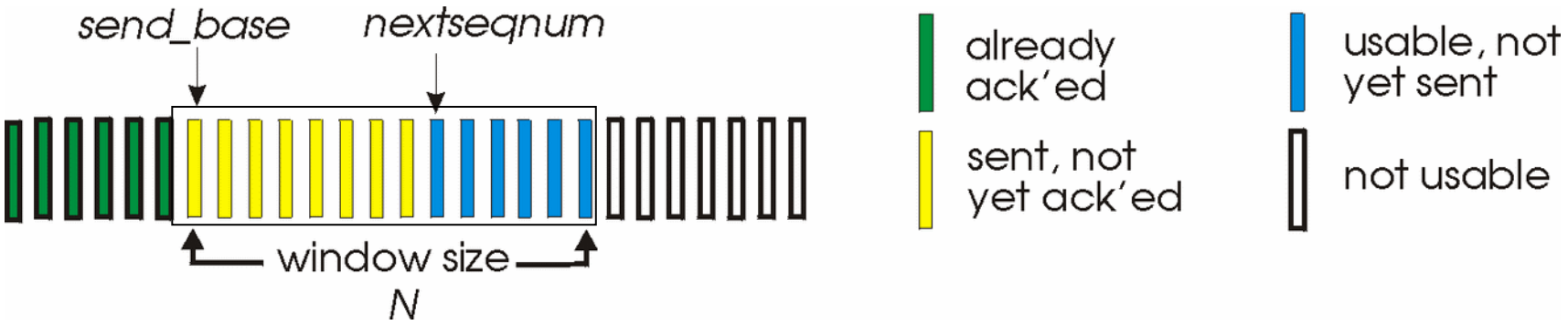
(b) Envio y verifico más tarde

Mejoras de Performance – Ventanas Deslizantes

- Le permite a la capa de transporte más libertad en el orden de transmisión y recepción de las TPDUs
- PERO: agrega complejidad
- El transmisor debe mantener copia de las TPDUs enviadas en un buffer o memoria para reenviarlas si no recibe el reconocimiento
- El receptor puede:
  - Aceptar solamente TPDUs en la secuencia correcta
    - Go Back N (si se pierden algunos tengo que retransmitir desde el último que llegó bien en adelante)
  - Aceptar TPDUs en desorden. En ese caso debe guardarlas hasta recibir las faltantes para entregarlas en orden a la capa de aplicación.
    - Selective Repeater

Mejoras de Performance – Ventanas Deslizantes

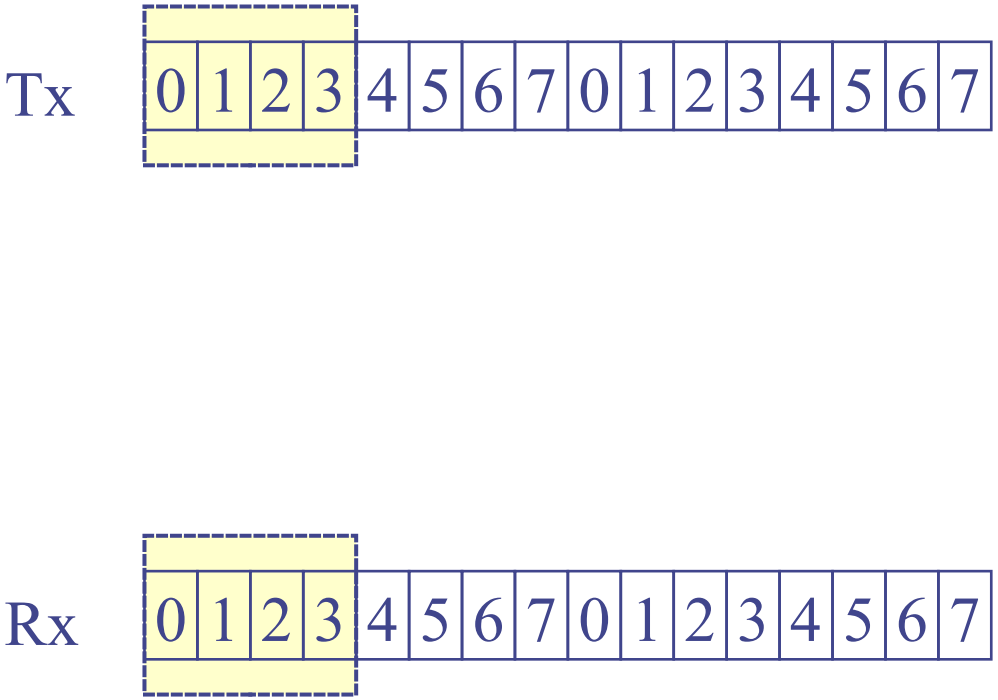
- **Ventana de transmisión (TX)**
  - números de secuencia de las TPDU que puedo enviar, o ya han sido enviadas y aún no han sido reconocidas
- **Ventana de recepción (RX)**
  - números de secuencia de las TPDU que se pueden aceptar en un determinado instante
- Las ventanas de TX y RX no tienen por que tener los mismos límites ni el mismo tamaño



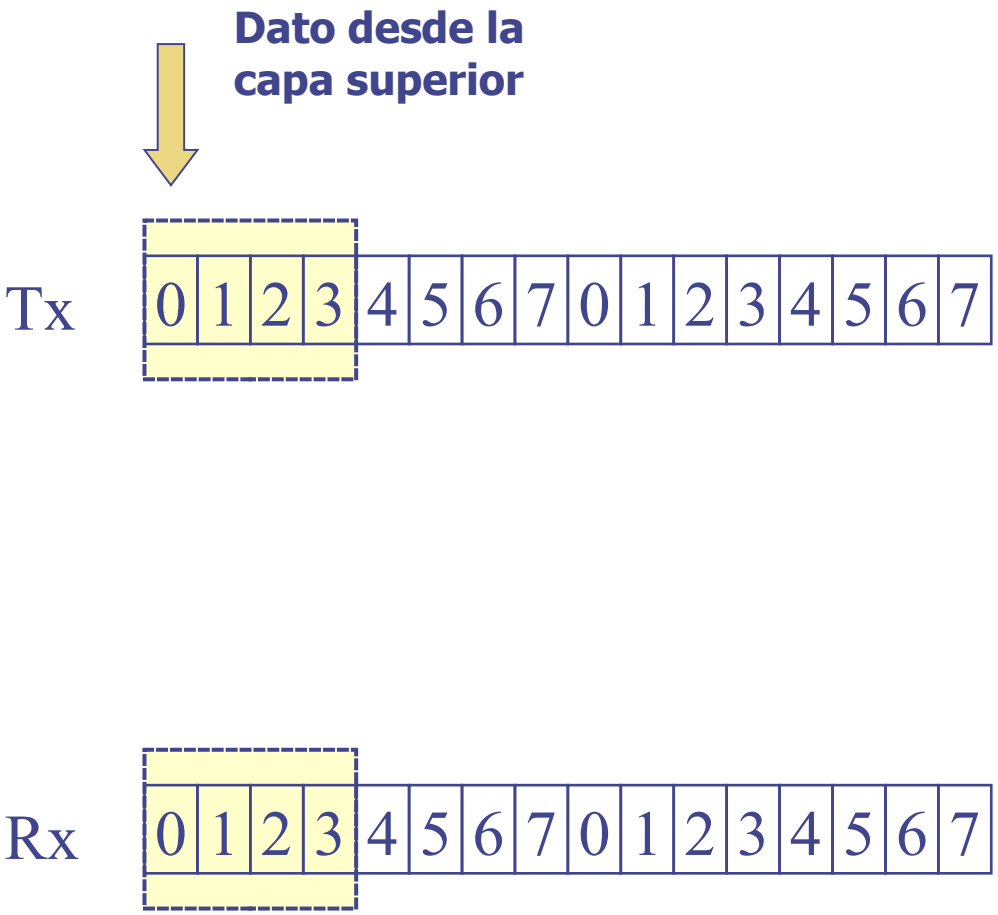
Mejoras de Performance – Ventanas Deslizantes

- Si no se dispone lugar en la ventana de TX no se deben aceptar datos de la aplicación
- La ventana del receptor son los números de secuencia que puede aceptar
- Si se recibe algo fuera de la ventana se descarta
- Puede ser necesario enviar un reconocimiento
- Si lo recibido coincide con el límite inferior, se entrega a la capa de aplicación los datos, se envía ACK y se avanza la ventana (“desliza”)

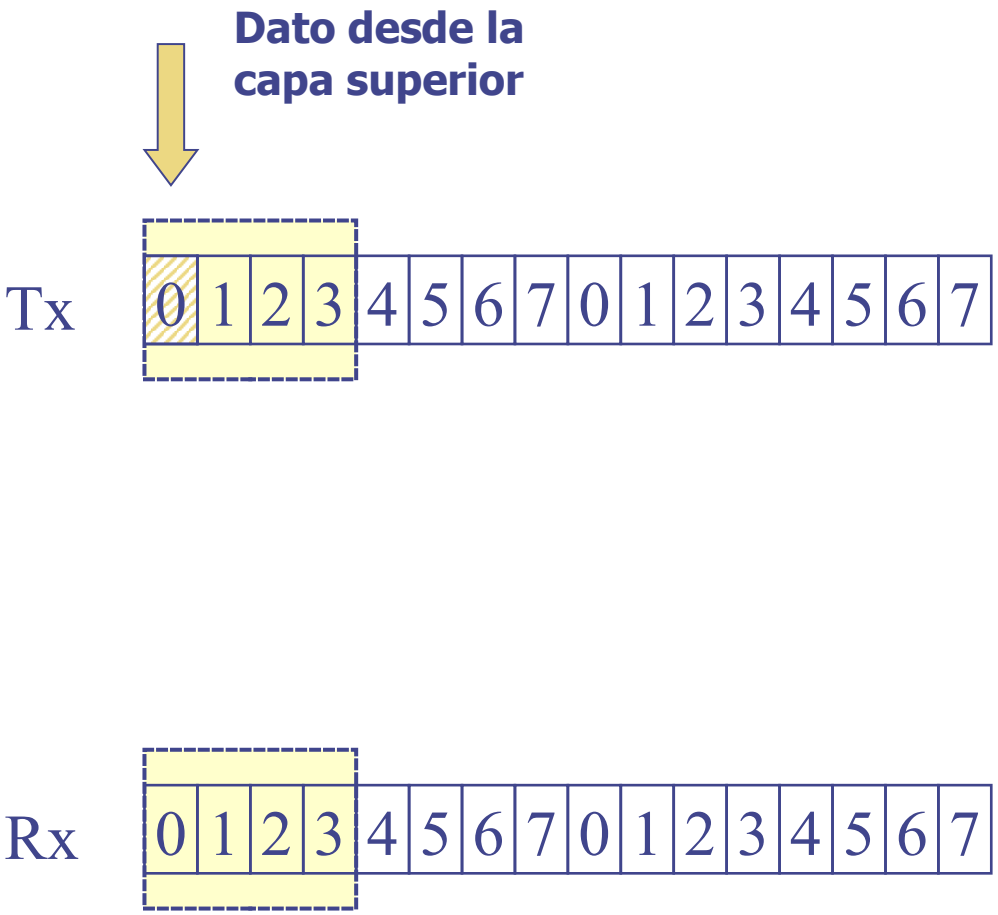
Ventanas Delizantes



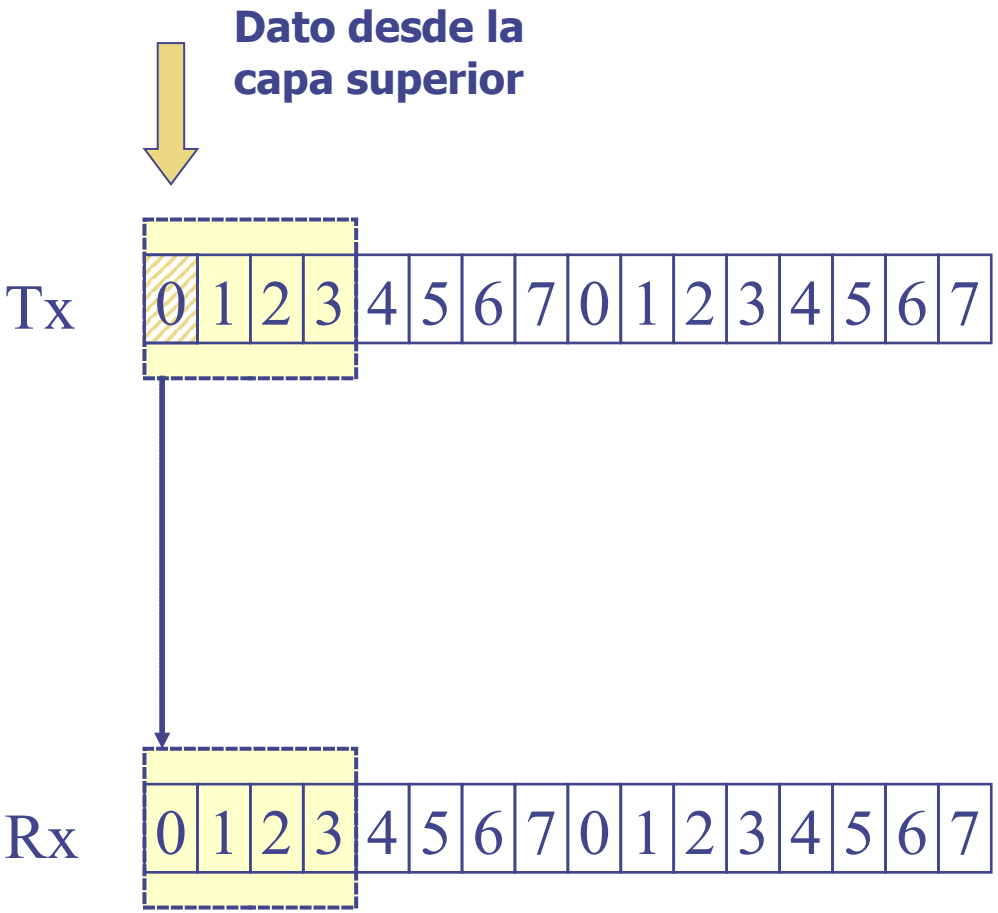
Ventanas Delizantes



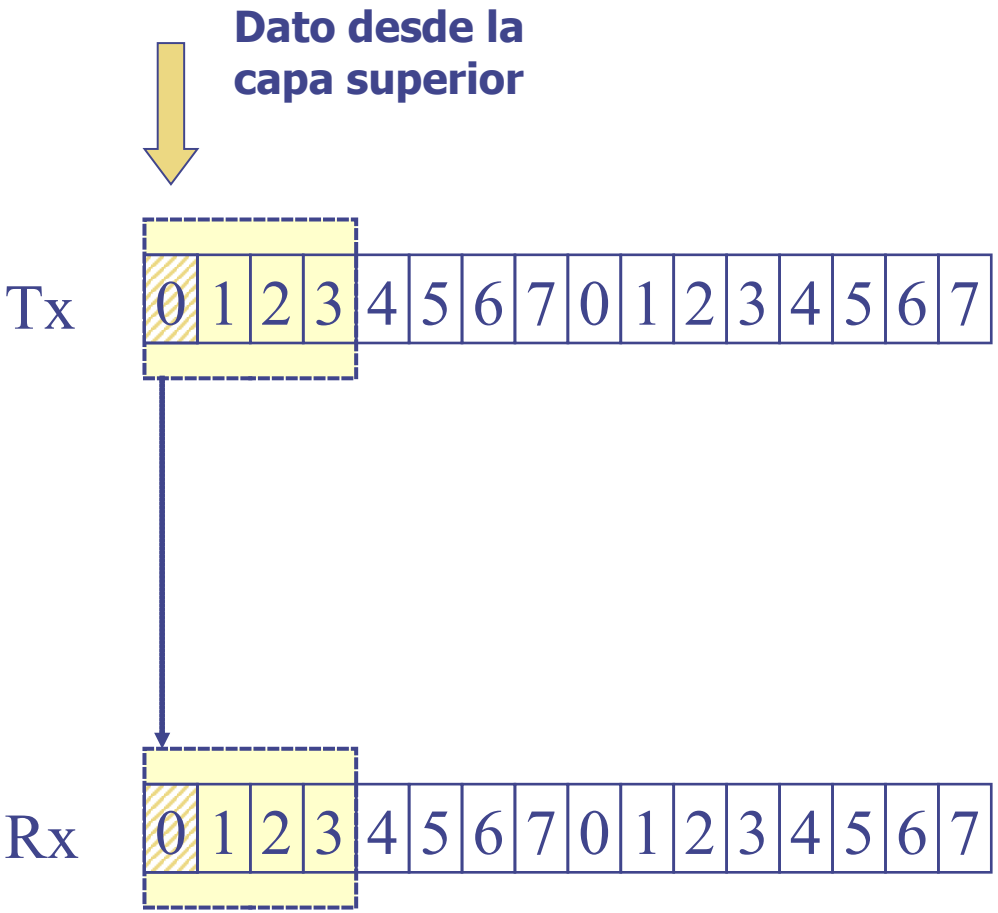
Ventanas Delizantes



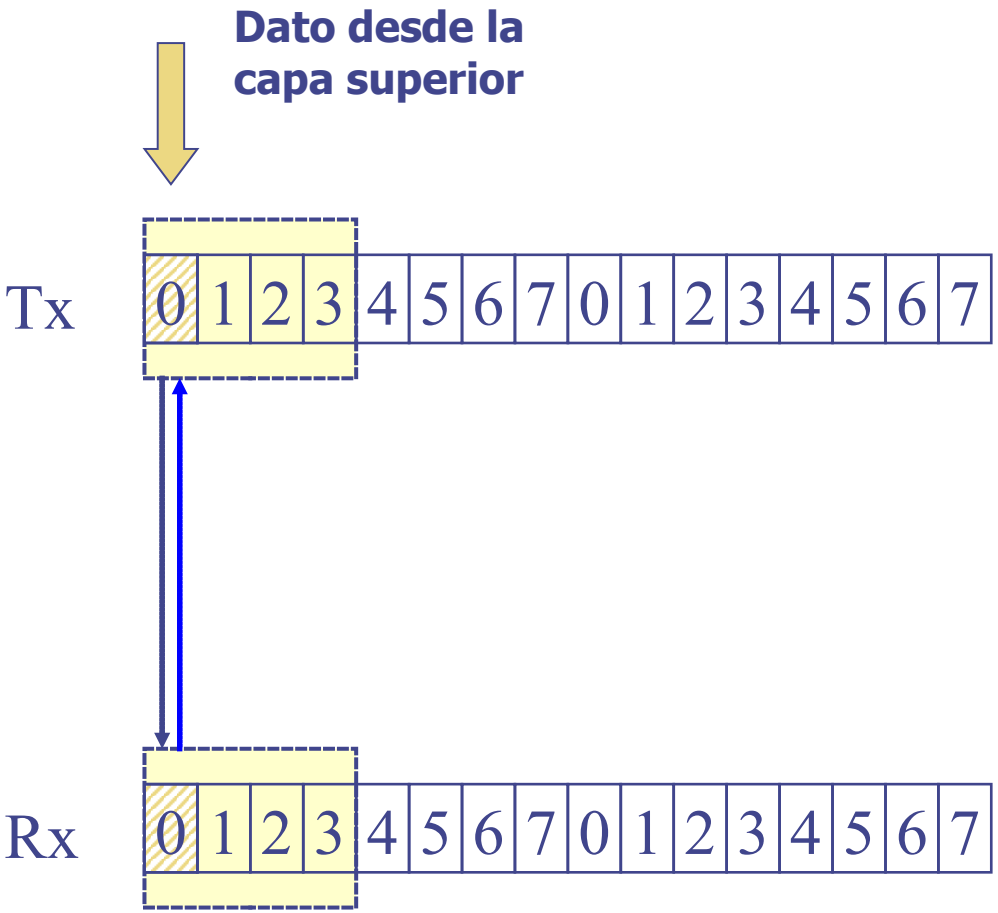
Ventanas Delizantes



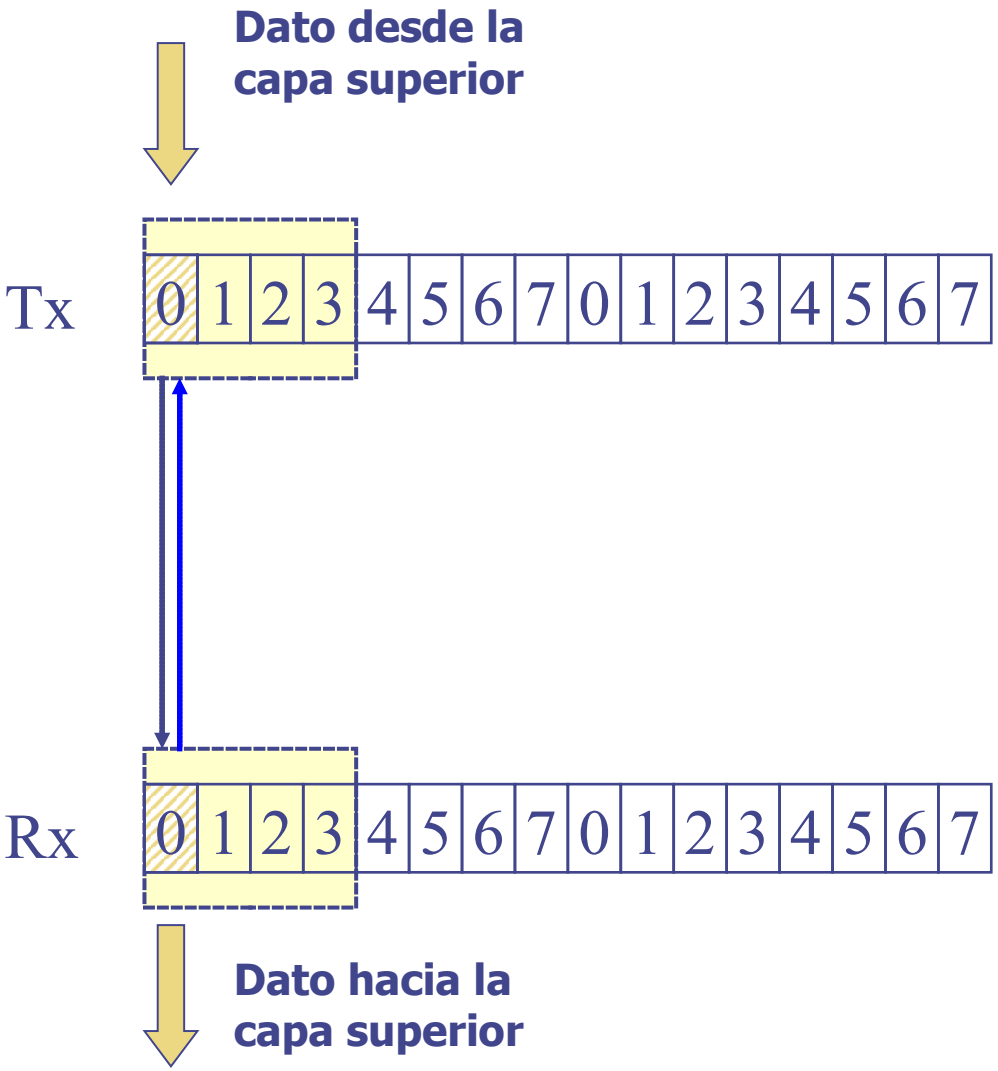
Ventanas Delizantes



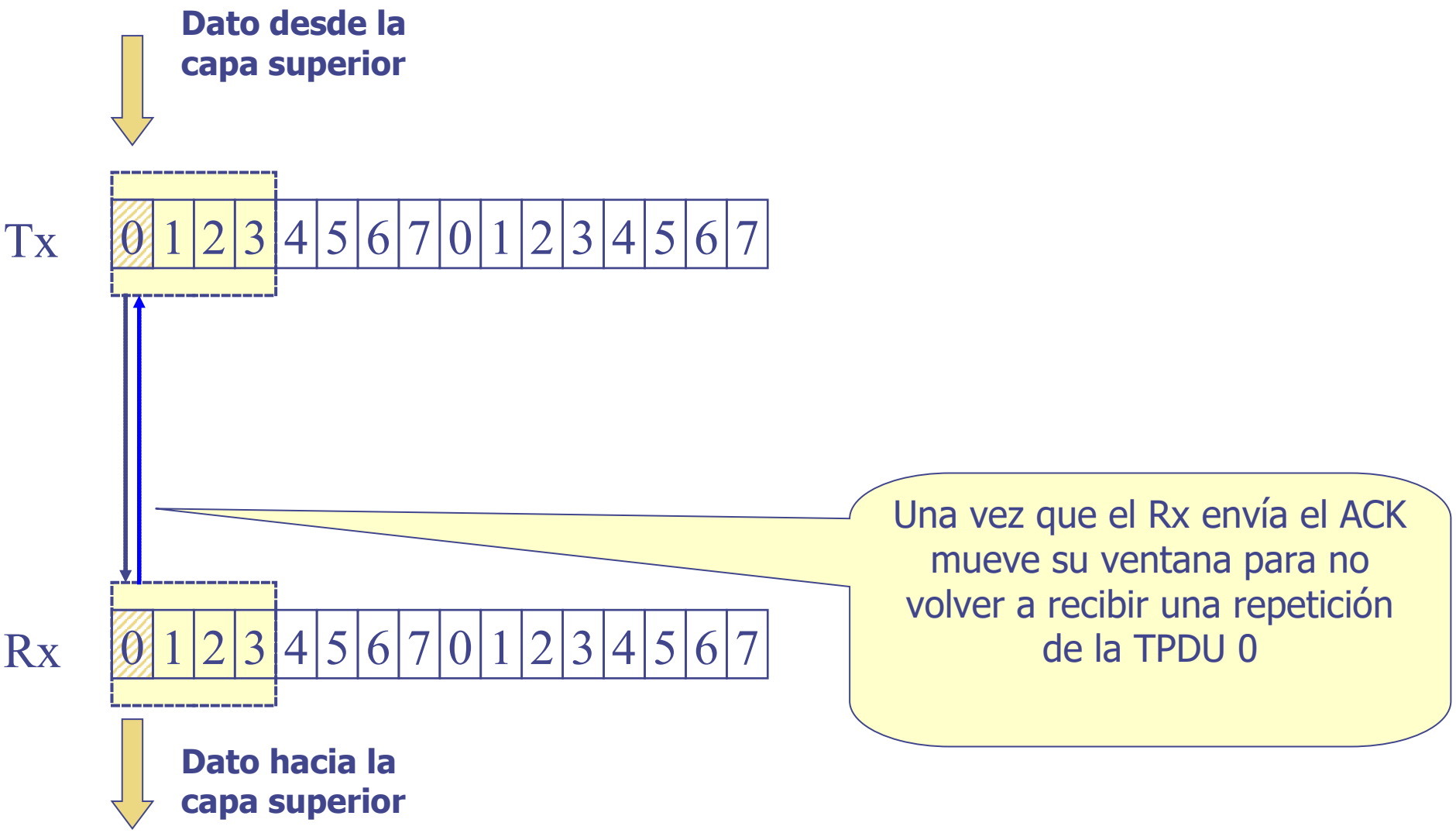
Ventanas Delizantes



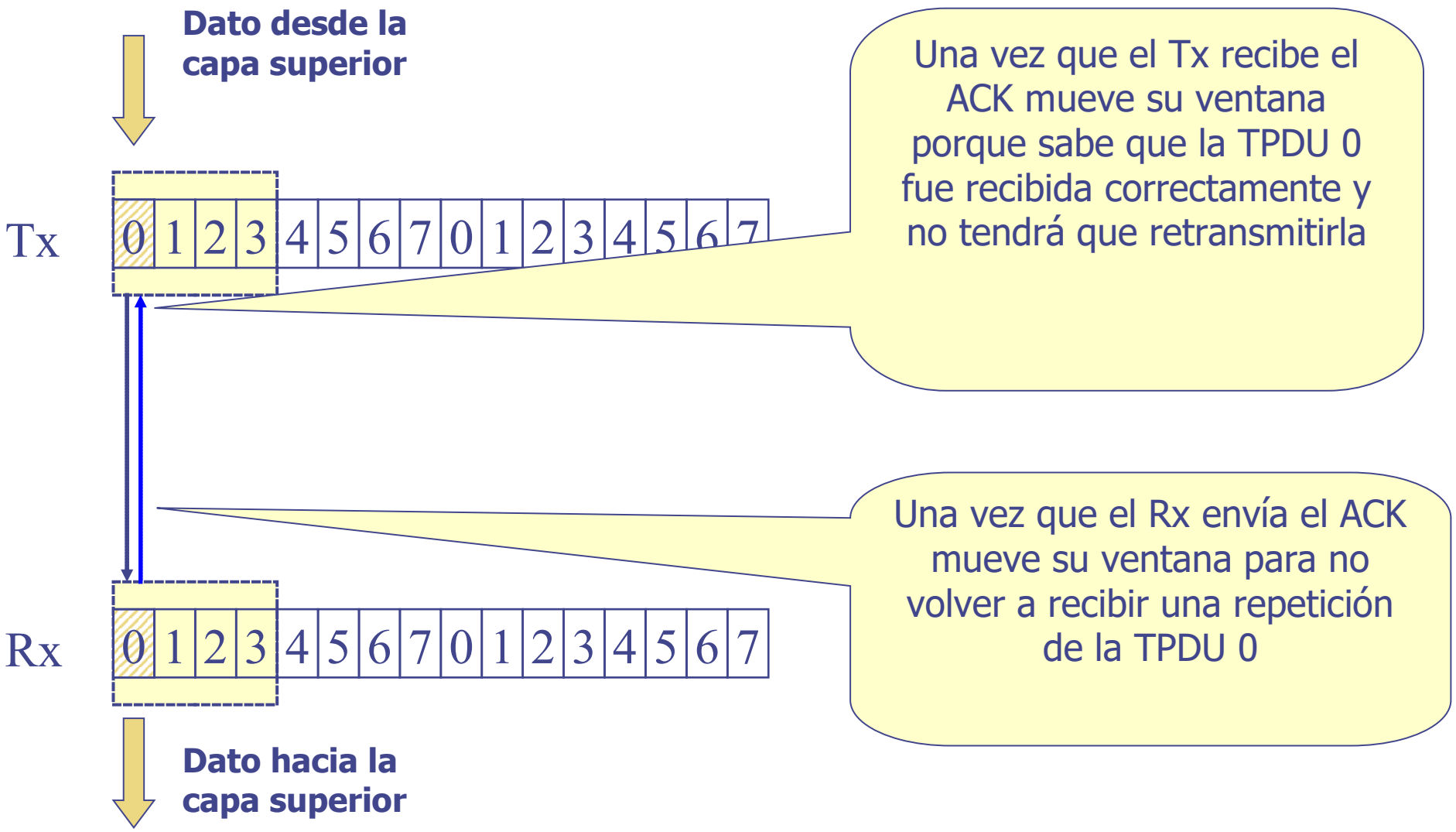
Ventanas Delizantes



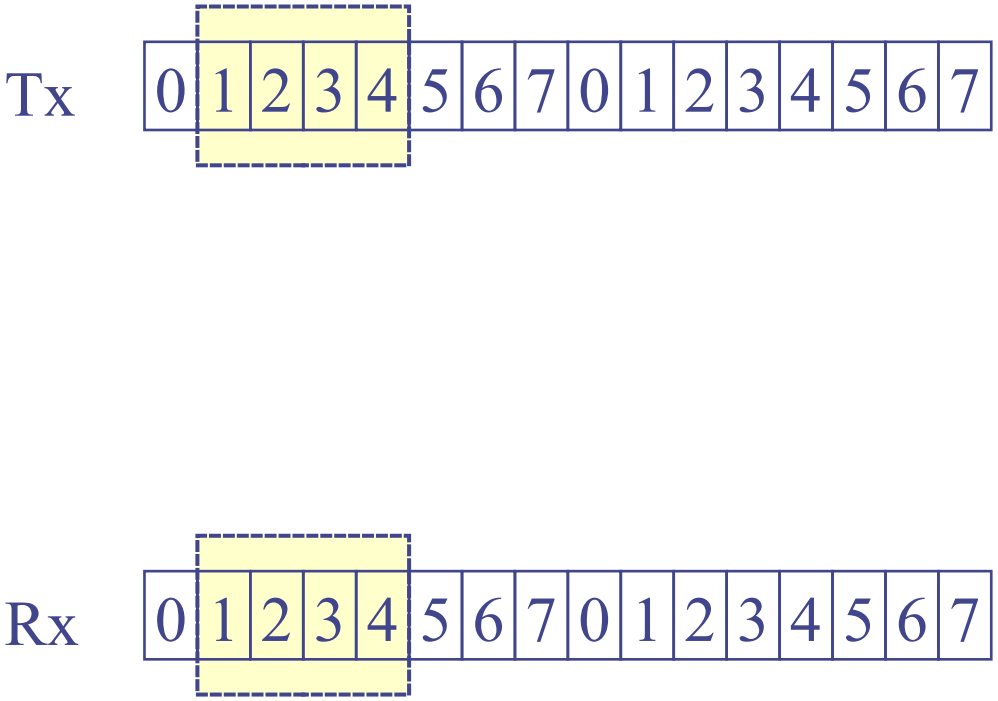
Ventanas Delizantes



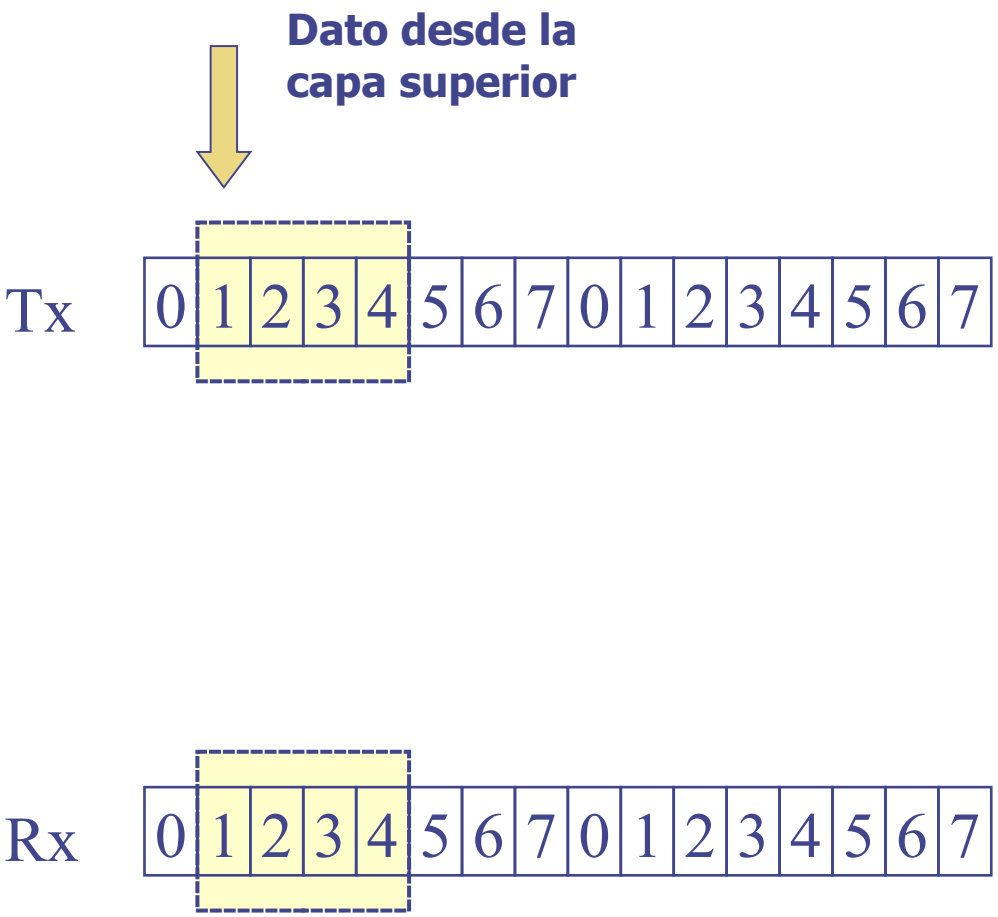
Ventanas Delizantes



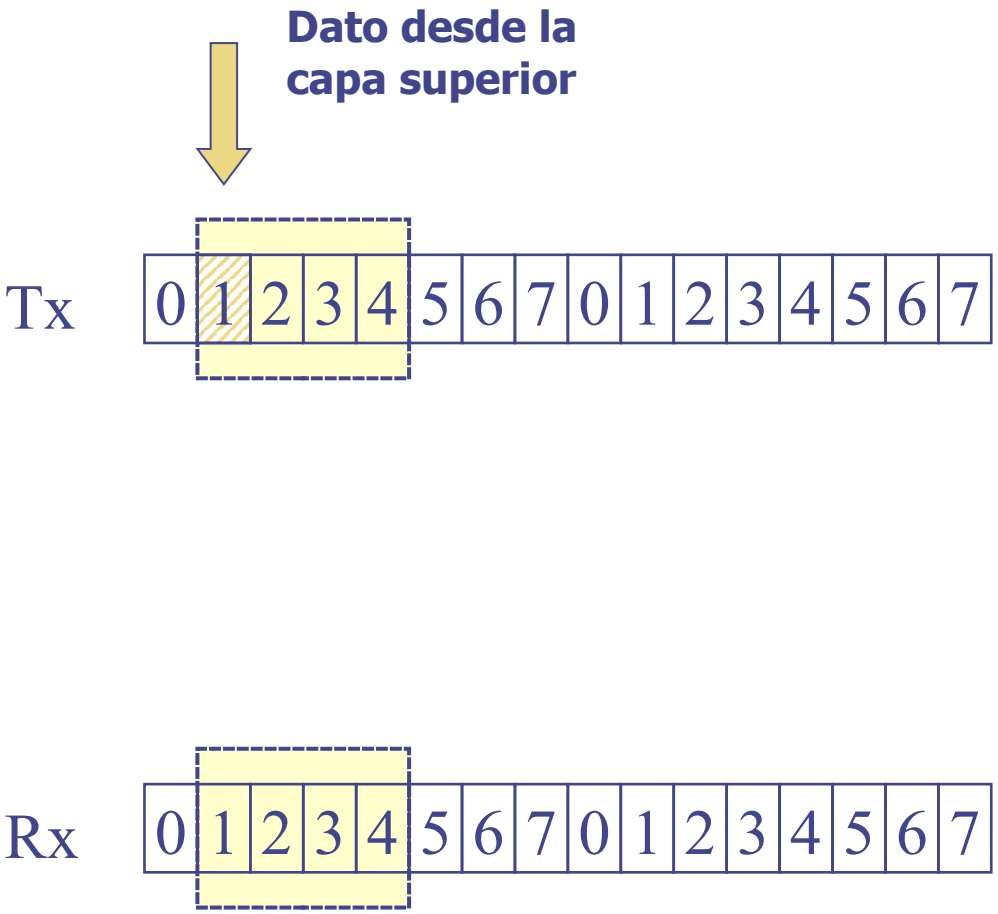
Ventanas Delizantes



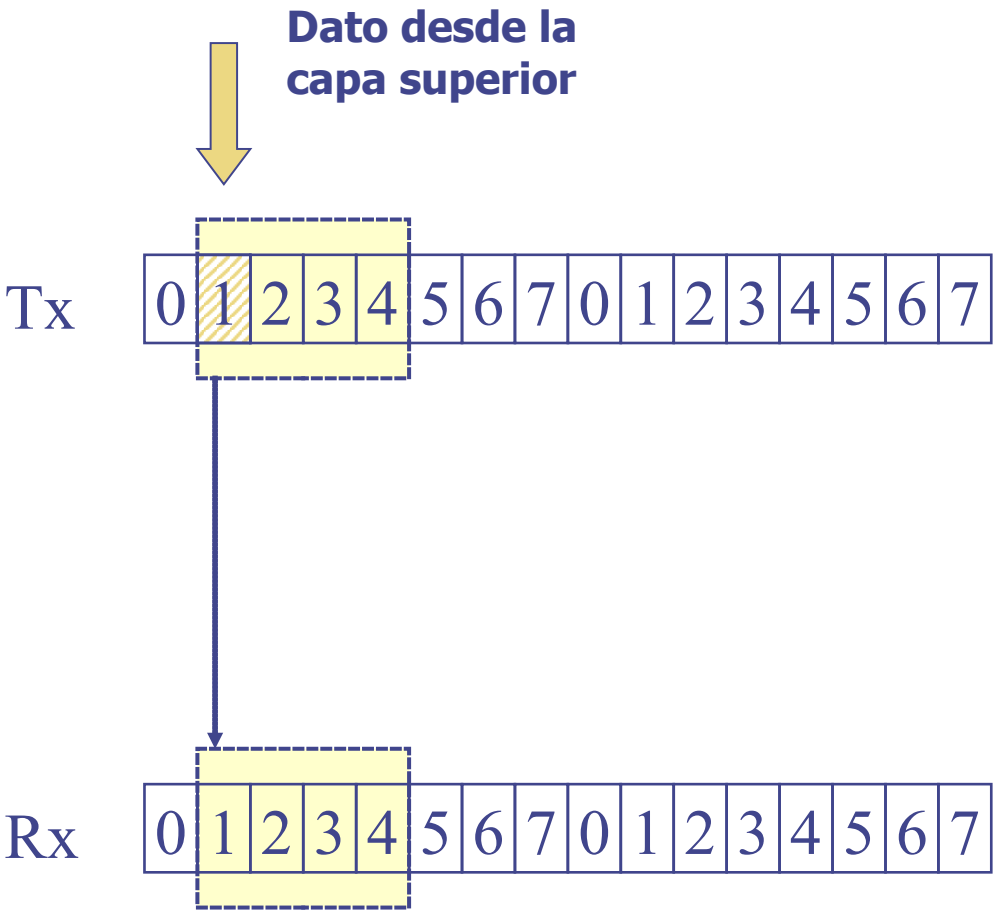
Ventanas Delizantes



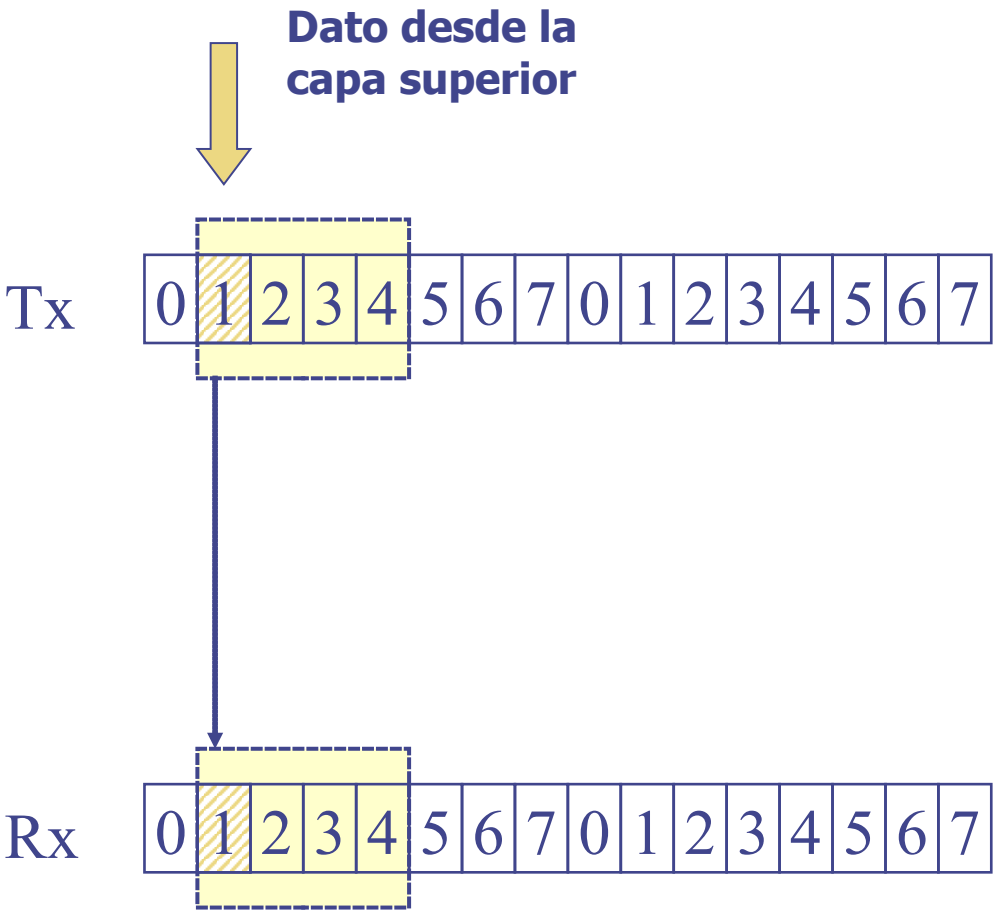
Ventanas Delizantes



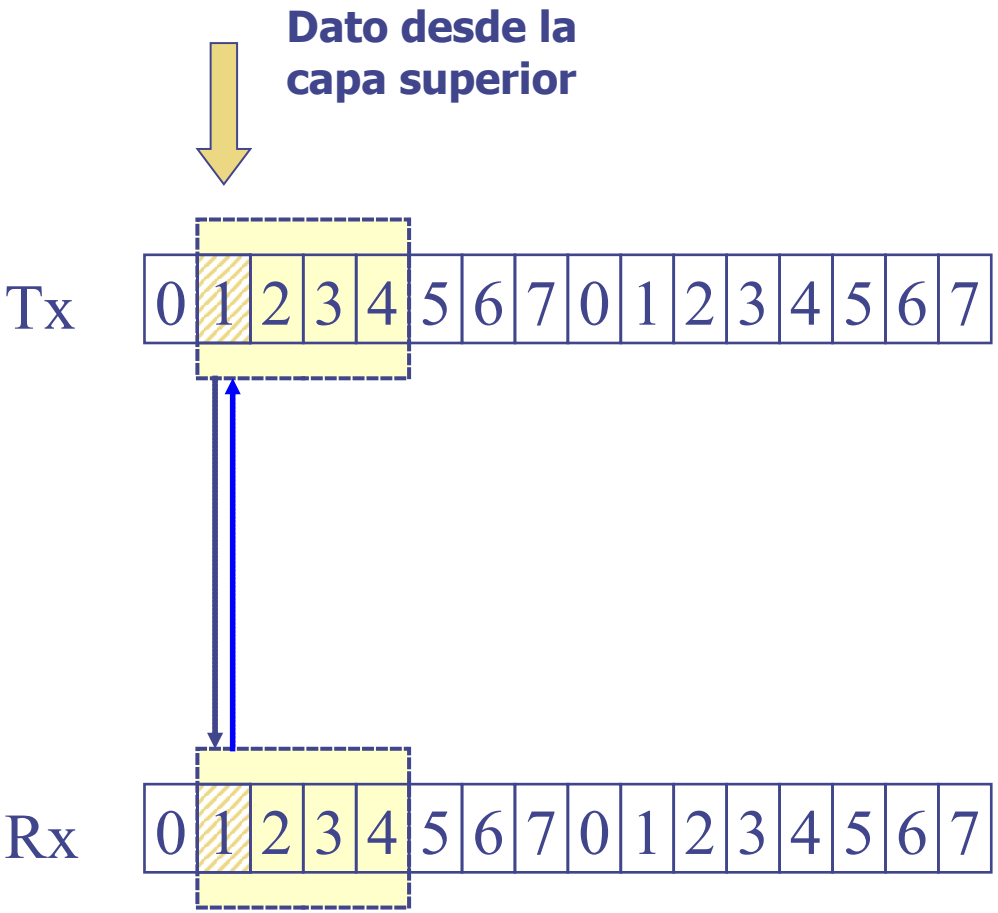
Ventanas Delizantes



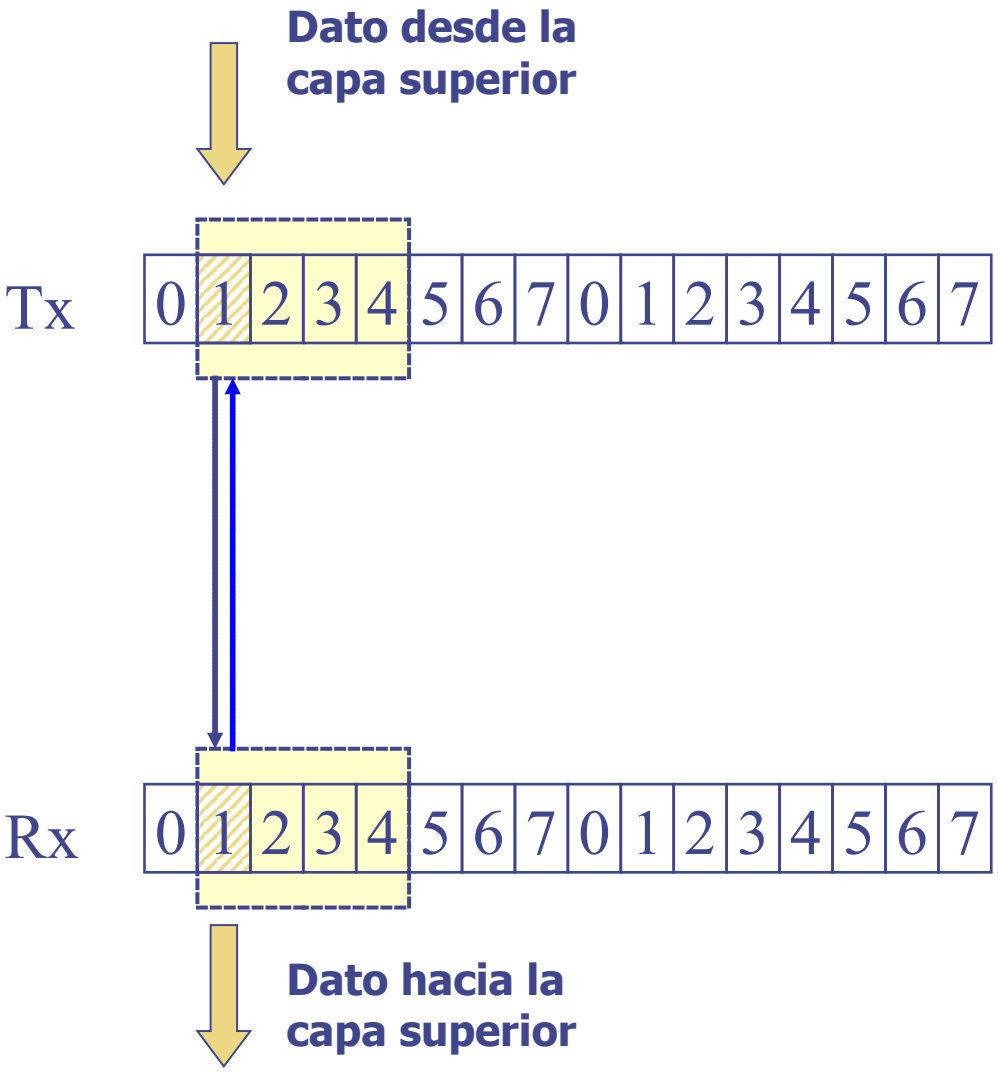
Ventanas Delizantes



Ventanas Delizantes

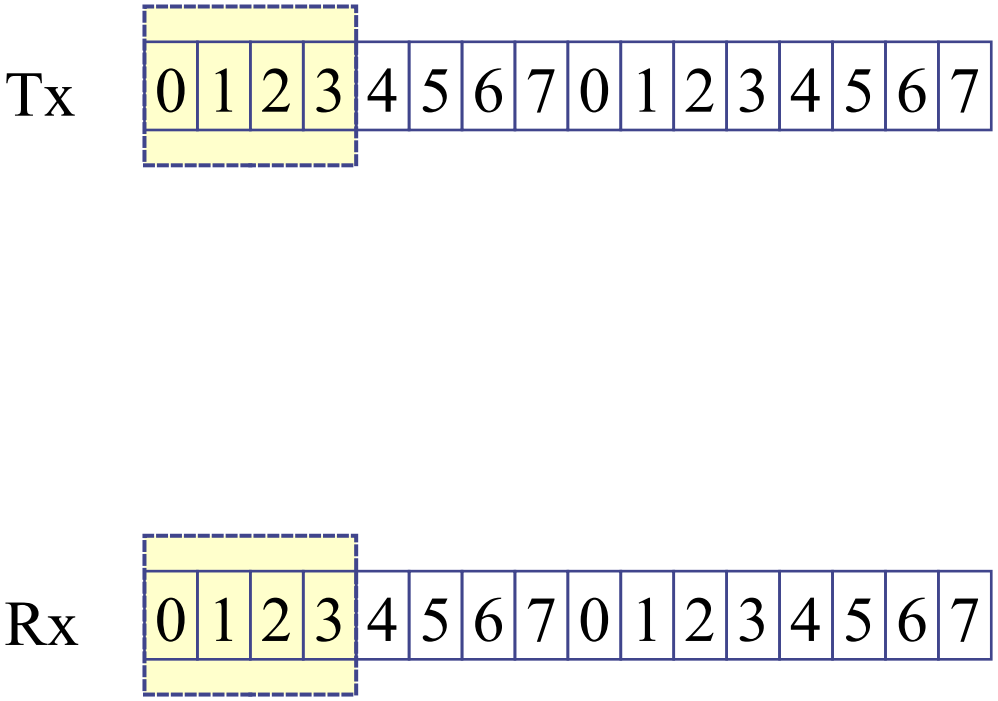


Ventanas Delizantes



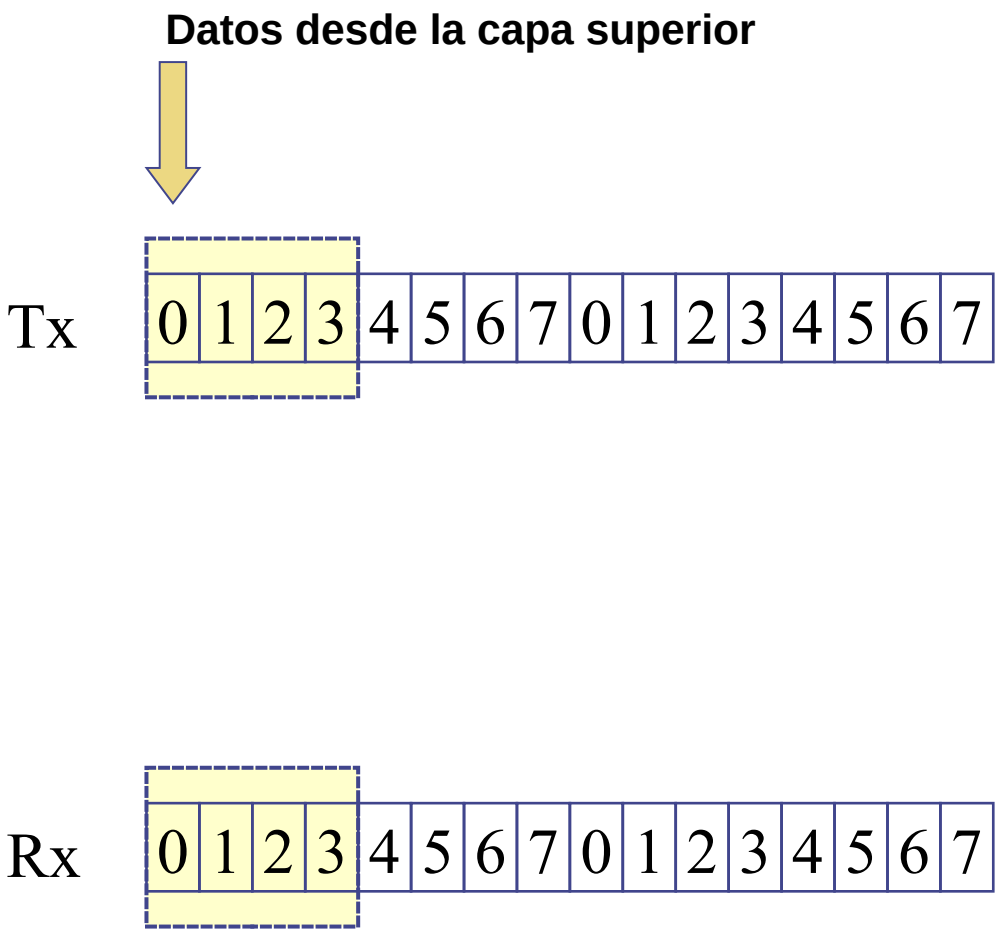
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



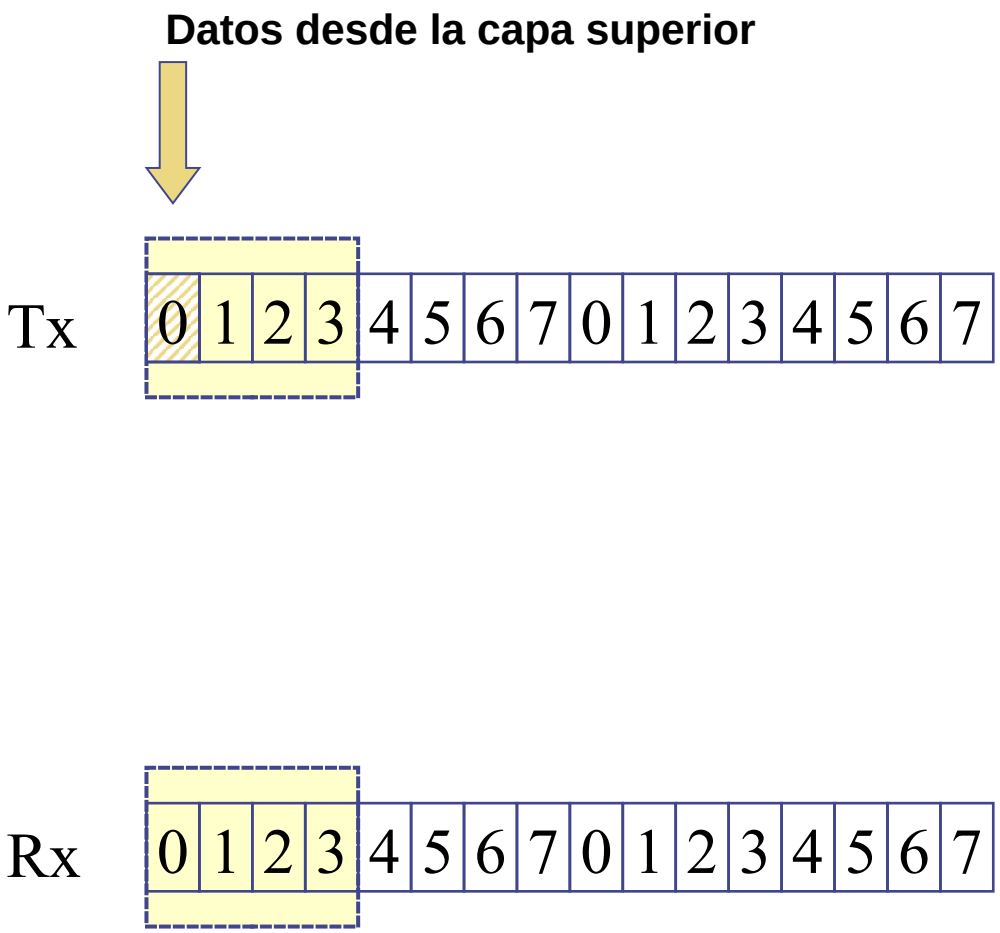
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



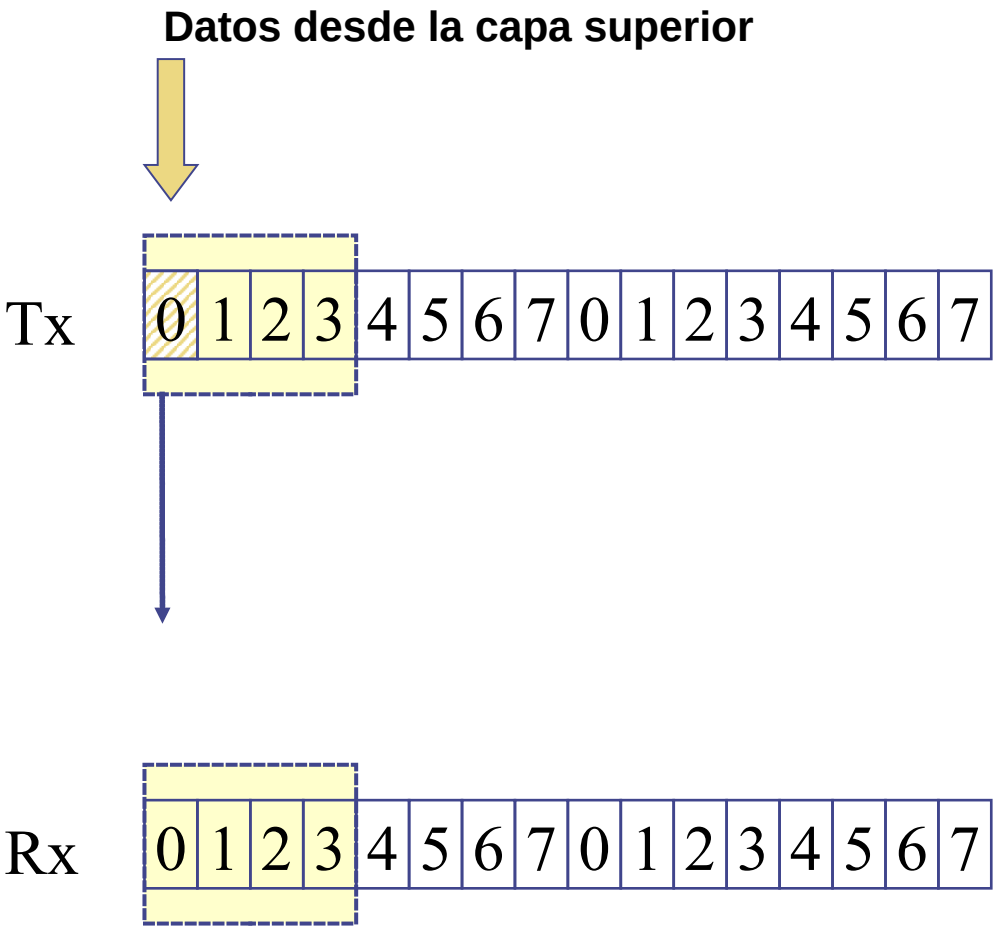
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



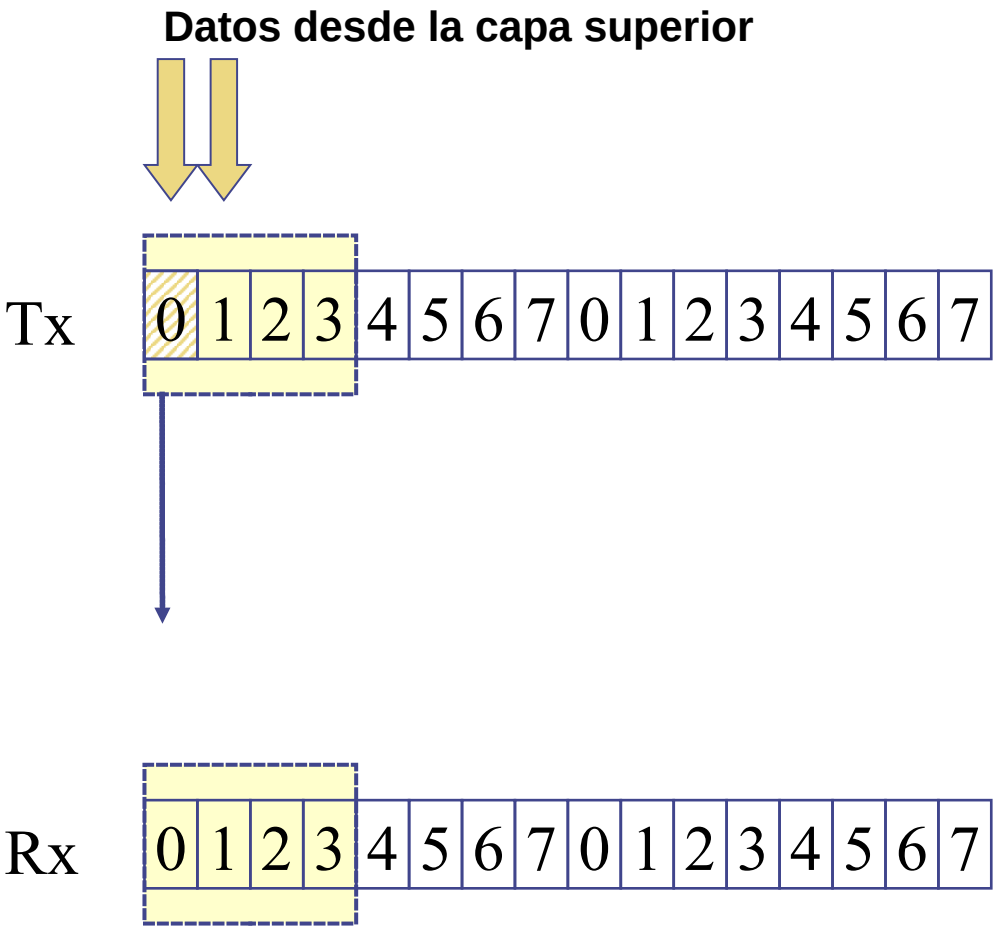
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



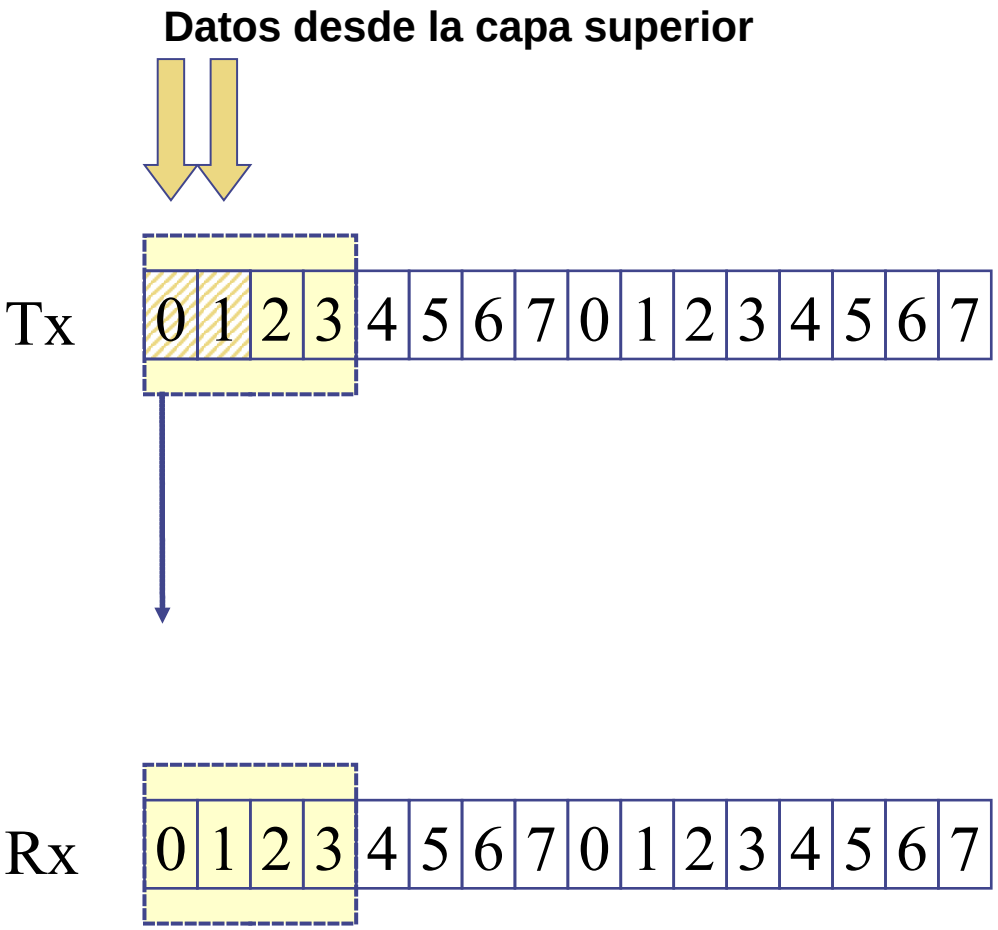
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



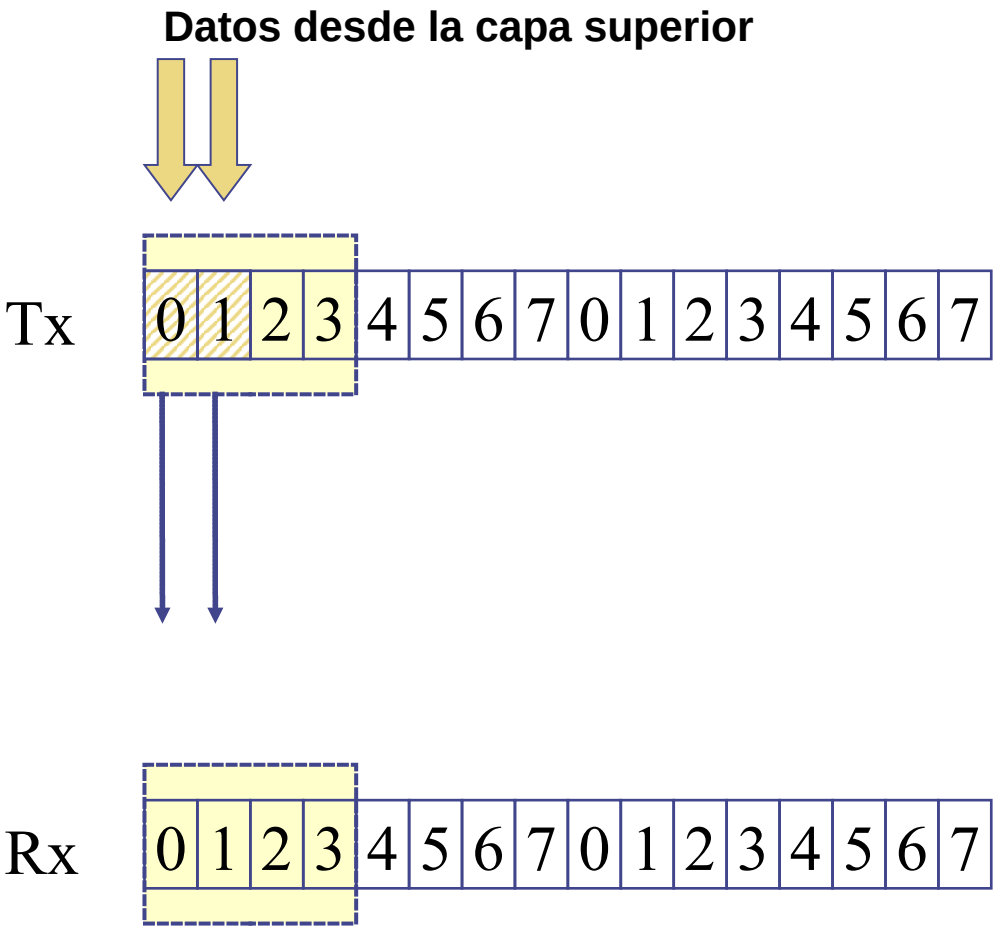
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



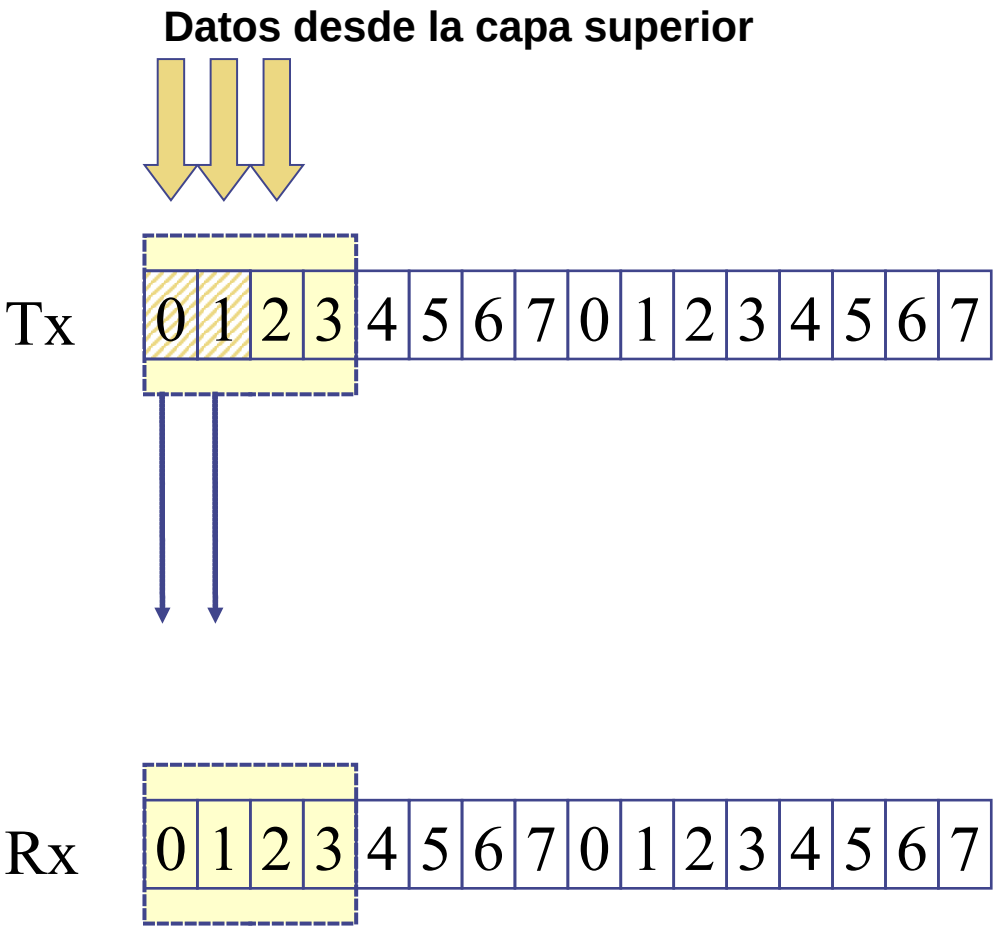
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



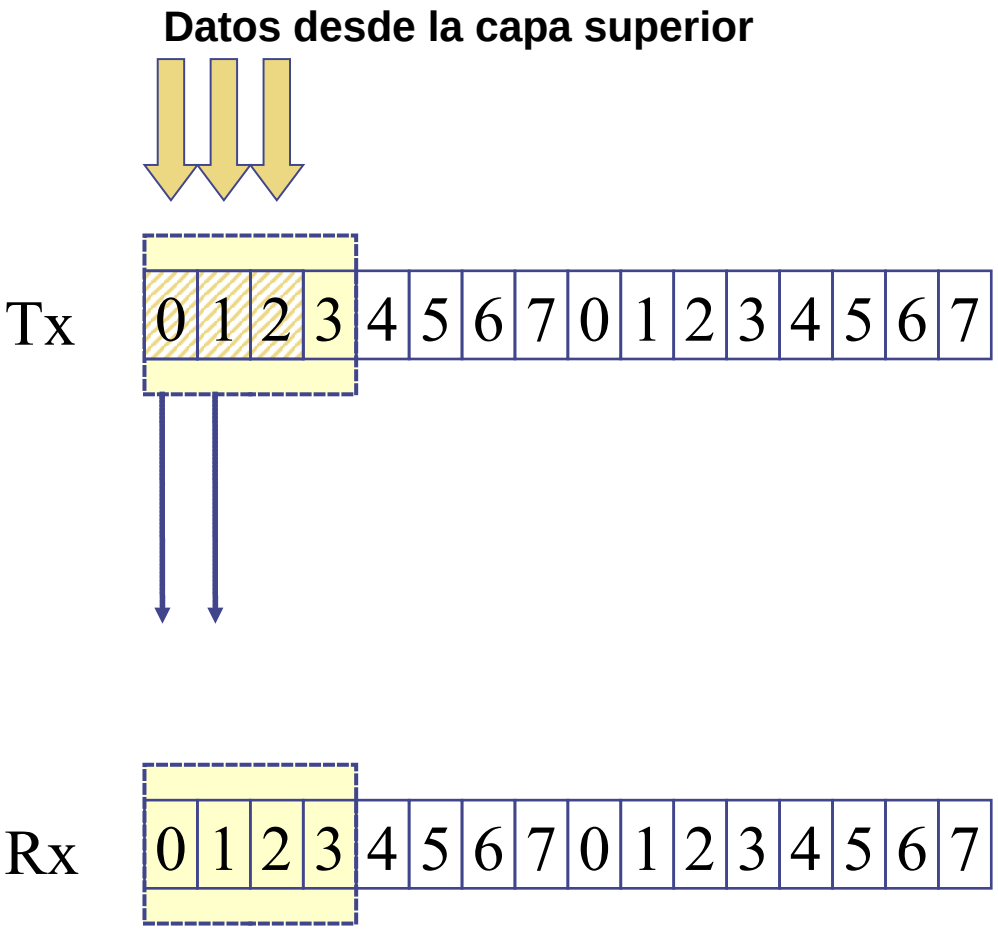
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



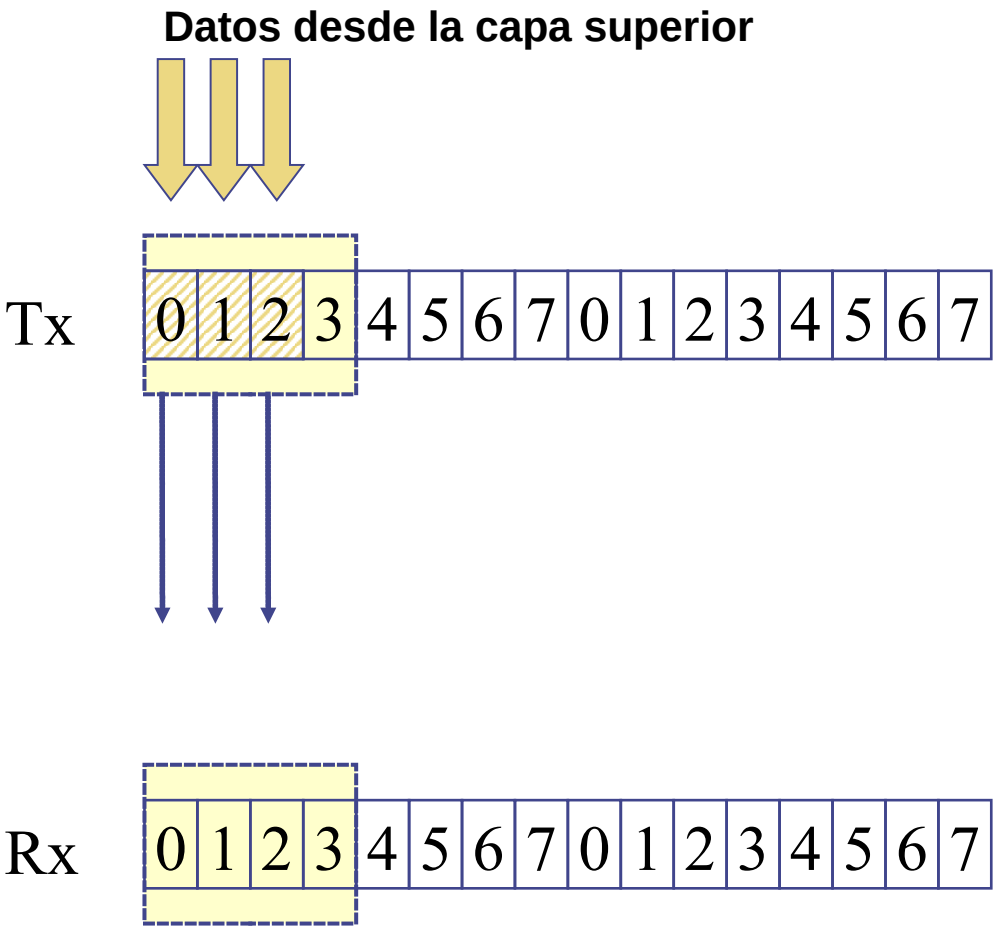
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



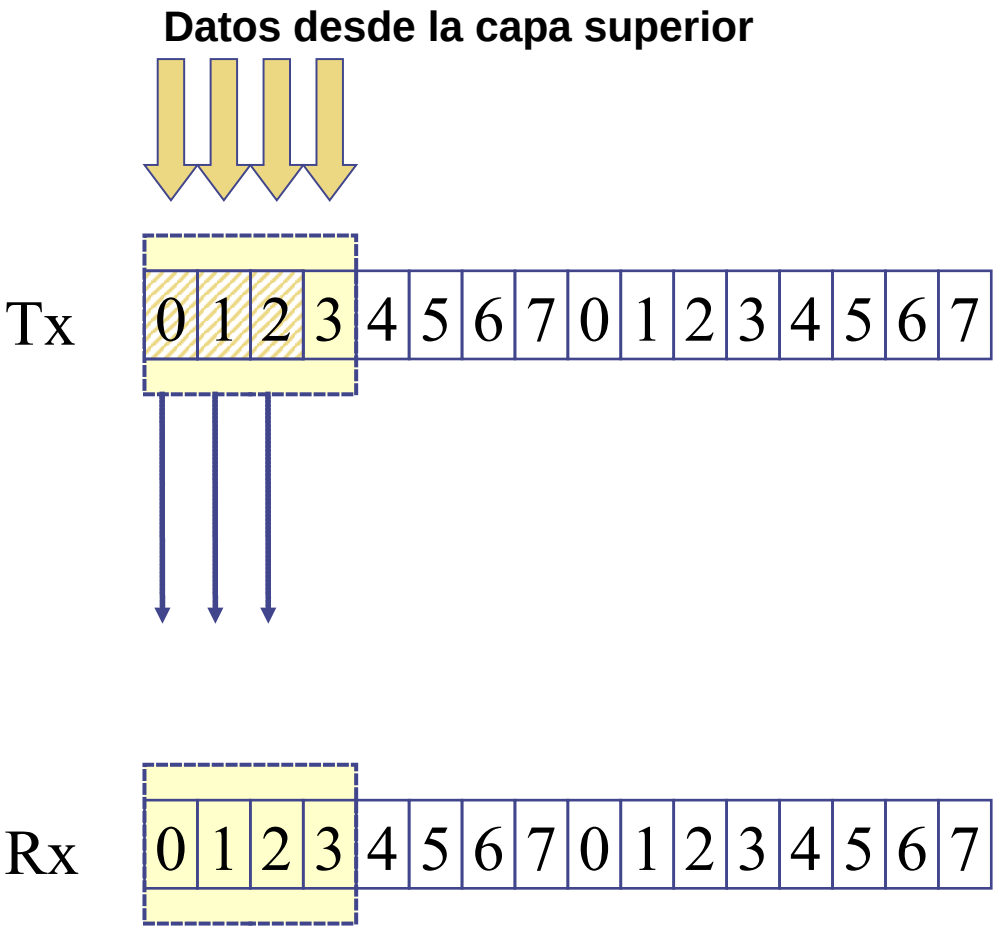
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



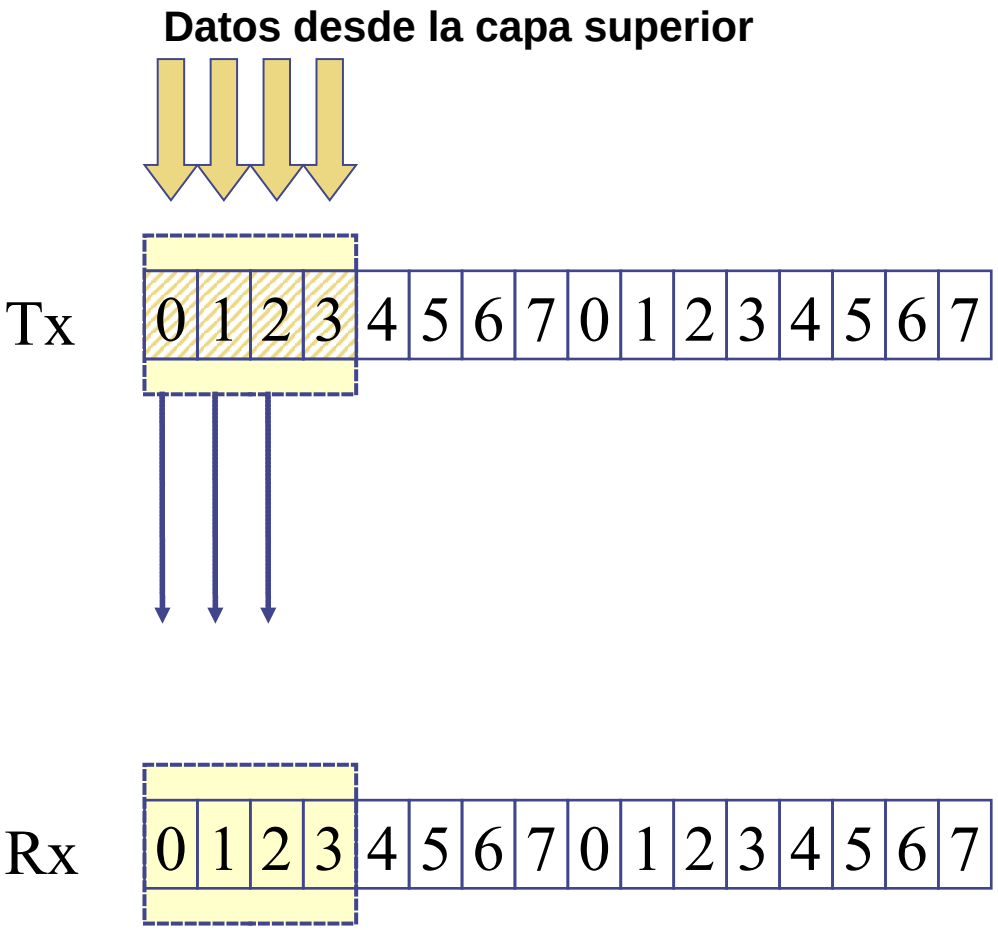
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



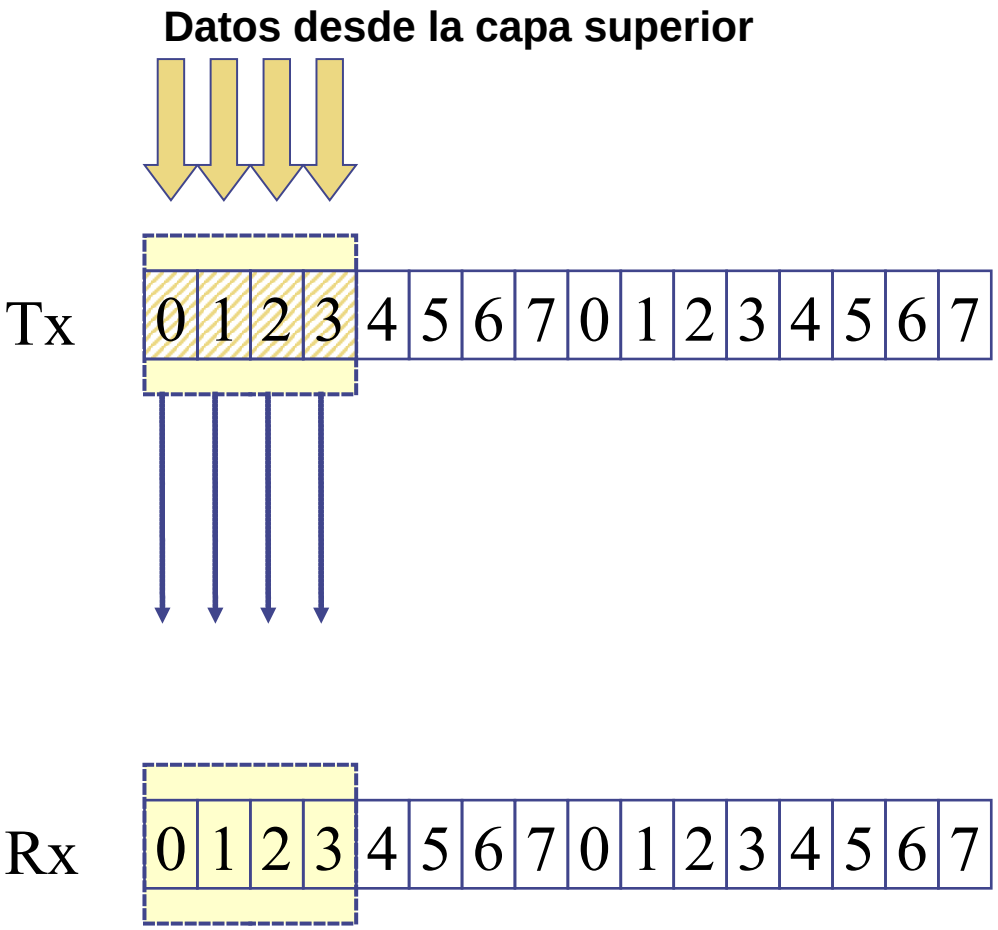
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



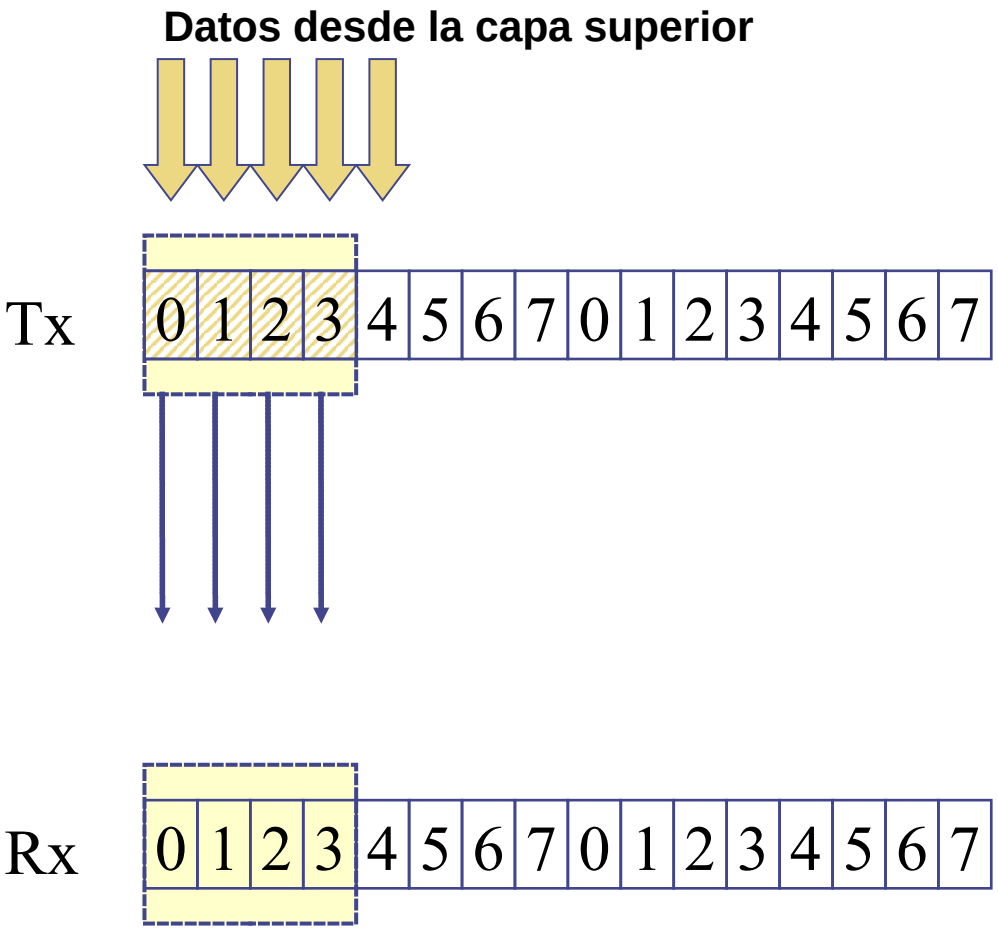
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



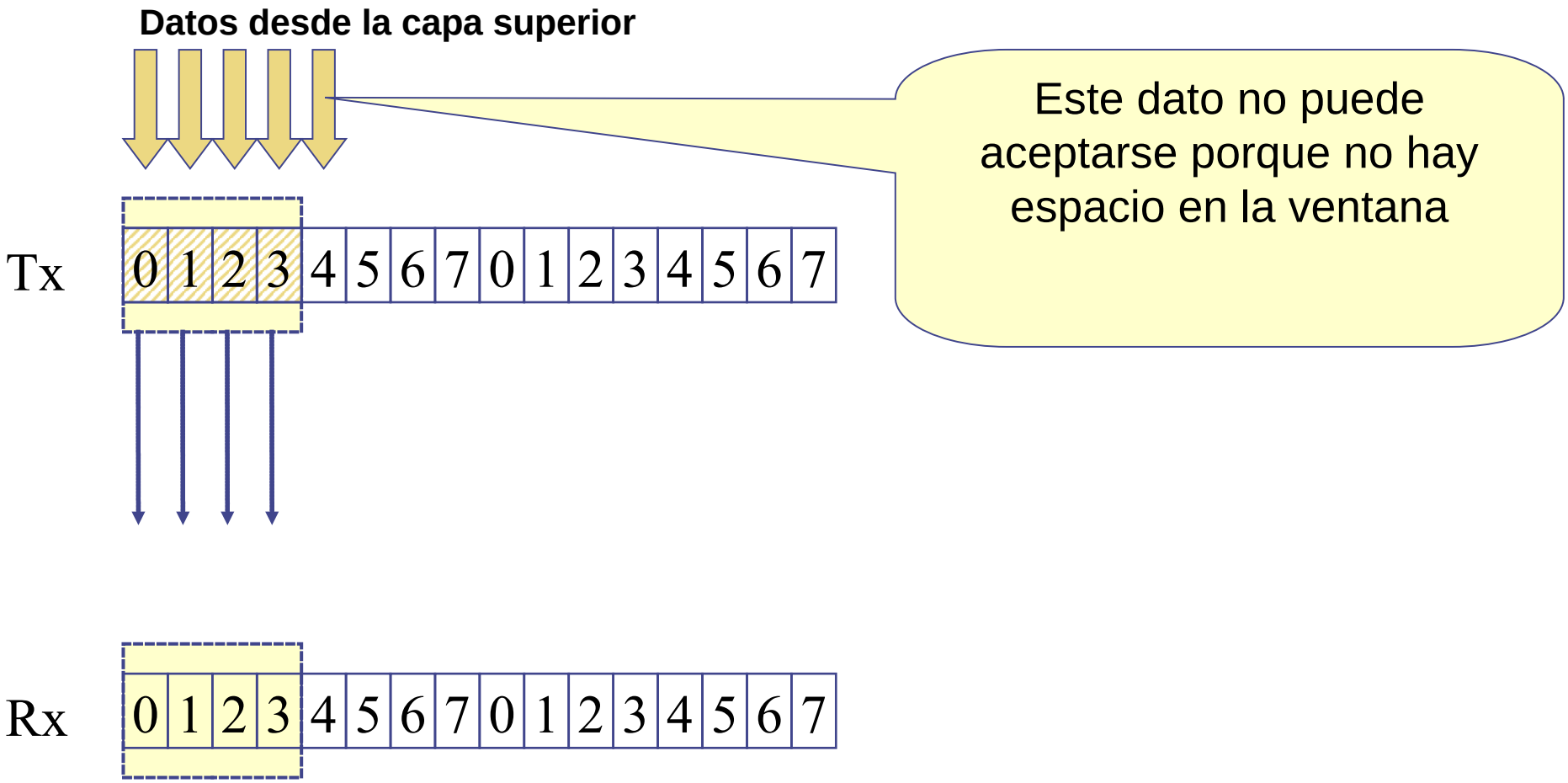
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



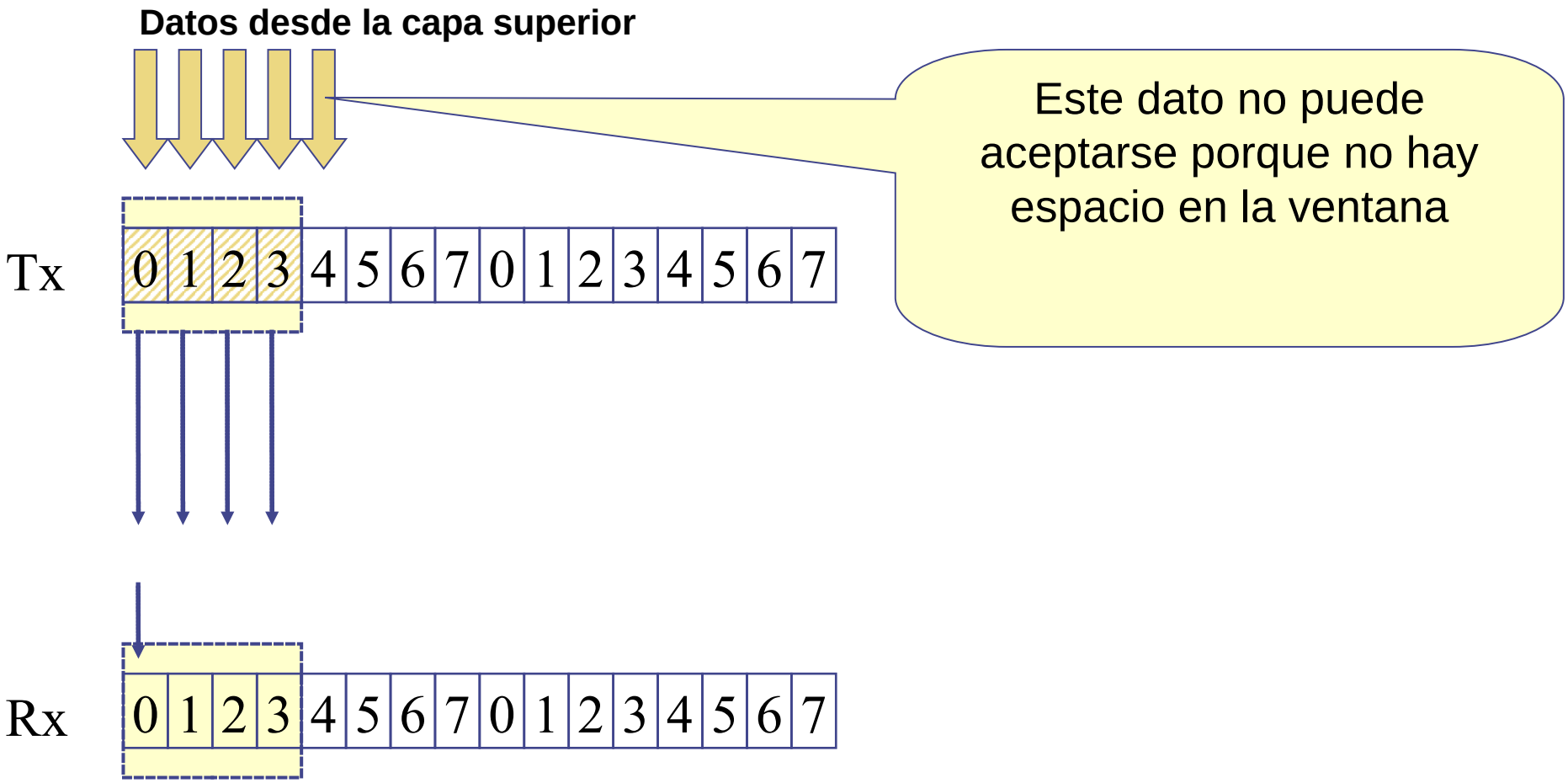
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



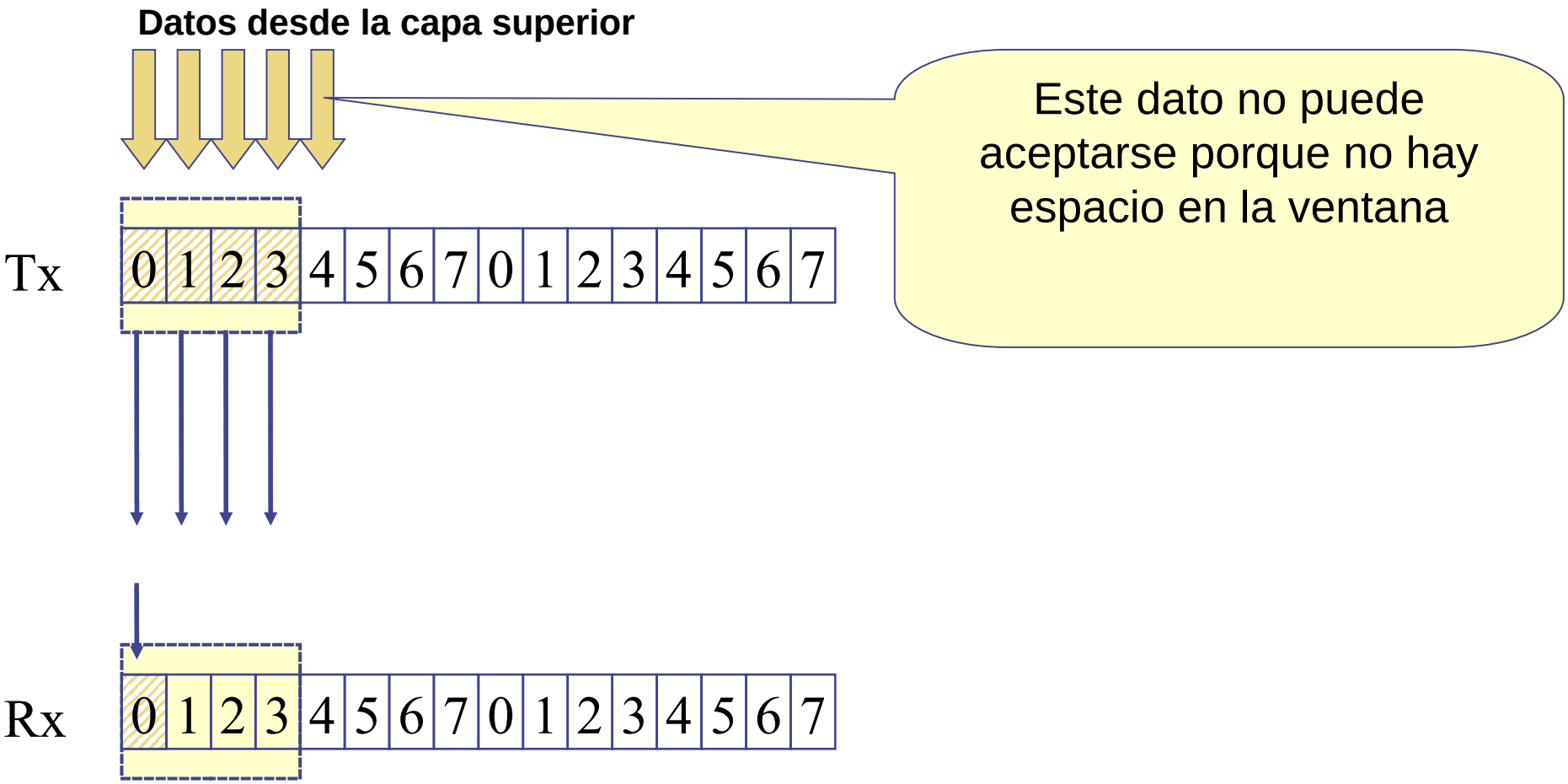
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



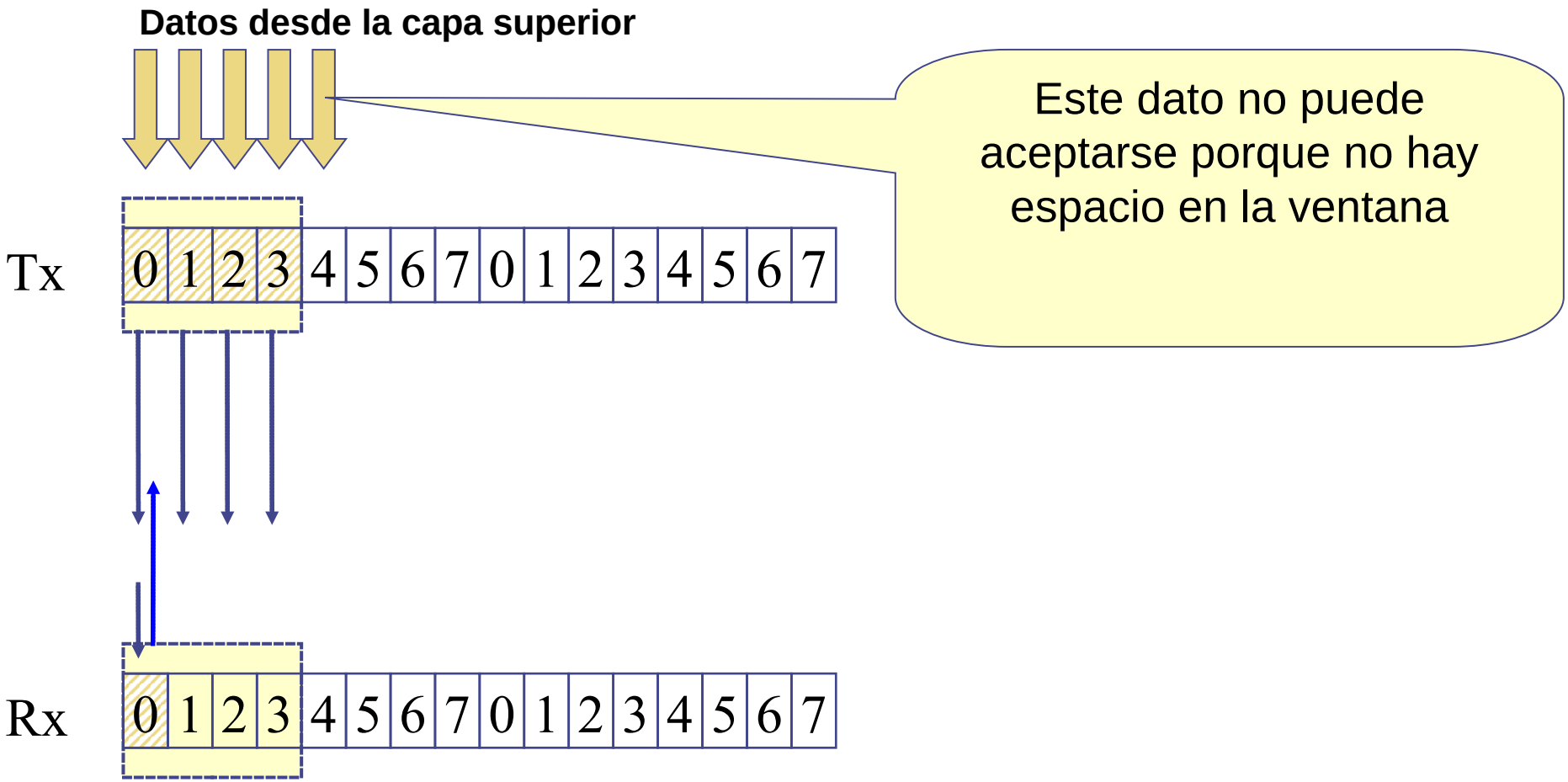
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



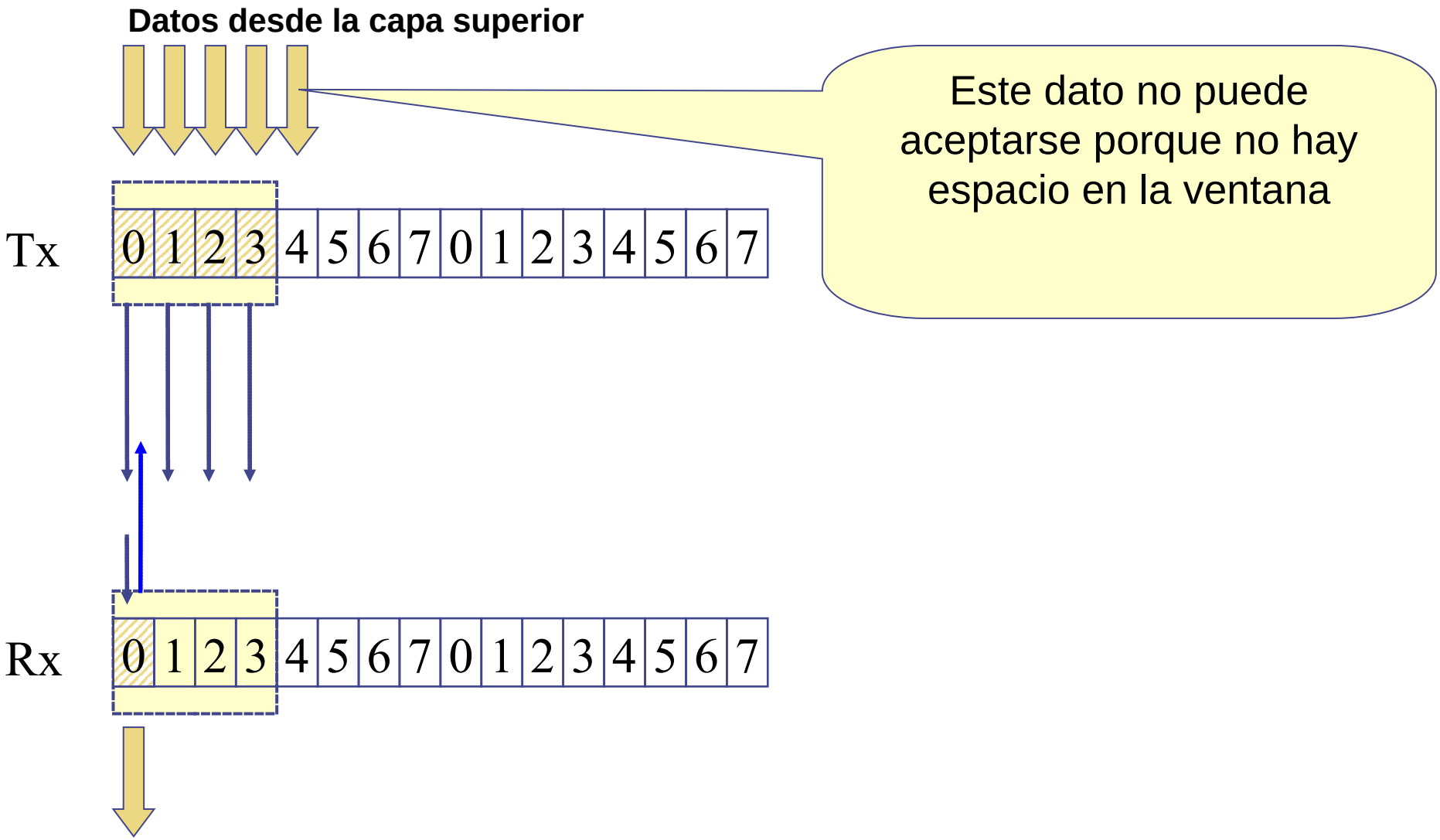
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



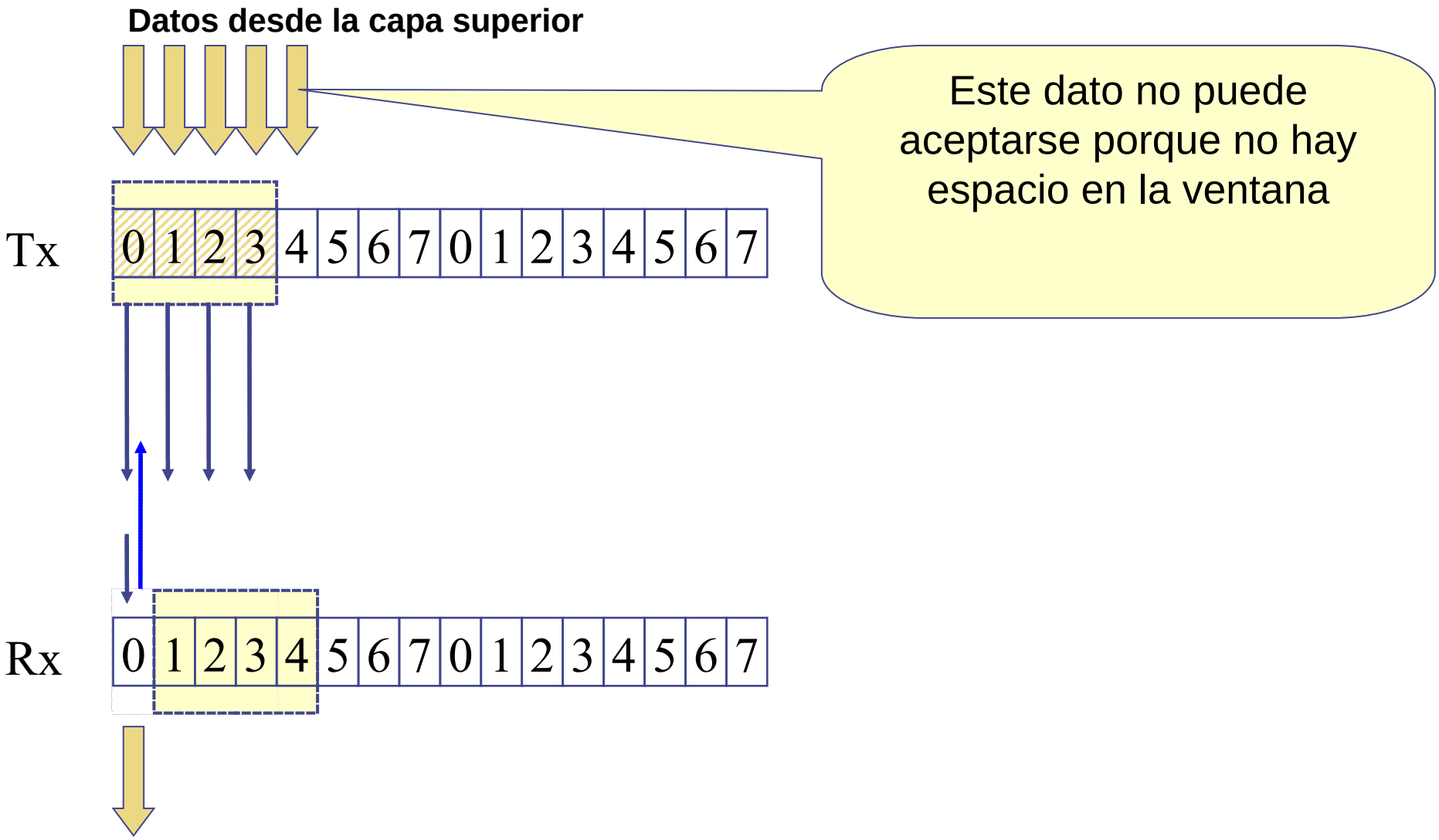
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



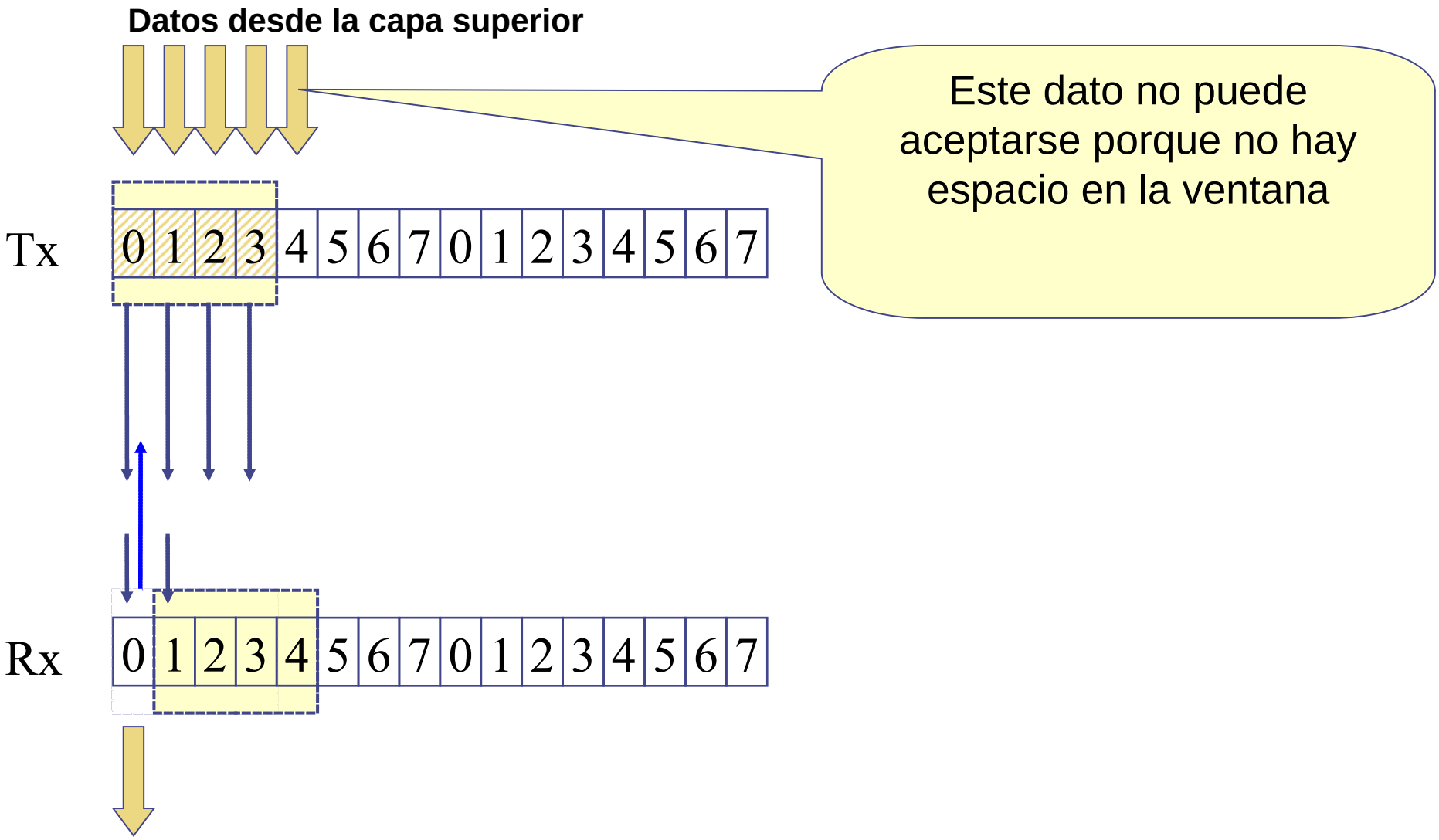
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



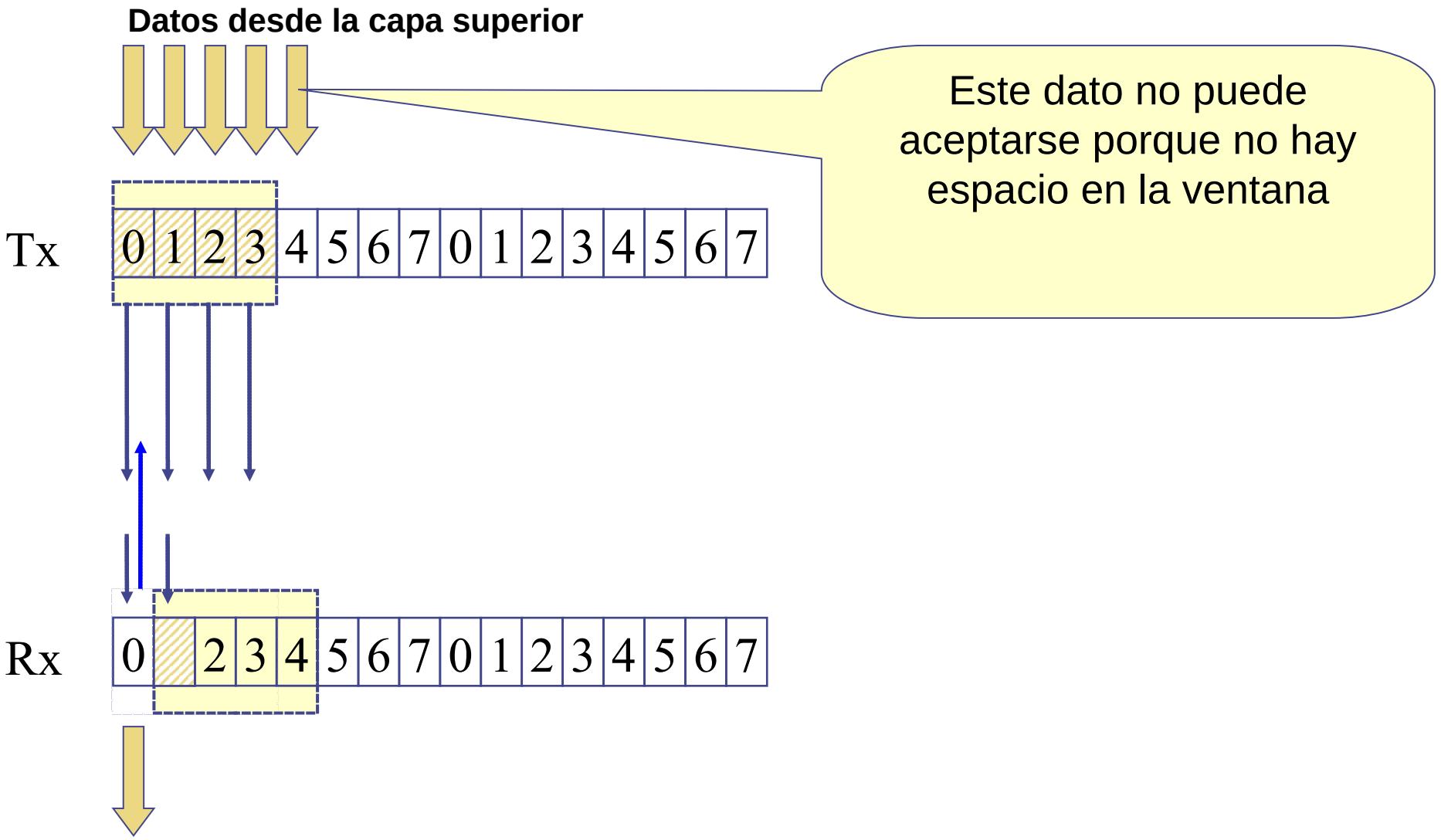
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



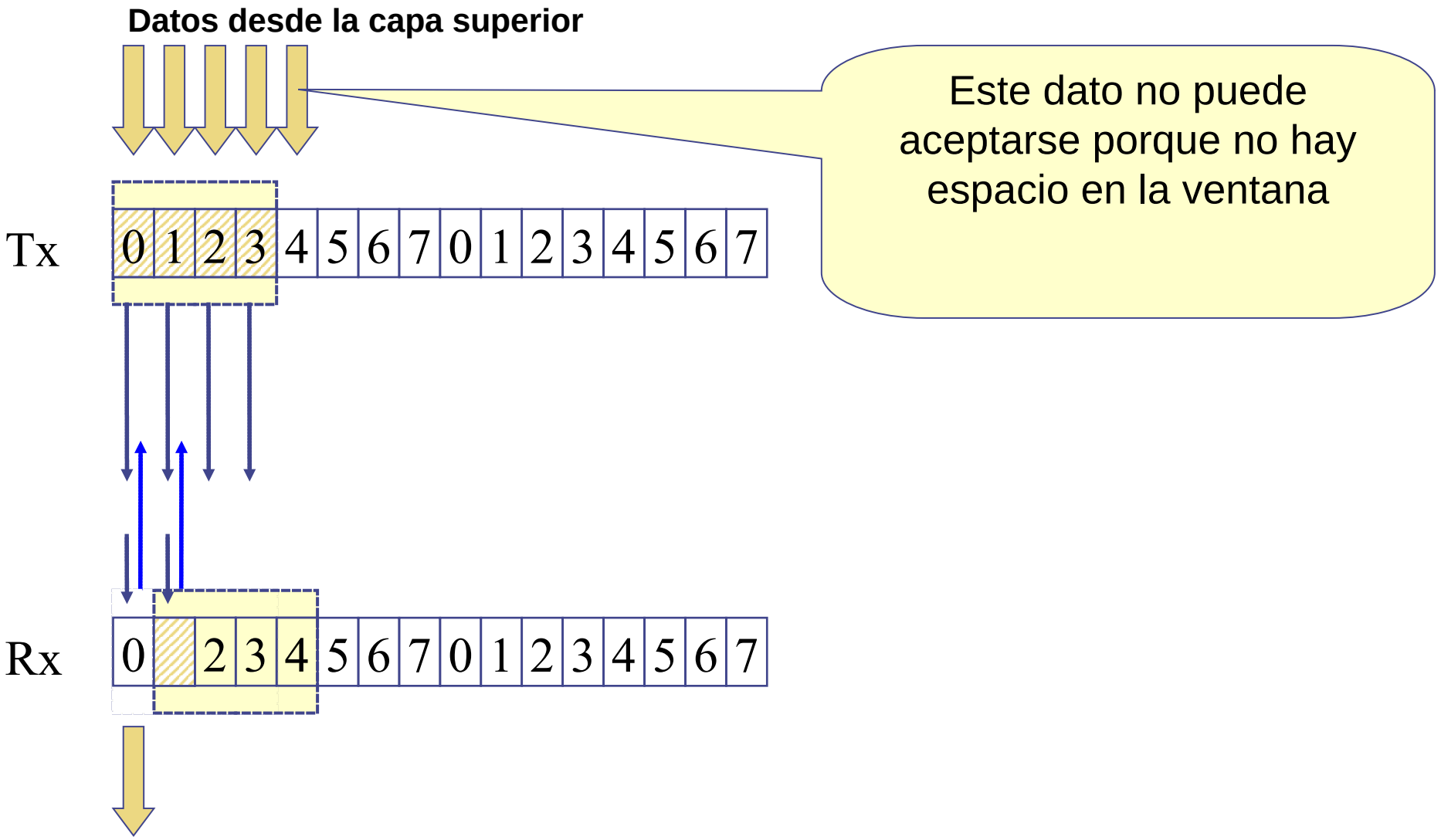
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



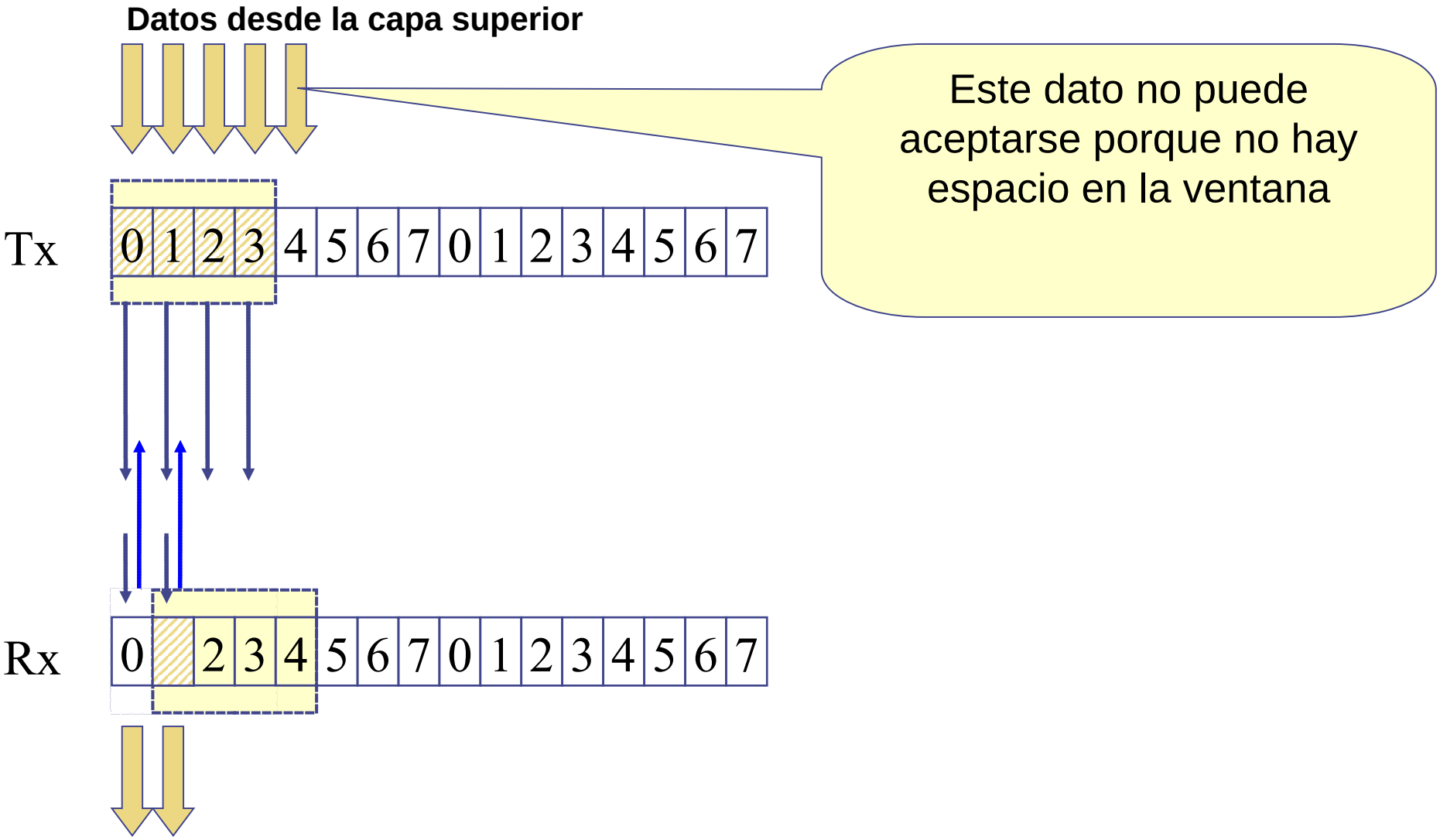
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



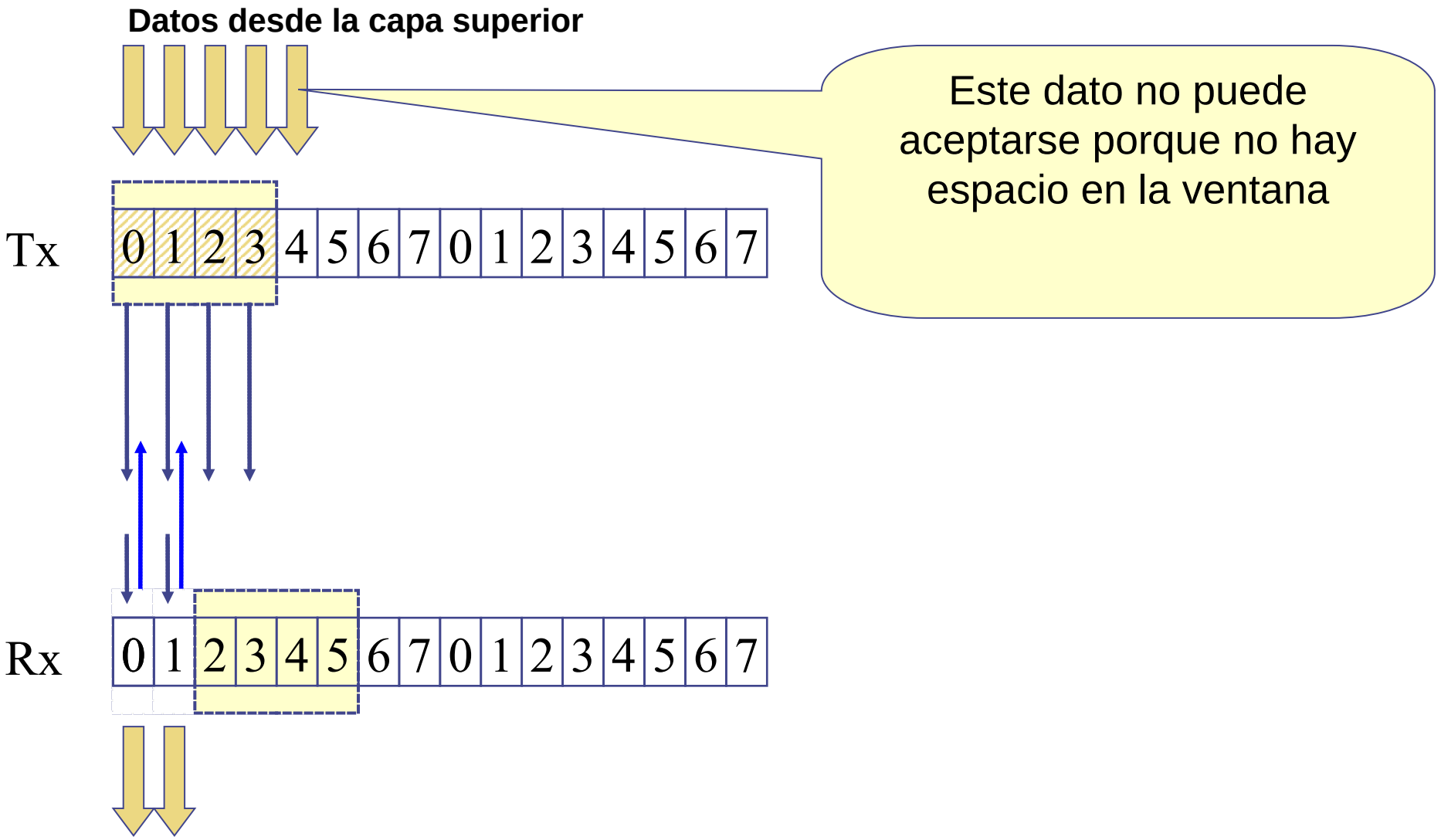
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



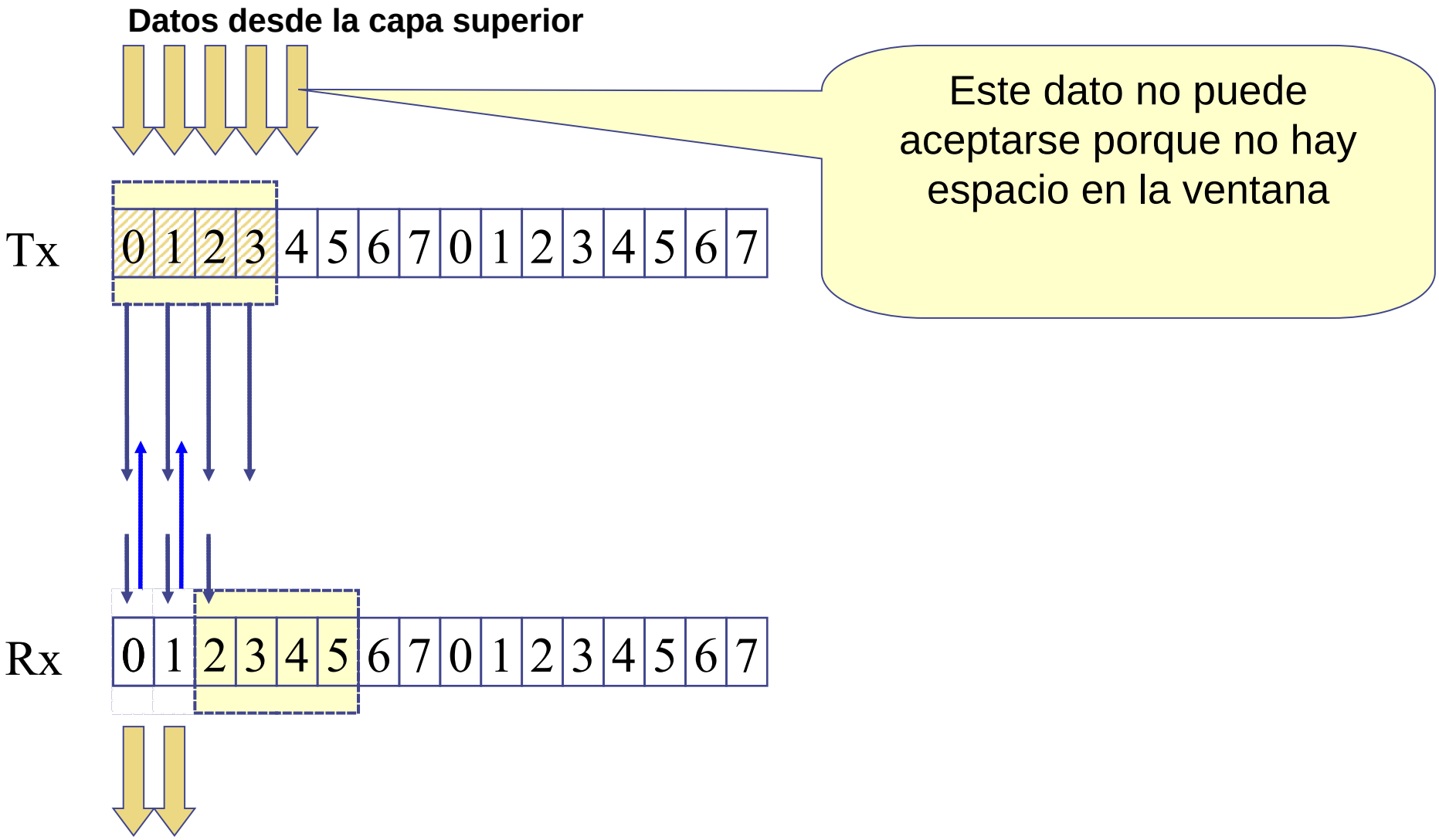
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



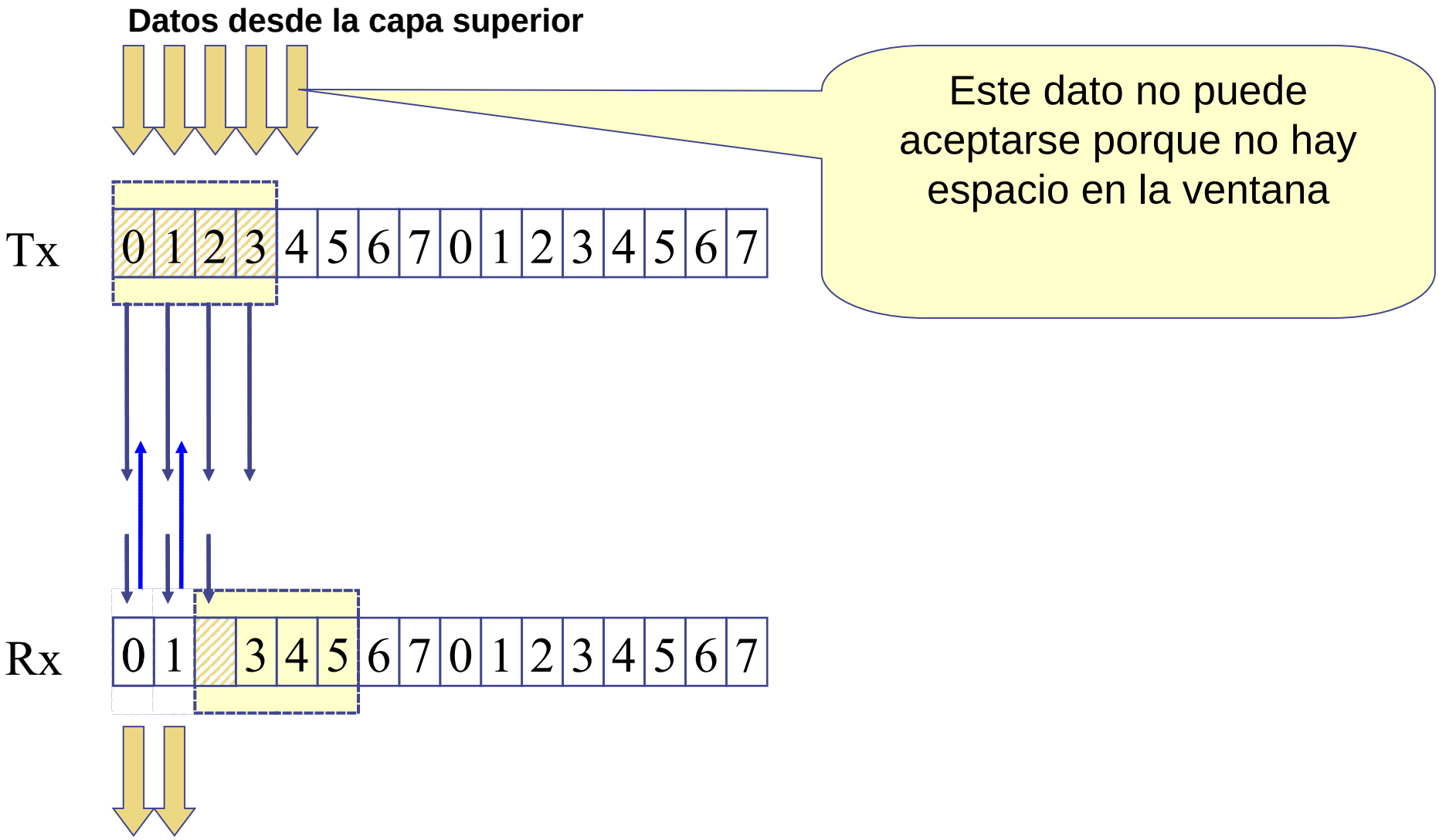
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



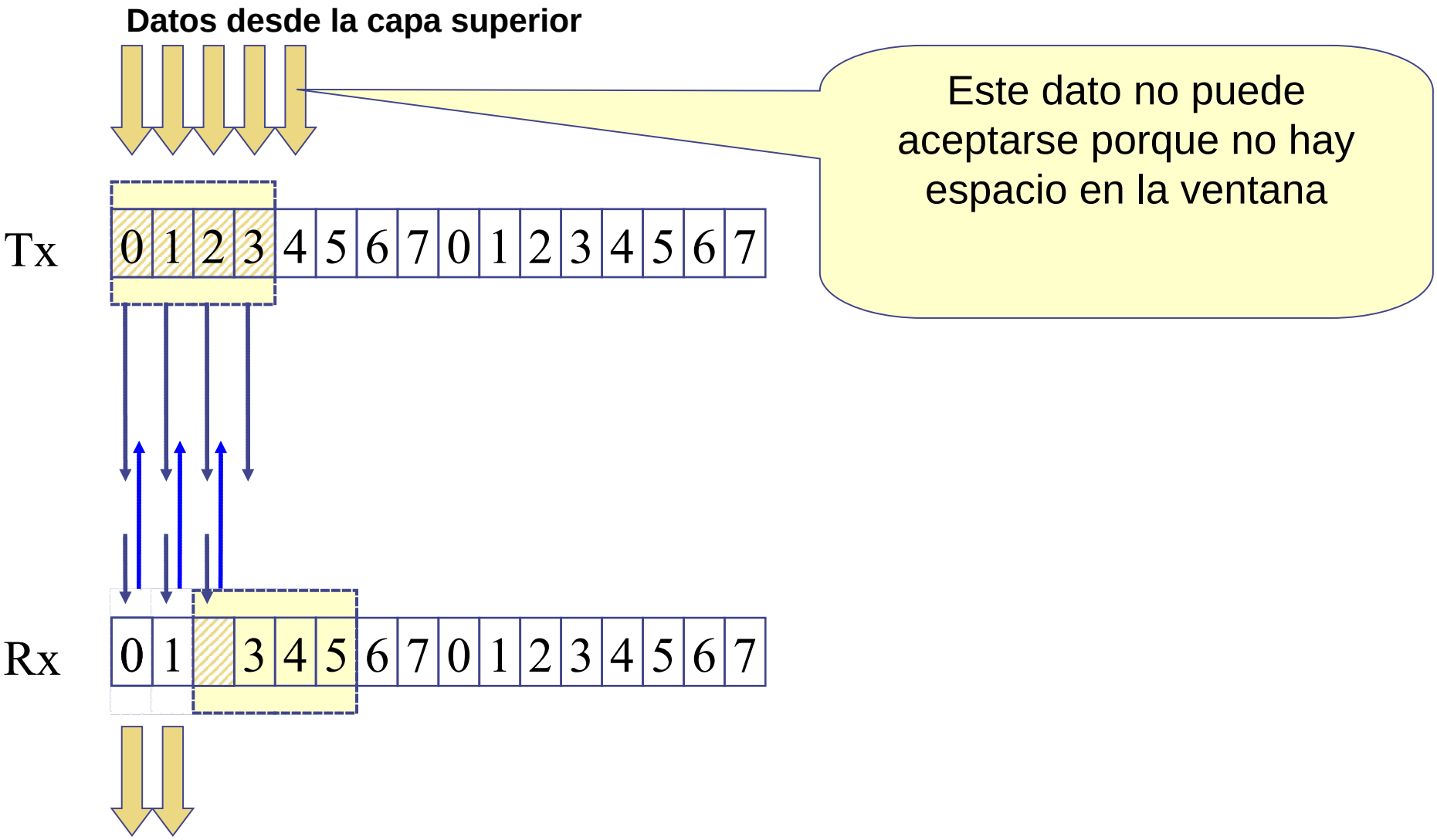
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



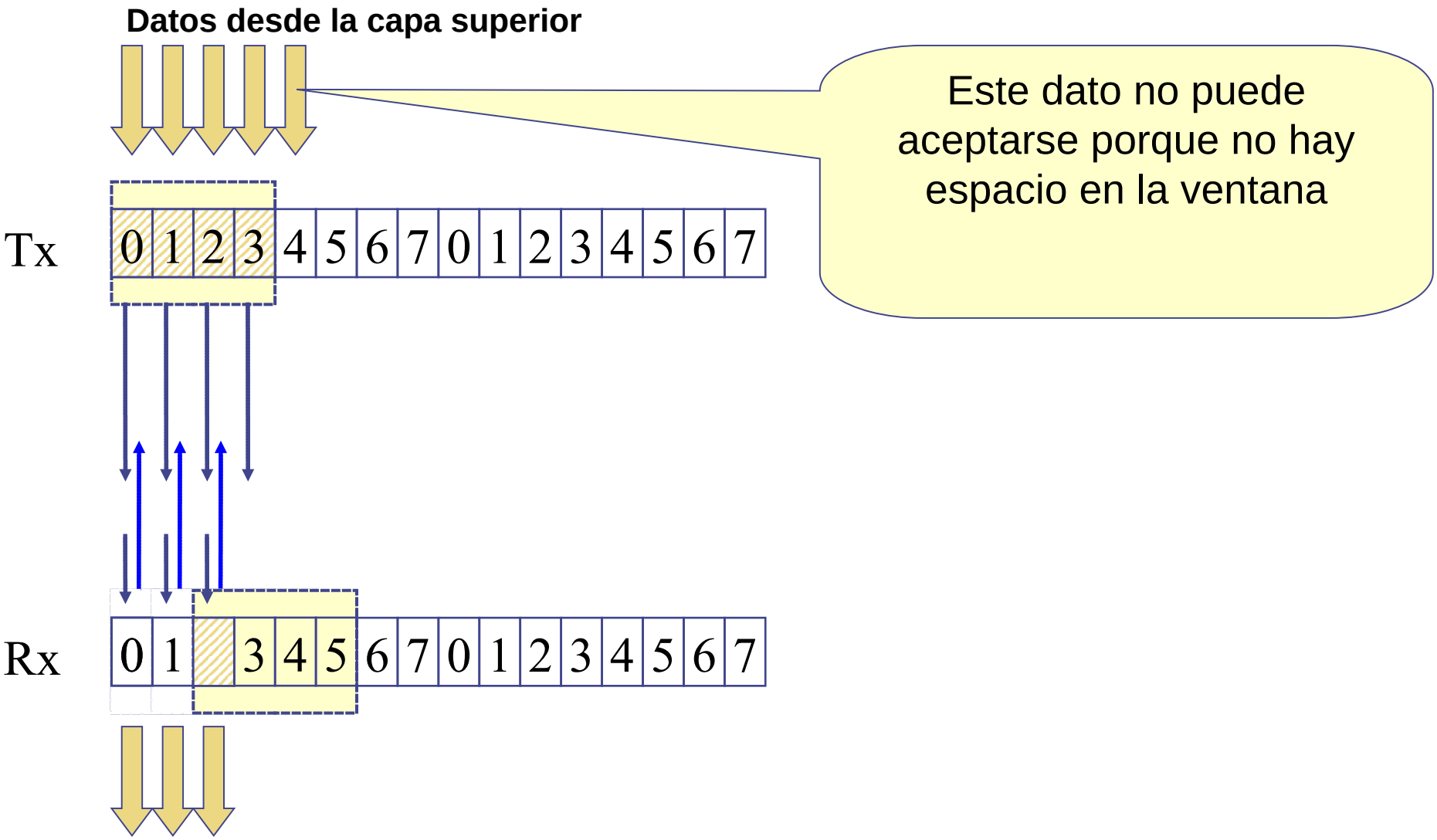
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



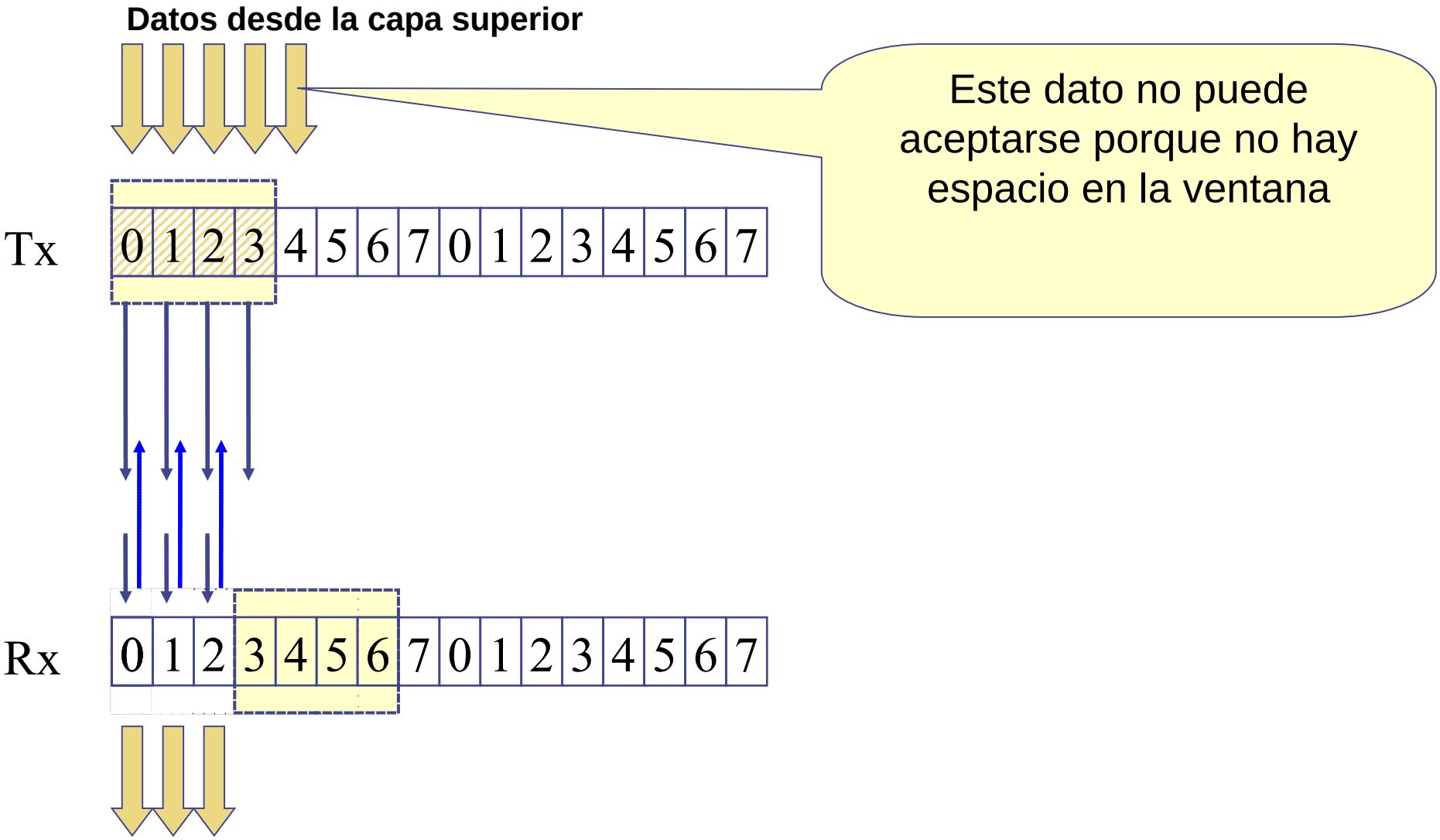
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



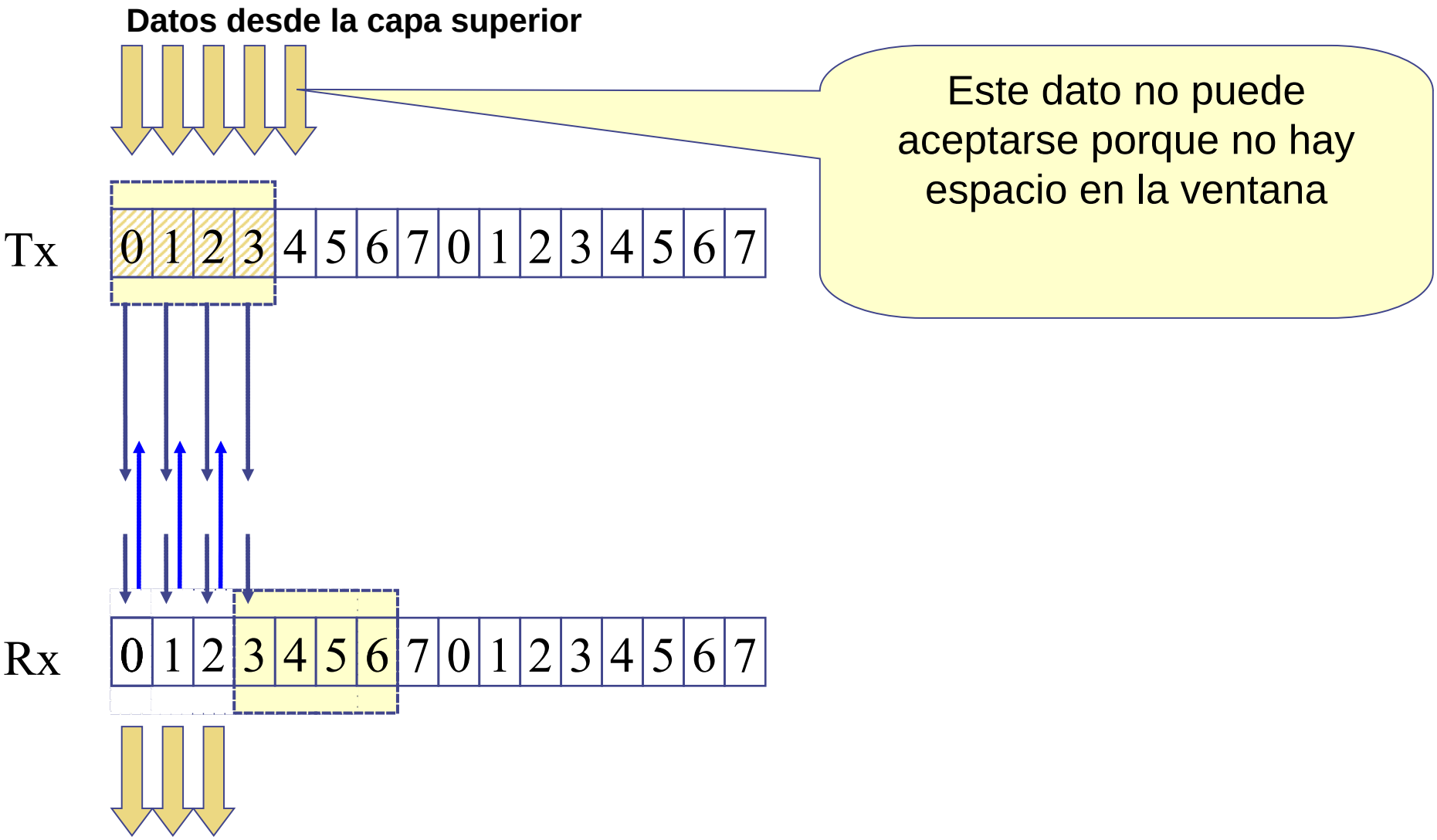
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



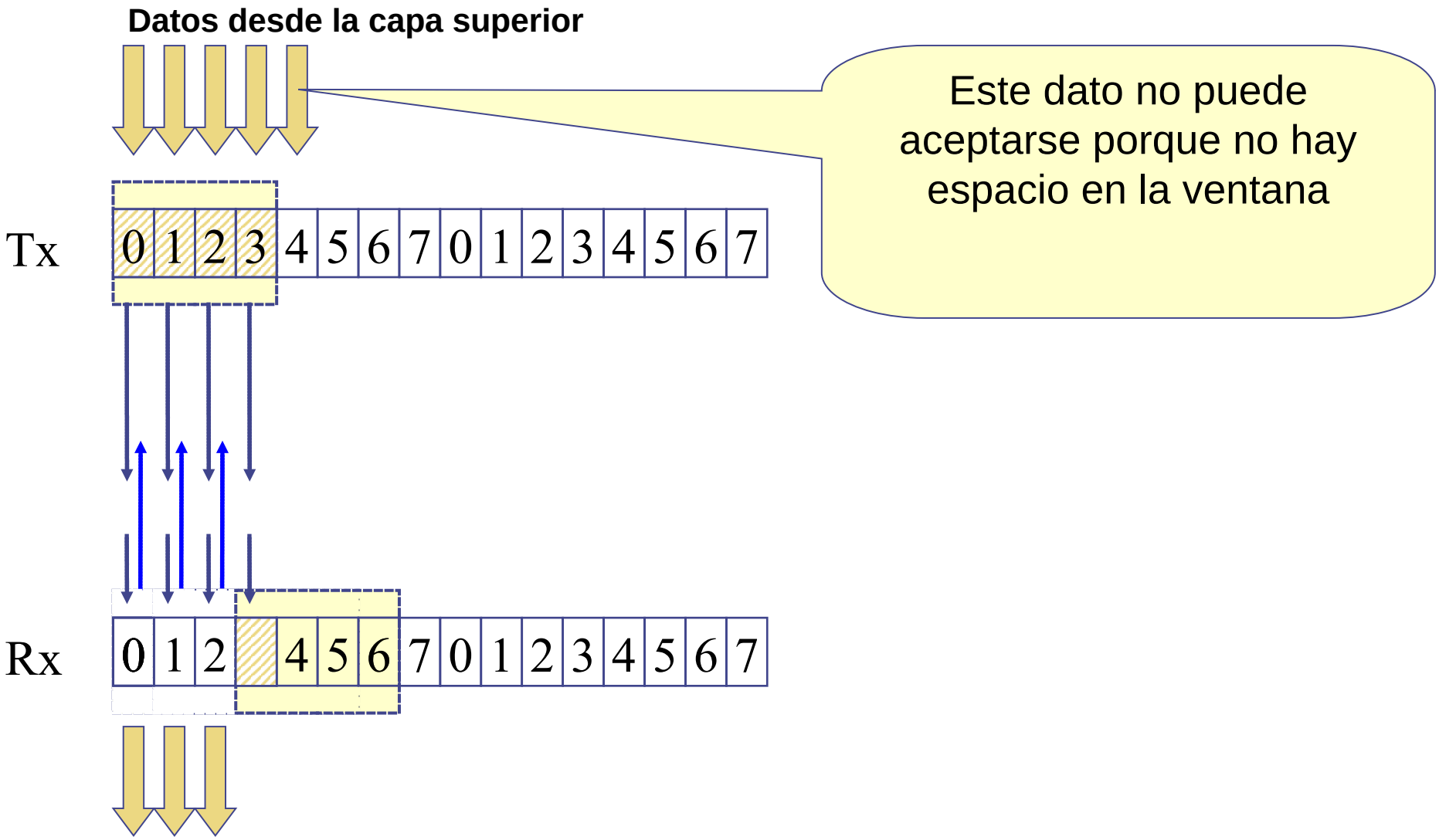
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



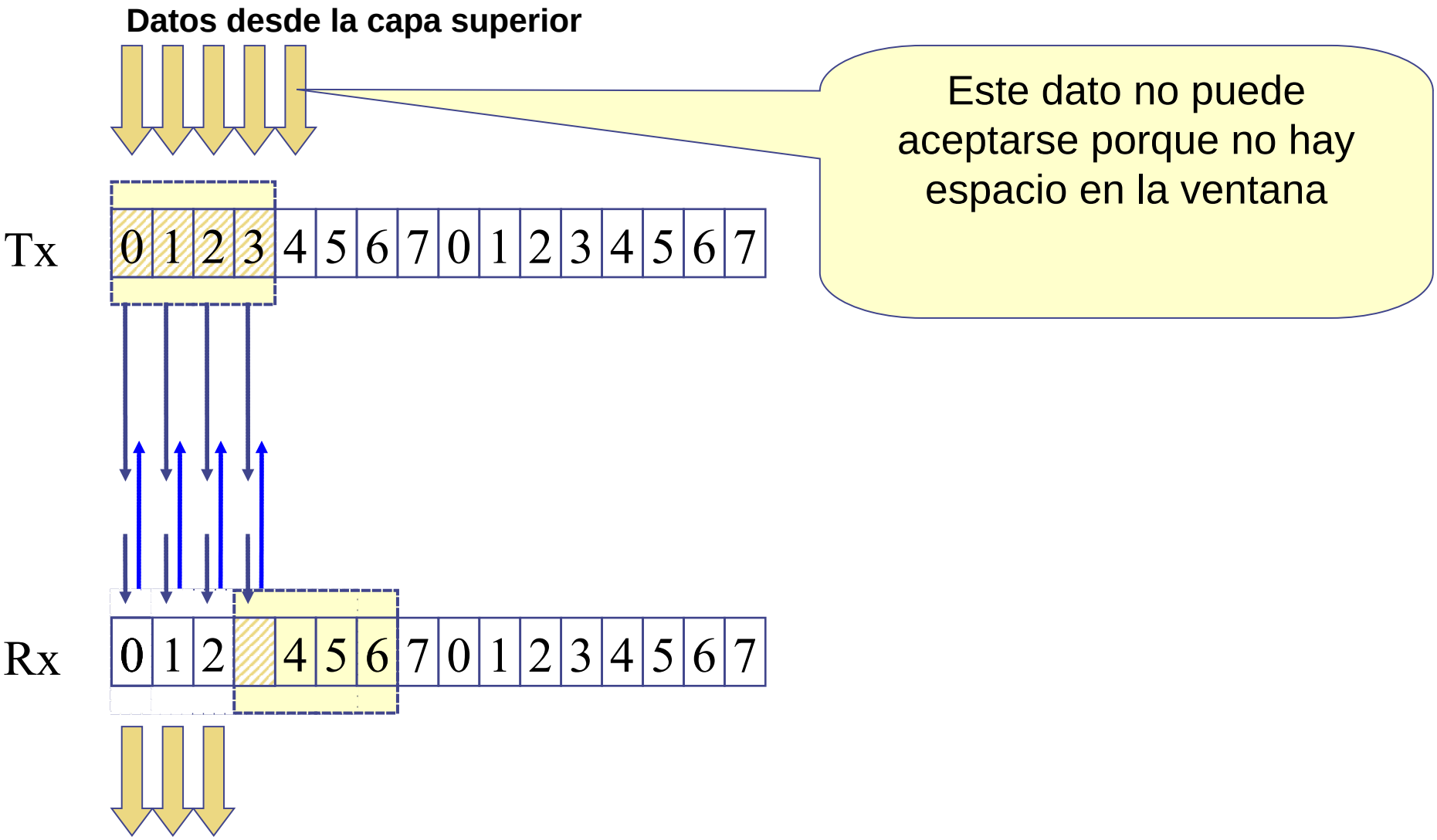
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



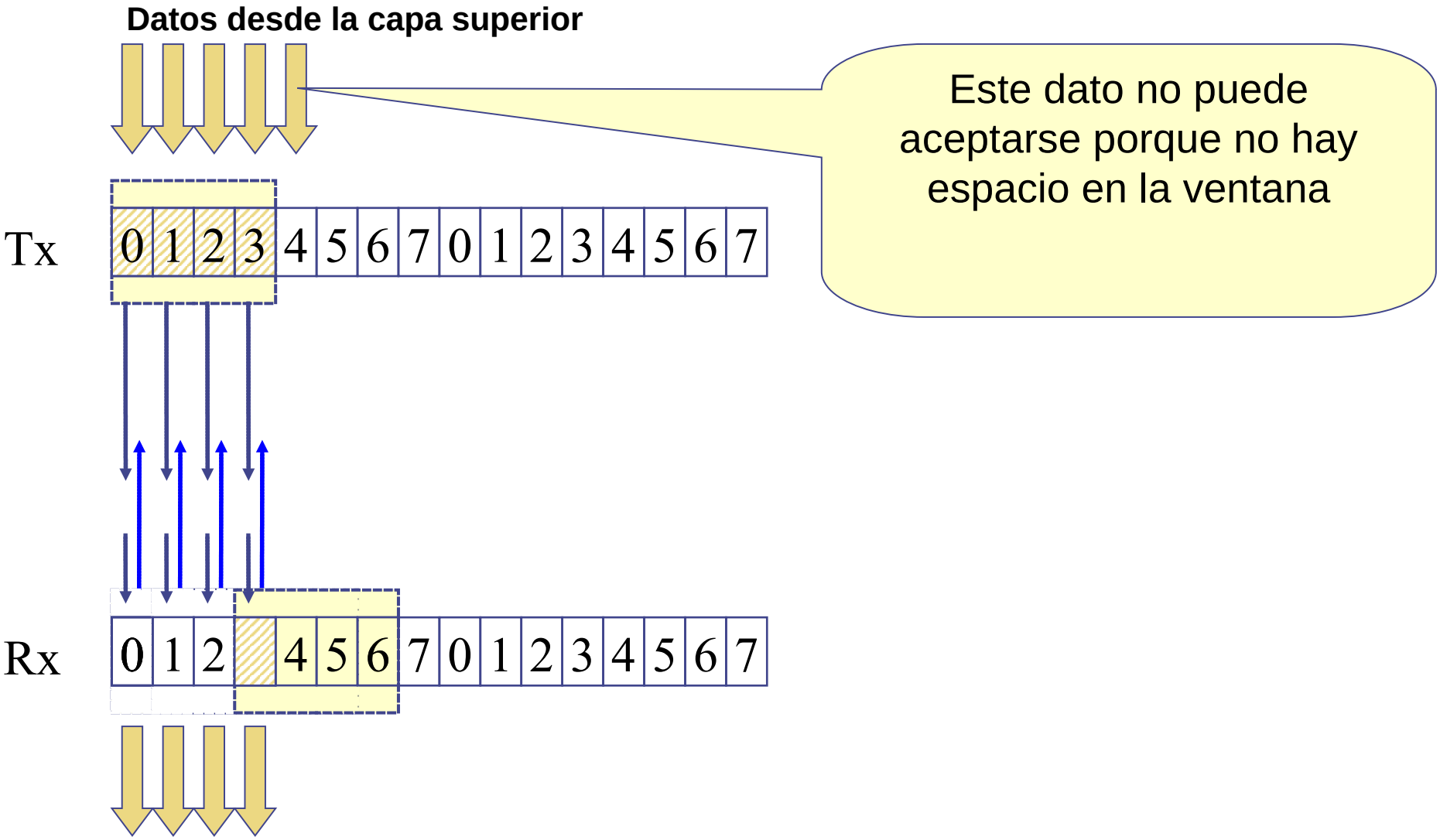
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



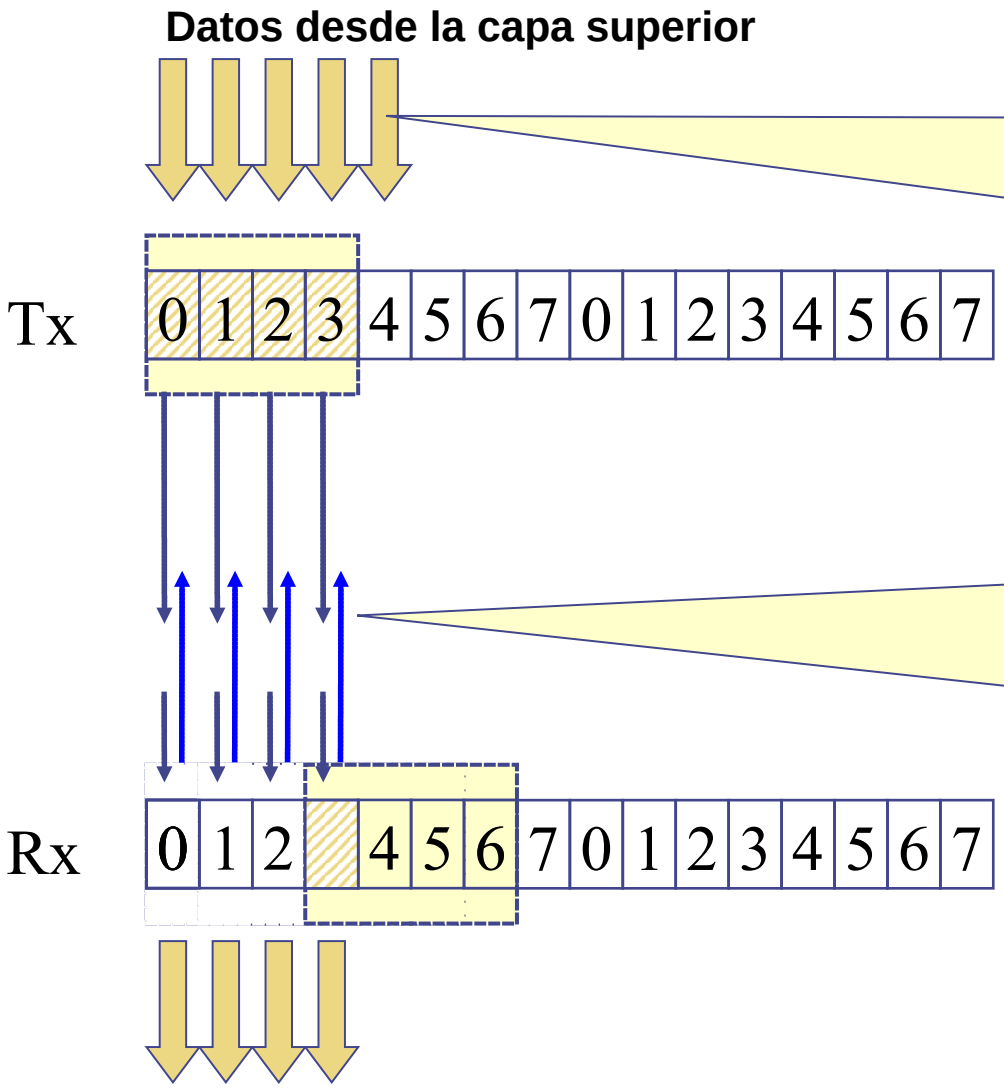
Ventanas Delizantes

- **Cuando hay retardo de origen a destino**



Ventanas Delizantes

- **Cuando hay retardo de origen a destino**

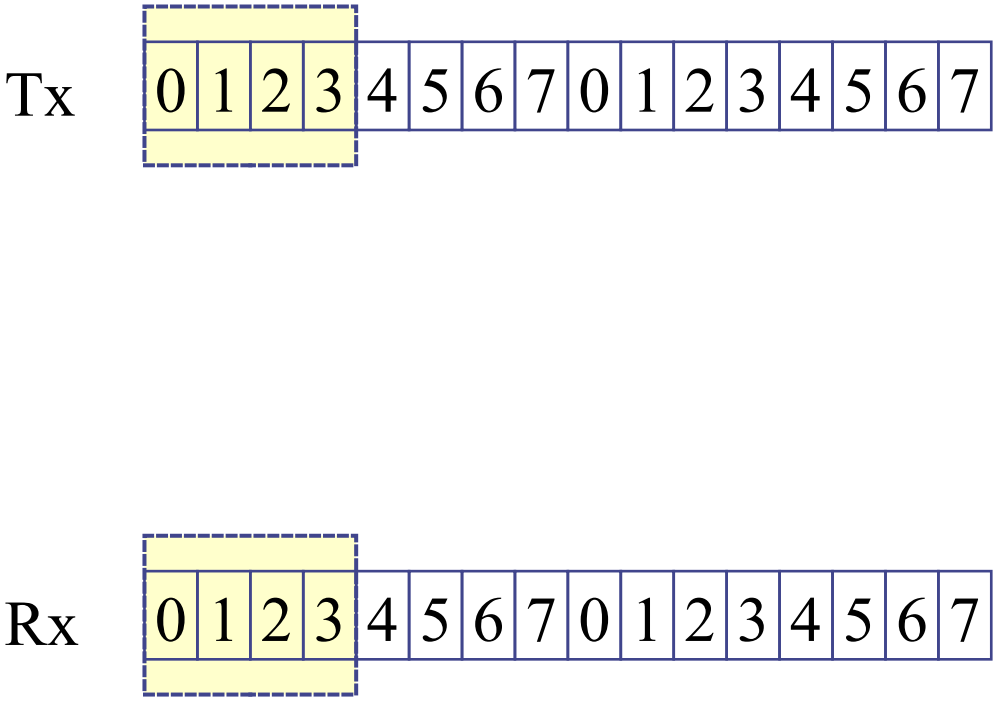


Este dato no puede aceptarse porque no hay espacio en la ventana

A medida que el Tx comienza a recibir los ACKs puede mover su ventana y aceptar nuevos datos de la capa superior

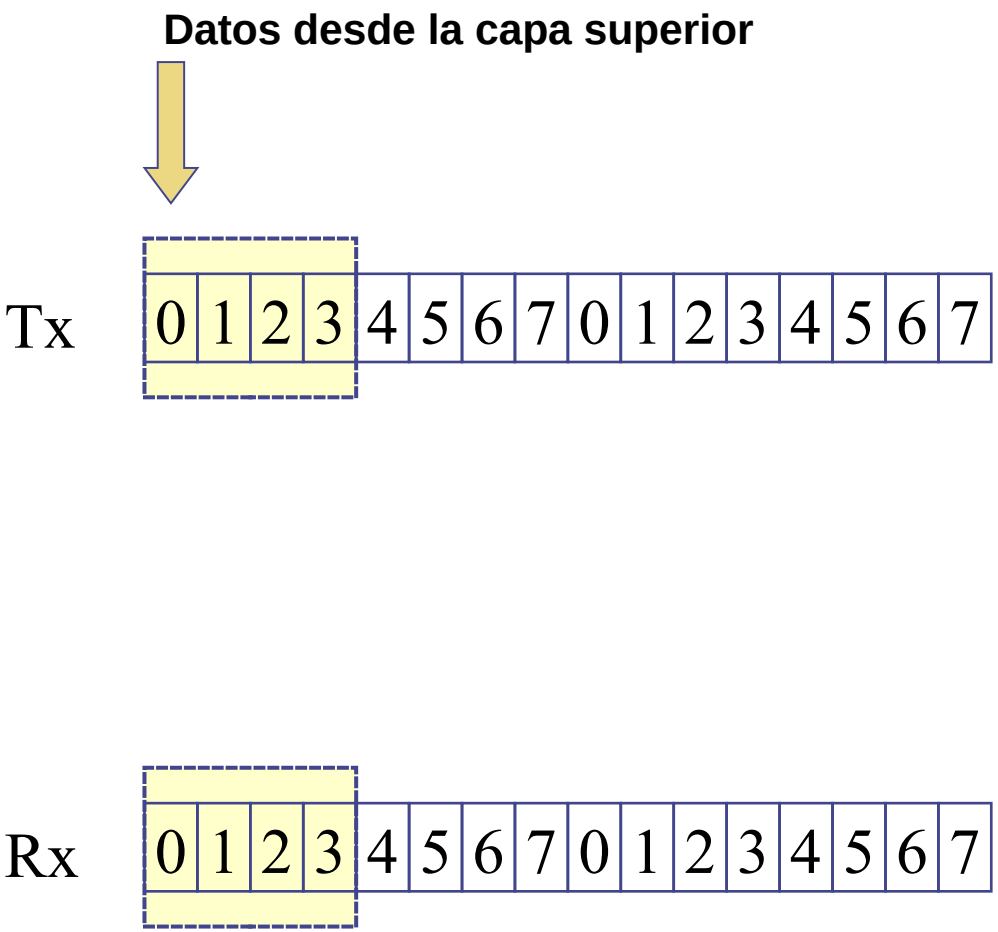
Ventanas Delizantes

- **Cuando hay errores**



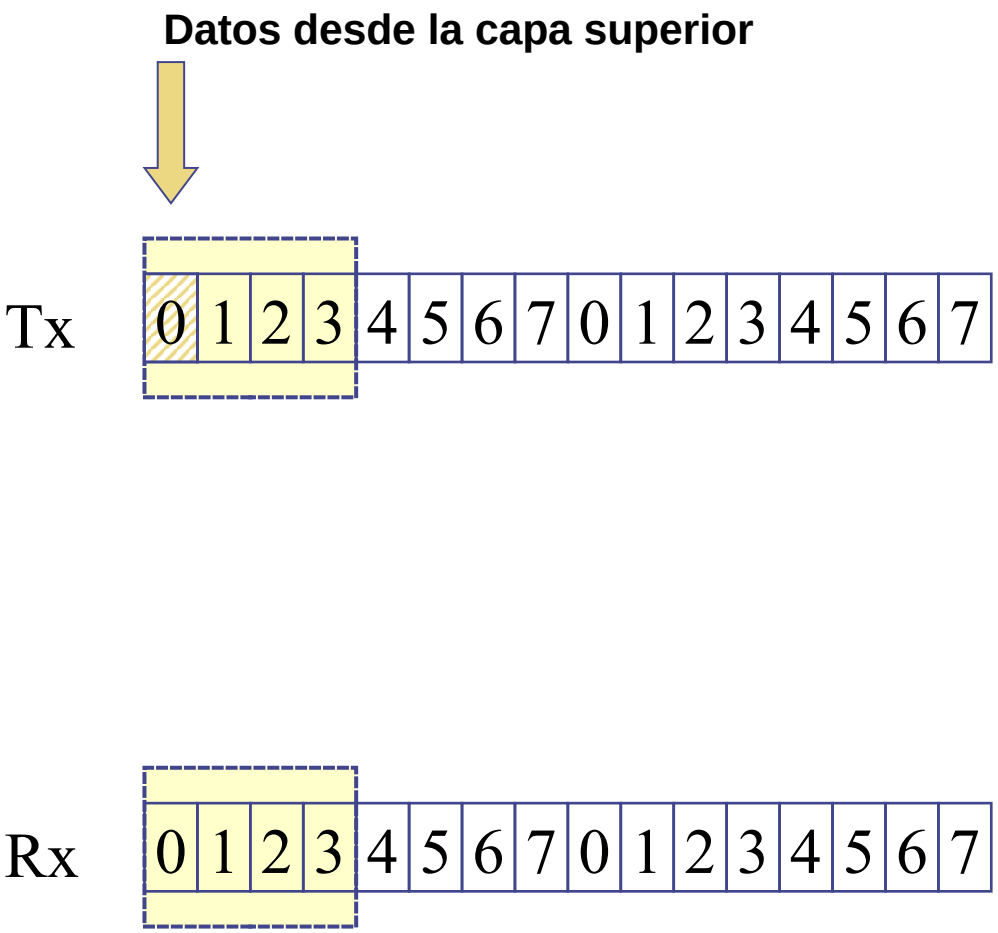
Ventanas Delizantes

- **Cuando hay errores**



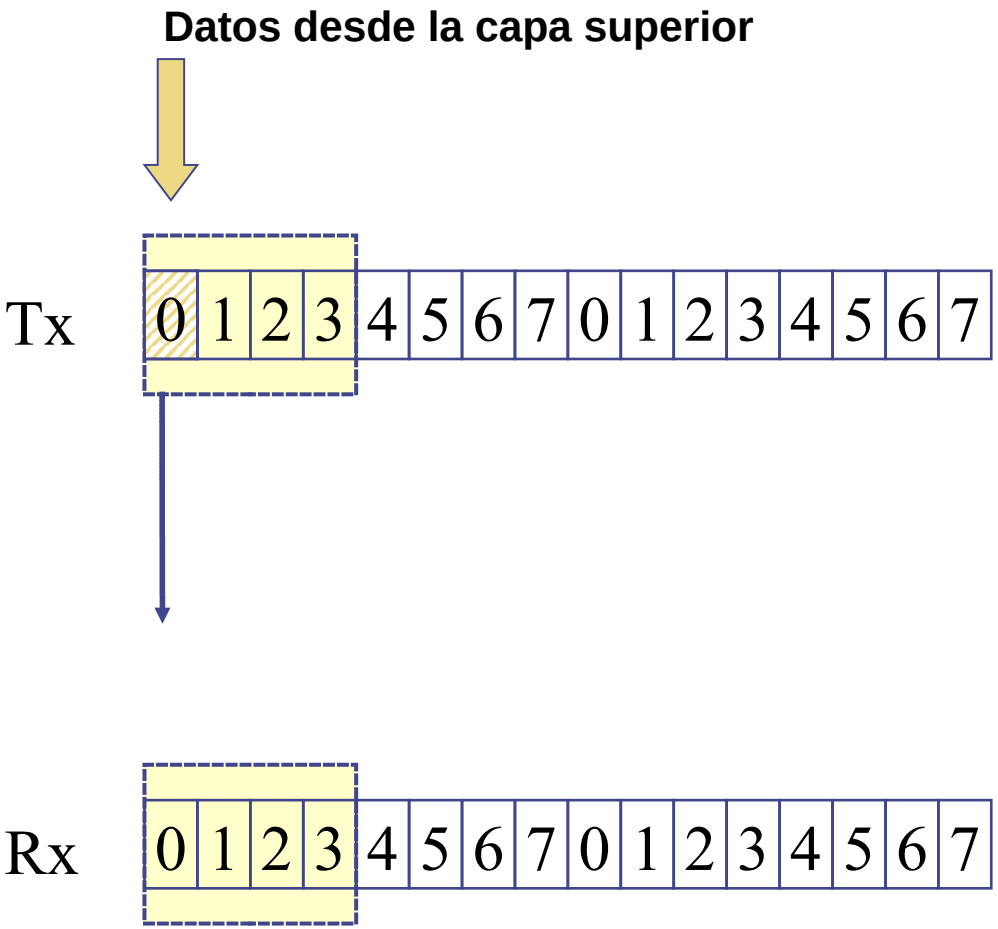
Ventanas Delizantes

- **Cuando hay errores**



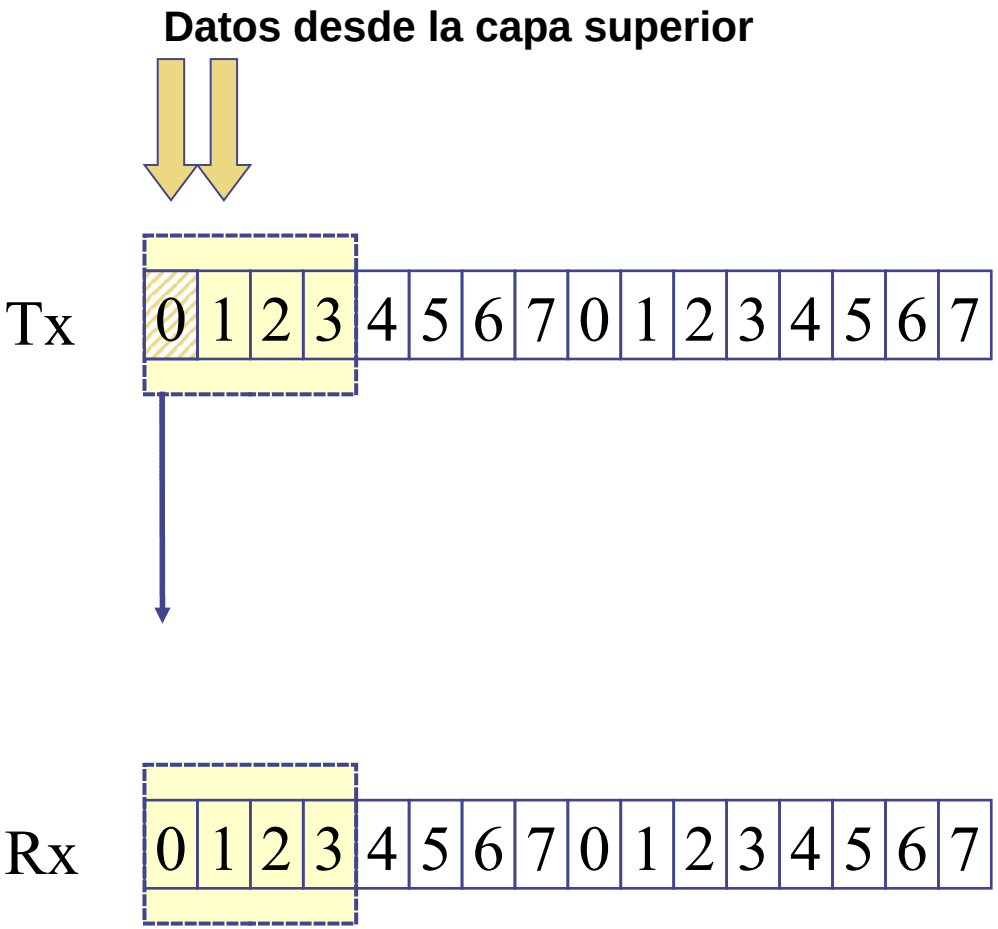
Ventanas Delizantes

- **Cuando hay errores**



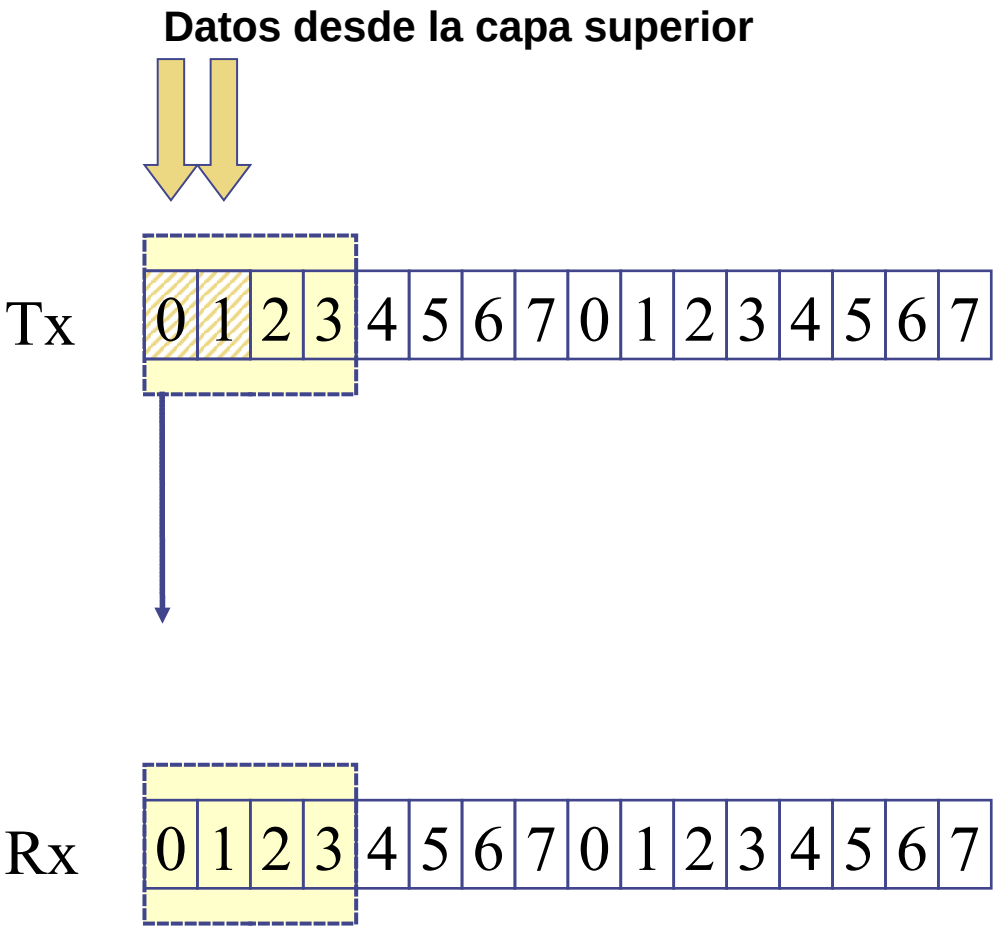
Ventanas Delizantes

- **Cuando hay errores**



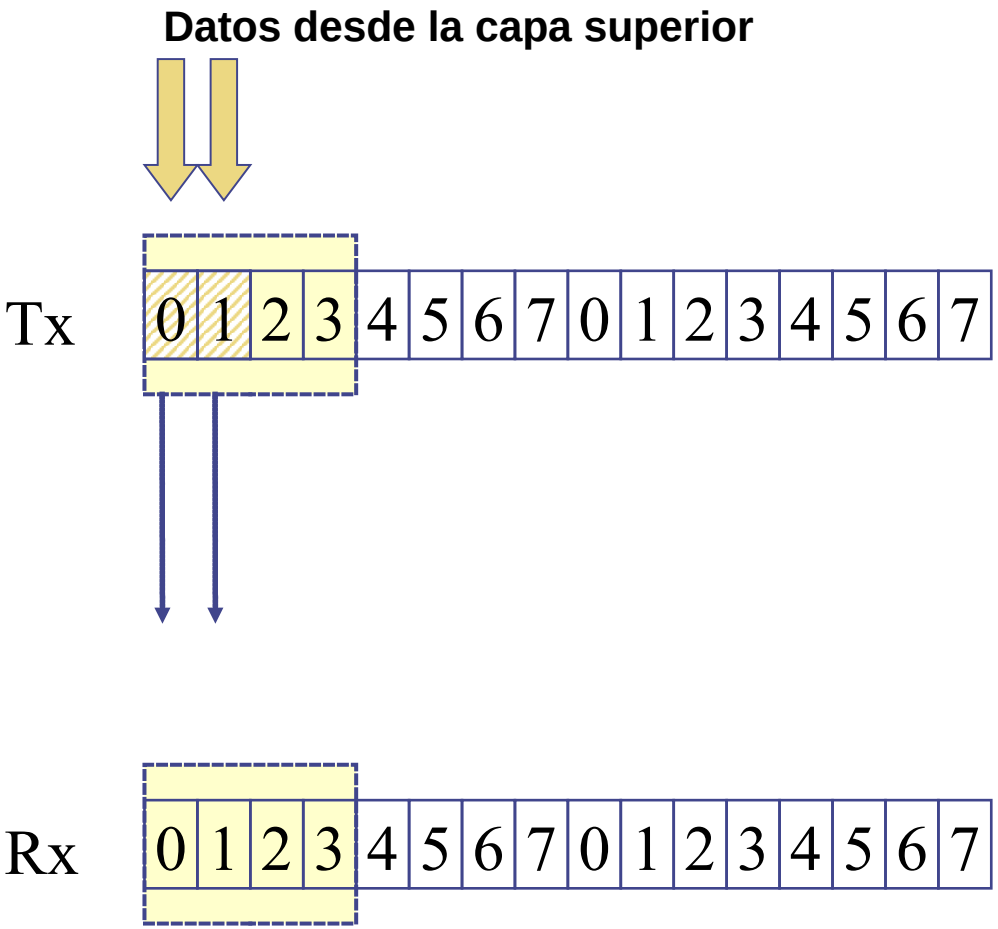
Ventanas Delizantes

- **Cuando hay errores**



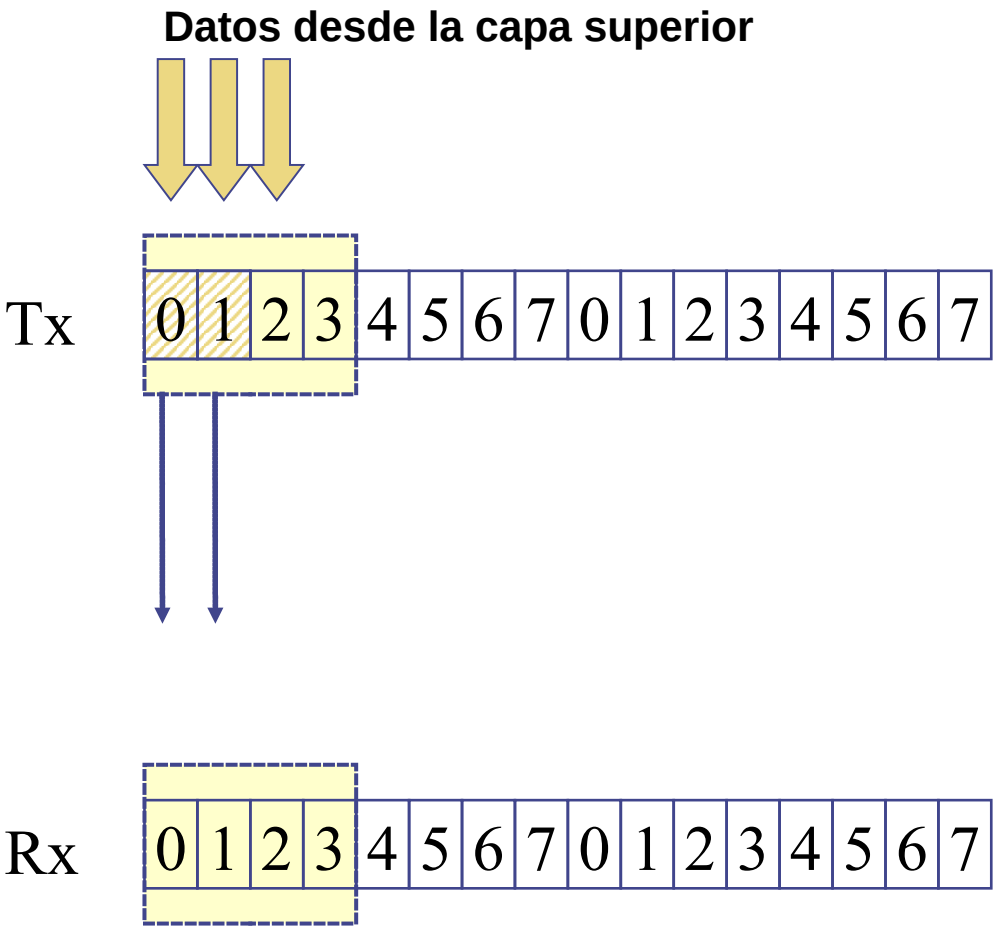
Ventanas Delizantes

- **Cuando hay errores**



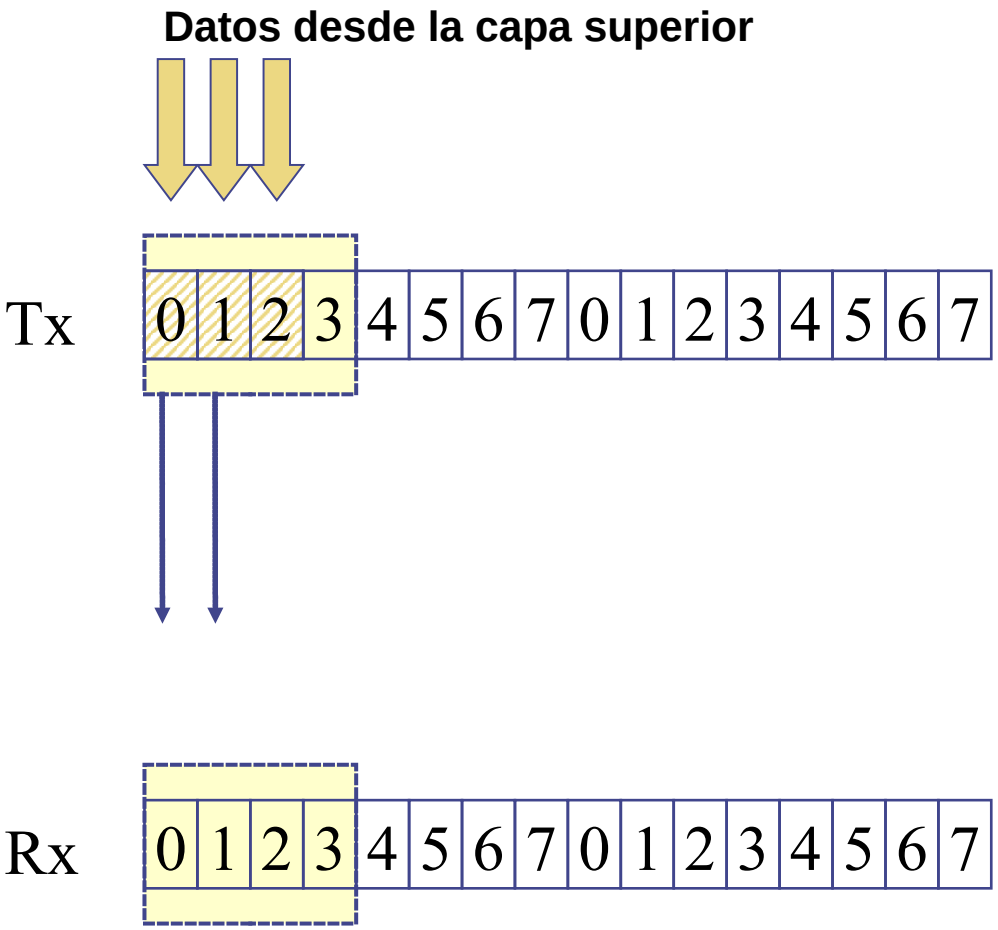
Ventanas Delizantes

- **Cuando hay errores**



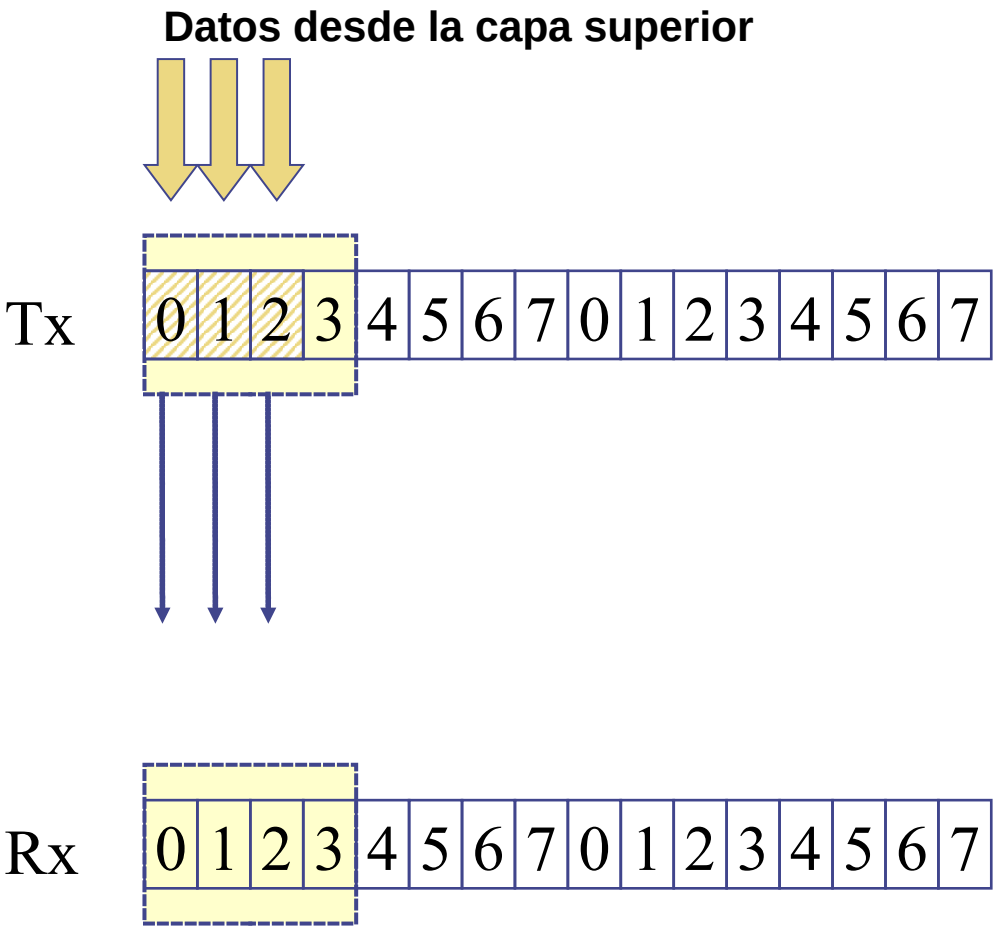
Ventanas Delizantes

- **Cuando hay errores**



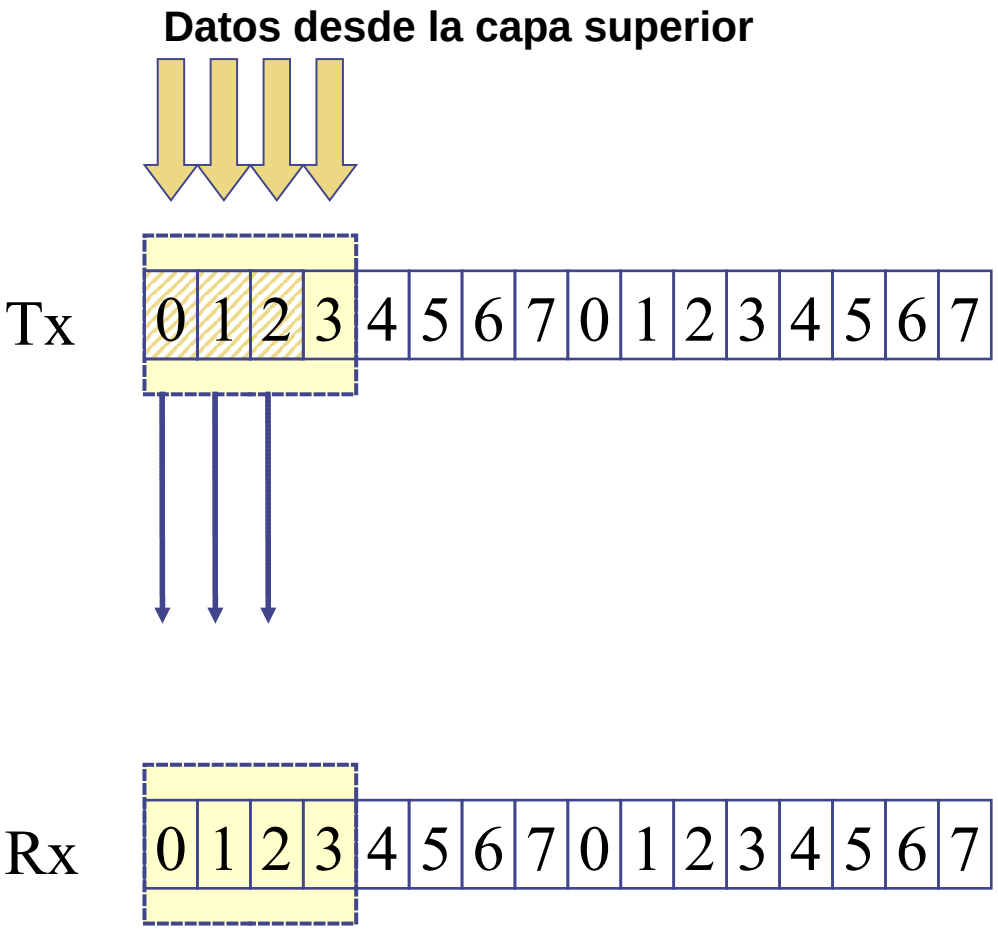
Ventanas Delizantes

- **Cuando hay errores**



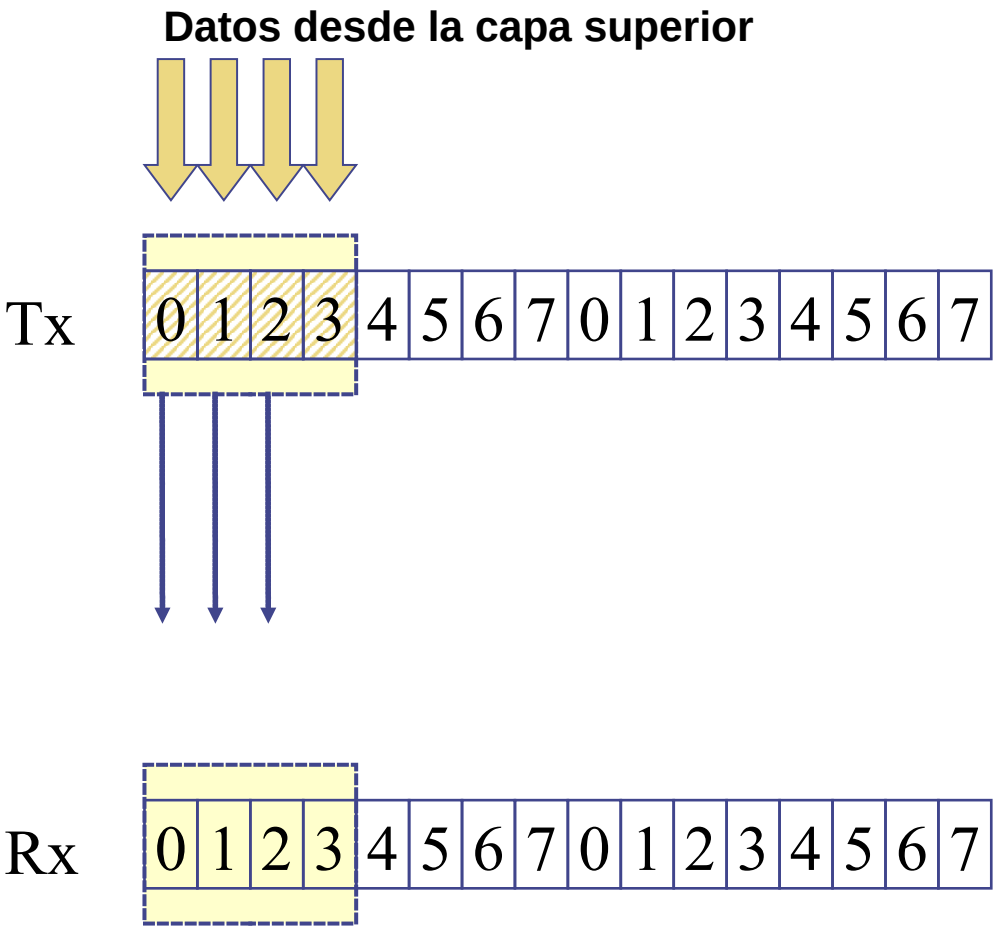
Ventanas Delizantes

- **Cuando hay errores**



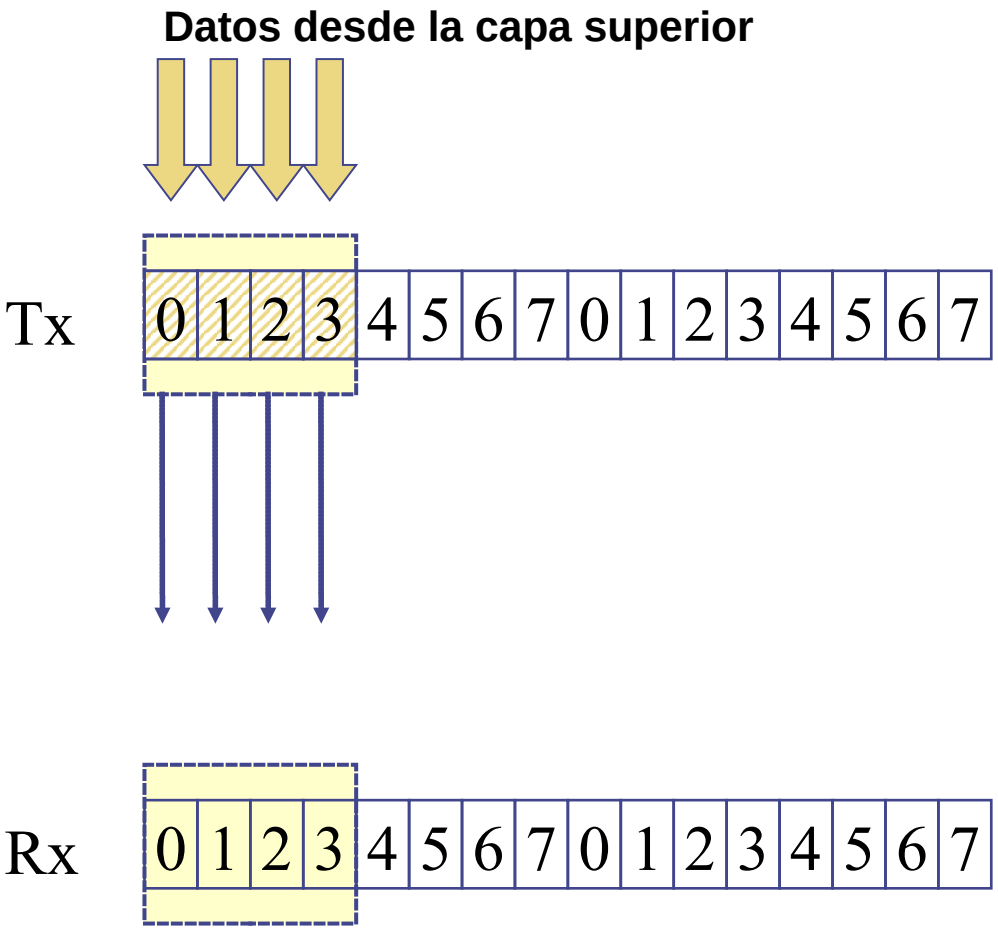
Ventanas Delizantes

- **Cuando hay errores**



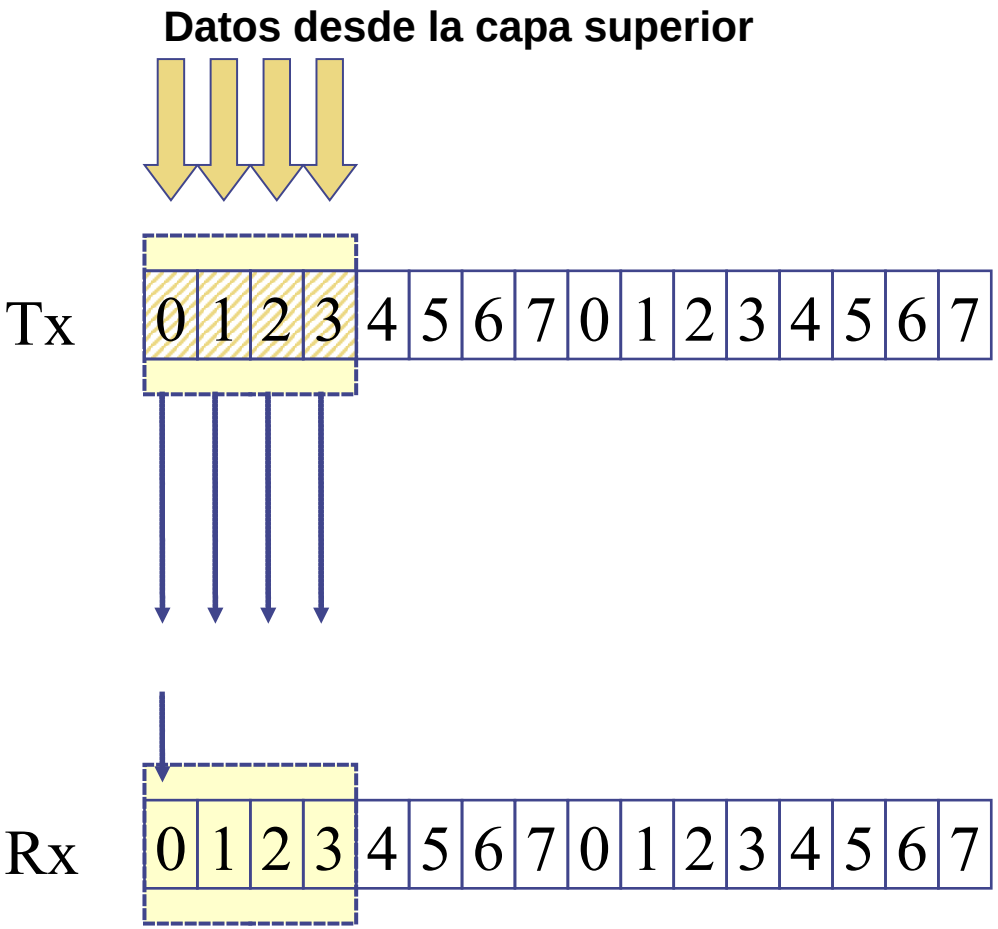
Ventanas Delizantes

- **Cuando hay errores**



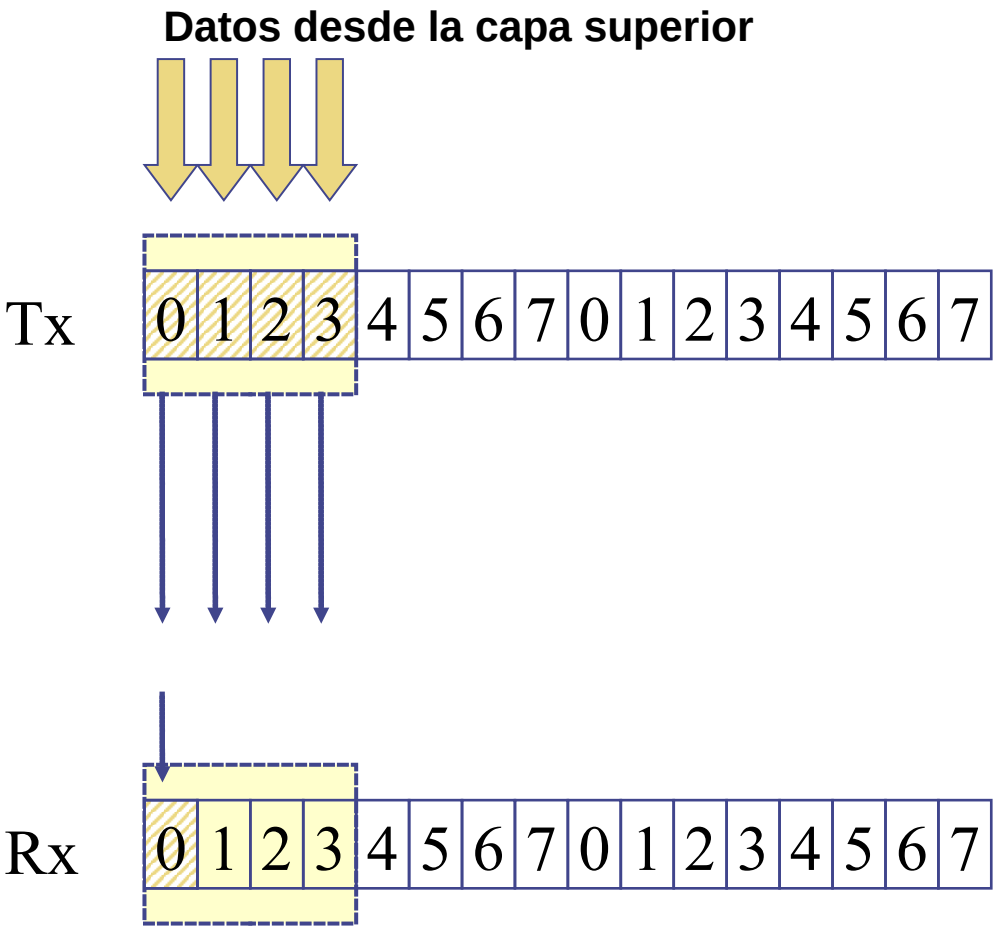
Ventanas Delizantes

- **Cuando hay errores**



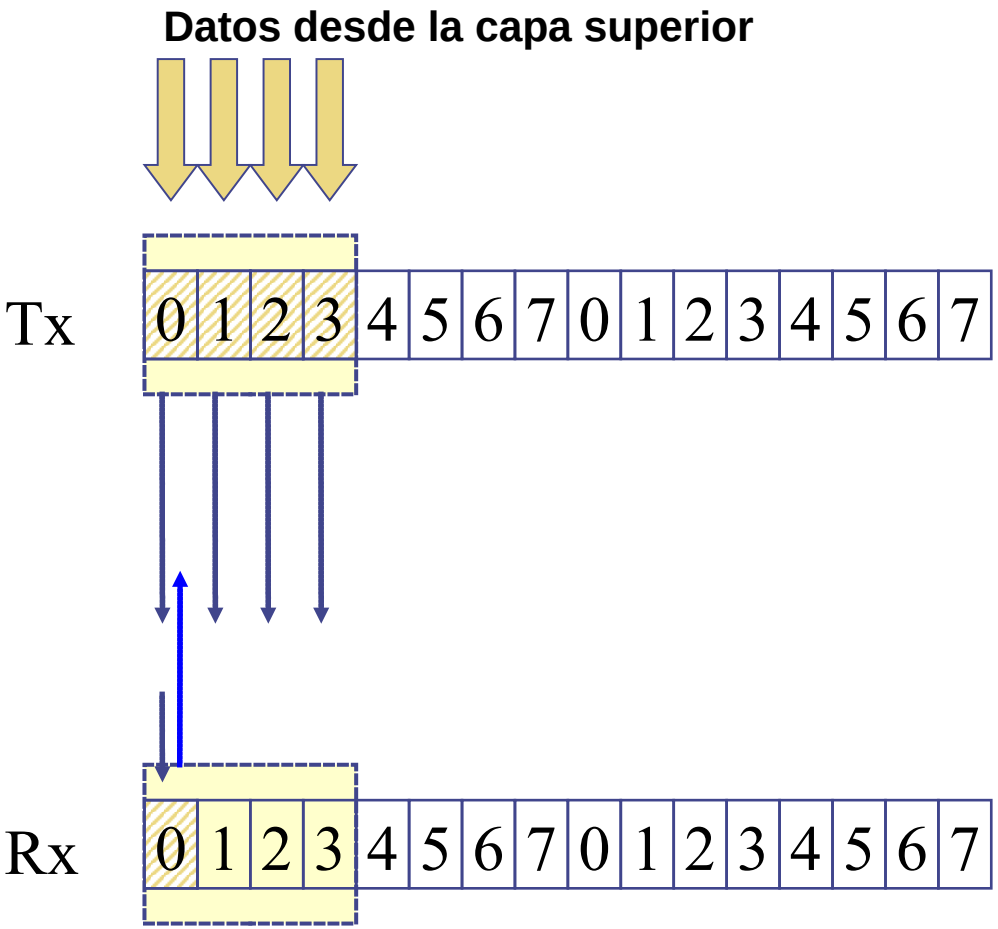
Ventanas Delizantes

- **Cuando hay errores**



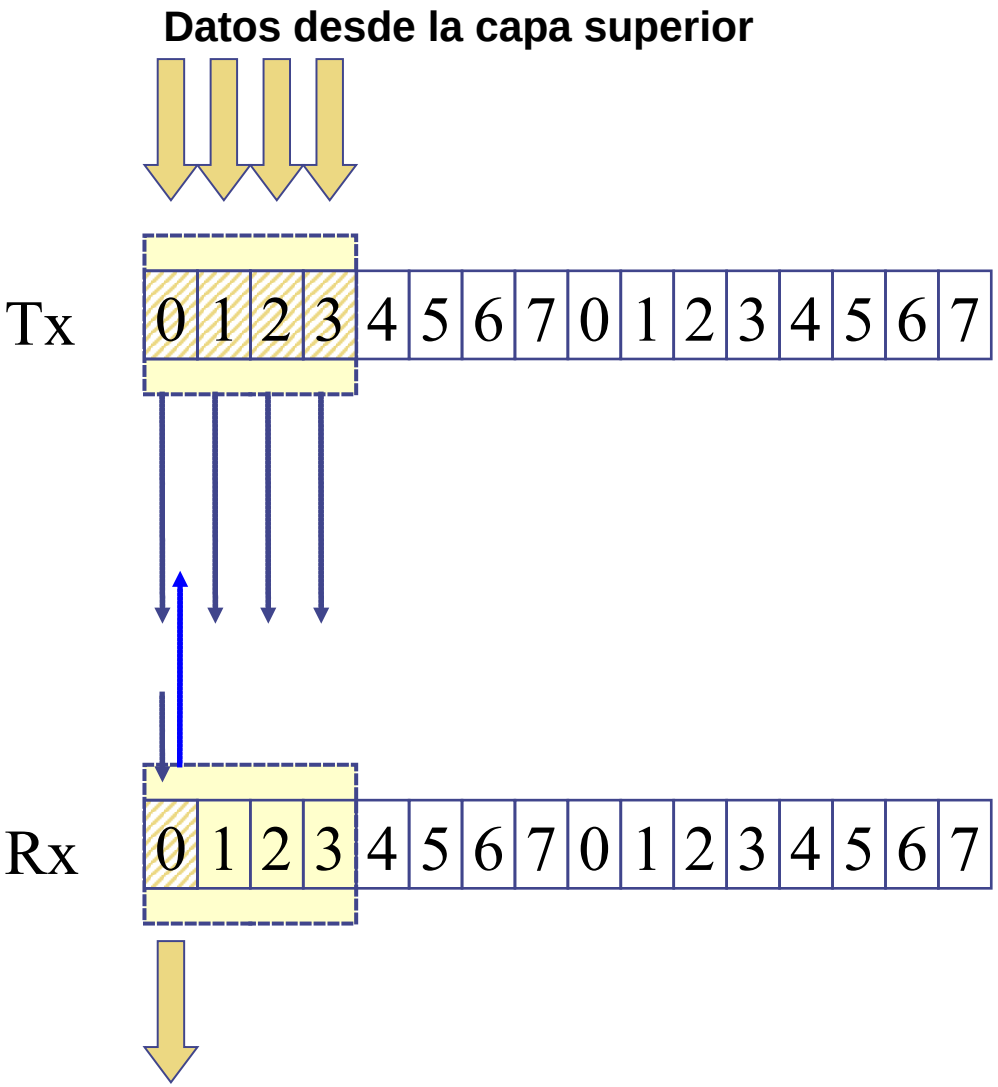
Ventanas Delizantes

- **Cuando hay errores**



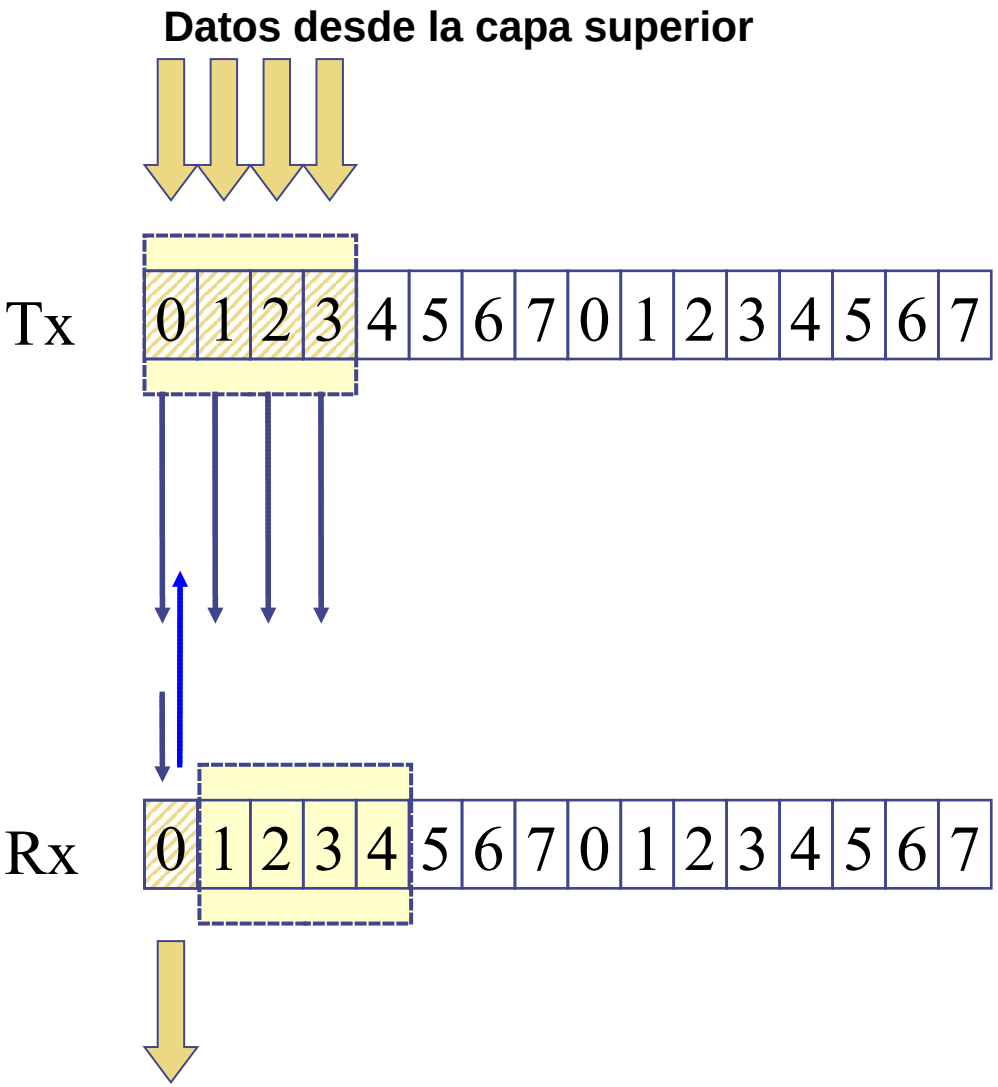
Ventanas Delizantes

- **Cuando hay errores**



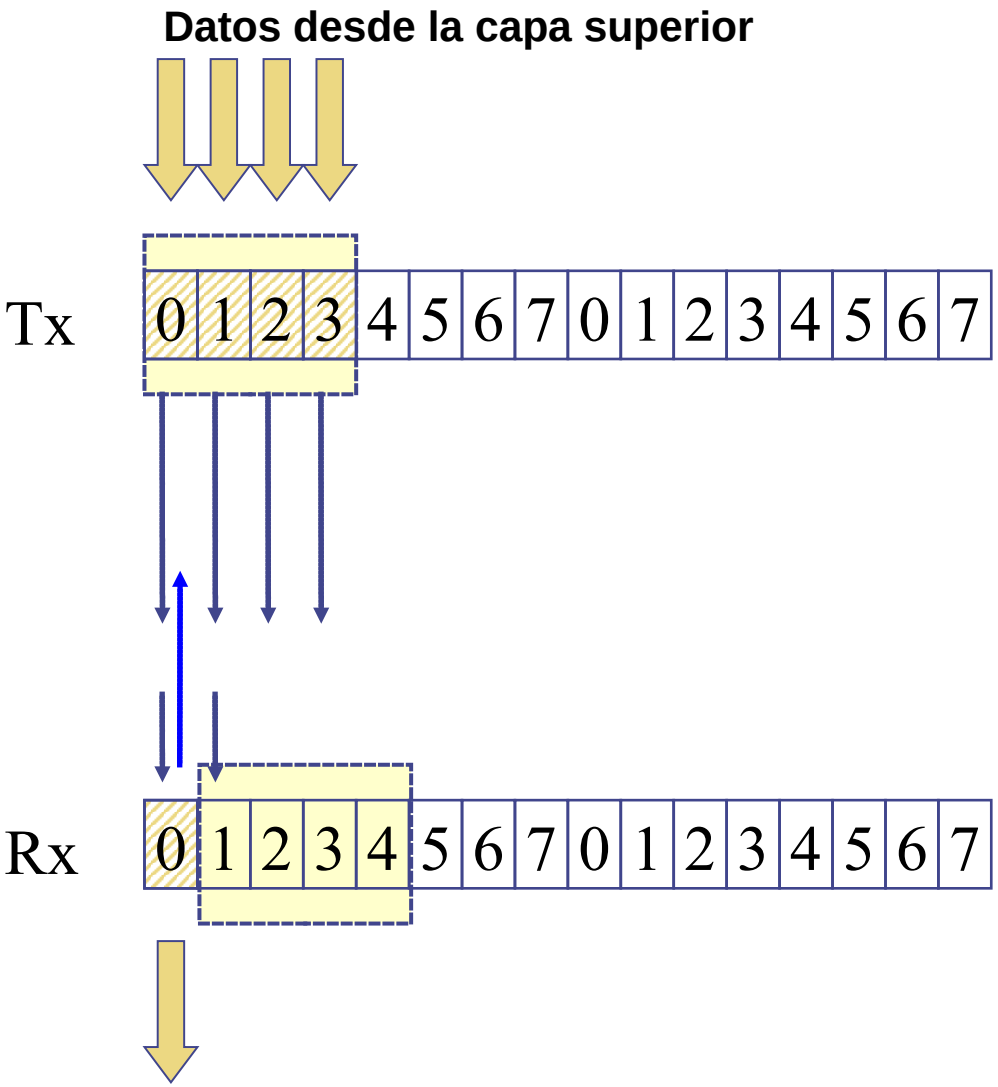
Ventanas Delizantes

- **Cuando hay errores**



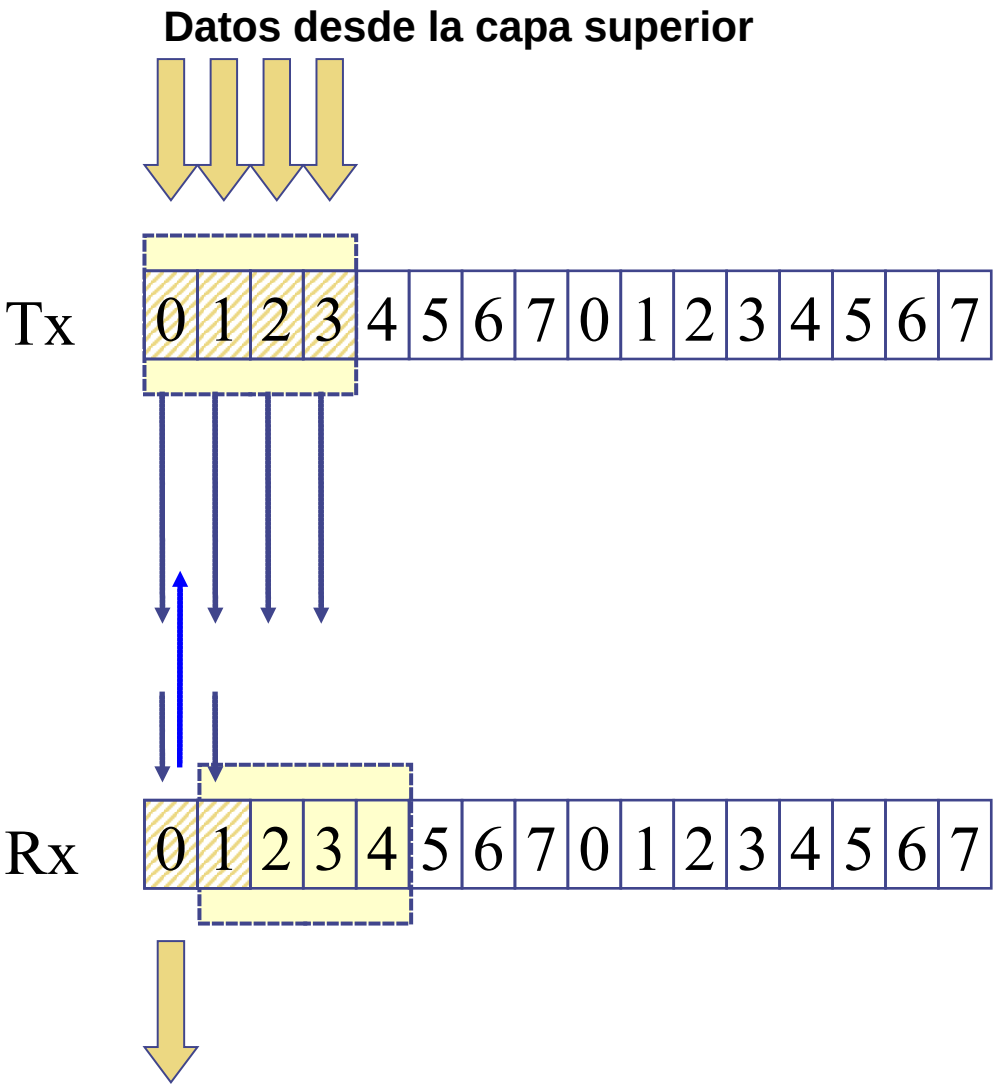
Ventanas Delizantes

- **Cuando hay errores**



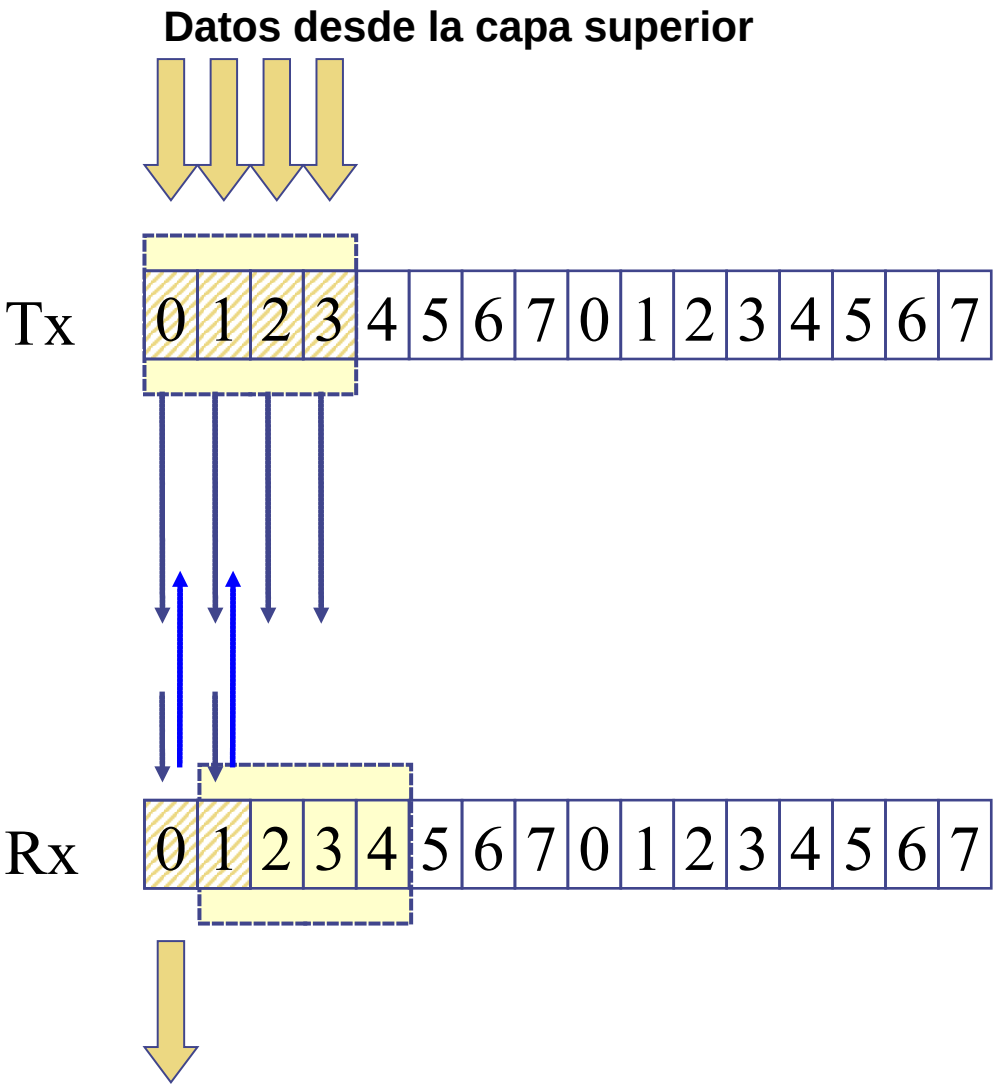
Ventanas Delizantes

- **Cuando hay errores**



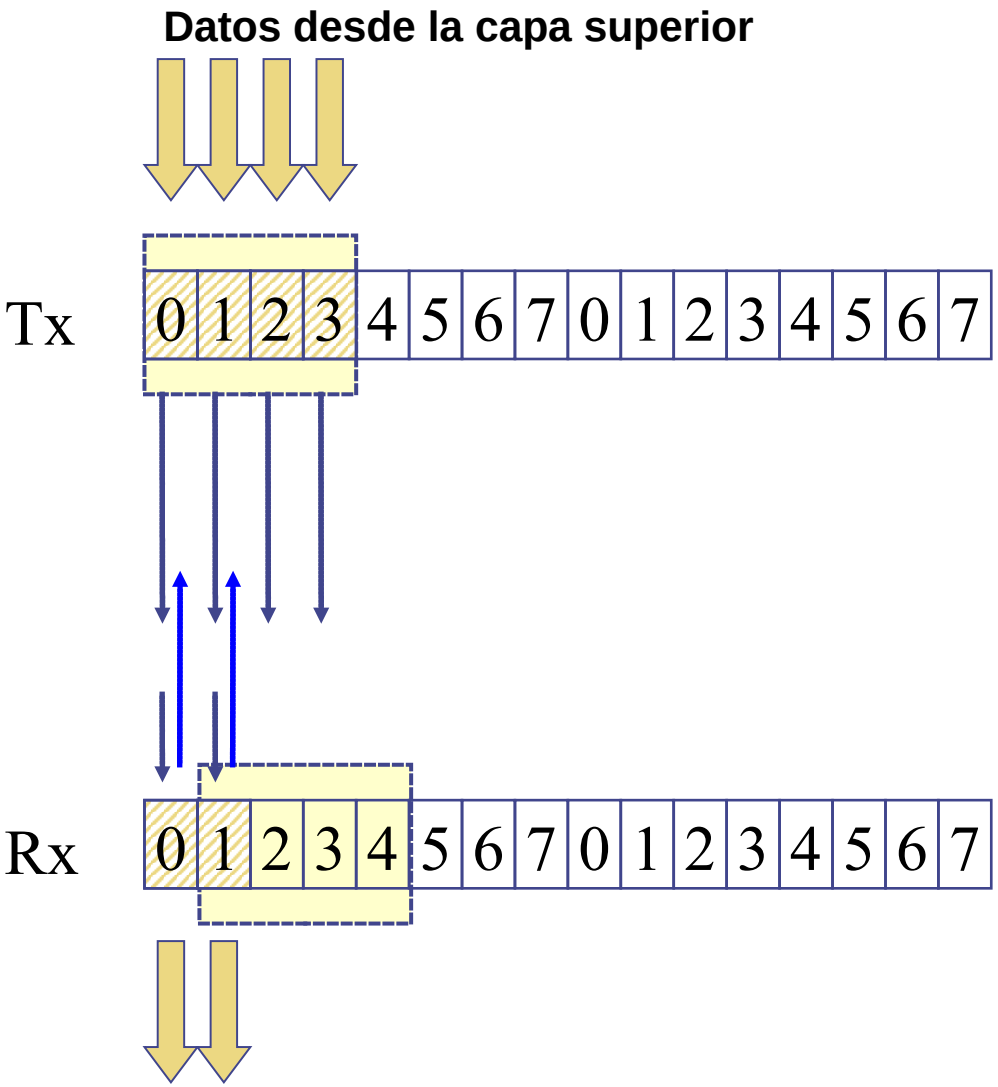
Ventanas Delizantes

- **Cuando hay errores**



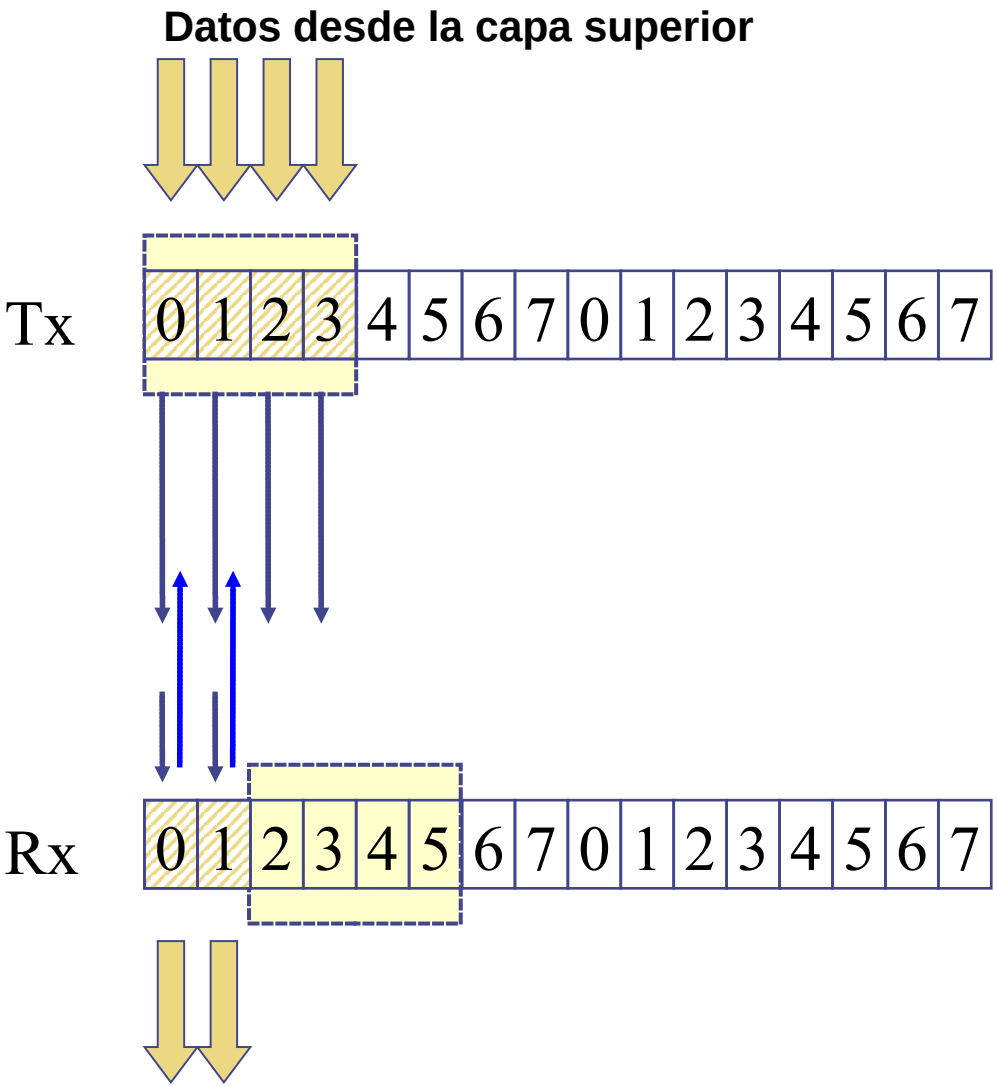
Ventanas Delizantes

- **Cuando hay errores**



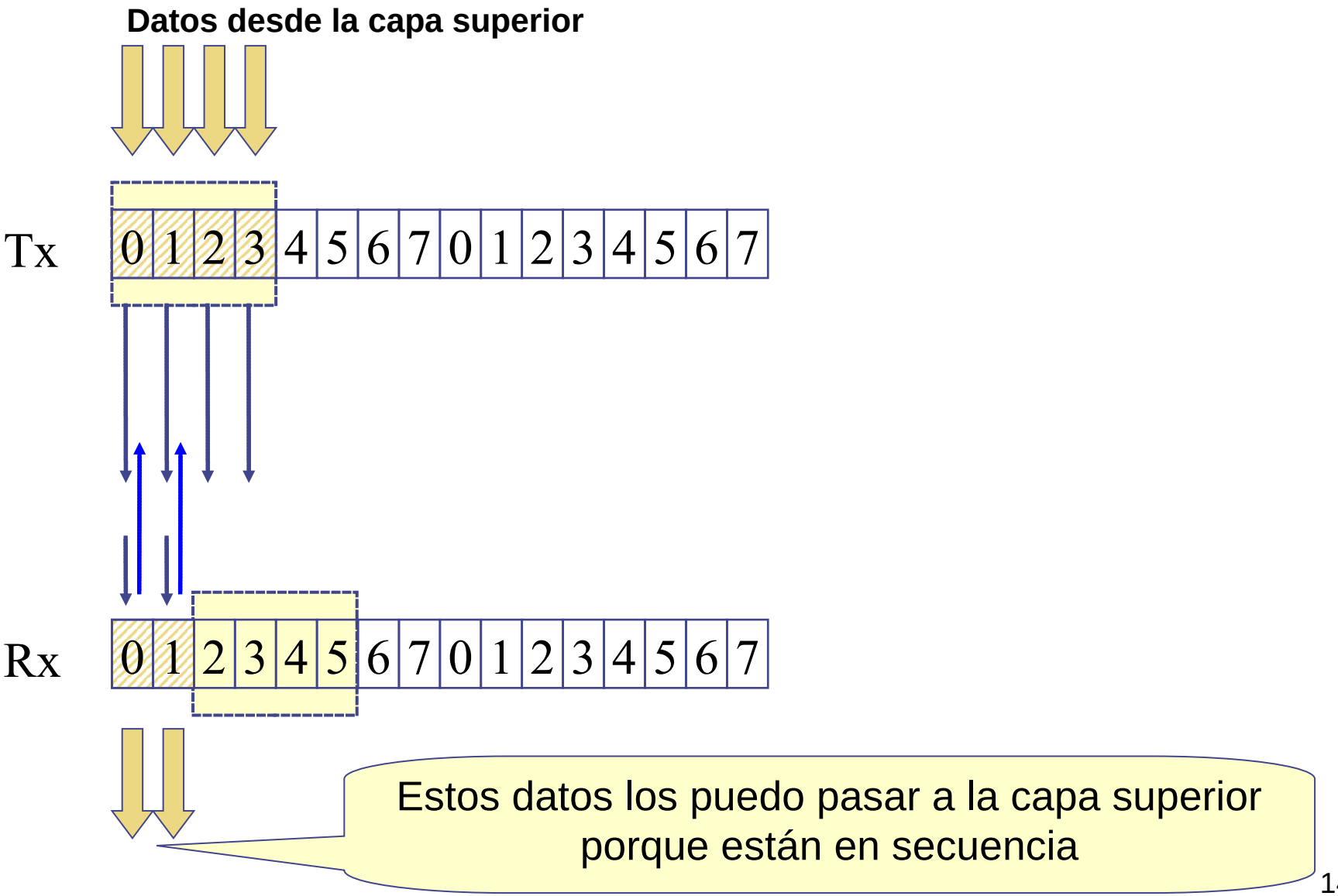
Ventanas Delizantes

- **Cuando hay errores**



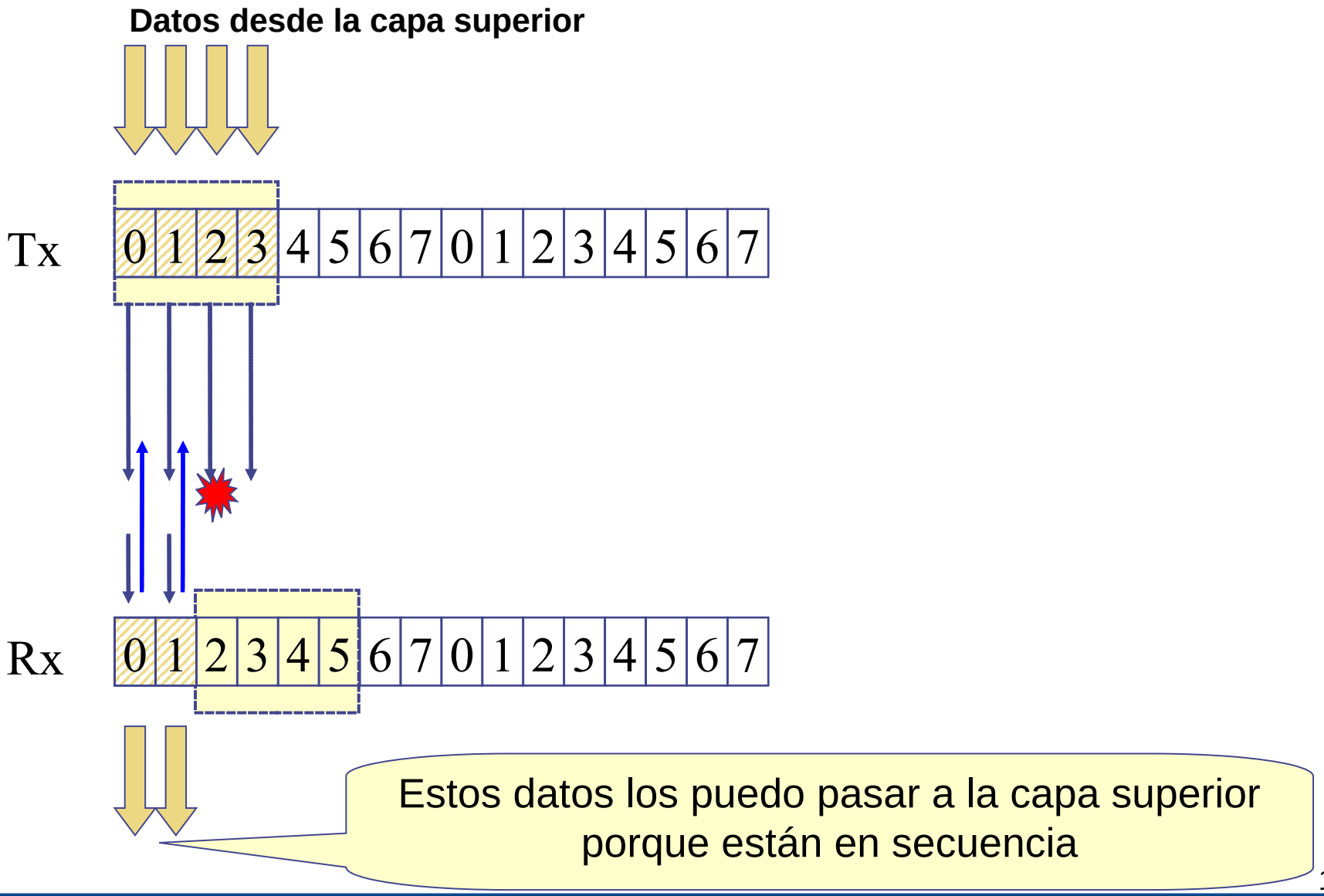
Ventanas Delizantes

- **Cuando hay errores**



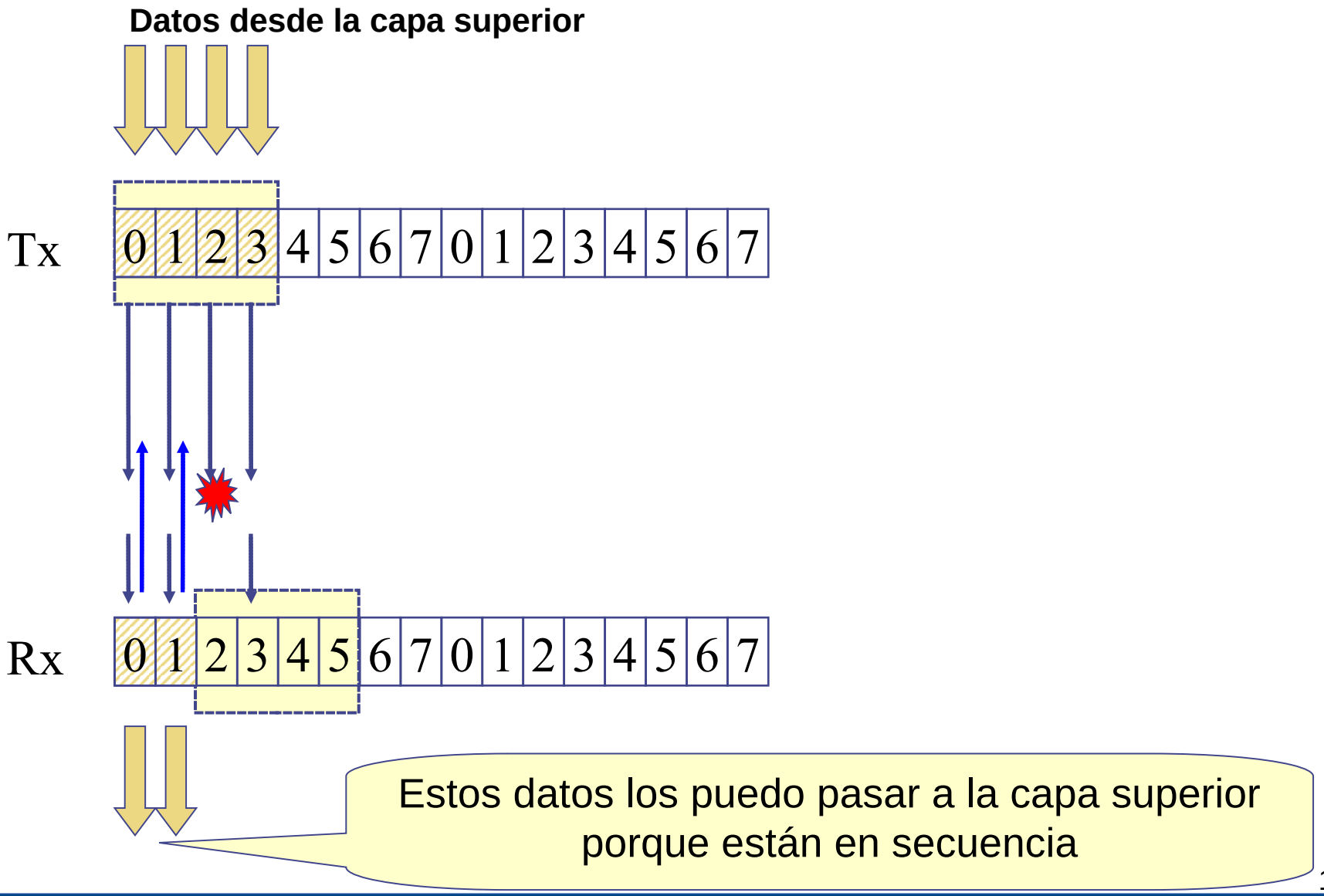
Ventanas Delizantes

- **Cuando hay errores**



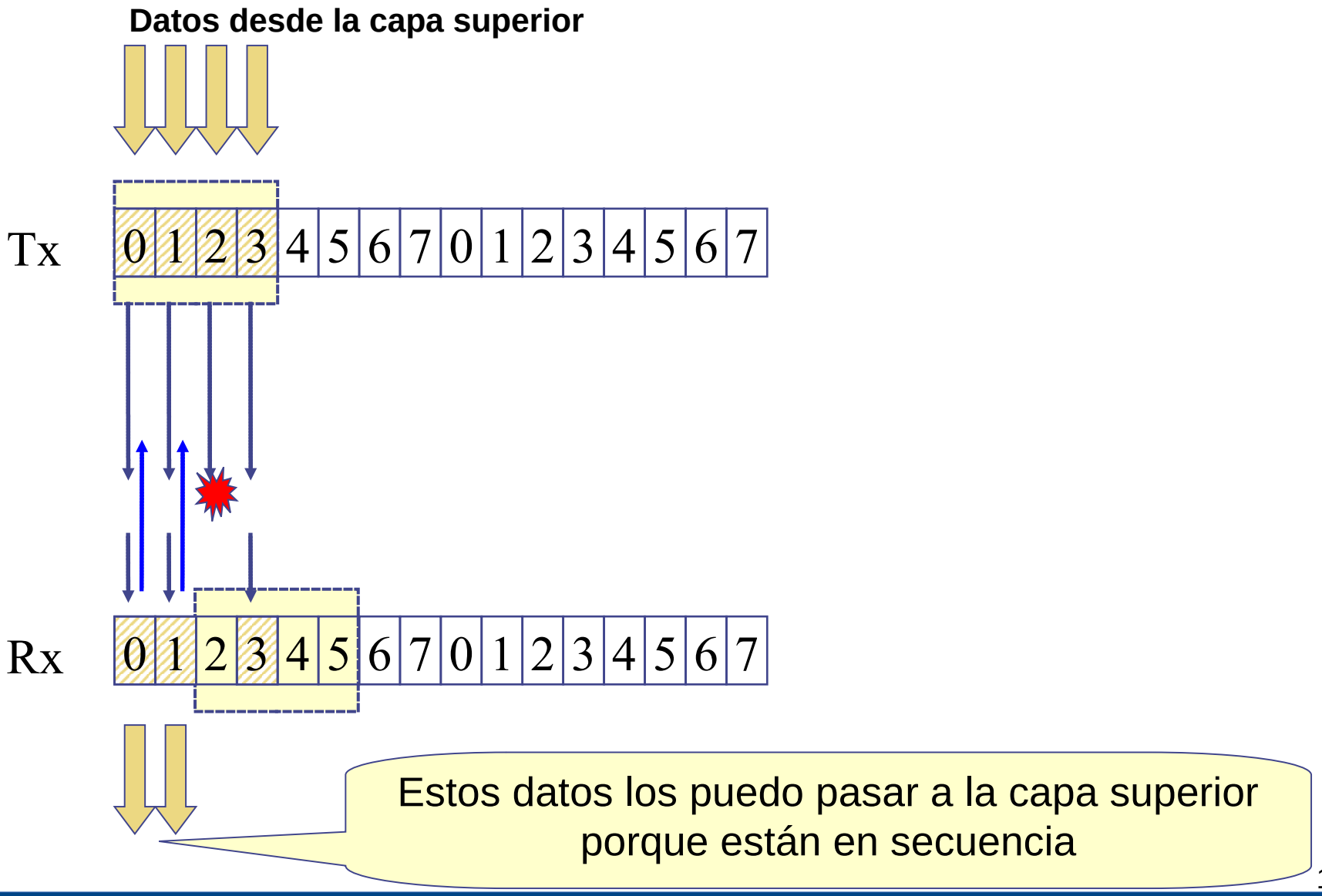
Ventanas Delizantes

- **Cuando hay errores**



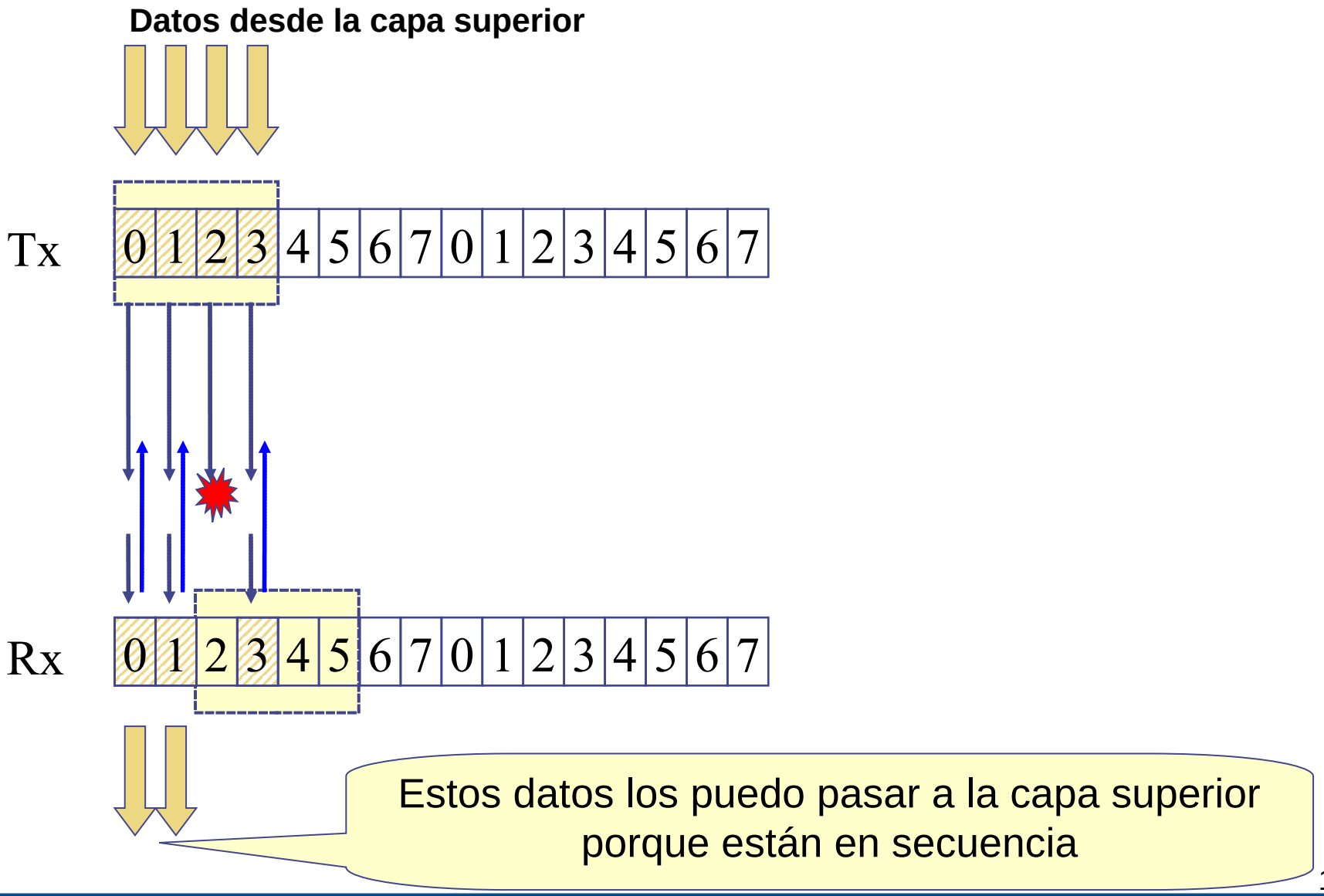
Ventanas Delizantes

- **Cuando hay errores**



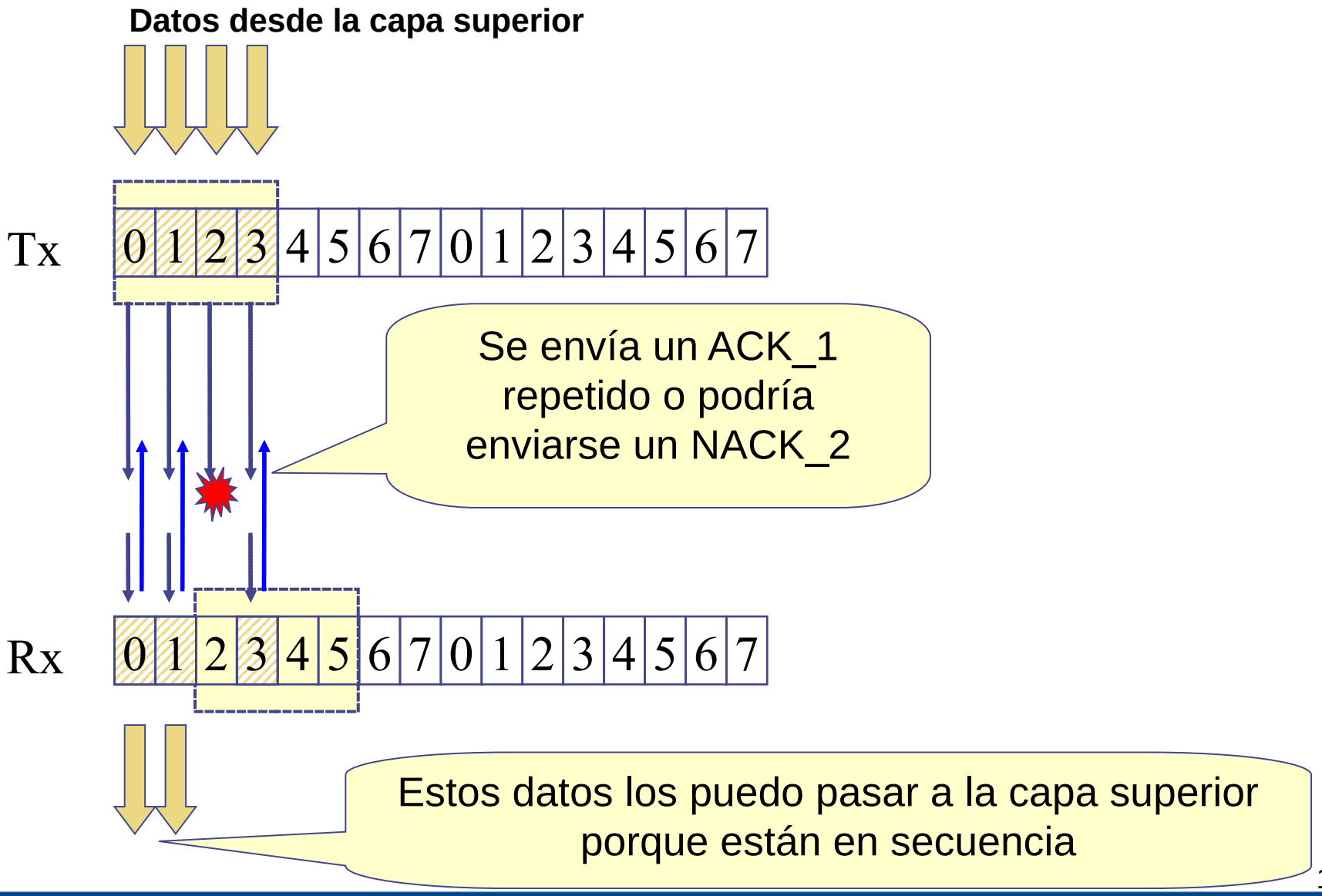
Ventanas Delizantes

- **Cuando hay errores**



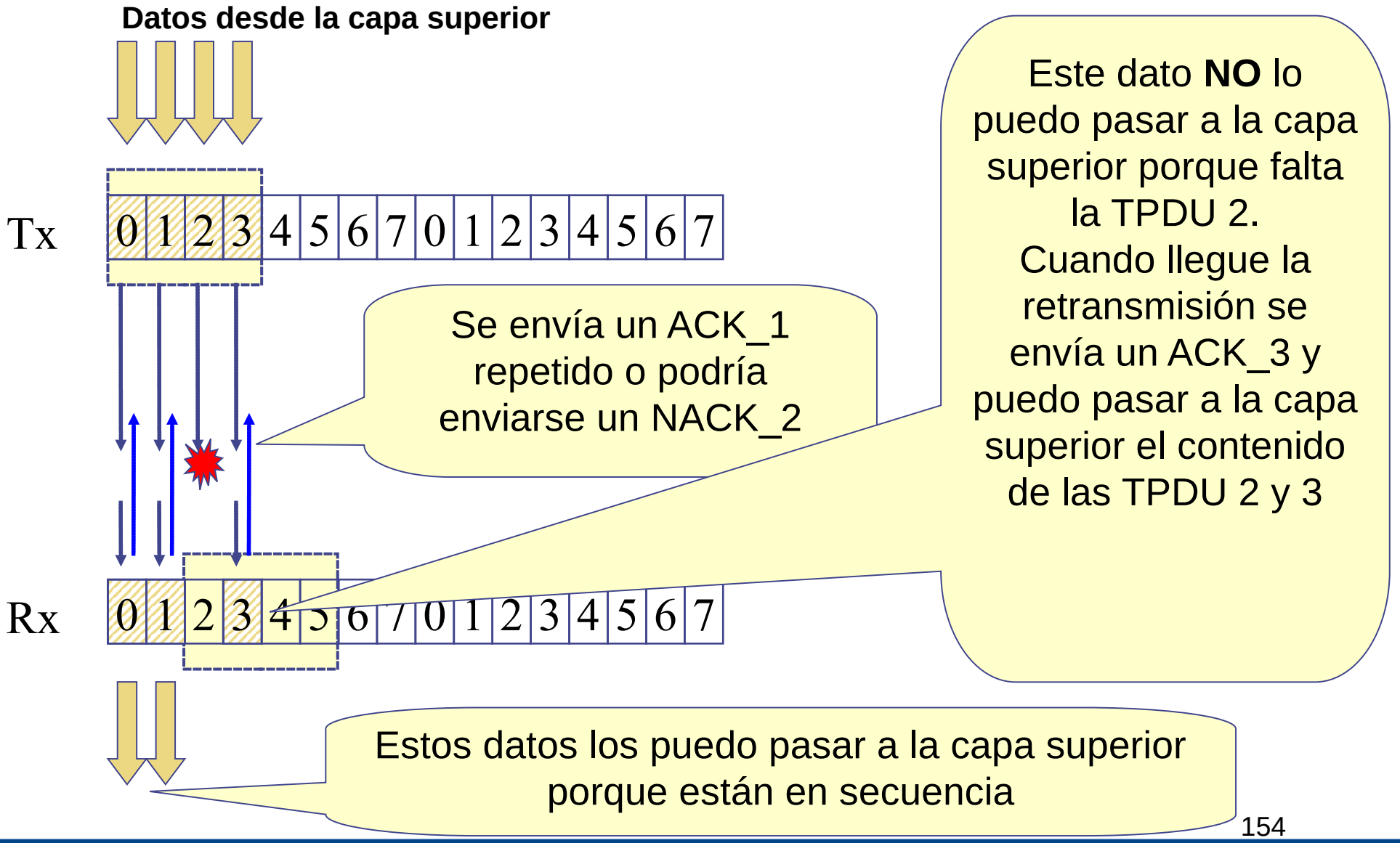
Ventanas Delizantes

- **Cuando hay errores**



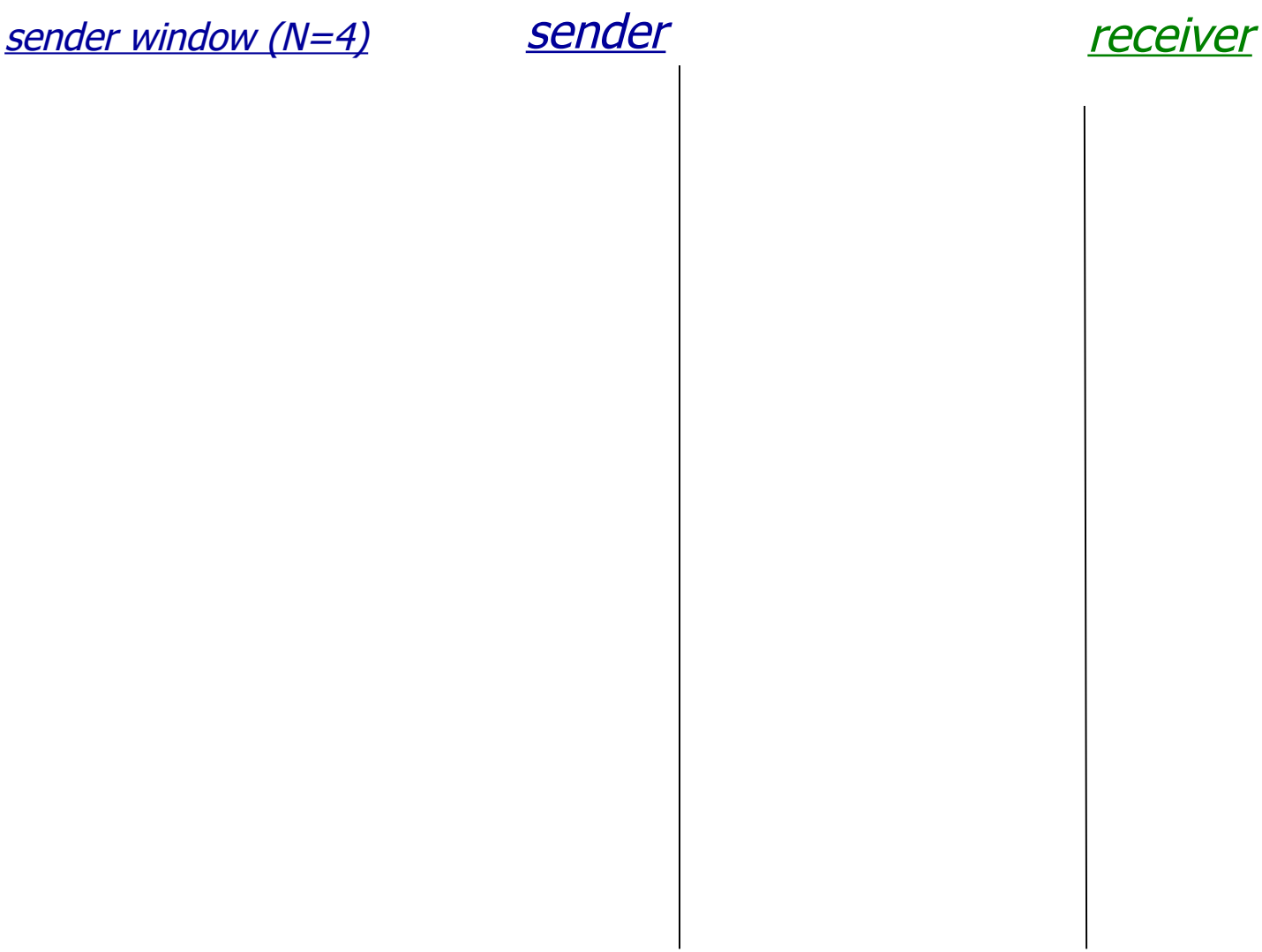
Ventanas Delizantes

- **Cuando hay errores**



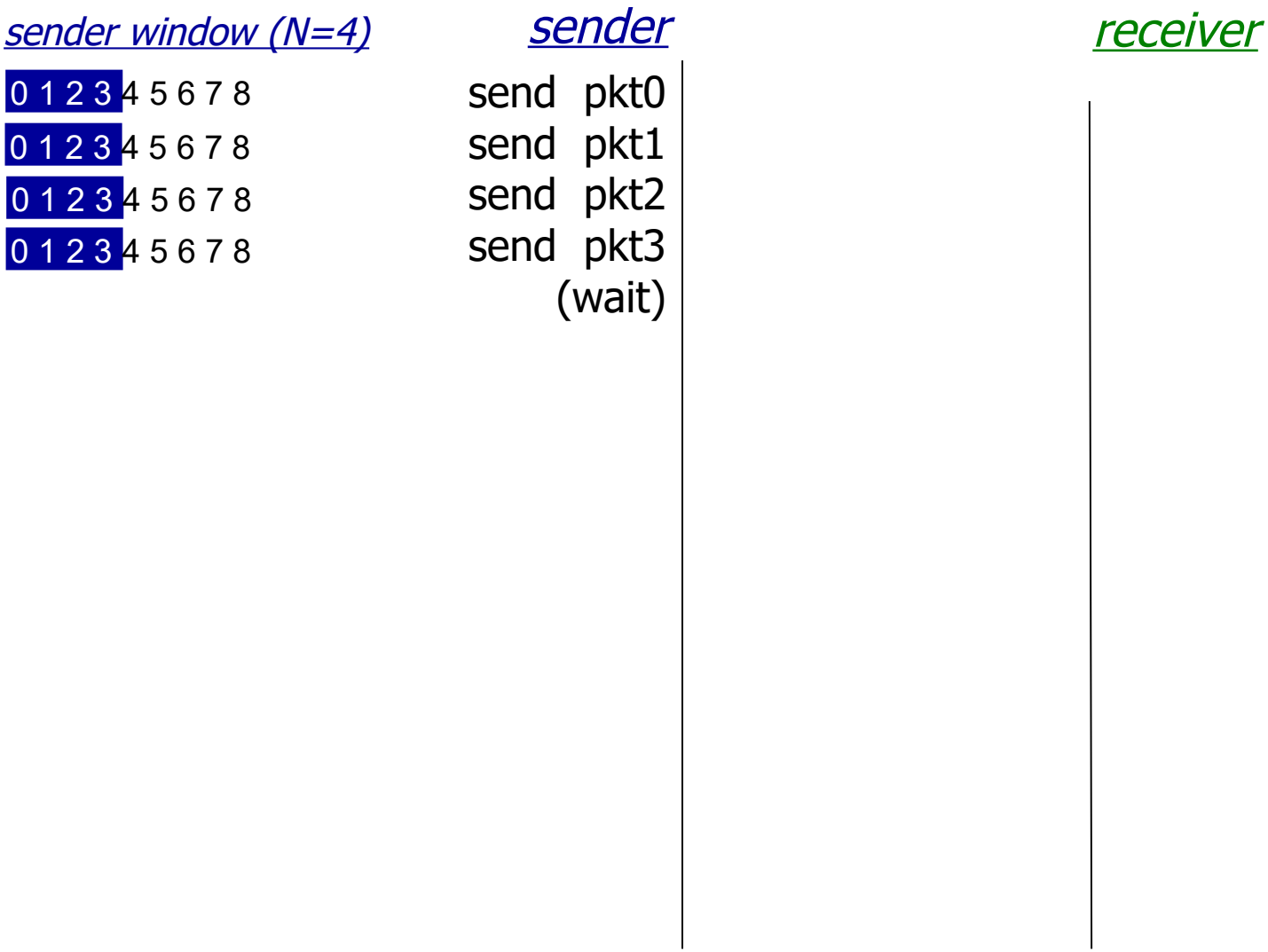
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



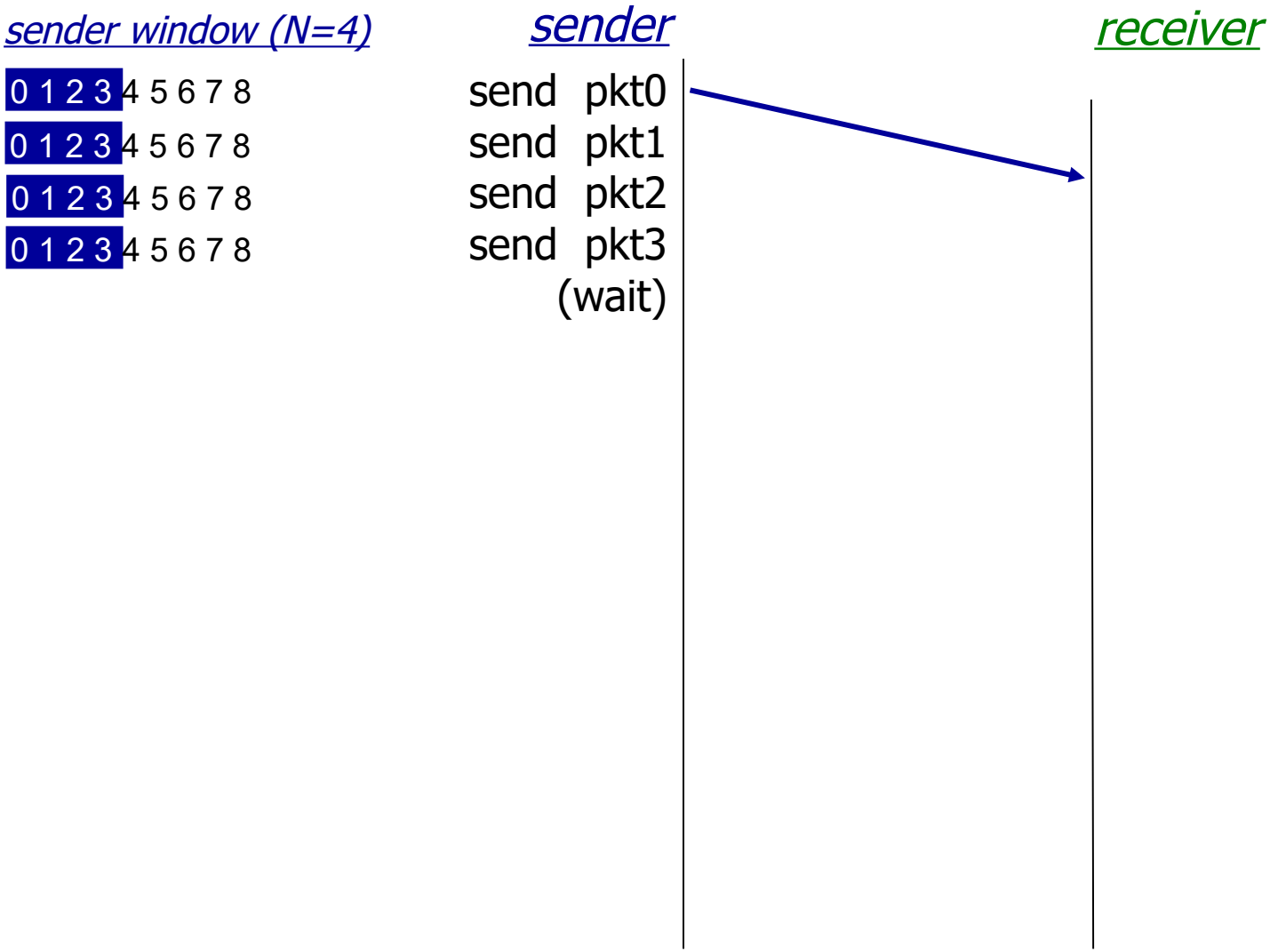
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



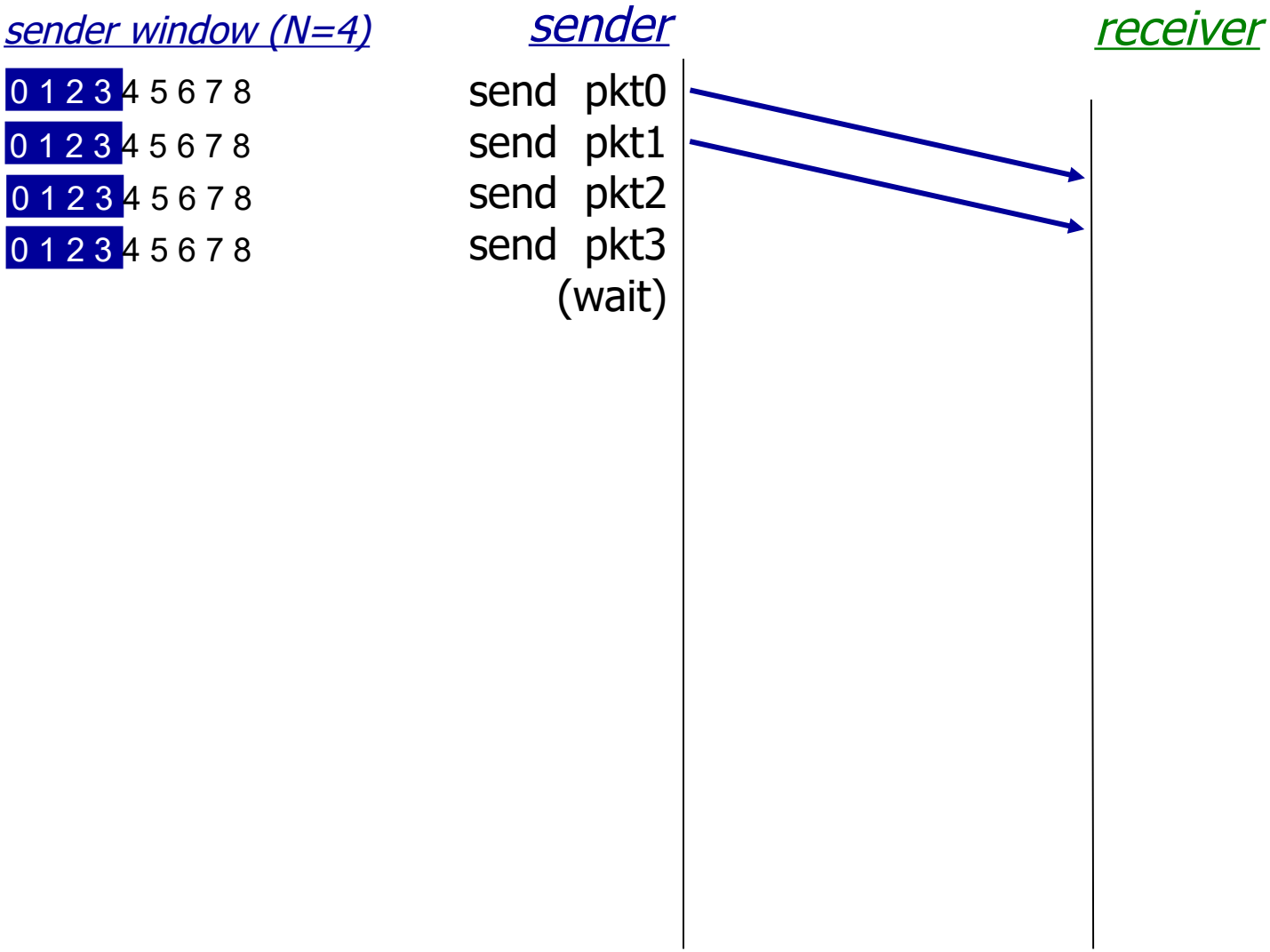
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



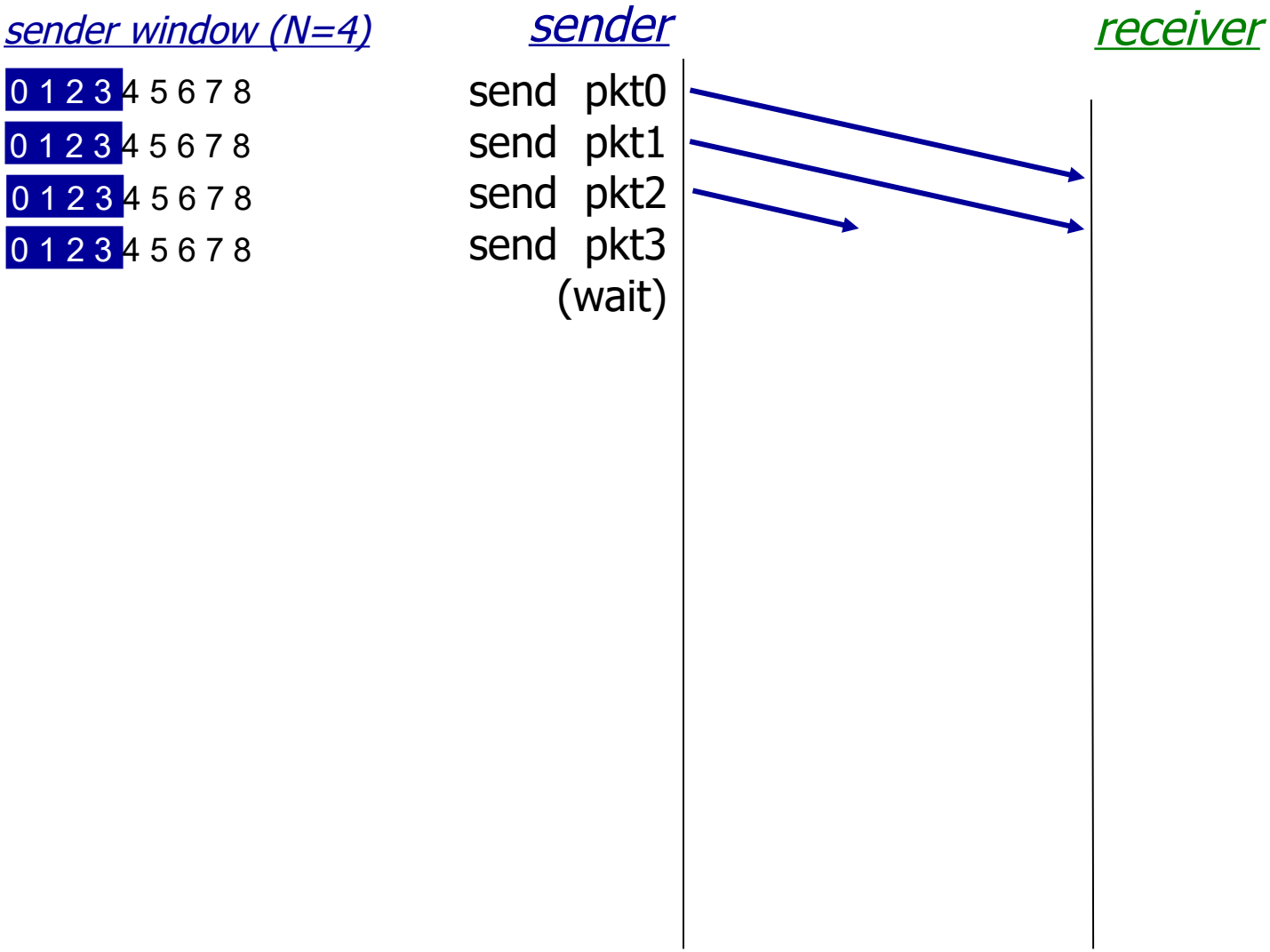
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



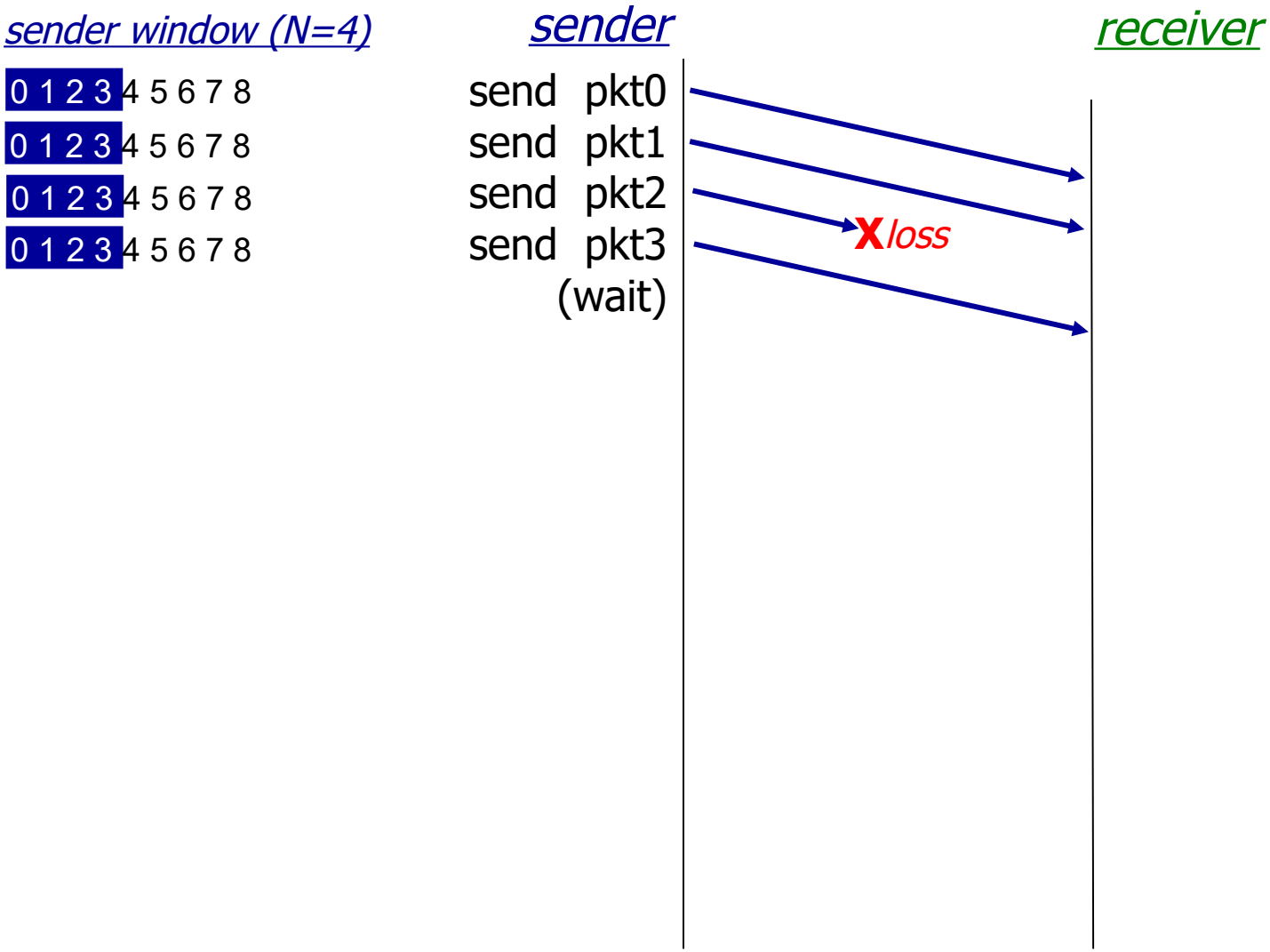
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



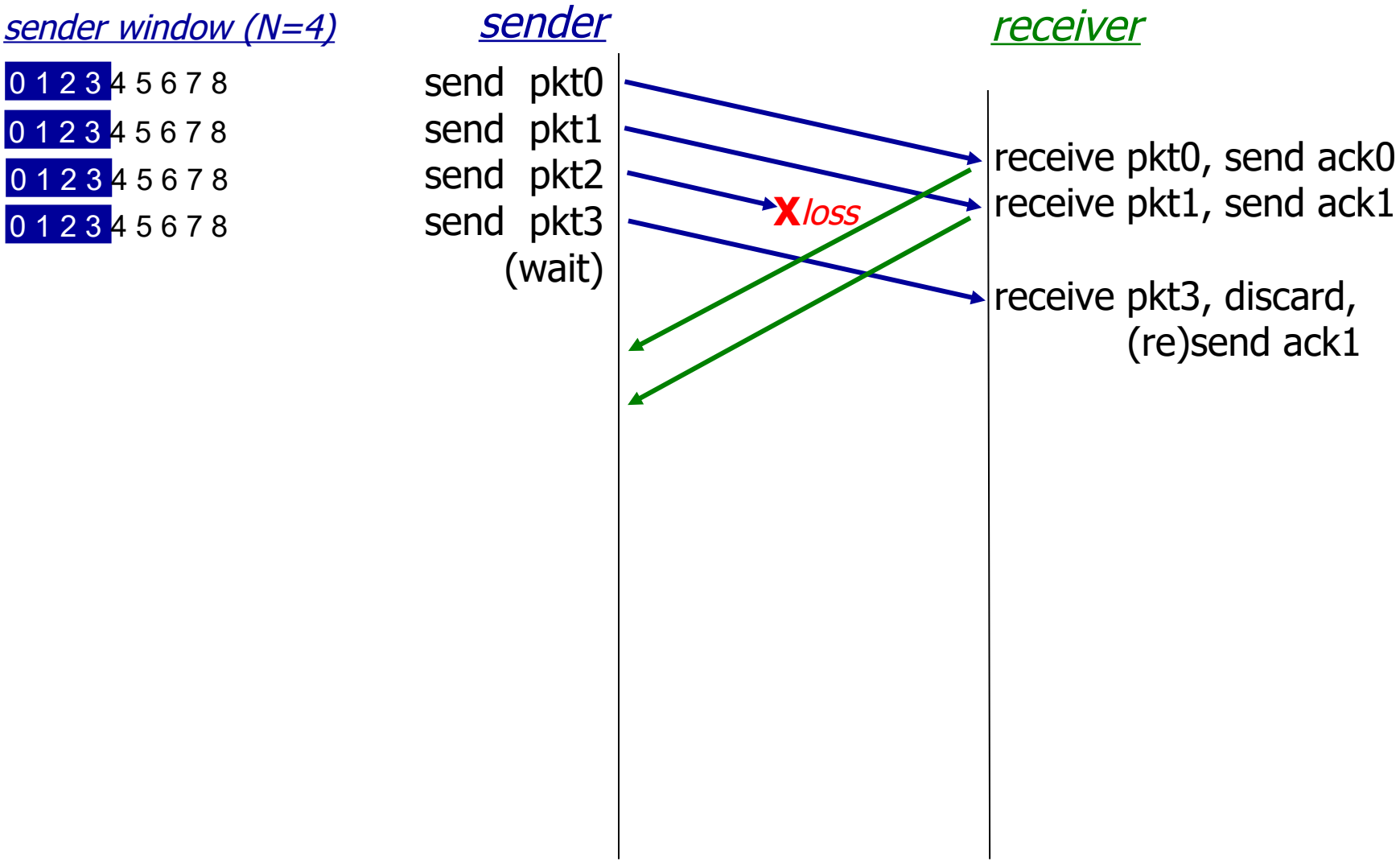
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



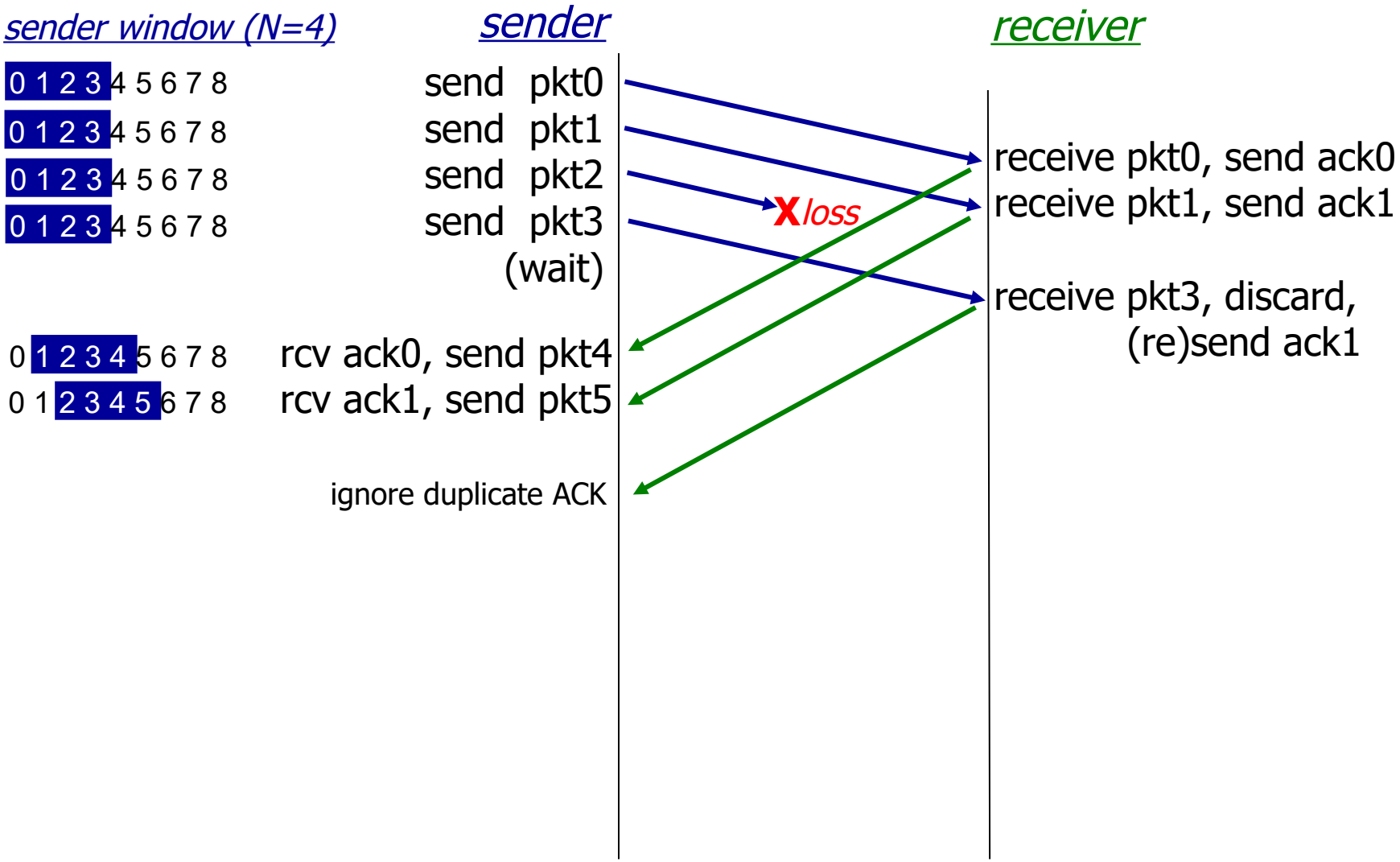
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



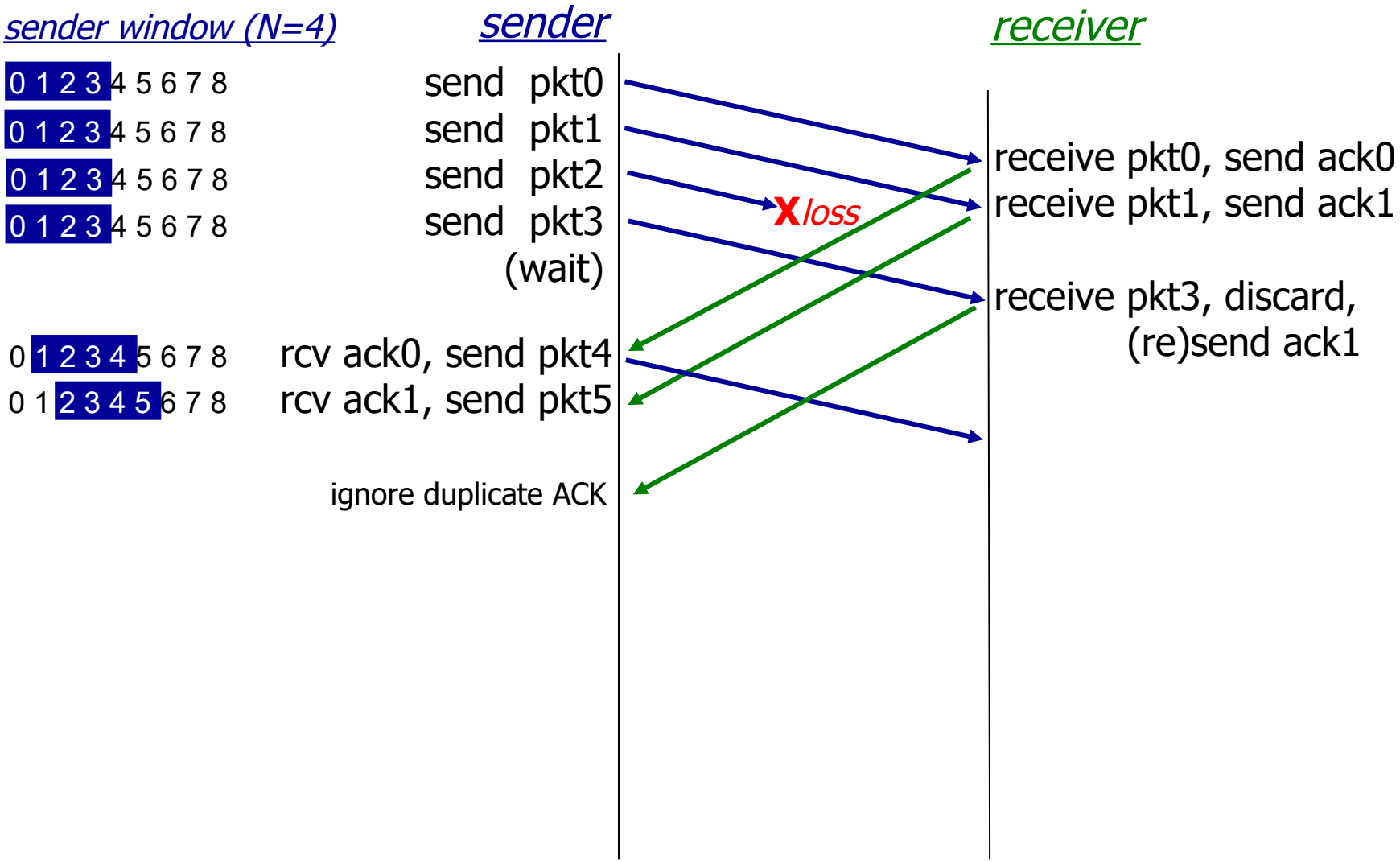
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



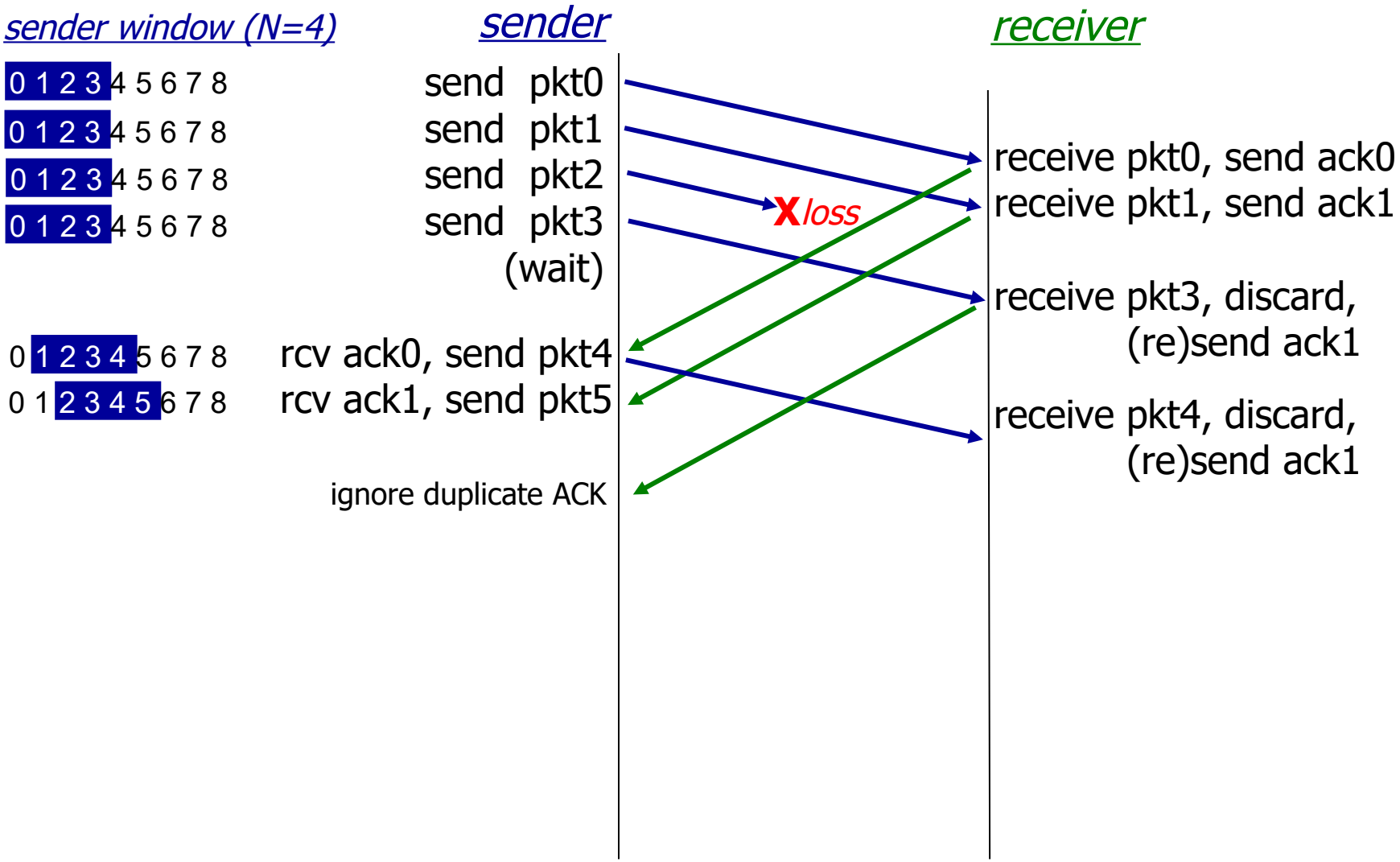
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



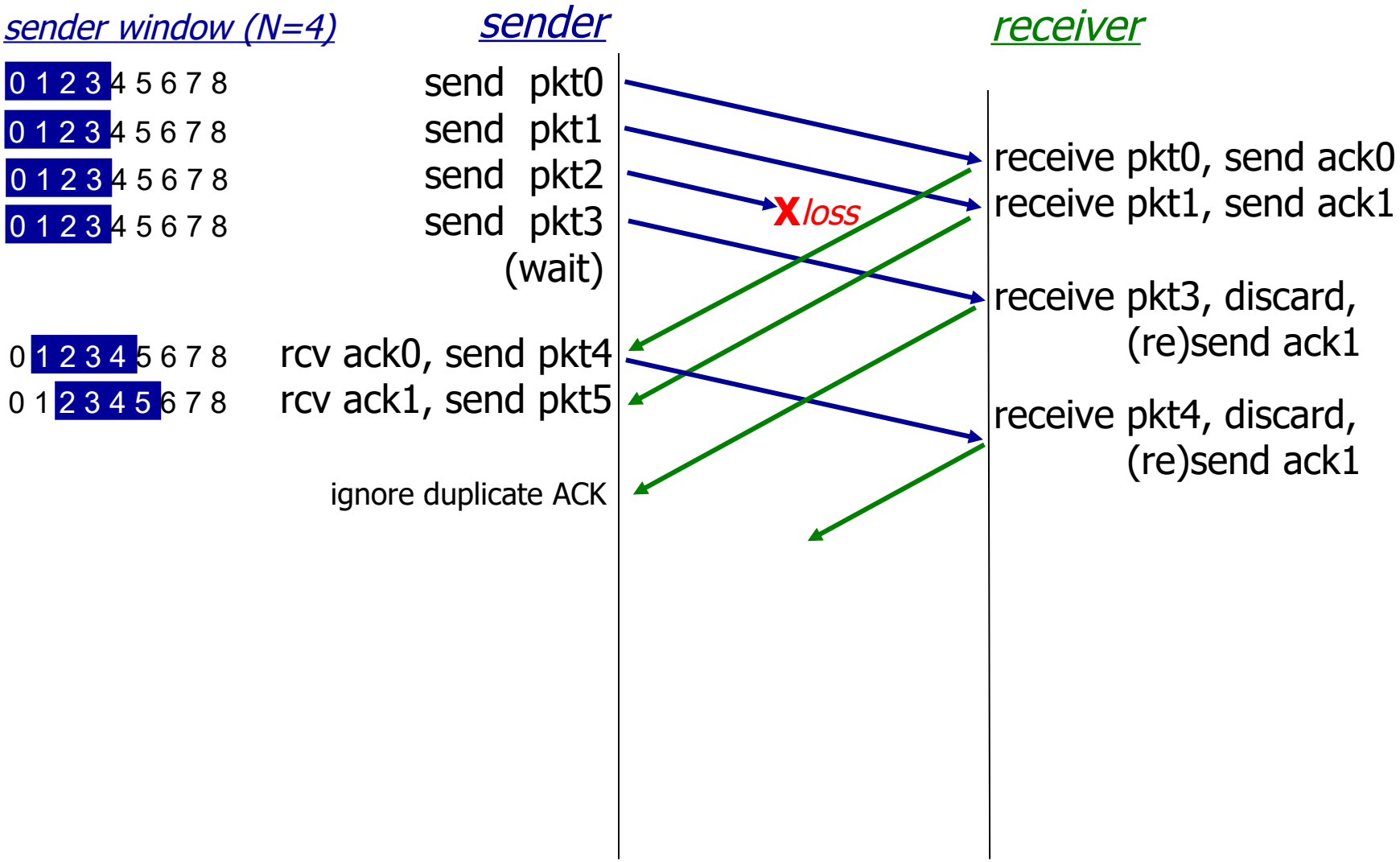
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



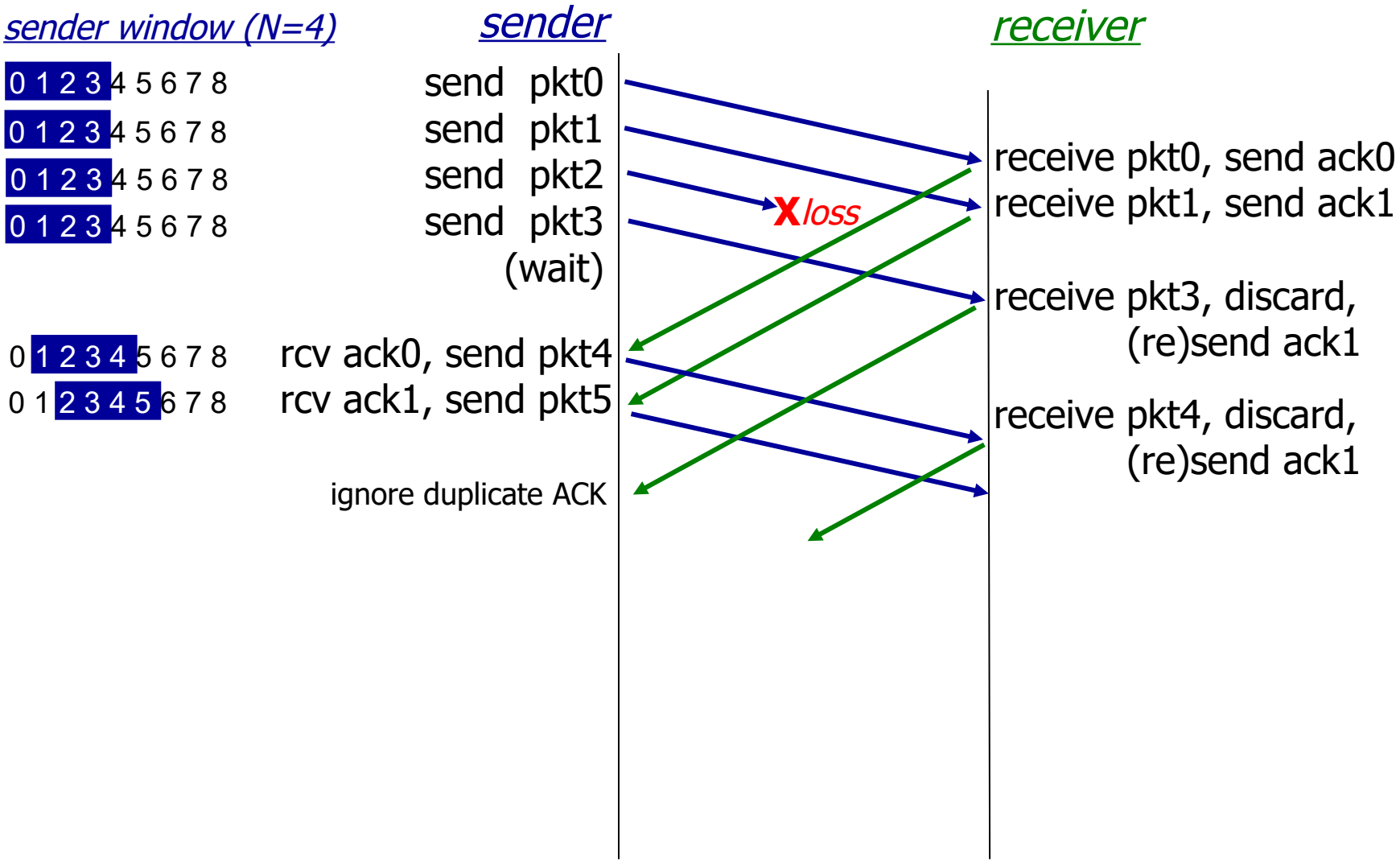
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



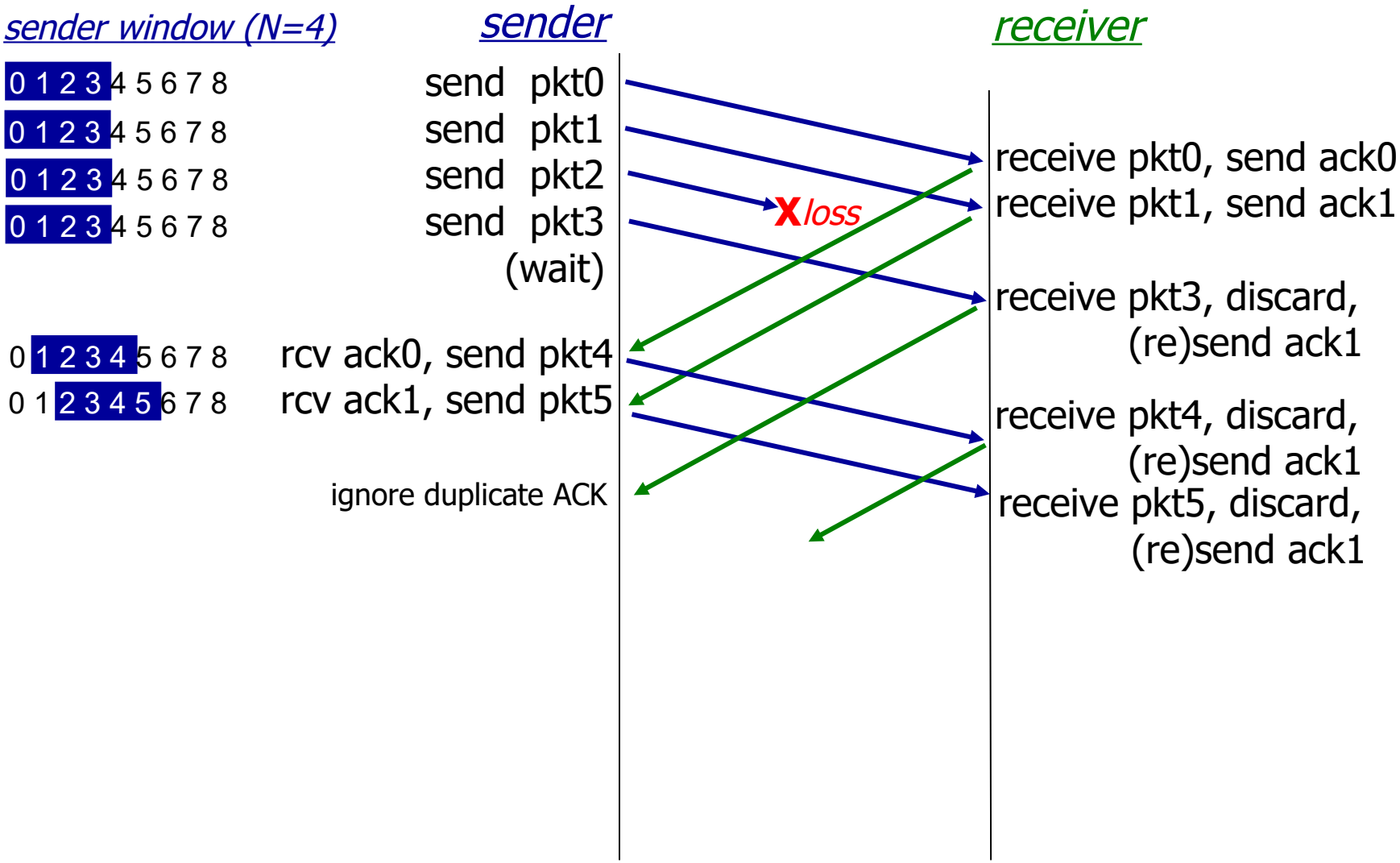
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



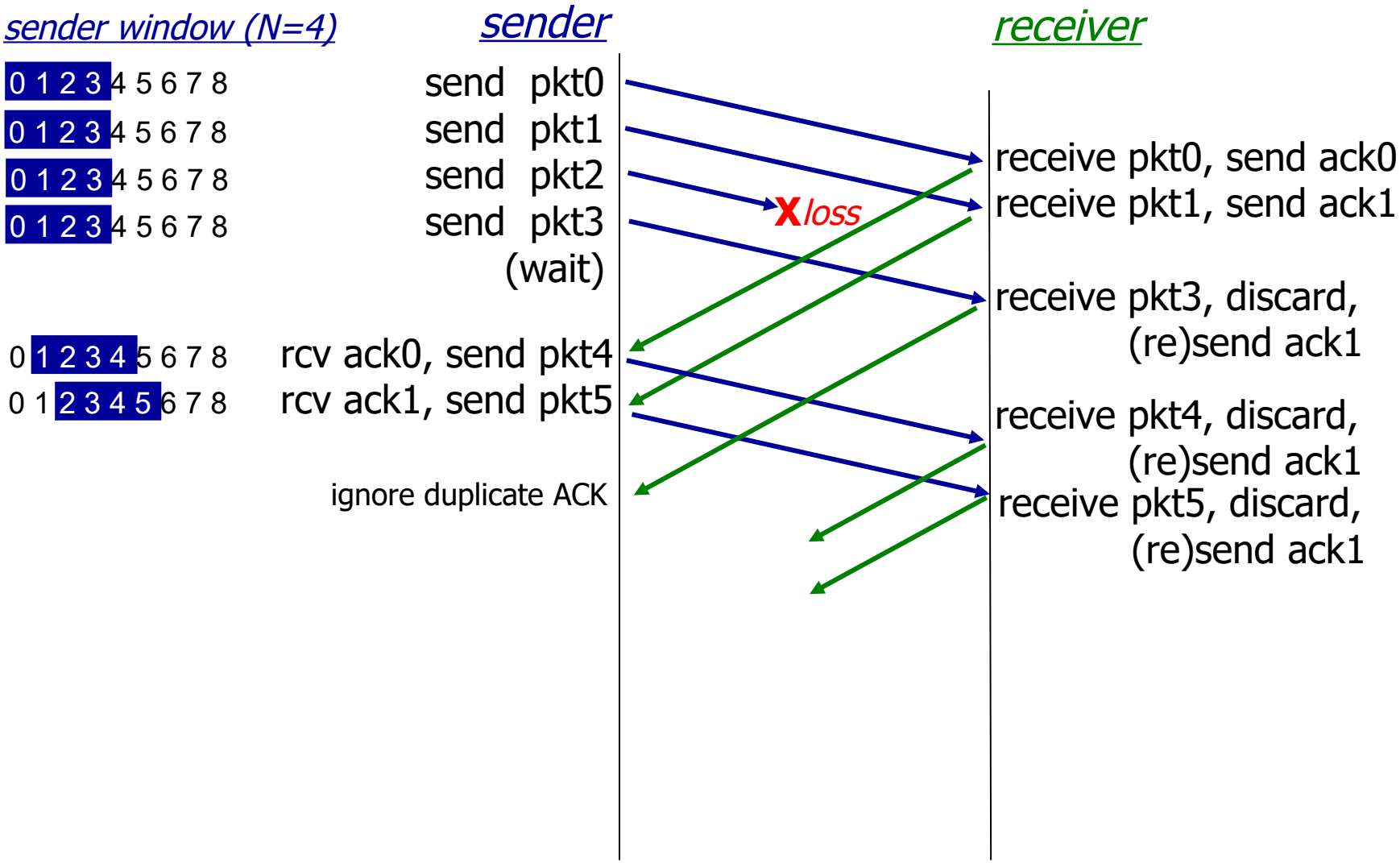
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



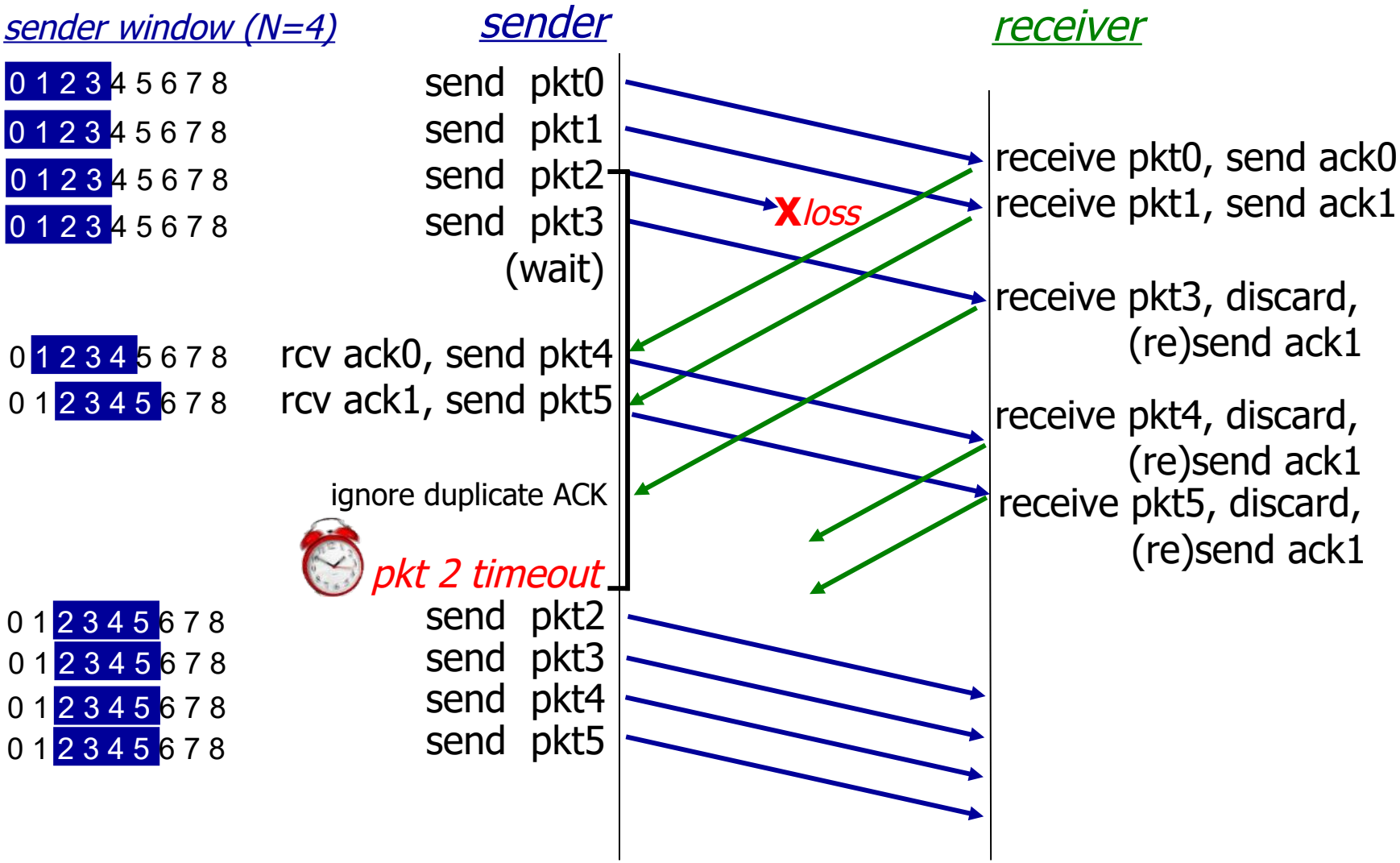
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



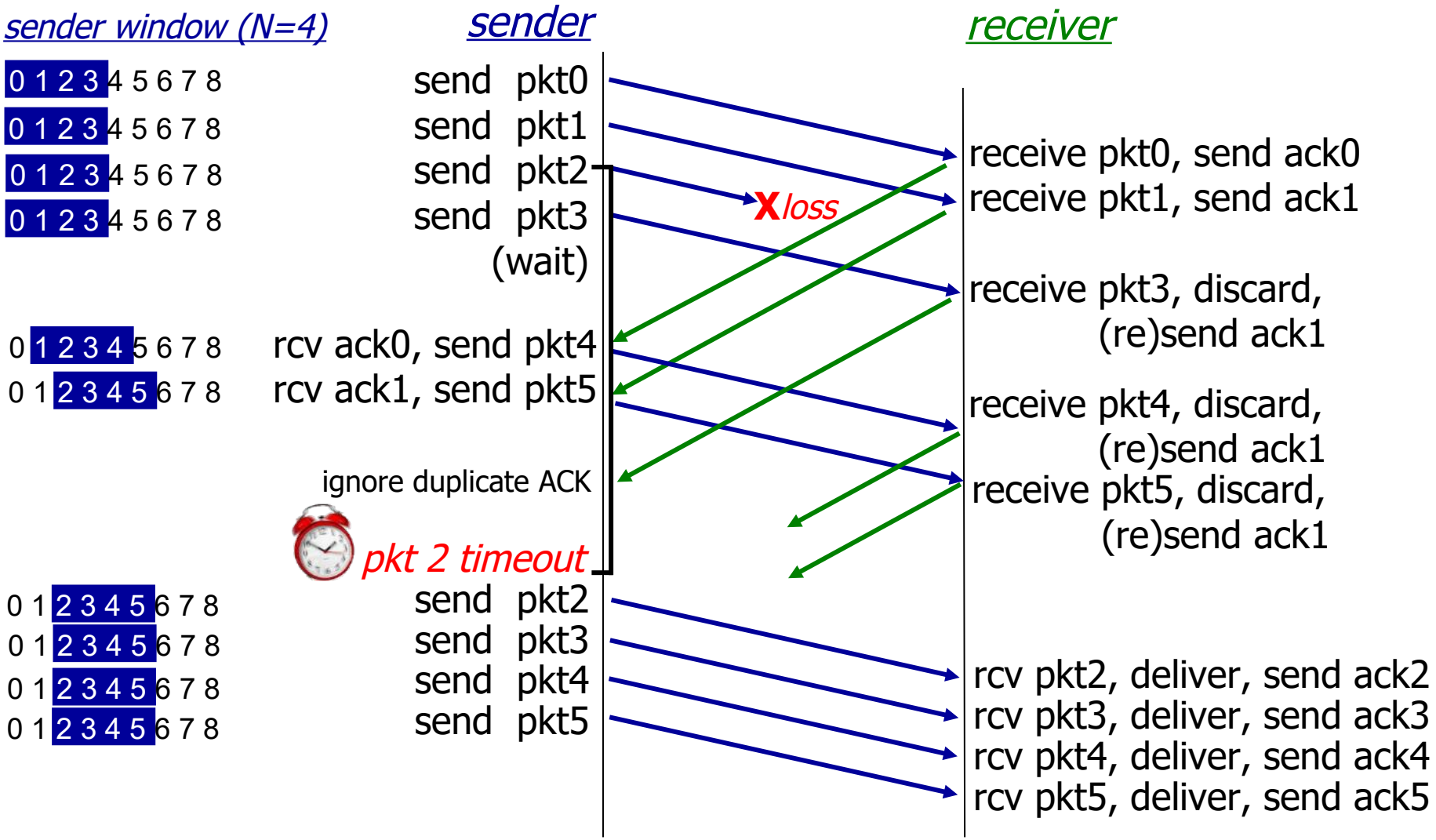
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



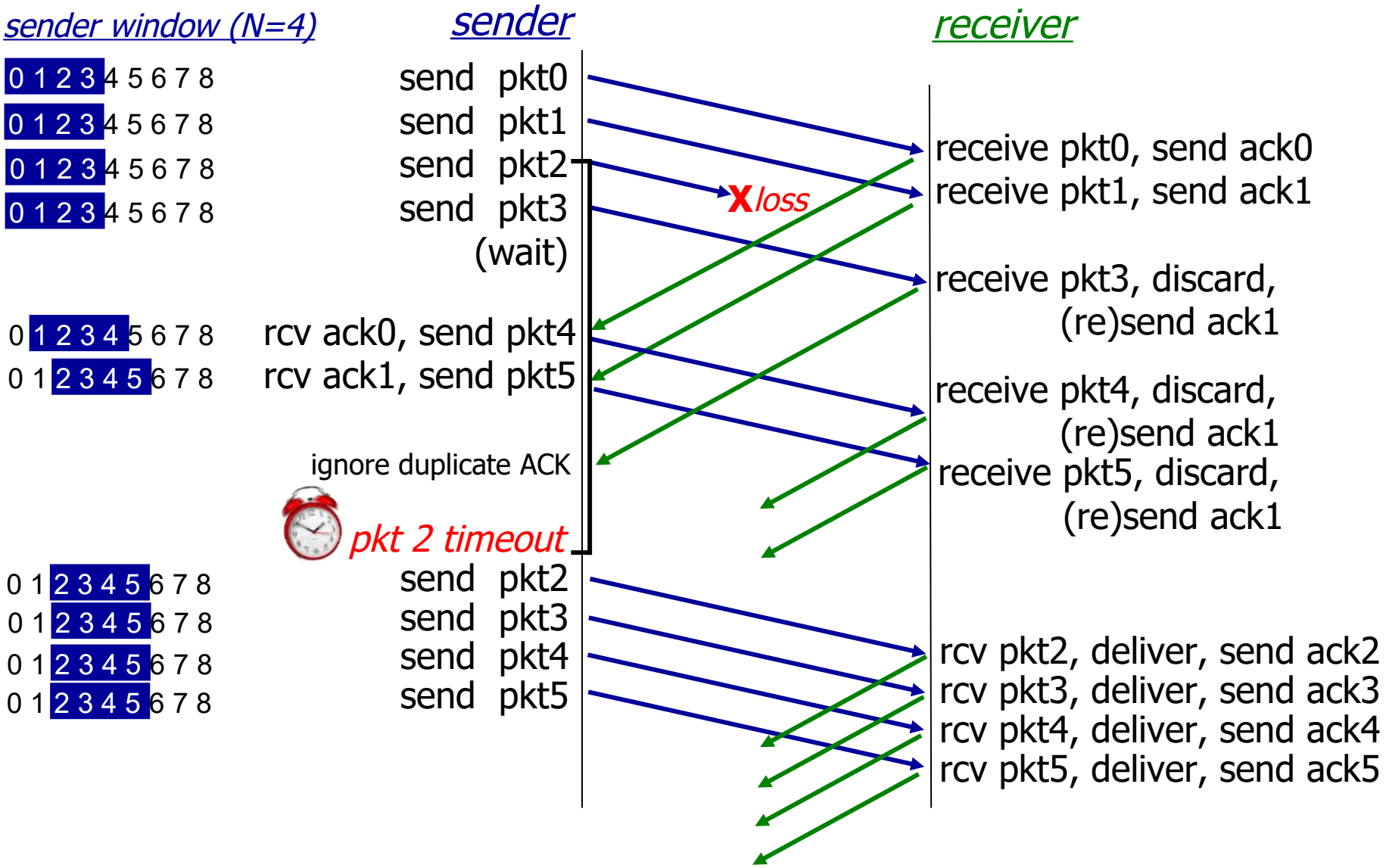
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



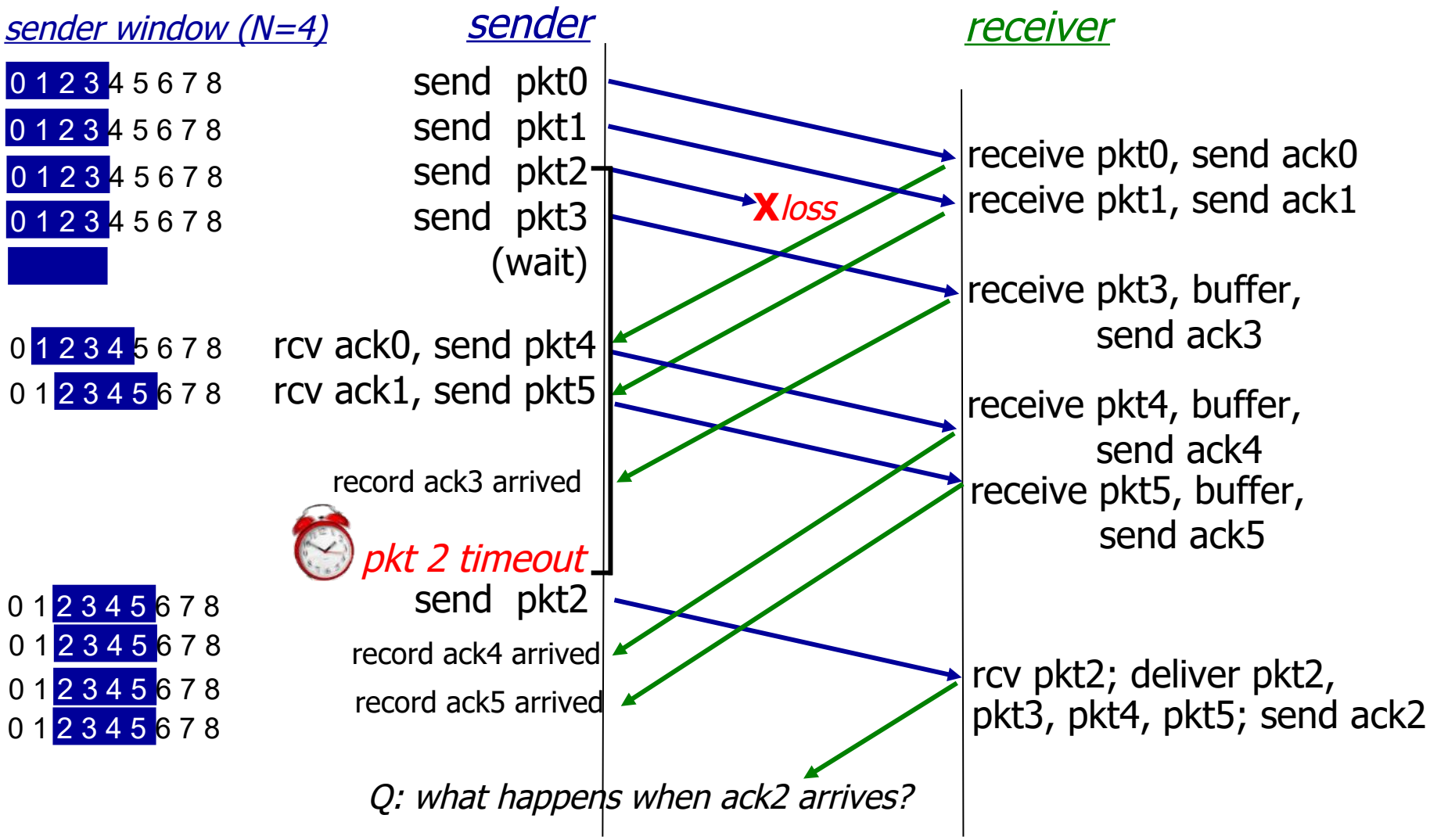
Go Back N

- Los ACK son acumulativos, ACK=X reconoce todos los números de secuencia hasta X, frente a la detección de una pérdida, reitero el último ACK



Selective Repeat

- Los ACK son selectivos, ACK=X solo reconoce el número de secuencia X. El transmisor podría detectar la pérdida y acelar la re-transmisión.
- Existe la alternativa de enviar NACK (negative ACK).



Ventanas Deslizantes - Tamaño Máximo de Ventana

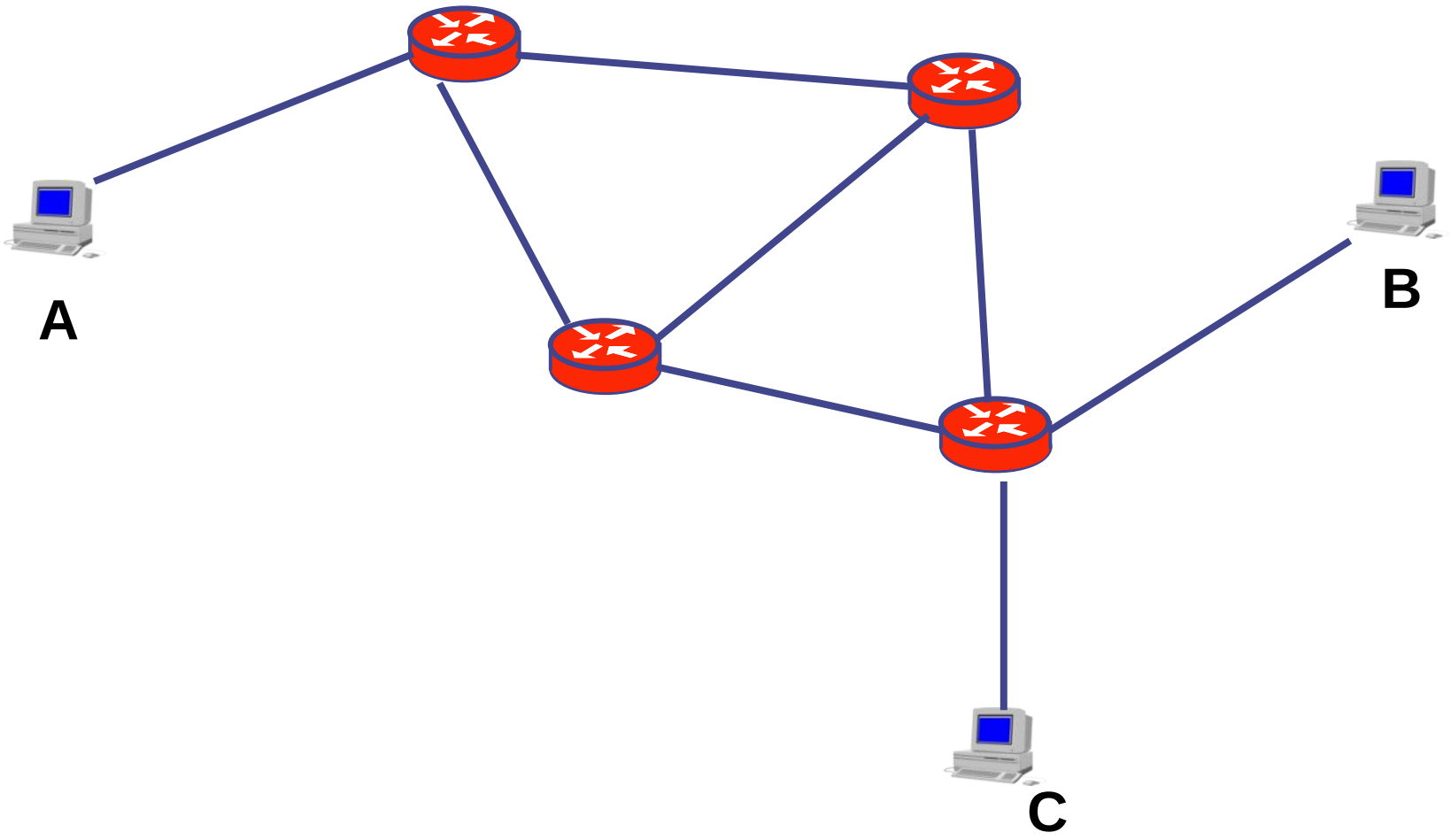
- **Ventana de transmisión TXWin**
- Si TXWin = MAX_SEQ + 1 (Ejemplo: MAX_SEQ=7)
  - Se envían 0 a 7 (ventana 8)
  - se pierden los reconocimientos
  - retransmito 0 a 7
  - se interpretan como información nueva
- Si el receptor acepta TPDU en desorden (ventana de recepción >1),
  - TXWin <= (MAX_SEQ+1)/2
- Si ventana recepción 1 (no acepta TPDU desordenadas), entonces:
  - TXWIN <= MAX_SEQ

Ventanas Deslizantes – Tamaño Máximo de Ventana

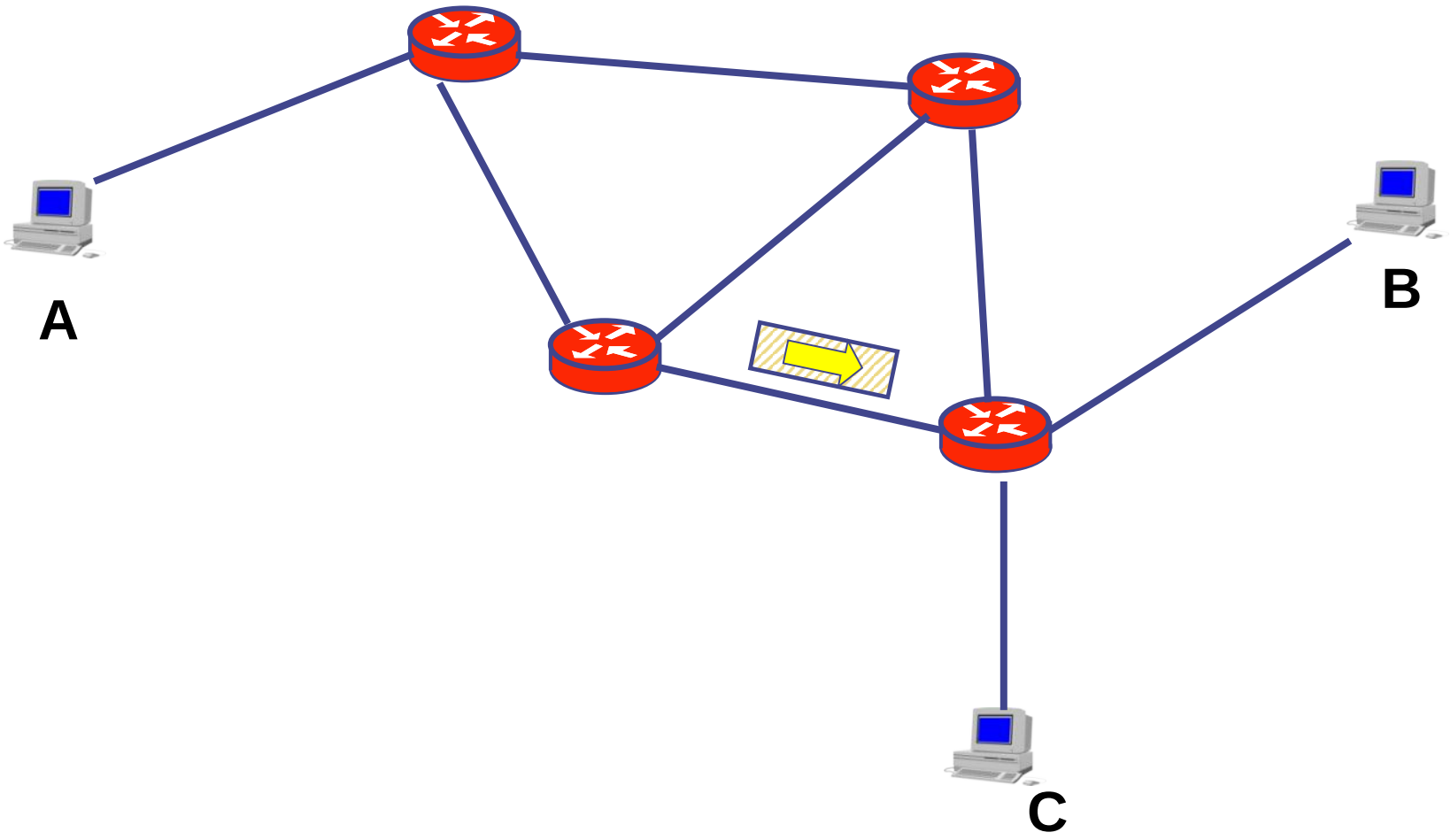
- **Ventana de recepción RXWin**
  - Si RXWin = MAX_SEQ (Ejemplo: MAX_SEQ=7)
    - TX envía 0 a 3, RX los recibe bien y envía ACKs
    - RX avanza la ventana para admitir 4 a 2
    - Se pierden los ACKs y el TX retransmite el 0
    - El “0” cae en la ventana del RX, se acepta como nuevo
  - Falla: la nueva ventana se superpone con la vieja
- Para evitar la falla: RXWin <=(MAX_SEQ+1)/2

- **Buffers (memoria) VS Ventana**
  - “número de buffers disponibles del receptor” = **ventana de recepción** y no la cantidad de números de secuencia
  - Es posible utilizar de un espacio de números de secuencia grande, e igual trabajar con equipos con poca capacidad de memoria.

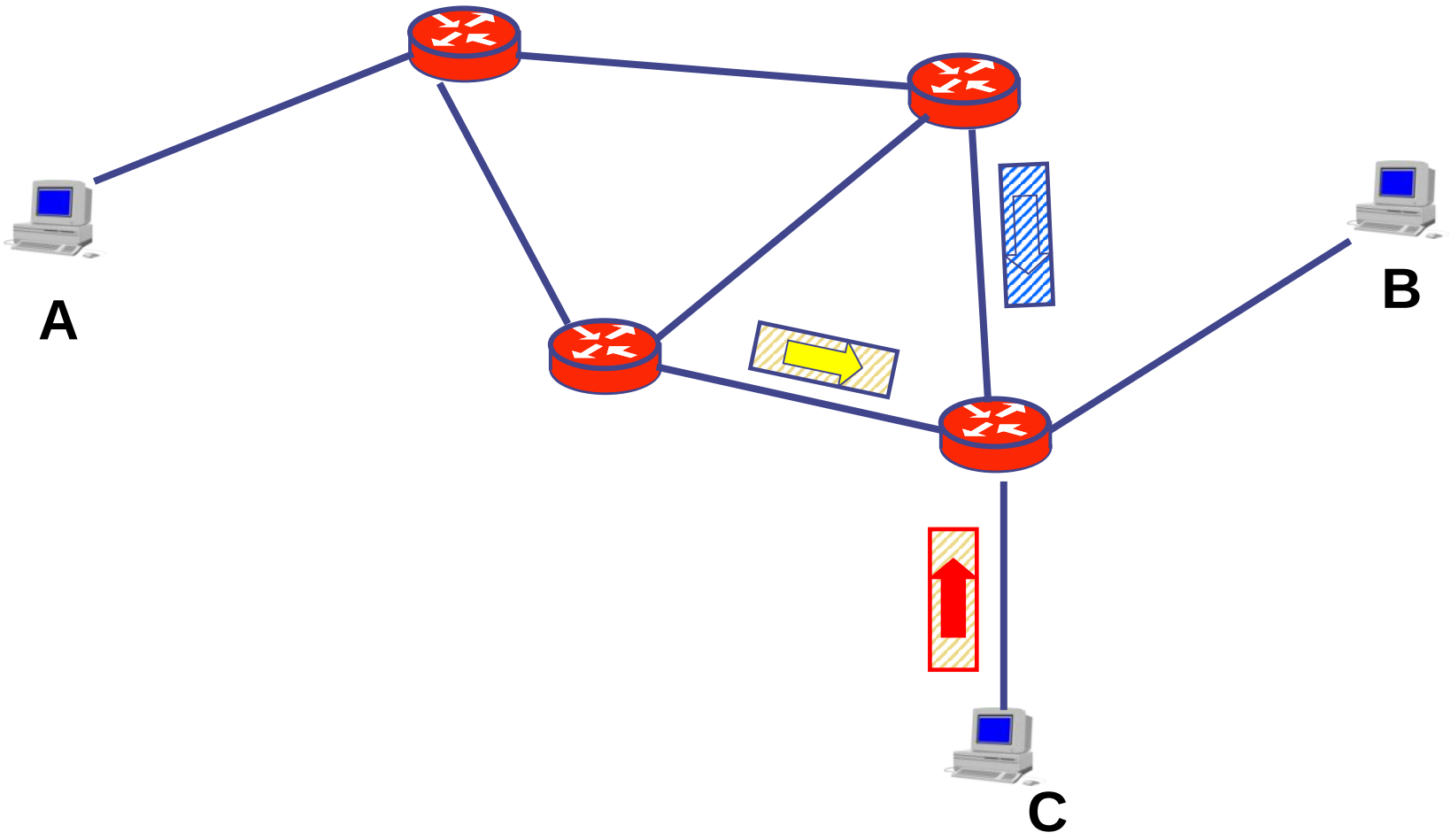
Motivando Conceptos – Retardo Variable y Descartes



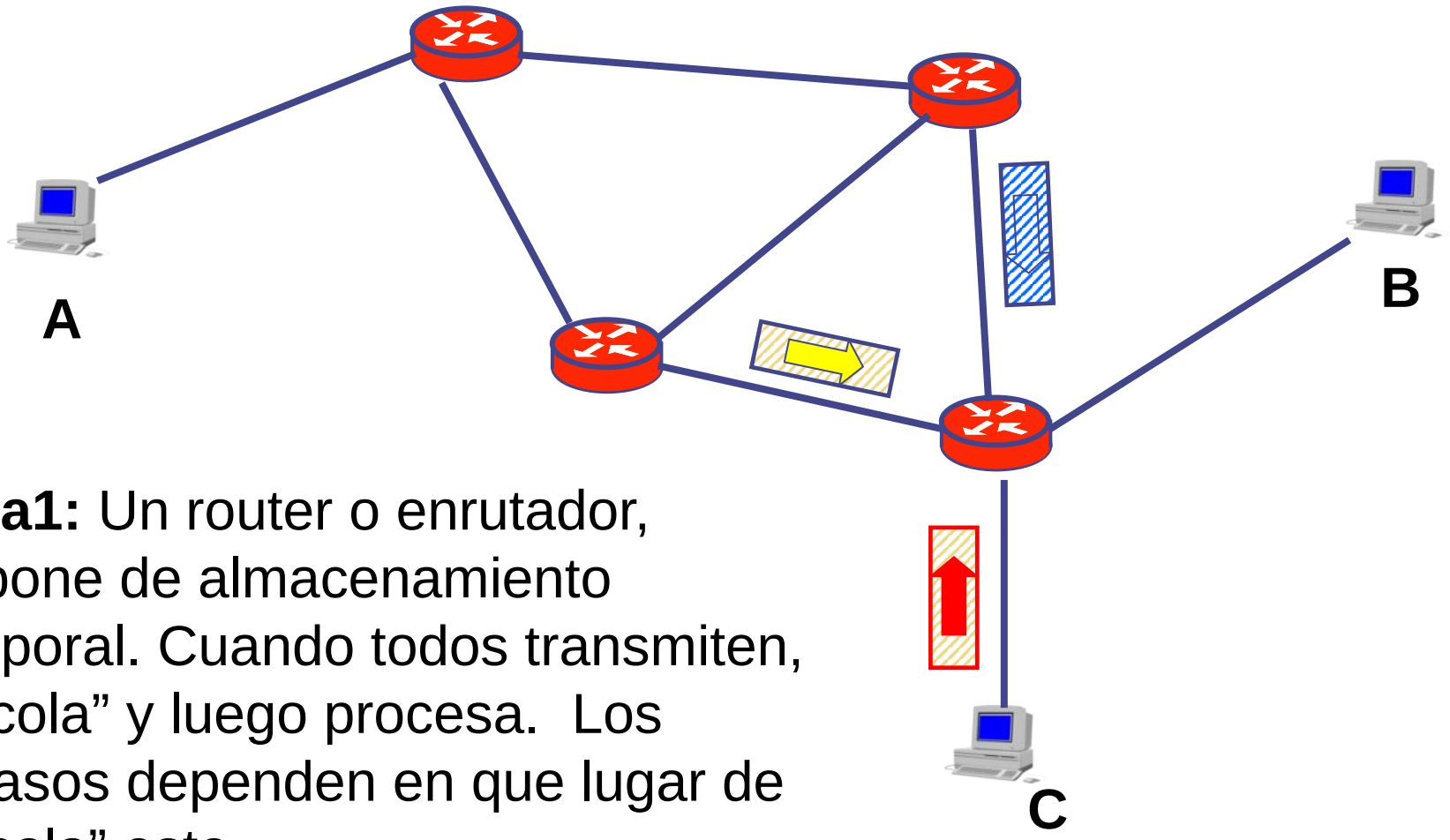
Motivando Conceptos – Retardo Variable y Descartes



Motivando Conceptos – Retardo Variable y Descartes



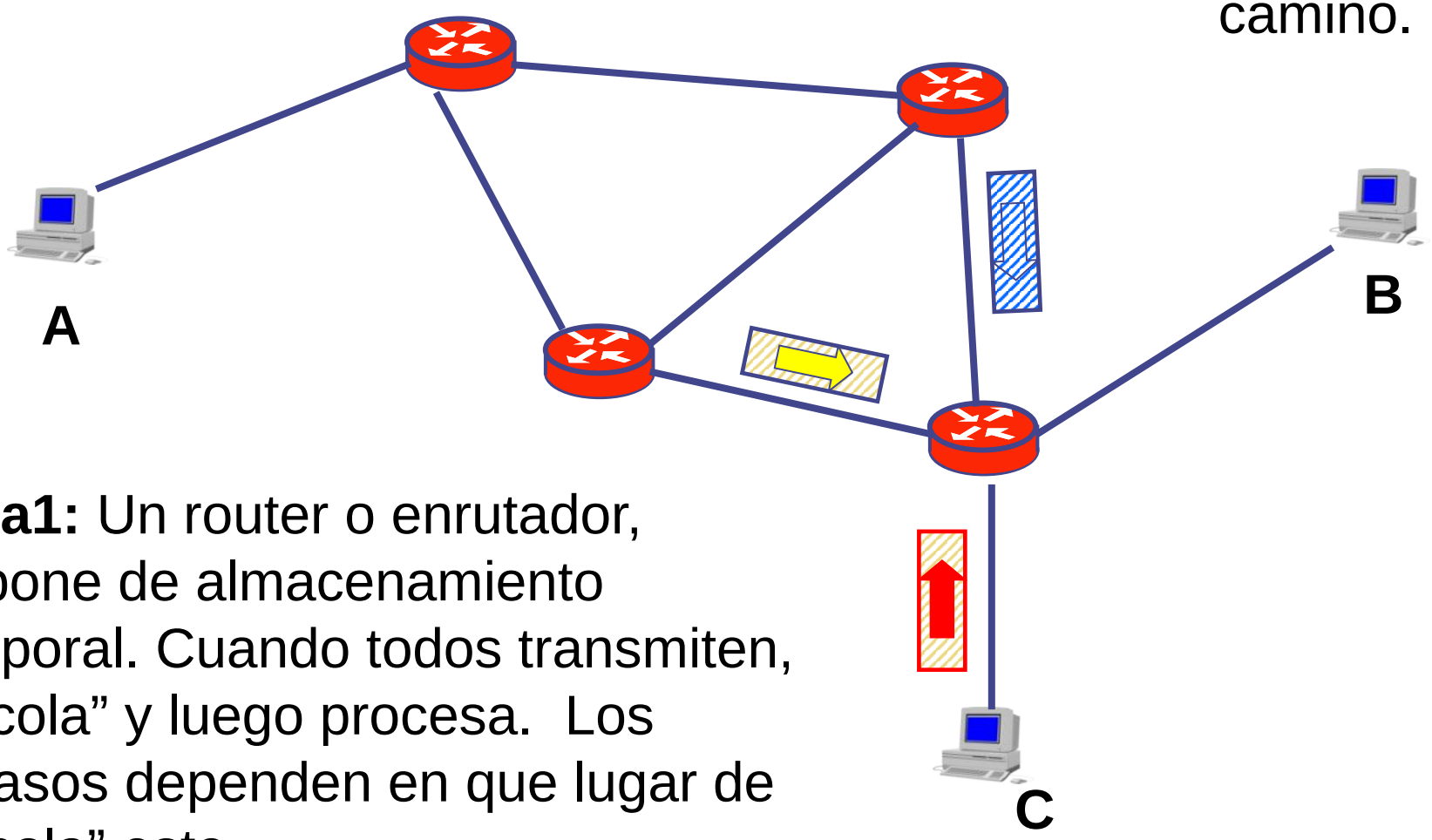
Motivando Conceptos – Retardo Variable y Descartes



Nota1: Un router o enrutador, dispone de almacenamiento temporal. Cuando todos transmiten, "encola" y luego procesa. Los retrasos dependen en que lugar de la "cola" este.

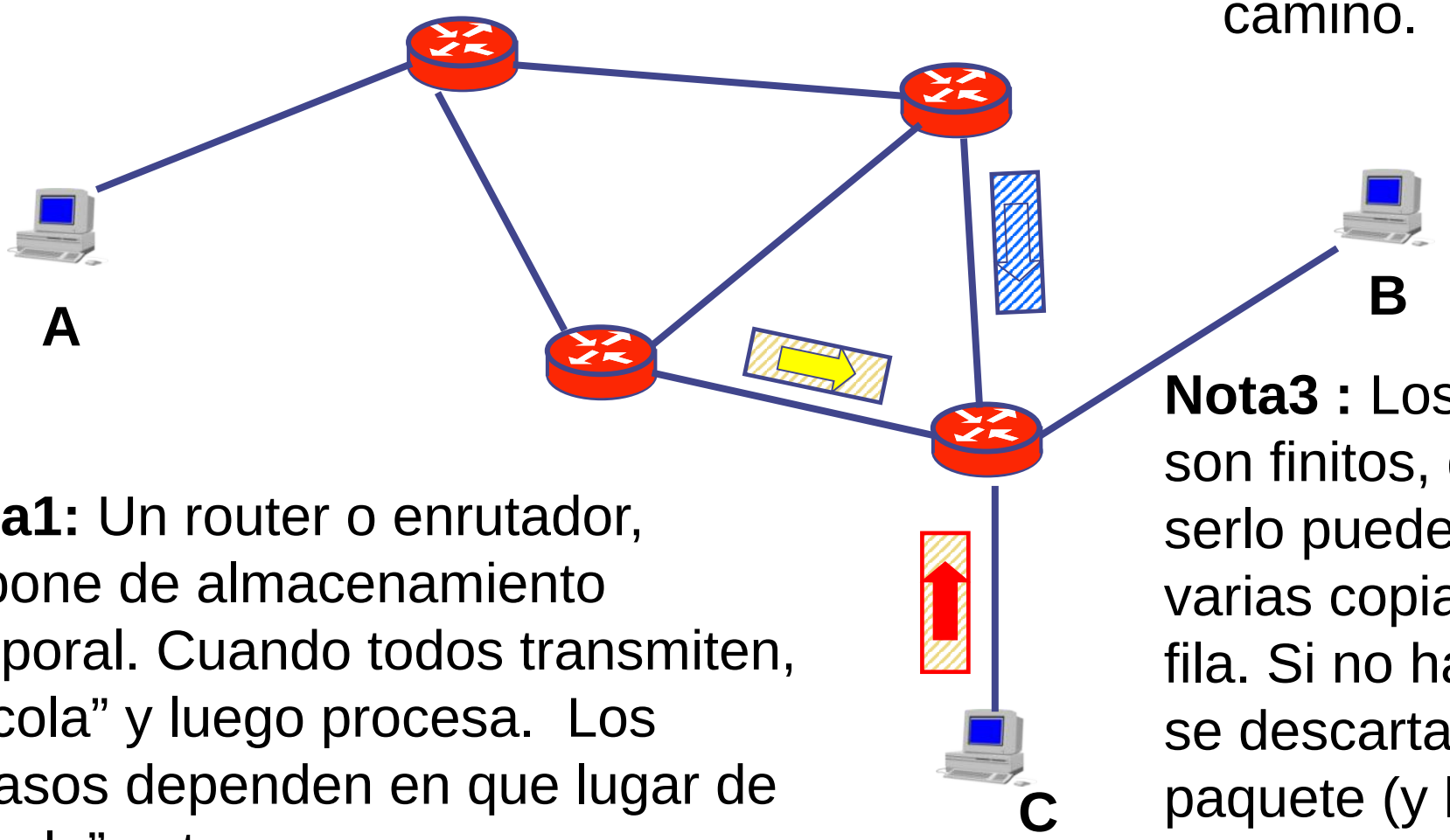
Motivando Conceptos – Retardo Variable y Descartes

Nota2 : Tantas “colas” (buffers) como routers en el camino.



Nota1: Un router o enrutador, dispone de almacenamiento temporal. Cuando todos transmiten, “encola” y luego procesa. Los retrasos dependen en que lugar de la “cola” este.

Motivando Conceptos – Retardo Variable y Descartes



Nota1: Un router o enrutador, dispone de almacenamiento temporal. Cuando todos transmiten, "encola" y luego procesa. Los retrasos dependen en que lugar de la "cola" este.

Nota2 : Tantas "colas" (buffers) como routers en el camino.

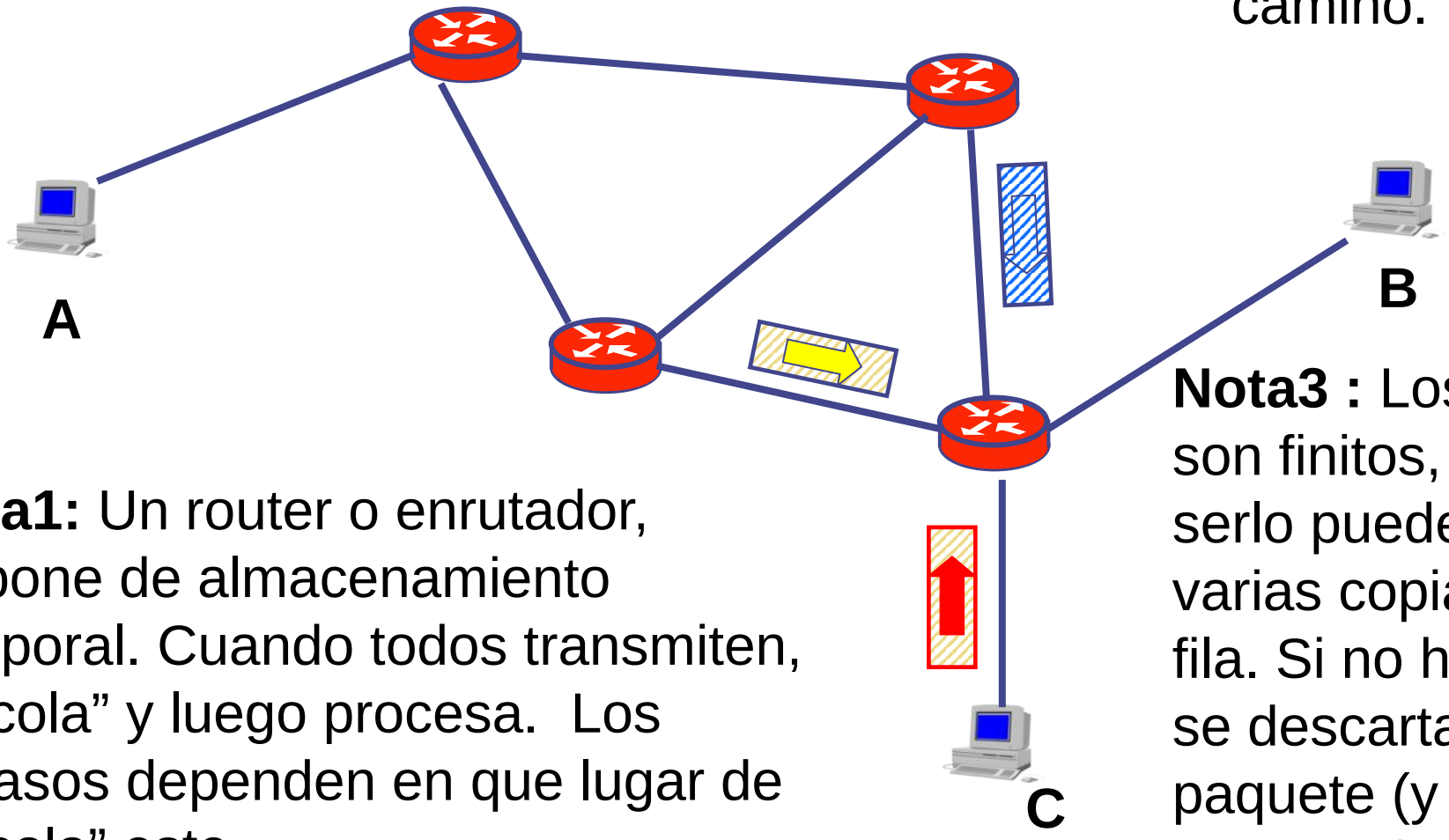
Nota3 : Los buffers son finitos, de no serlo pueden haber varias copias en la fila. Si no hay lugar, se descarta el paquete (y la TPDU que lleva)

Motivando Conceptos – Retardo Variable y Descartes

Recordar : Una TPDU (segmento) "viaja" en la carga útil de un paquete (NPDU).

Lo que suceda con los paquetes, afecta a los segmentos que lleva.

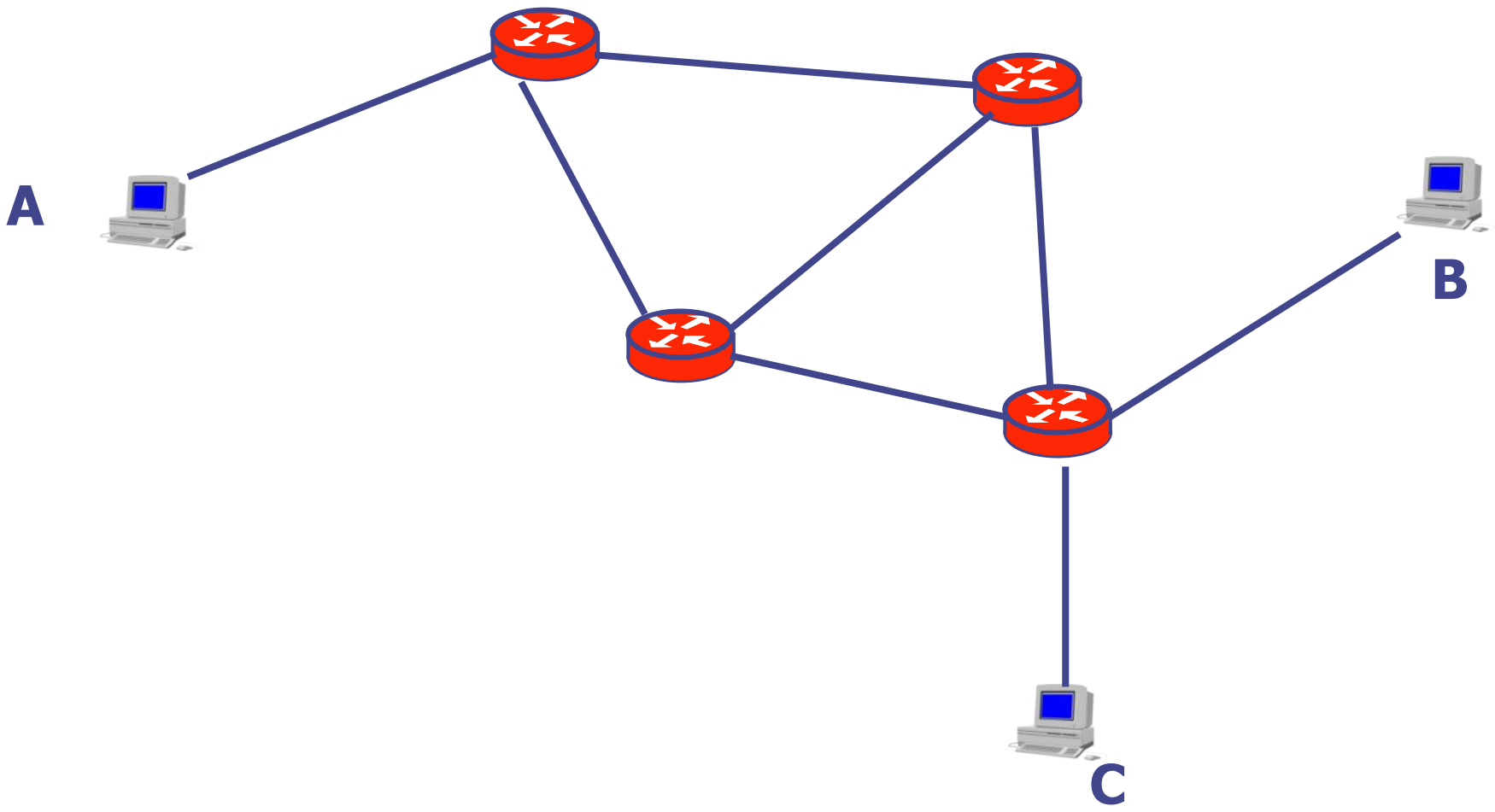
Nota2 : Tantas "colas" (buffers) como routers en el camino.



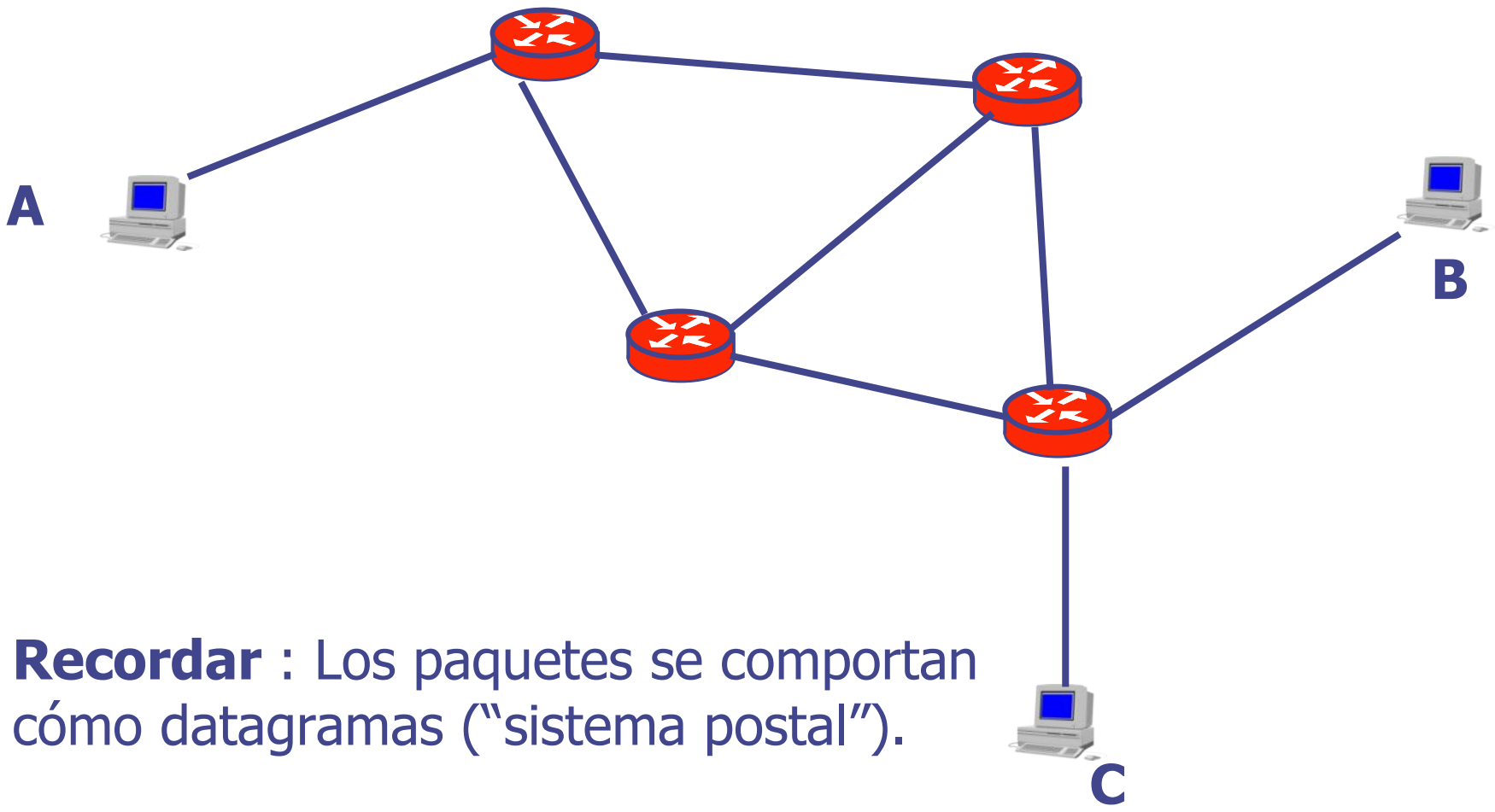
Nota1: Un router o enrutador, dispone de almacenamiento temporal. Cuando todos transmiten, "encola" y luego procesa. Los retrasos dependen en que lugar de la "cola" este.

Nota3 : Los buffers son finitos, de no serlo pueden haber varias copias en la fila. Si no hay lugar, se descarta el paquete (y la TPDU que lleva)

Motivando Conceptos – Duplicadas y Desordenadas



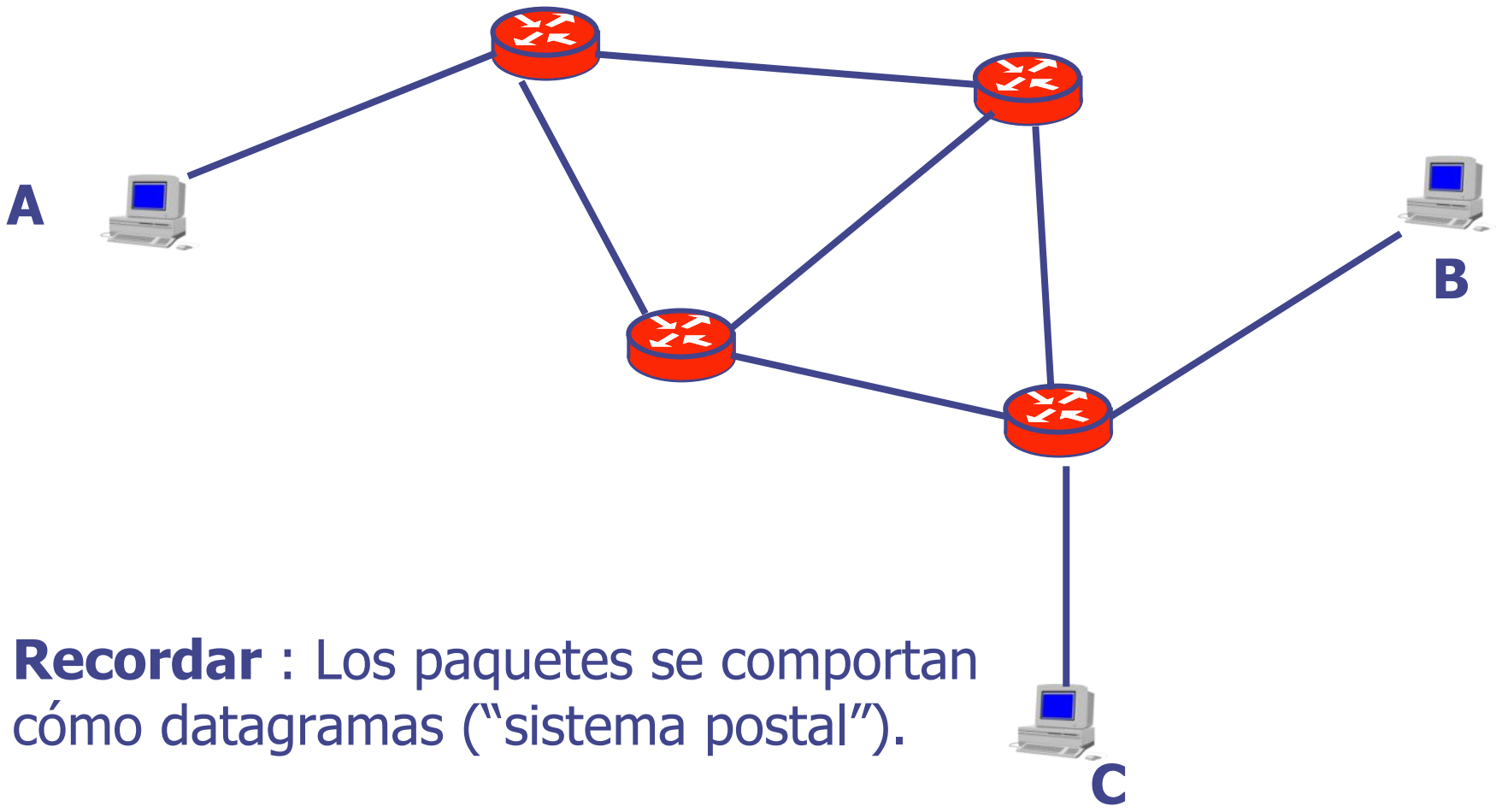
Motivando Conceptos – Duplicadas y Desordenadas



Recordar : Los paquetes se comportan cómo datagramas (“sistema postal”).

Motivando Conceptos – Duplicadas y Desordenadas

Duplicadas: Una TPDU y su reenvío pueden tomar diferentes caminos.



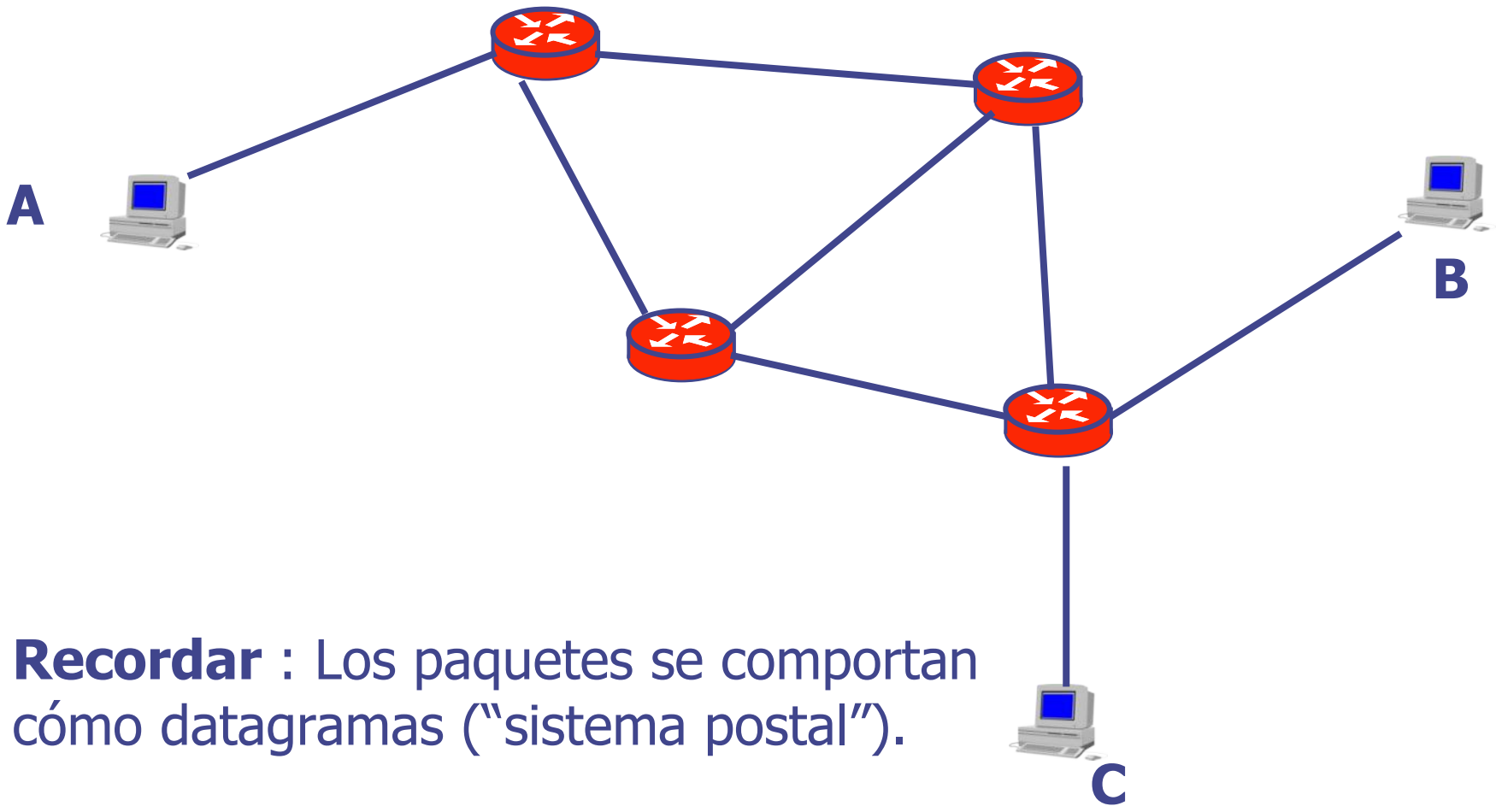
Recordar : Los paquetes se comportan cómo datagramas (“sistema postal”).

Motivando Conceptos – Duplicadas y Desordenadas

Recordar : Una TPDU "viaja" en la carga útil de un paquete (NPDU).

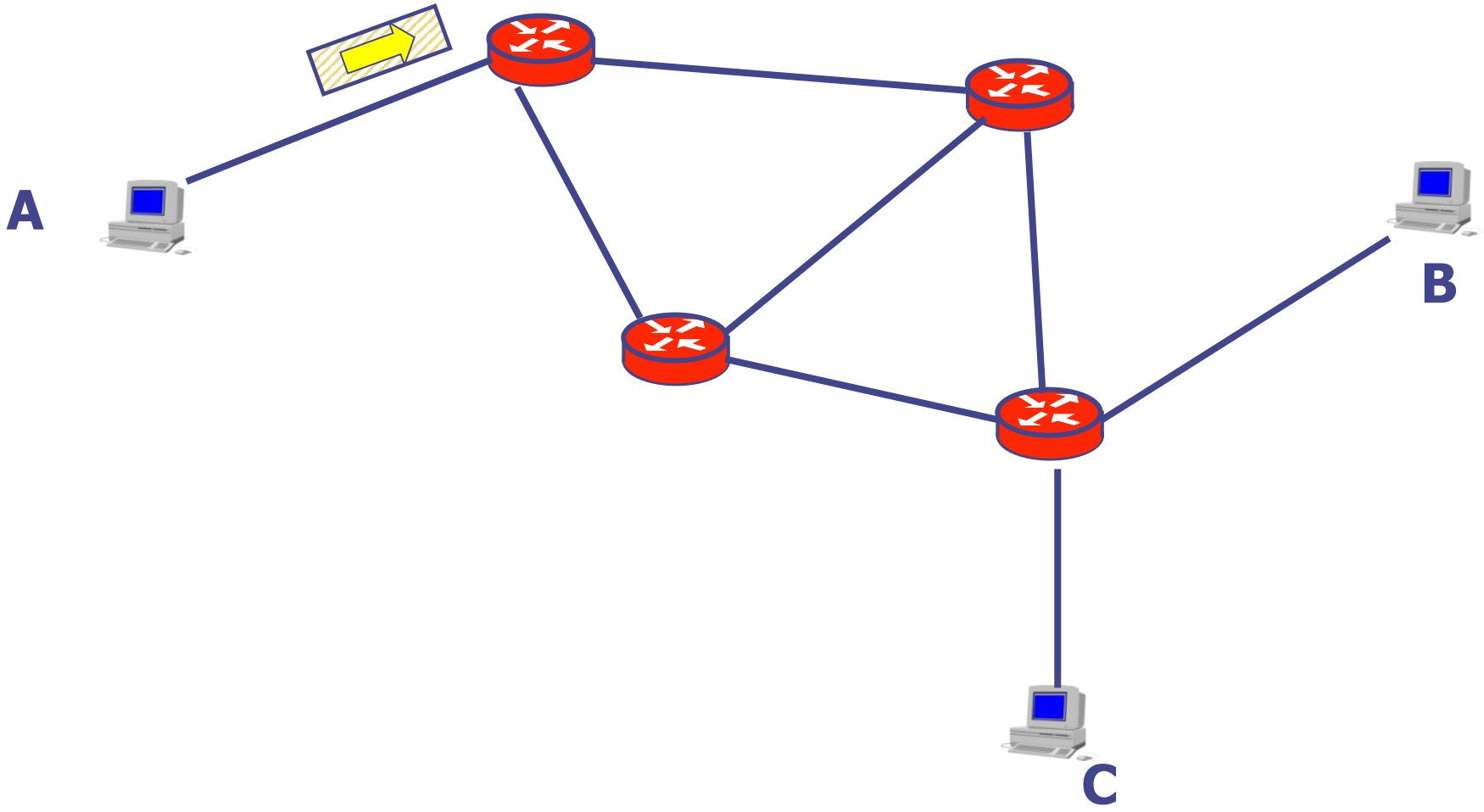
Duplicadas: Una TPDU y su reenvío pueden tomar diferentes caminos.

Lo que suceda con los paquetes, afecta a los segmentos que lleva.

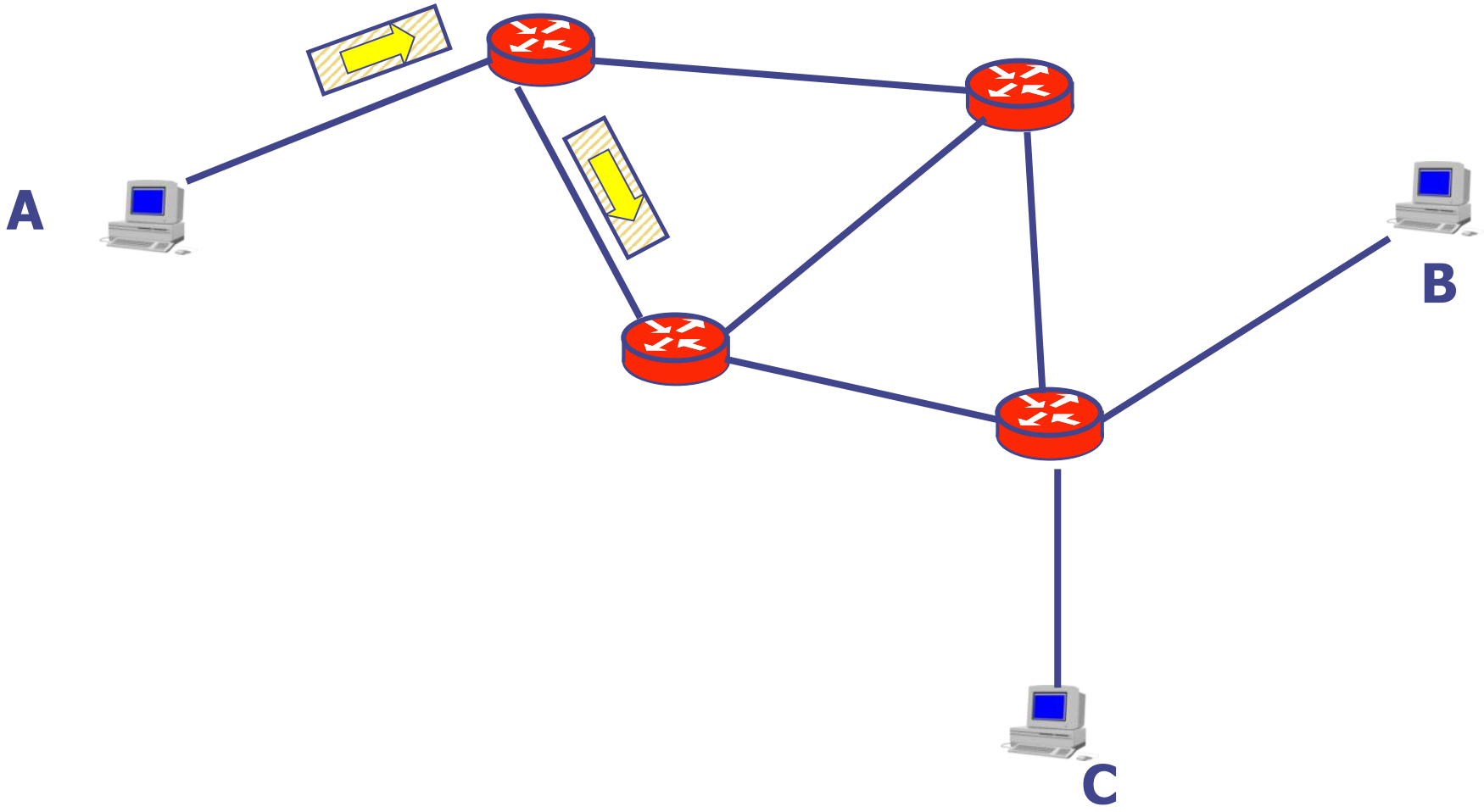


Recordar : Los paquetes se comportan cómo datagramas ("sistema postal").

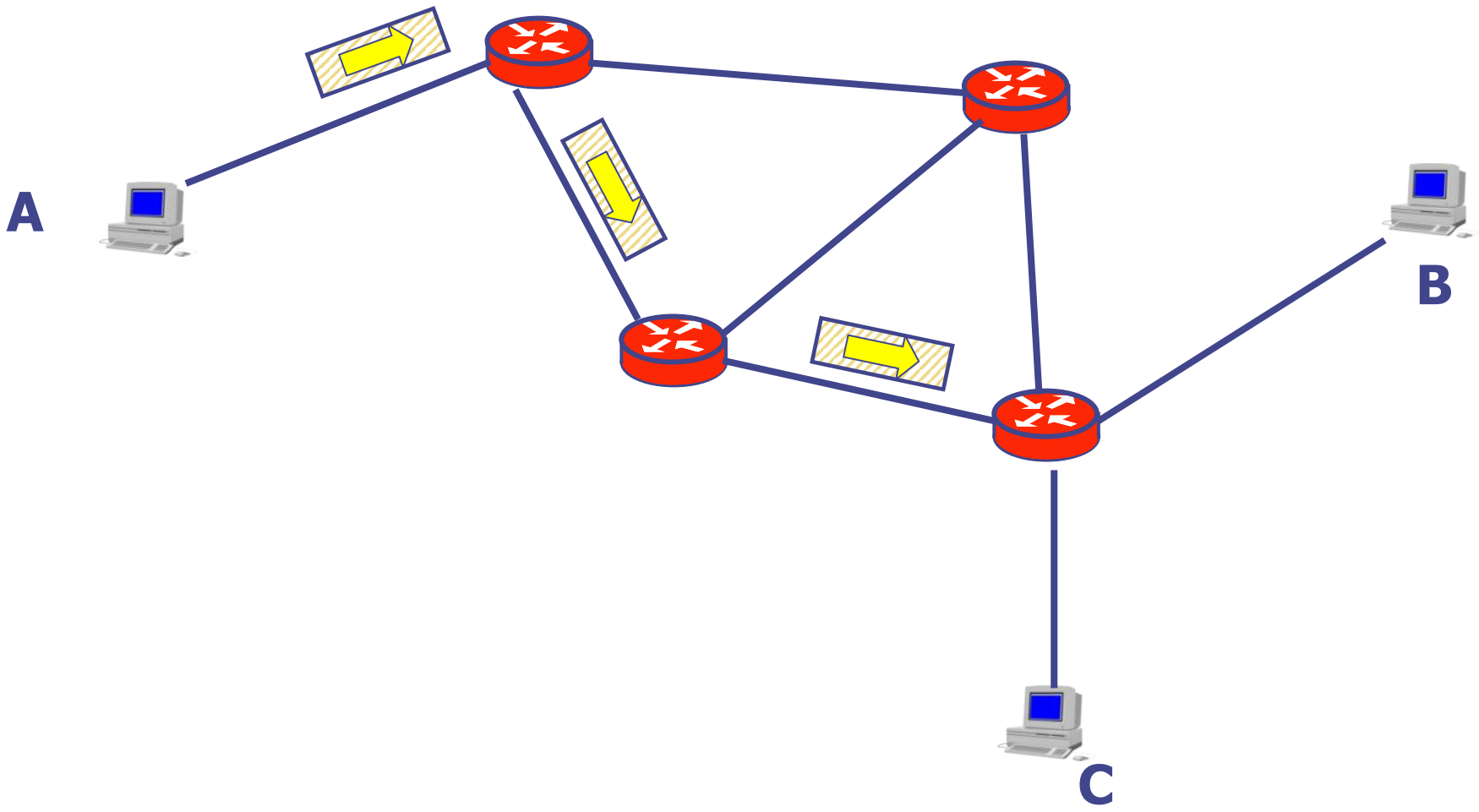
Motivando Conceptos – Duplicadas y Desordenadas



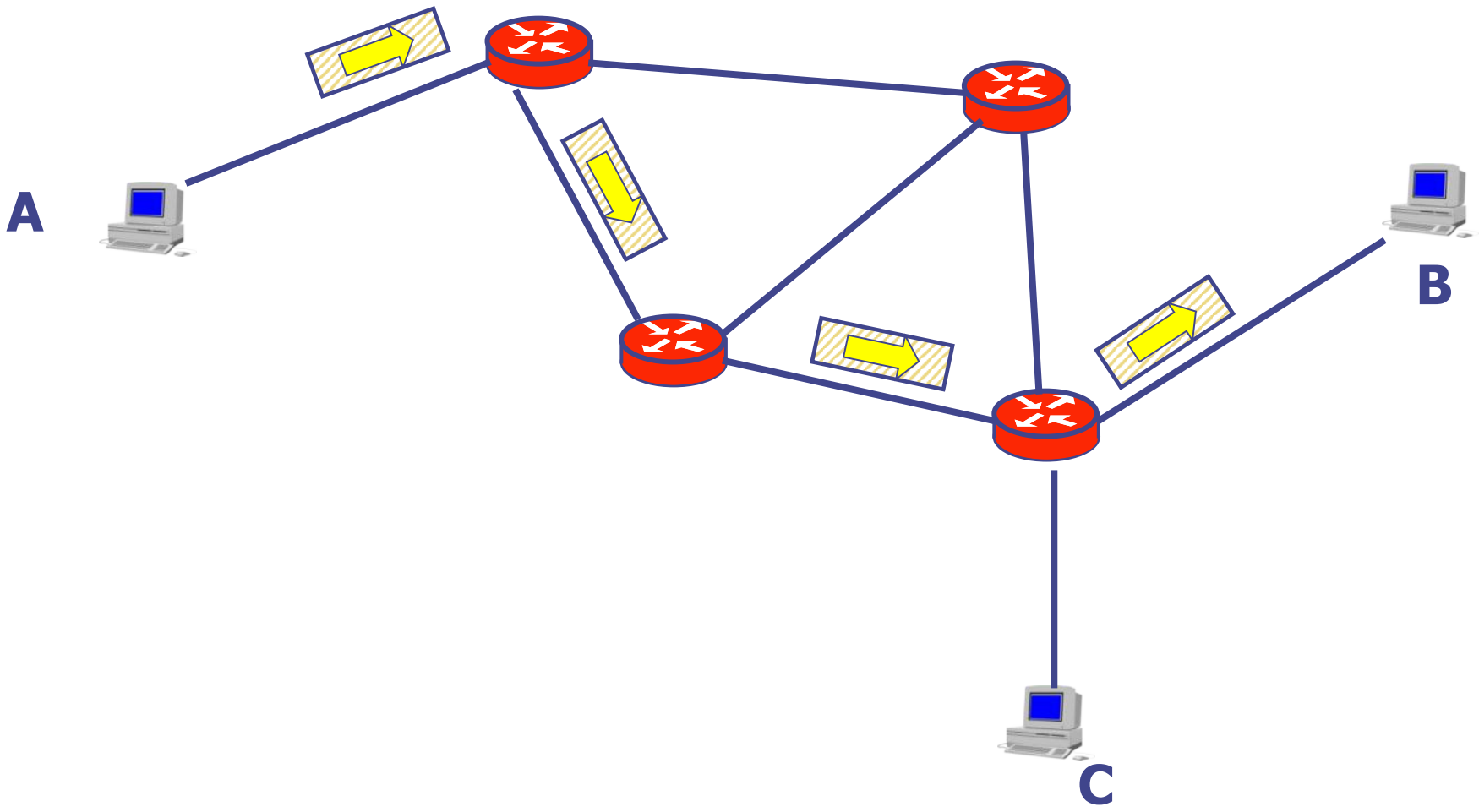
Motivando Conceptos – Duplicadas y Desordenadas



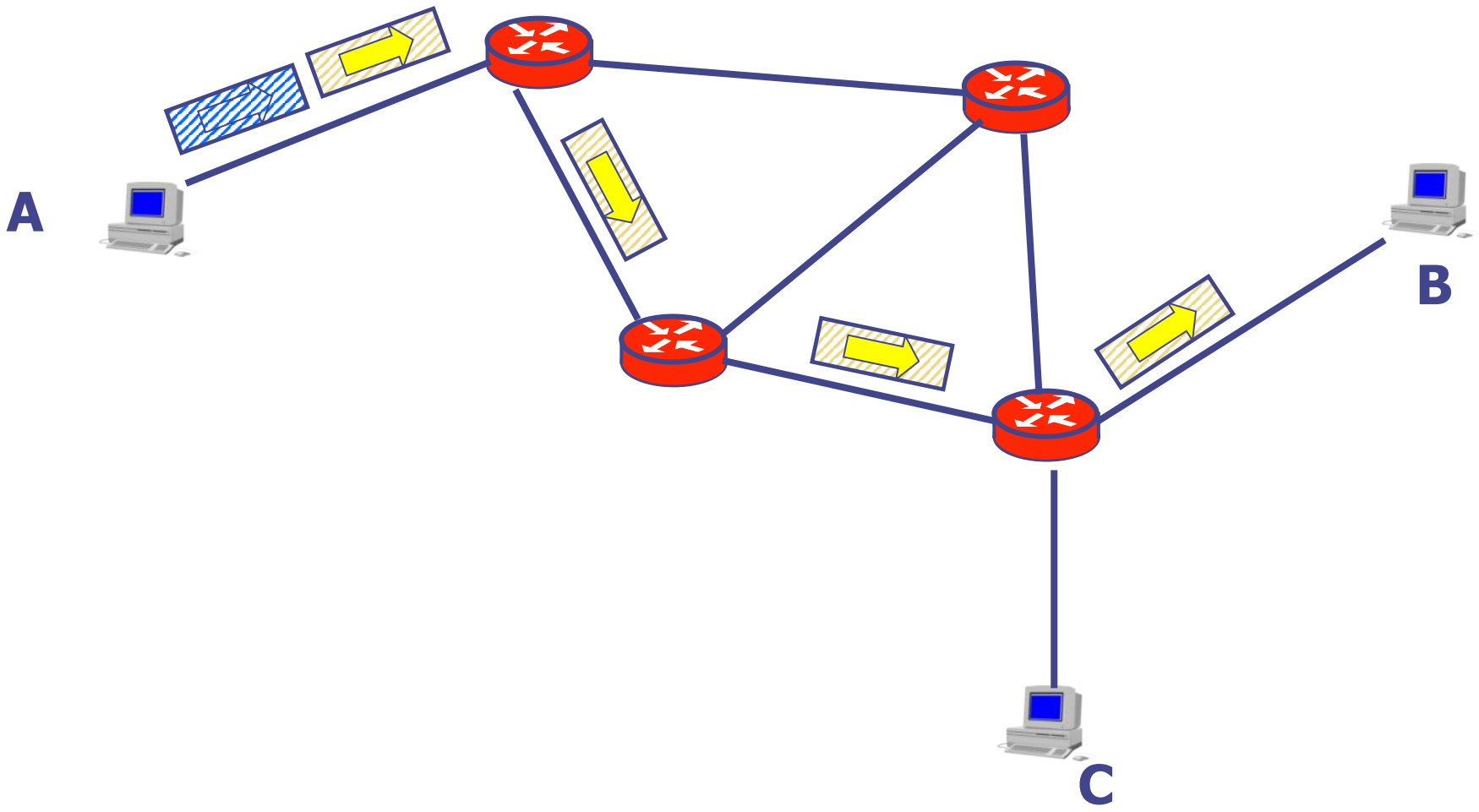
Motivando Conceptos – Duplicadas y Desordenadas



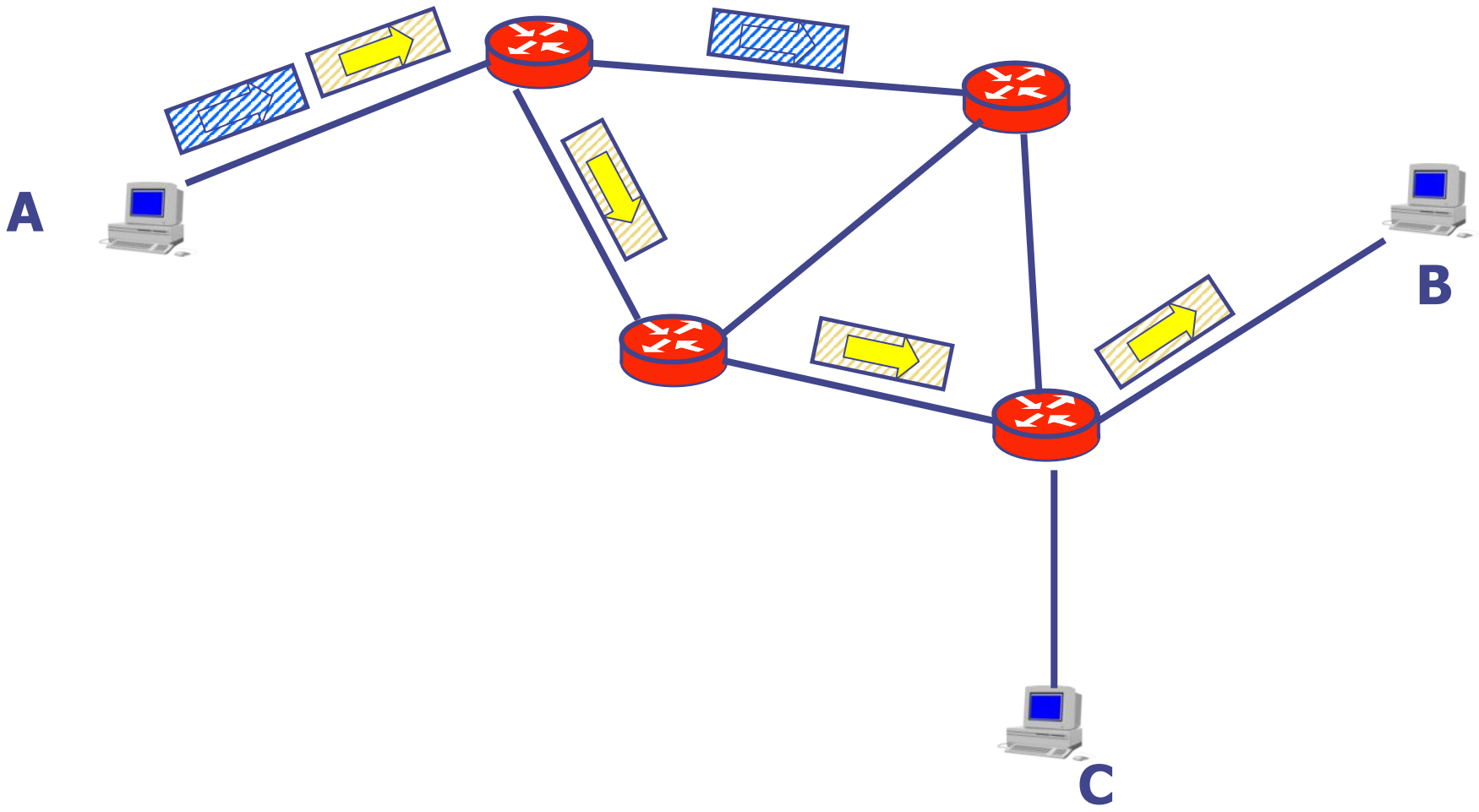
Motivando Conceptos – Duplicadas y Desordenadas



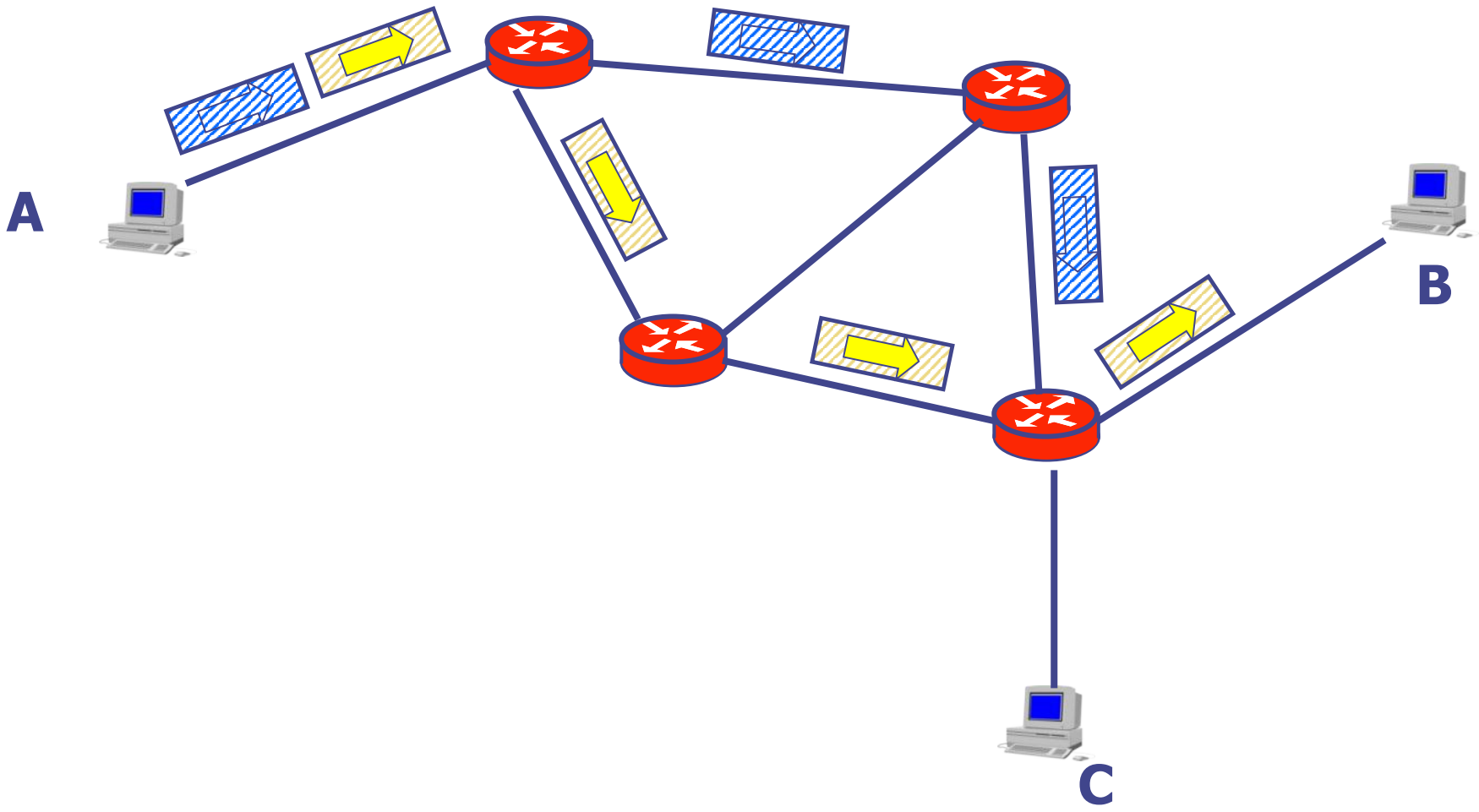
Motivando Conceptos – Duplicadas y Desordenadas



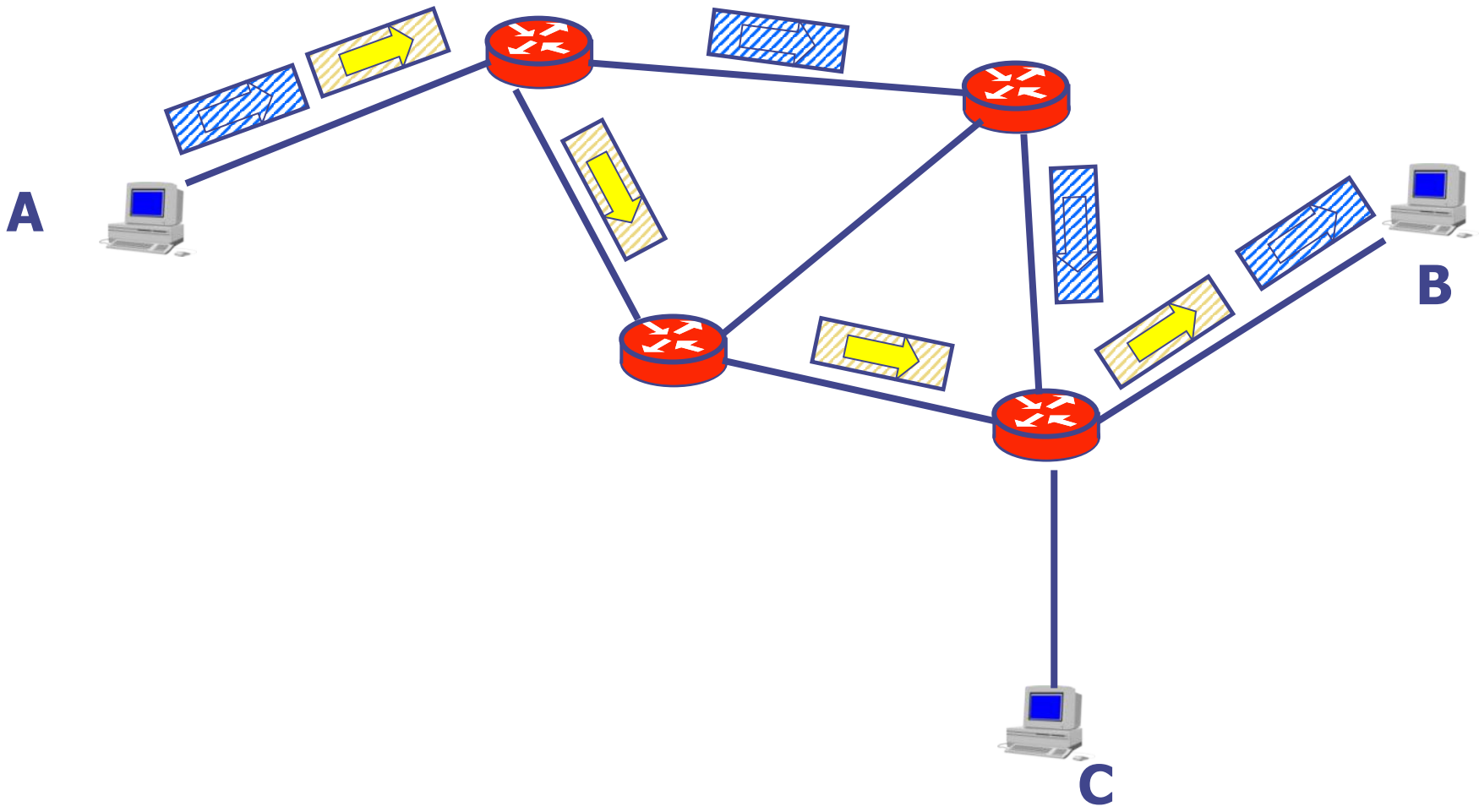
Motivando Conceptos – Duplicadas y Desordenadas



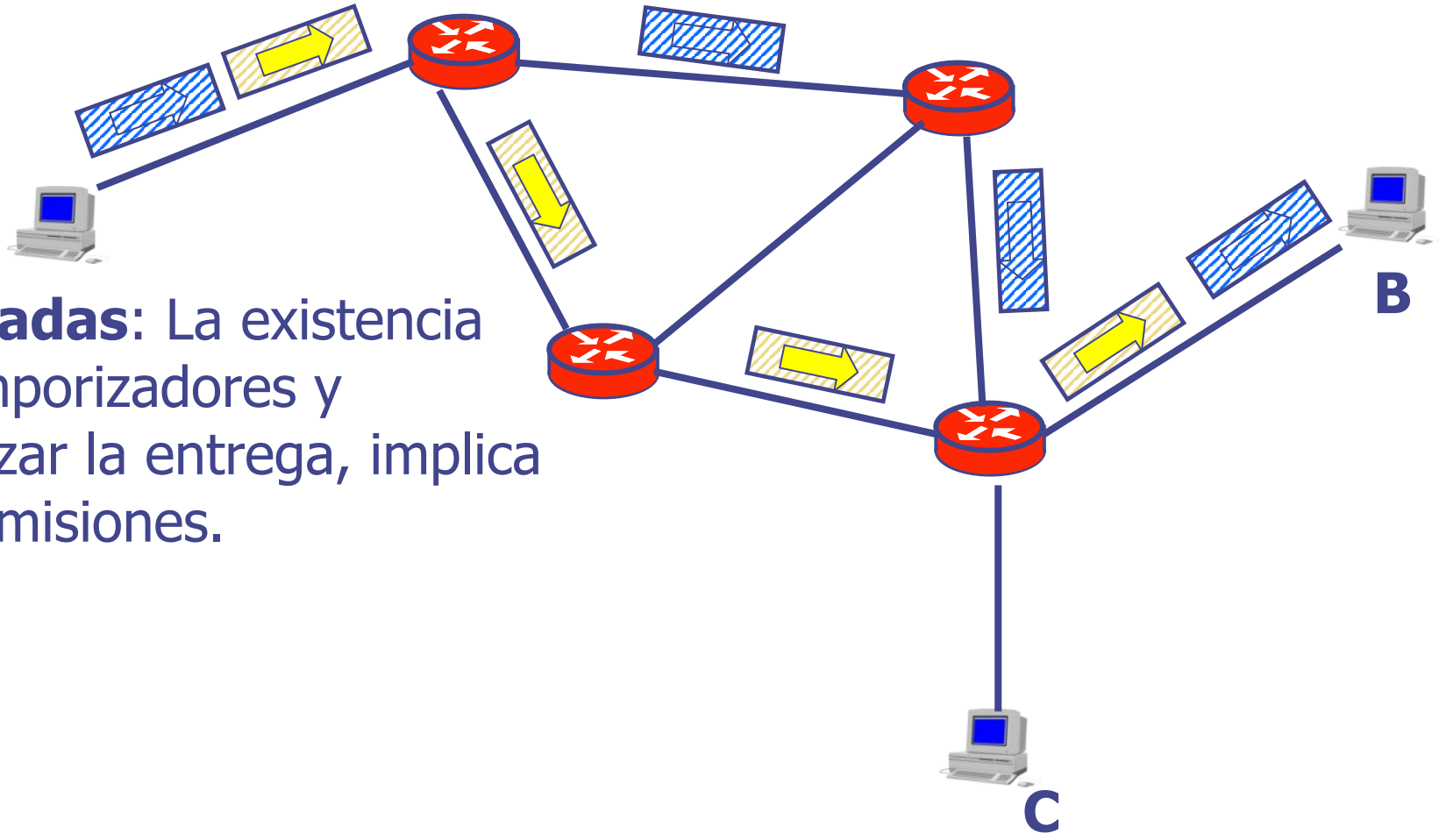
Motivando Conceptos – Duplicadas y Desordenadas



Motivando Conceptos – Duplicadas y Desordenadas

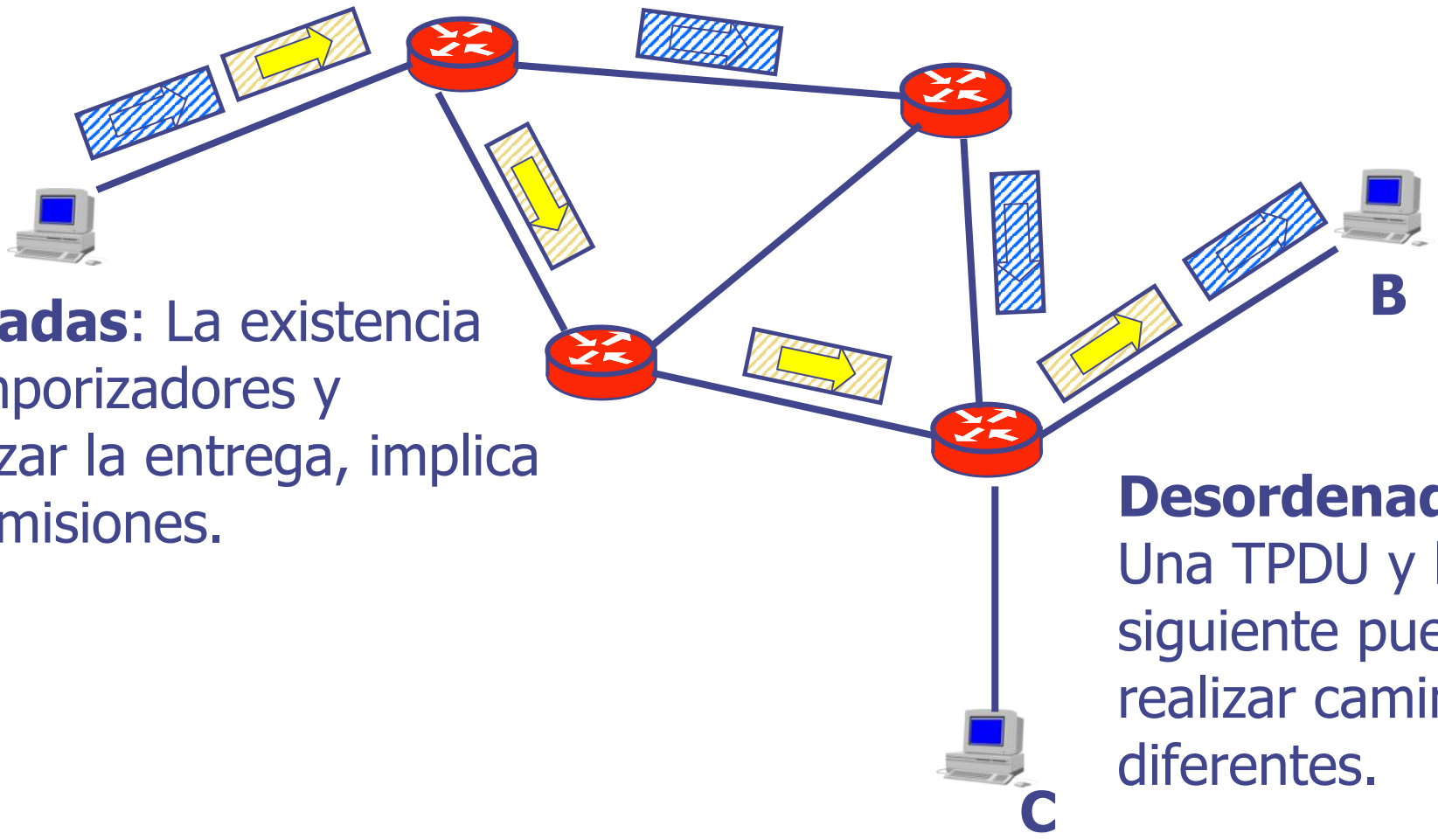


Motivando Conceptos – Duplicadas y Desordenadas



Duplicadas: La existencia de temporizadores y garantizar la entrega, implica retransmisiones.

Motivando Conceptos – Duplicadas y Desordenadas



Duplicadas: La existencia de temporizadores y garantizar la entrega, implica retransmisiones.

Desordenadas: Una TPDU y la siguiente pueden realizar caminos diferentes.

Ventanas Deslizantes y Números de Secuencia

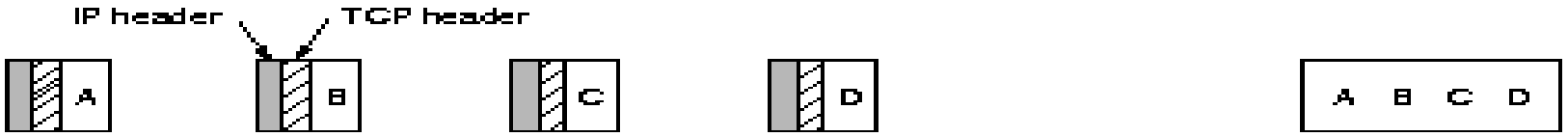
- Cada retransmisión por expiración del temporizador, genera un “potencial” duplicado.
- La subred podría tener duplicados que aparecen en el receptor cientos de milisegundos luego del arribo del paquete original, podrían tener un número de secuencia correspondiente a la nueva ventana
  - Si los números de secuencia se “reciclaron”
- Para poder solucionar este problema, se precisa tener una cota superior del tiempo de vida de un paquete (TPDU) en la red, y suficientes números de secuencia.
- Debe haber suficientes números de secuencia y utilizarse a una velocidad suficientemente baja como para que al reutilizar un número, todos los duplicados de TPDU anteriores (o sus ACK) con dicho número hayan desaparecido.
- **Se necesita muchos más números que si la red no pudiera re-ordenar los paquetes**

TCP – Connection Oriented Protocol

- **Orientado a Conexión:** previo al intercambio de datos, las entenidades de transporte deben acordar algunos parámetros. Estos se negocian en los primeros segmentos.
- **Objetivo:** Flujo confiable de bytes sobre una red no confiable
  - Conexión punto a punto
  - Conexión bi-direccional (la misma conexión permite enviar datos de A hacia B y de B hacia A)
  - Debe funcionar sobre IP (no da garantías)
  - Diferentes tecnologías de red en el medio
  - Robusto frente a problemas de la red
- Recibe flujo de la capa superior y lo divide en bloques que envía en segmentos independientes (uno en cada paquete IP)
- El receptor los reensambla
- **Los números de secuencia no son por segmento sino que identifican los bytes del flujo de datos de aplicación**
- **Se numeran los bytes del segmento y no los segmentos**

TCP – Connection Oriented Protocol

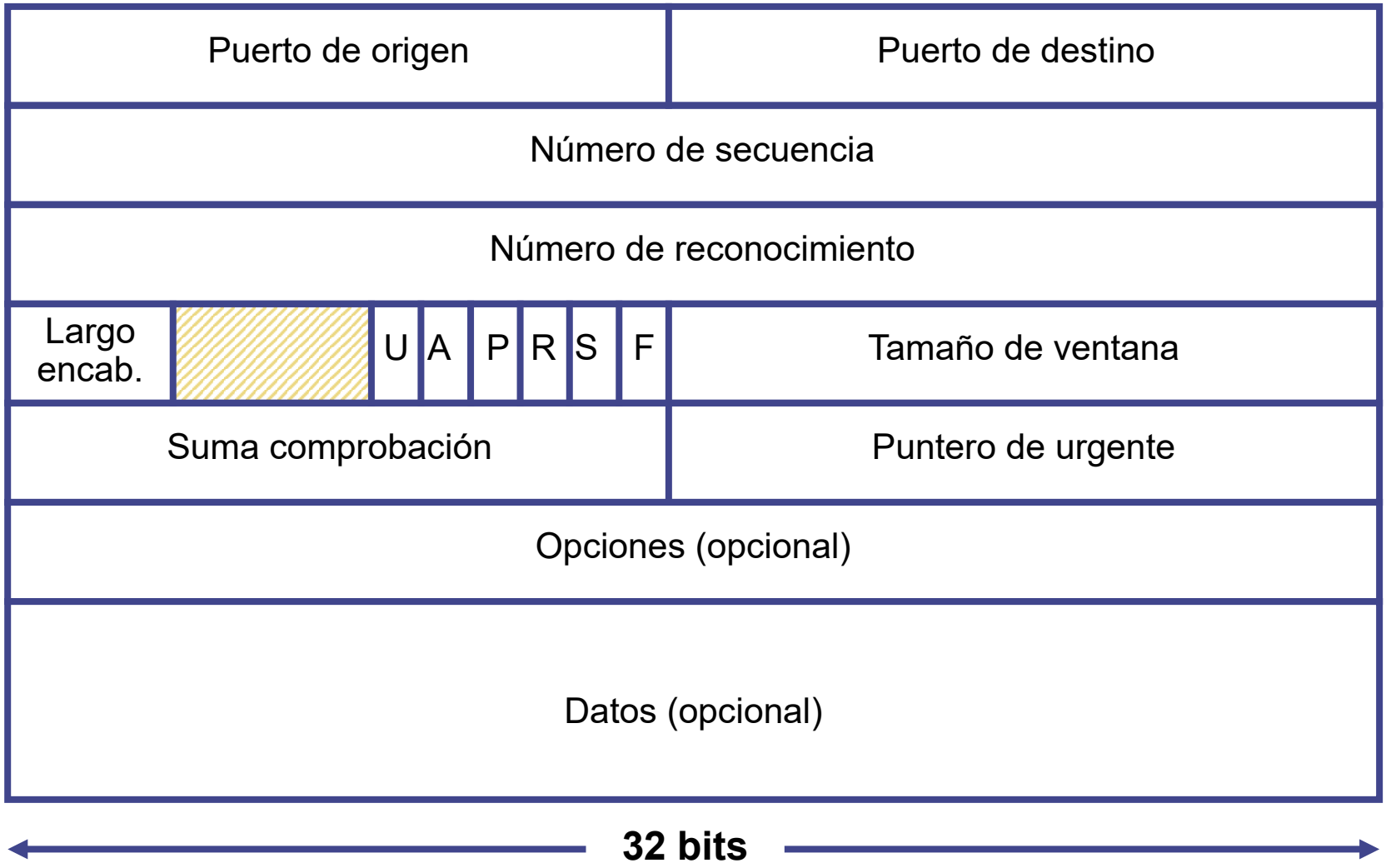
- TCP no mantiene las fronteras entre los bloques recibidos de la capa de aplicación, ni los recibidos de la red
- Ejemplo: Un mensaje (ABCD) es enviado en 4 paquetes de capa 3 separados, pero devuelto a la aplicación destino como un conjunto de bytes ABCD



Protocolo TCP

- Unidad de datos (TPDU) = Segmento
- Número de secuencia de 32 bits
- Se numeran los bytes, no los segmentos
- Reconocimientos acumulativos
  - ACK = **x**, reconoce los bytes **x-1**, **x-2**, **x- 3**, …..
  - El siguiente byte que espera es **x**
- Encabezado de 20 bytes (+ opciones)
- Tamaño máximo del segmento
  - carga del paquete IP: máximo 64 Kbytes
  - TCP trata de evitar fragmentación en capas inferiores. Por ello se autolimita a la MTU (maximum transfer unit) de la red, típico 1500 bytes.
- Utiliza protocolo de ventanas deslizantes de tamaño de ventana variable

Formato del Segmento TCP

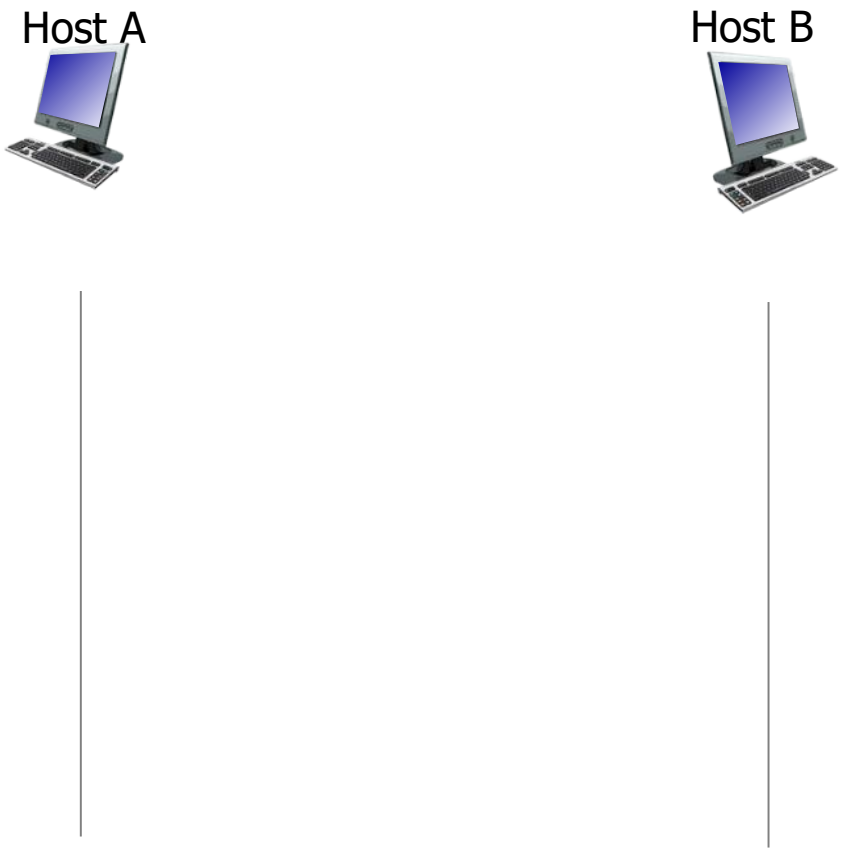


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

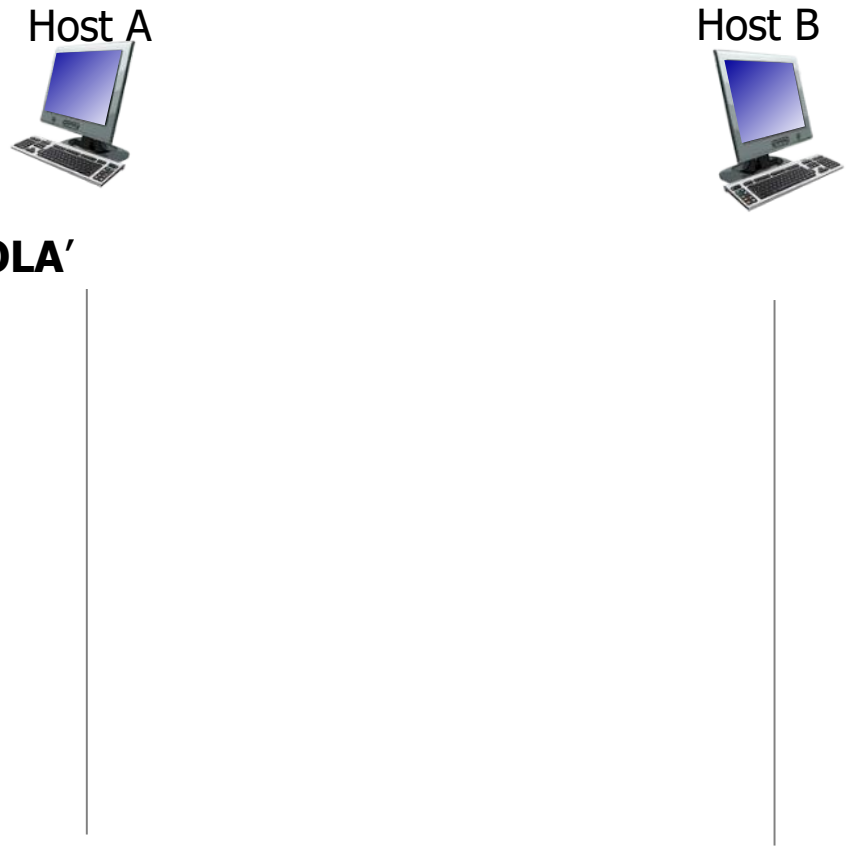


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

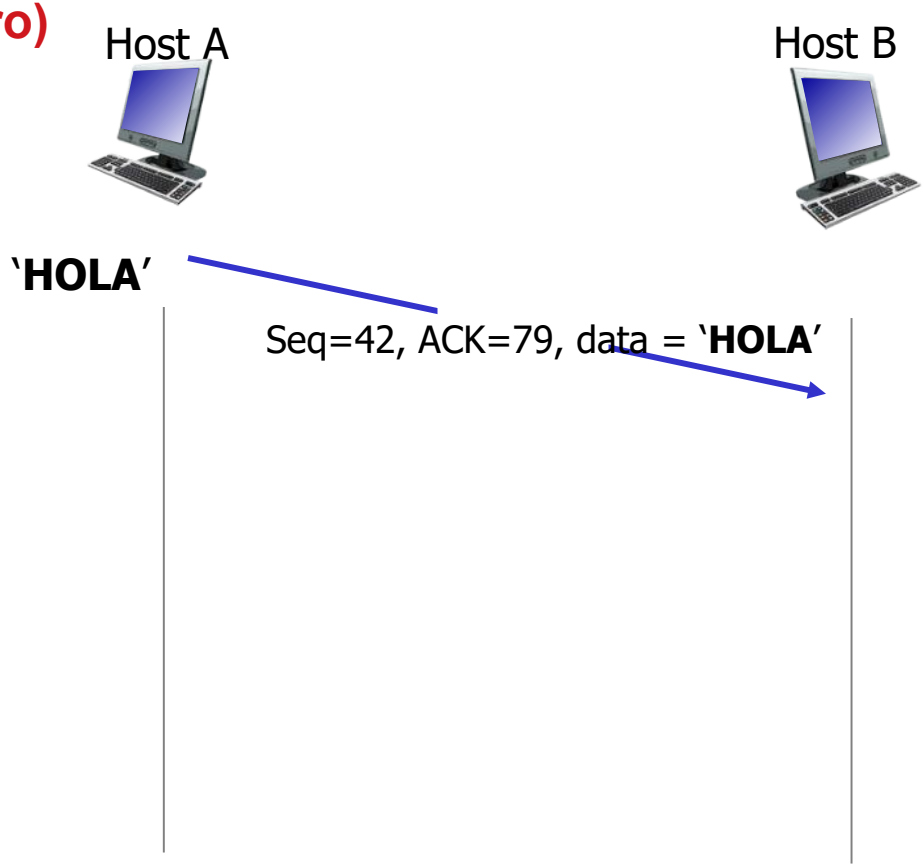


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

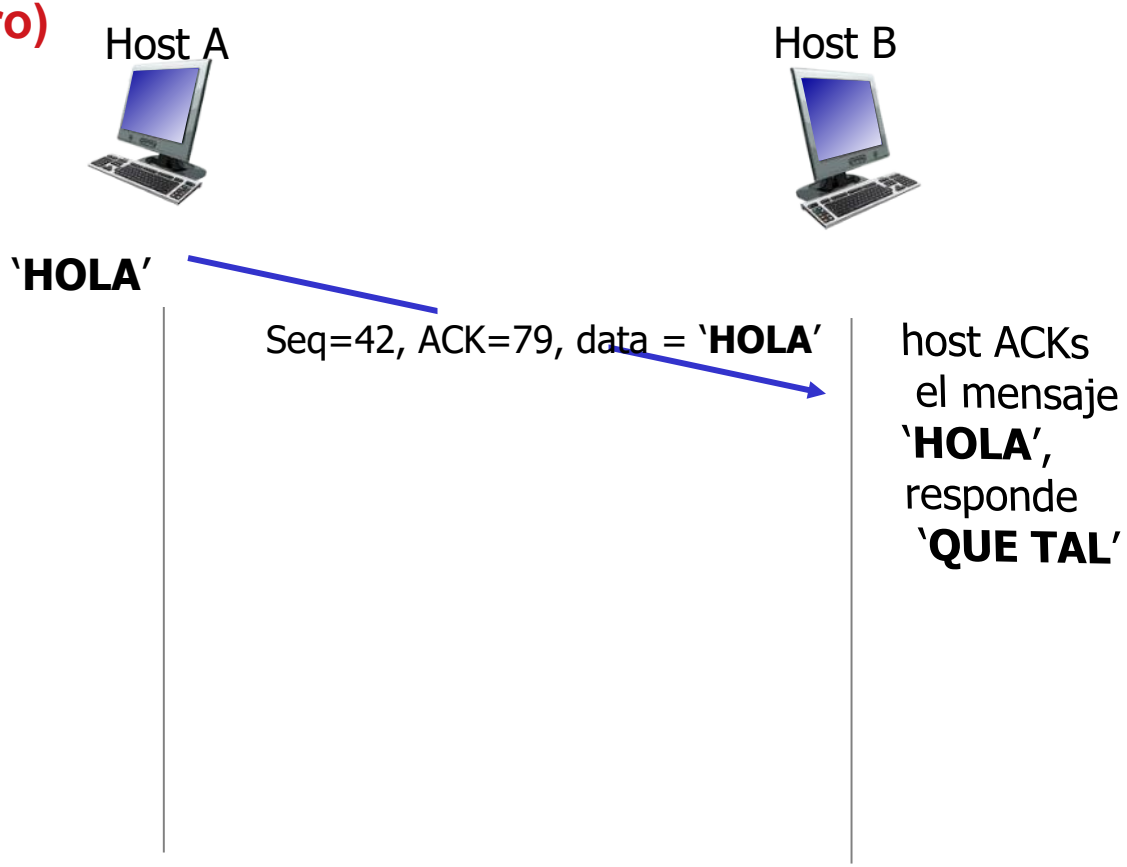


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

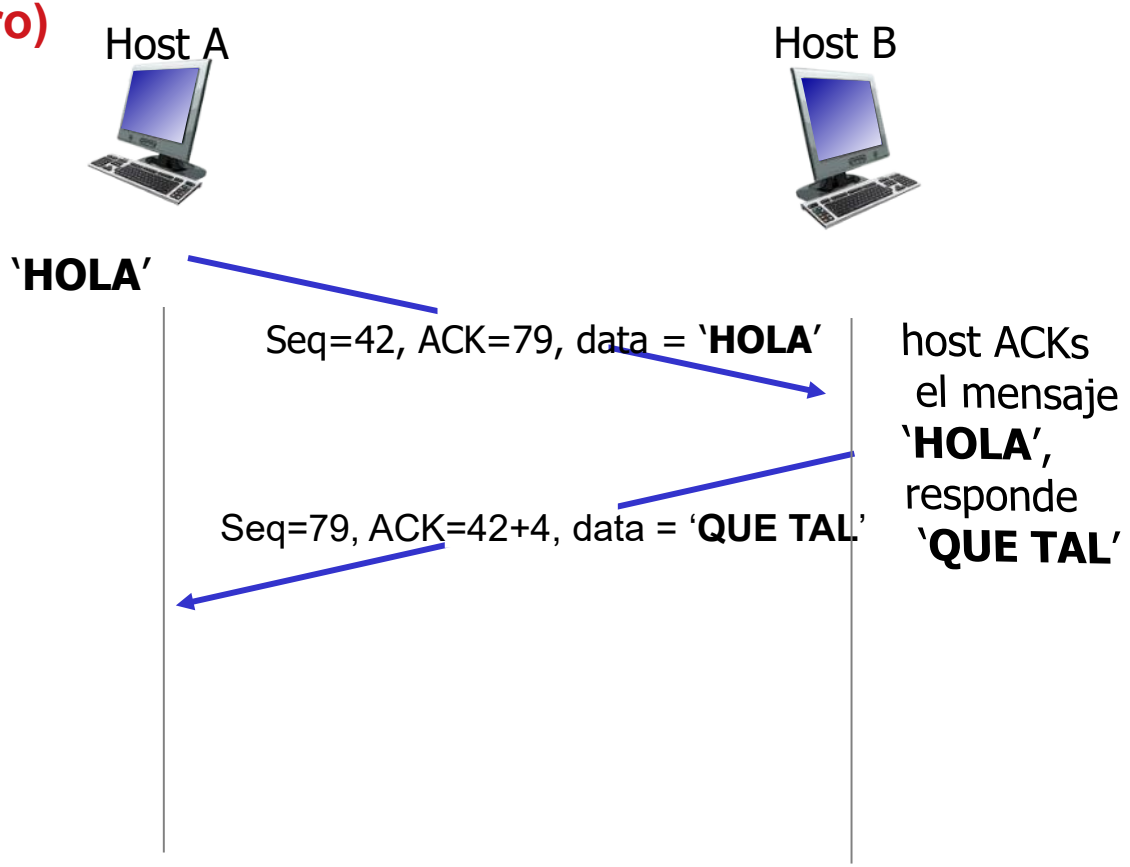


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

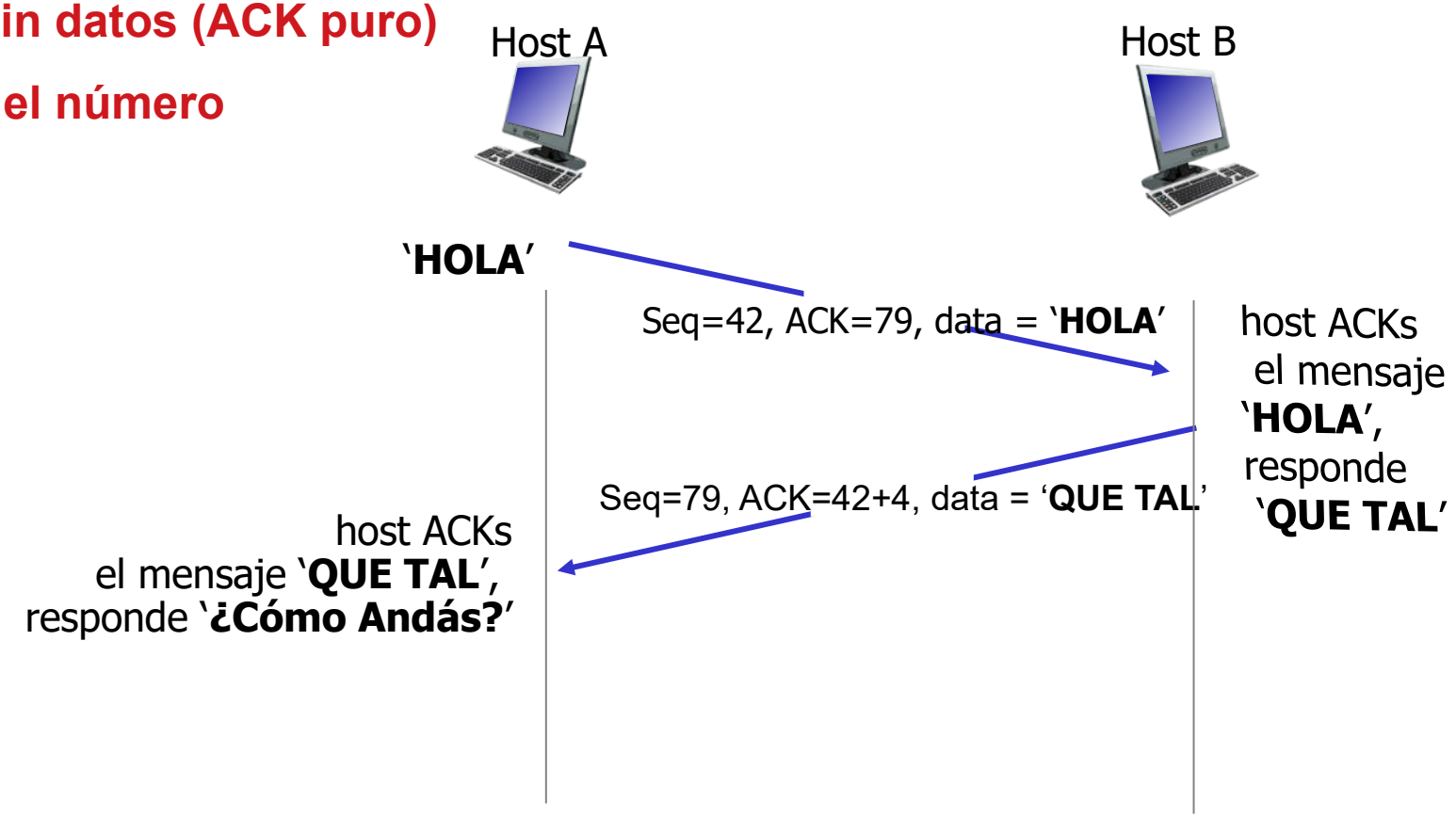


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

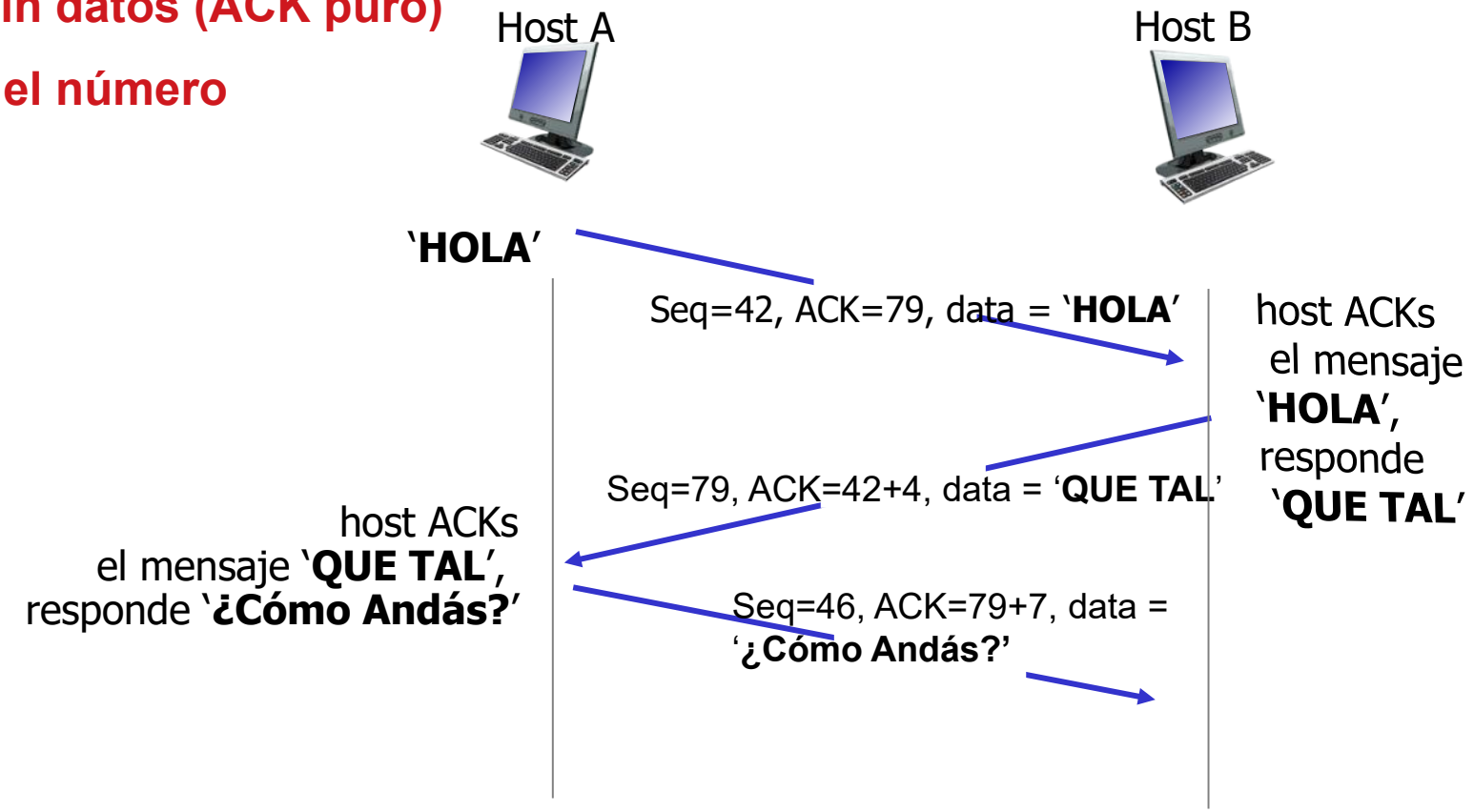


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

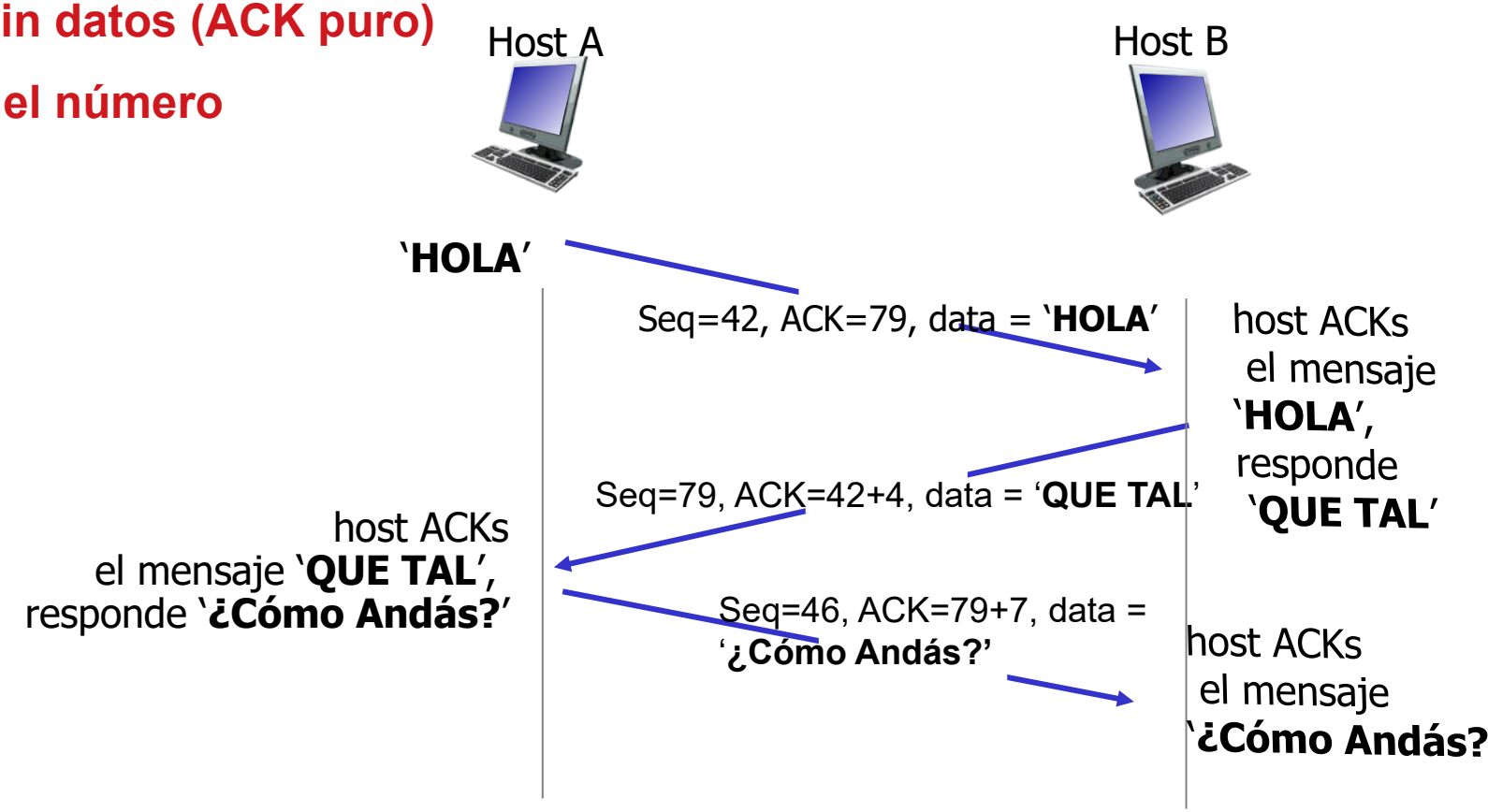


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia

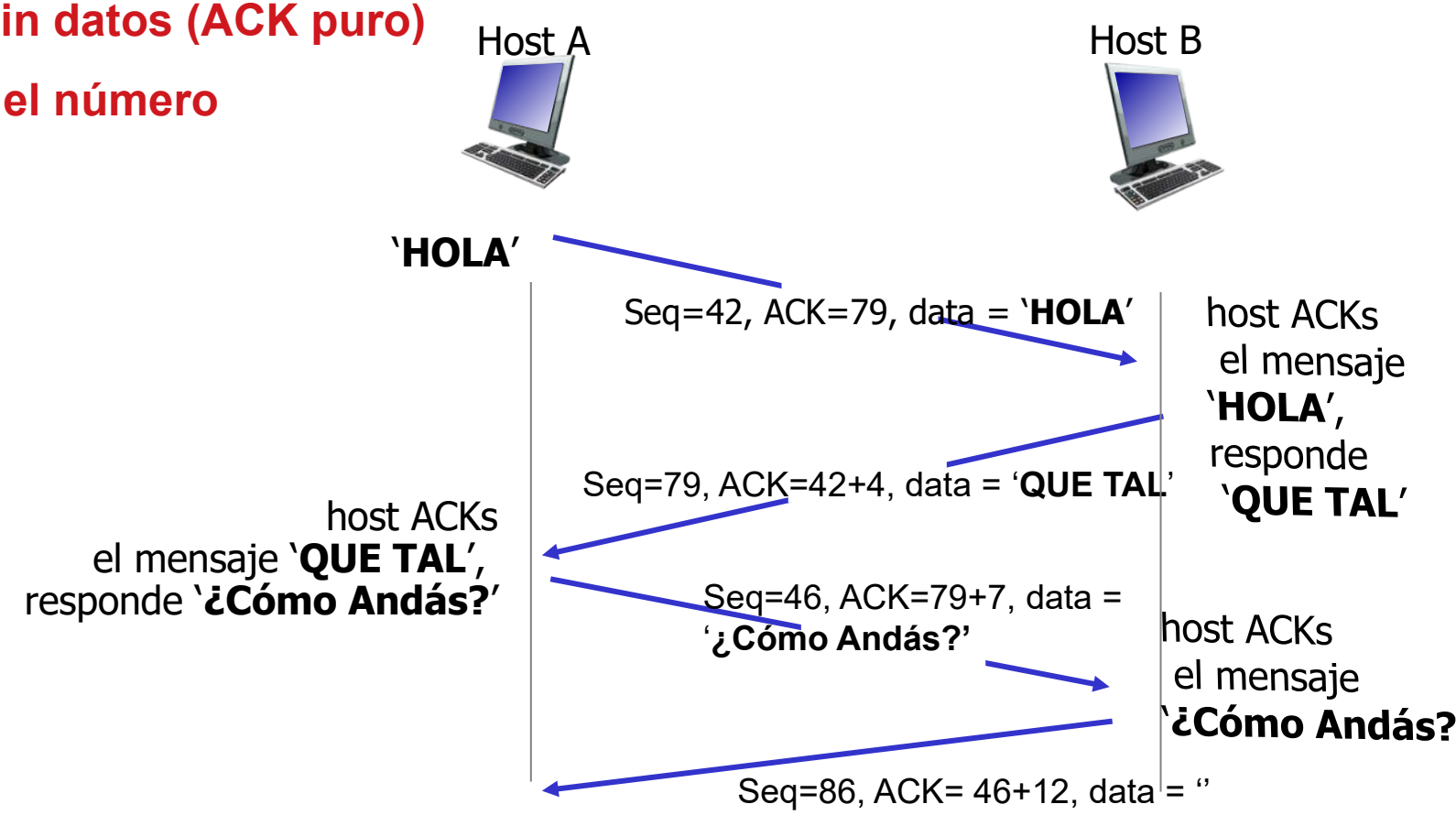


TCP – Reconocimientos y Números de Secuencia

- **Reconocimientos:**
  - Campo específico notificando que el segmento lleva un reconocimiento (flag ACK).
  - Número de reconocimiento (recordar acumulativo)
- **Número se secuencia**:
  - El número de secuencia del segmento corresponde al primer byte de datos de capa de aplicación.
  - **Un segmento sin datos (ACK puro)**

    No incrementa el número

    de secuencia



TCP - Banderas

- U - hay datos urgentes
- A - Campo de reconocimiento válido
- P - Push (se pide celeridad para enviar los datos a la capa de aplicación)
- R - Reset (cierre abrupto de conexión)
- S - Syn (sincronización inicial de números de secuencia. Establecimiento)
- F - Fin (solicitud de fin de conexión)

TCP – Opciones

- El campo de Opciones permite intercambiar datos no obligatorios
- Se han ido agregando nuevas opciones
  - Maximum Segment Size (MSS)
  - Escala de la ventana (WSCALE)
  - Asentimiento selectivo (SACK)
  - Timestamp
  - Asentimiento negativo (NAK)
  - Otras

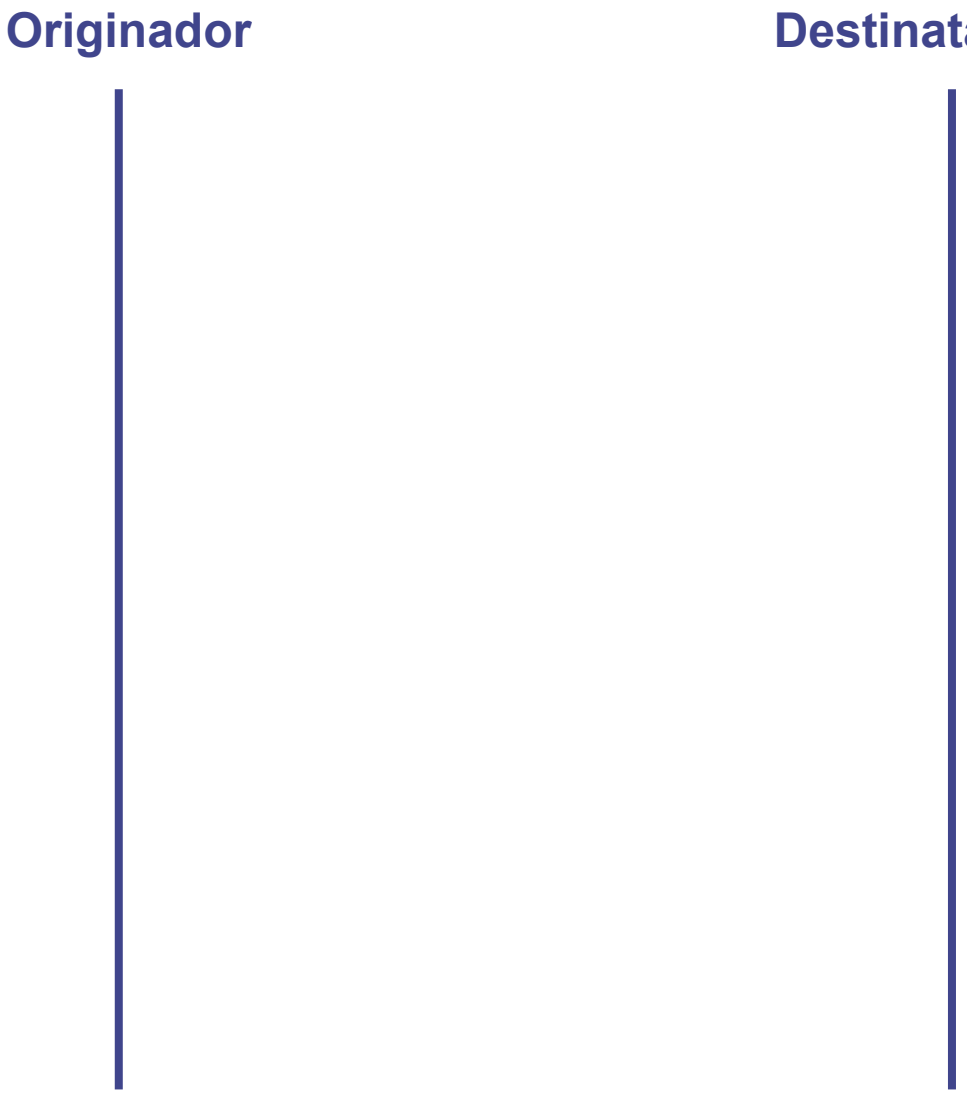
TCP – Opciones

- **MSS:** observamos el tamaño máximo de paquete que podemos mandar (dado por las capas inferiores), y descontamos los encabezados. Se envía en primer segmento
  - Por ejemplo, en ethernet el máximo es 1500 bytes, por lo que MSS es 1460
- NACK: avisar que no se recibió un determinado segmento (no se usa)
- Otras: veremos luego

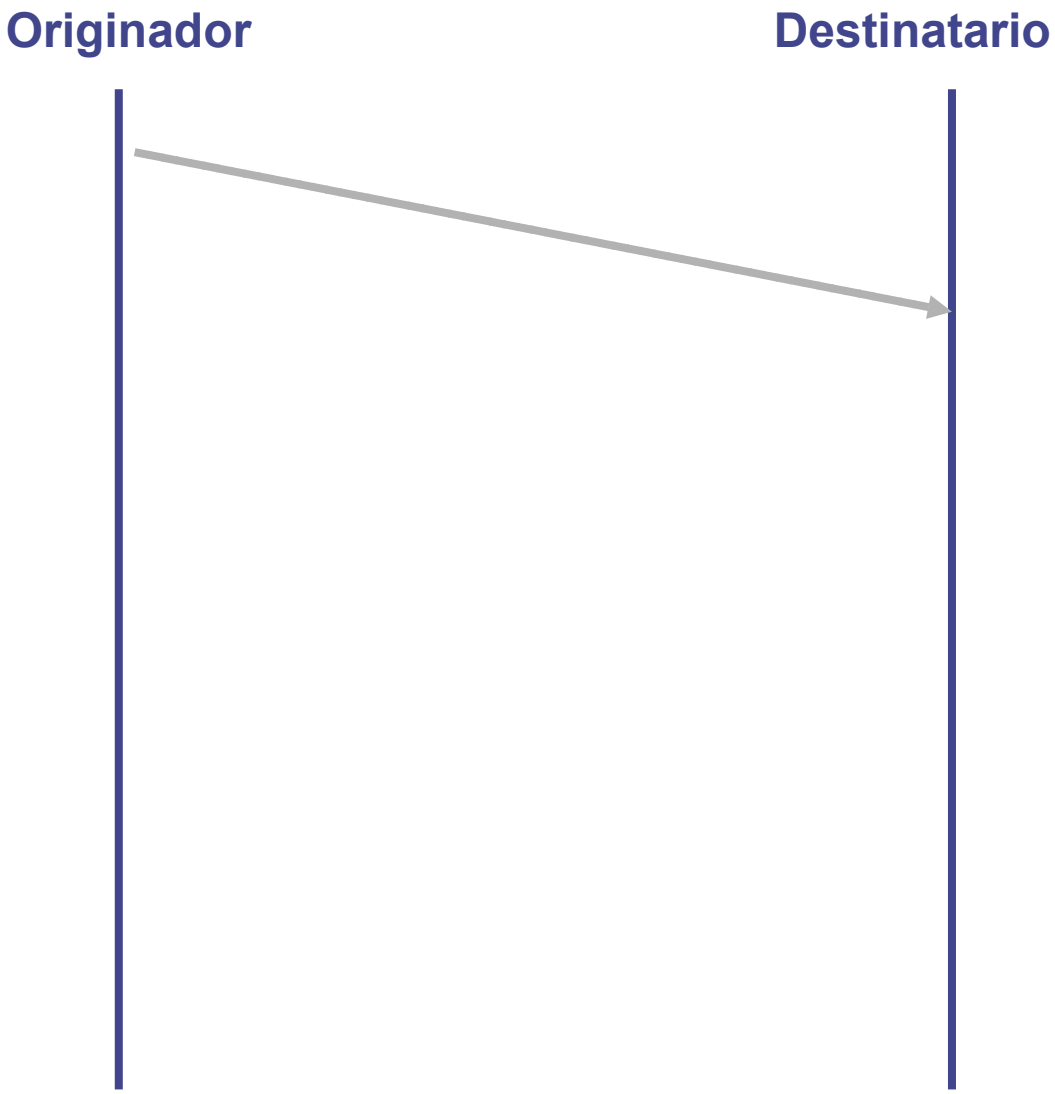
TCP

- Inicio y fin de conexión
- Manejo de números de secuencia
- Control de Flujo
- Control de Congestión
- Estados y temporizadores

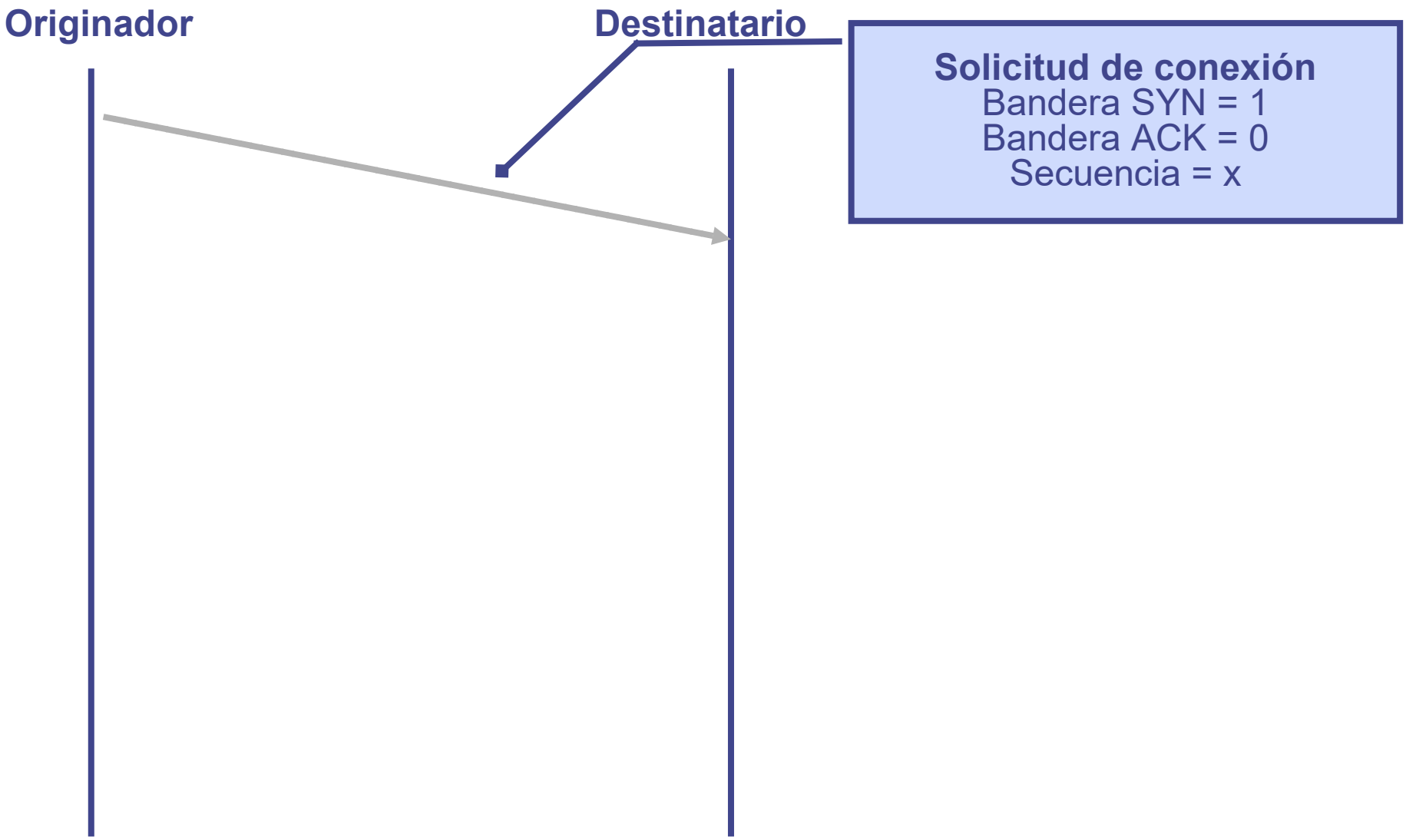
TCP – Inicio de Conexión en 3 vías



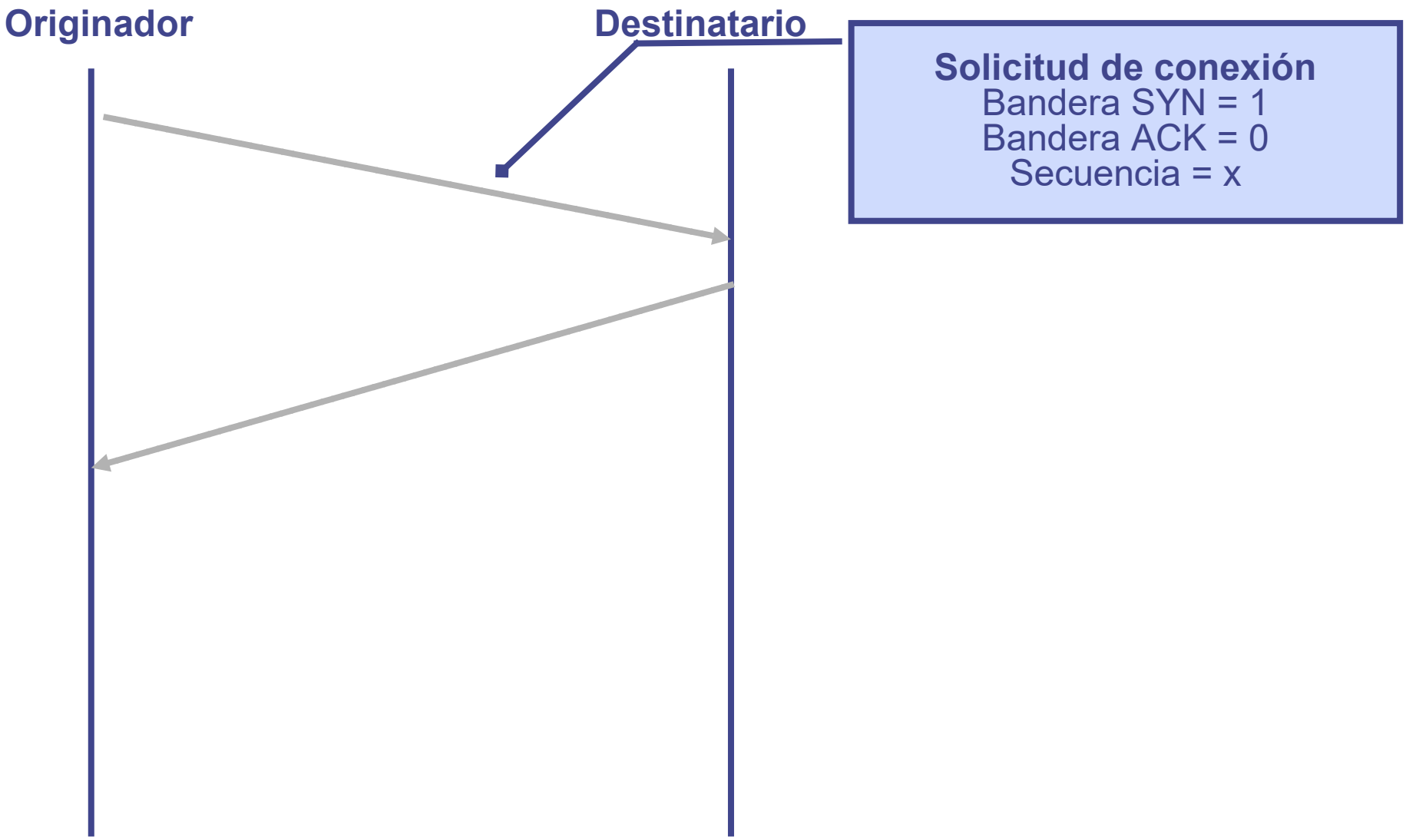
TCP – Inicio de Conexión en 3 vías



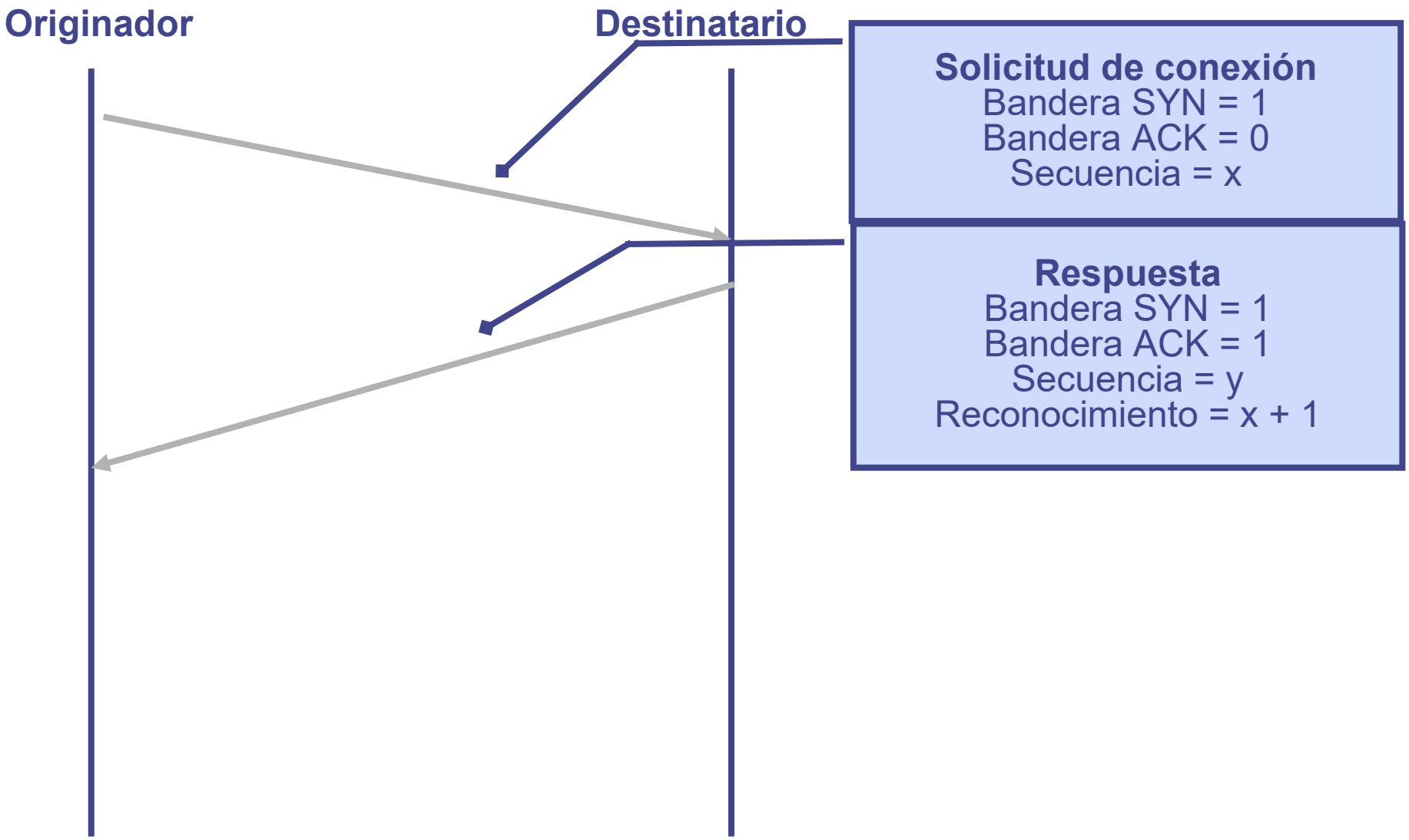
TCP – Inicio de Conexión en 3 vías



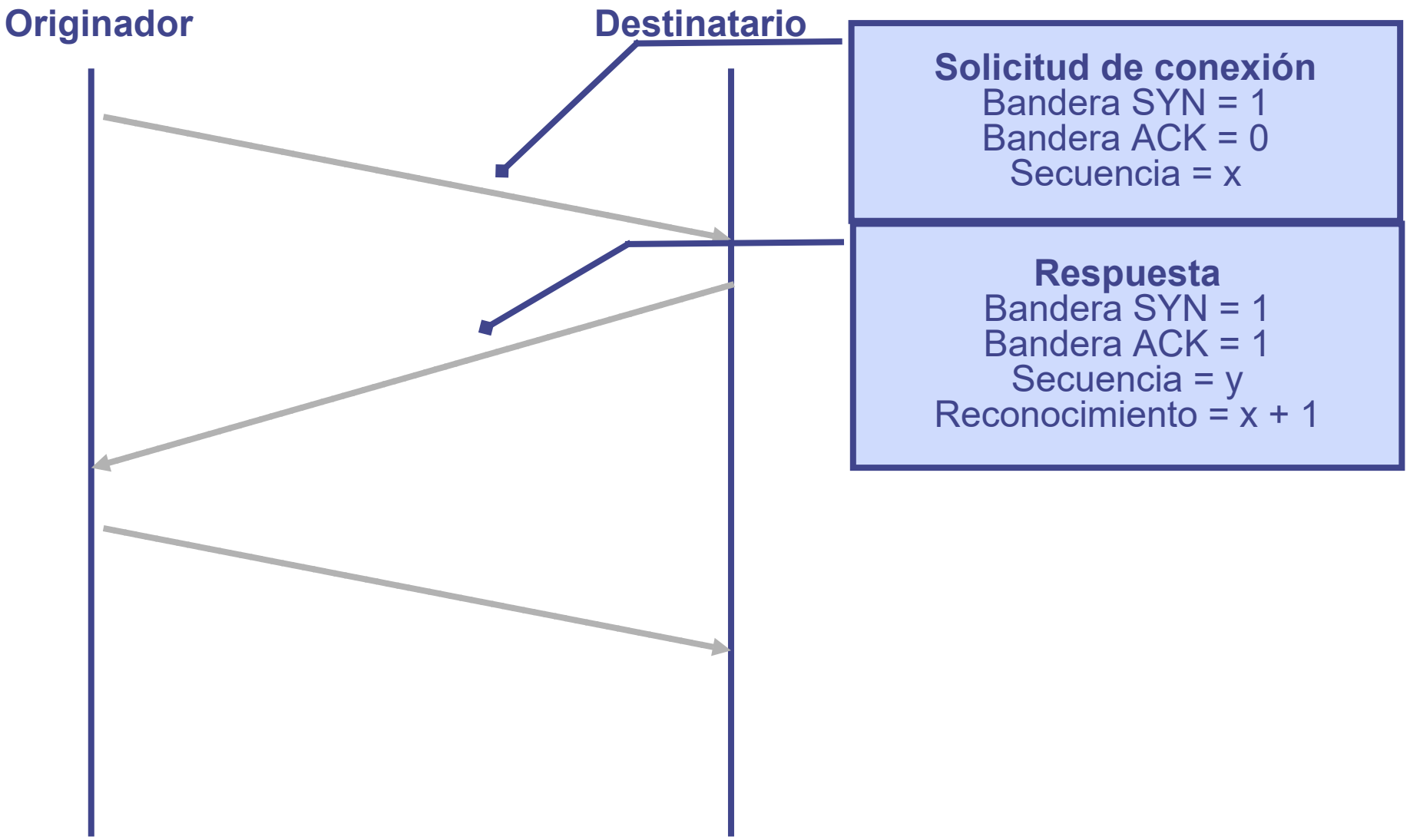
TCP – Inicio de Conexión en 3 vías



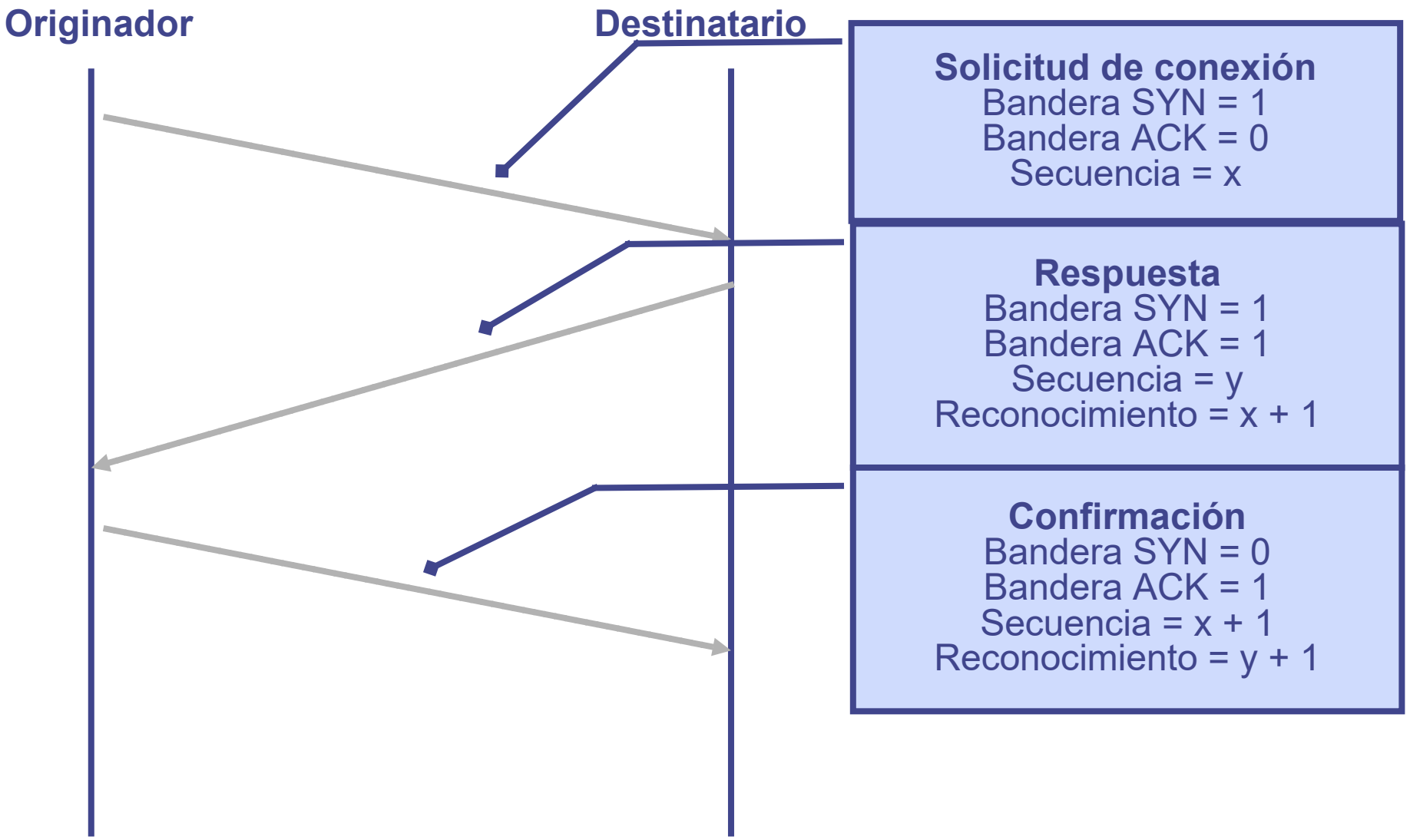
TCP – Inicio de Conexión en 3 vías



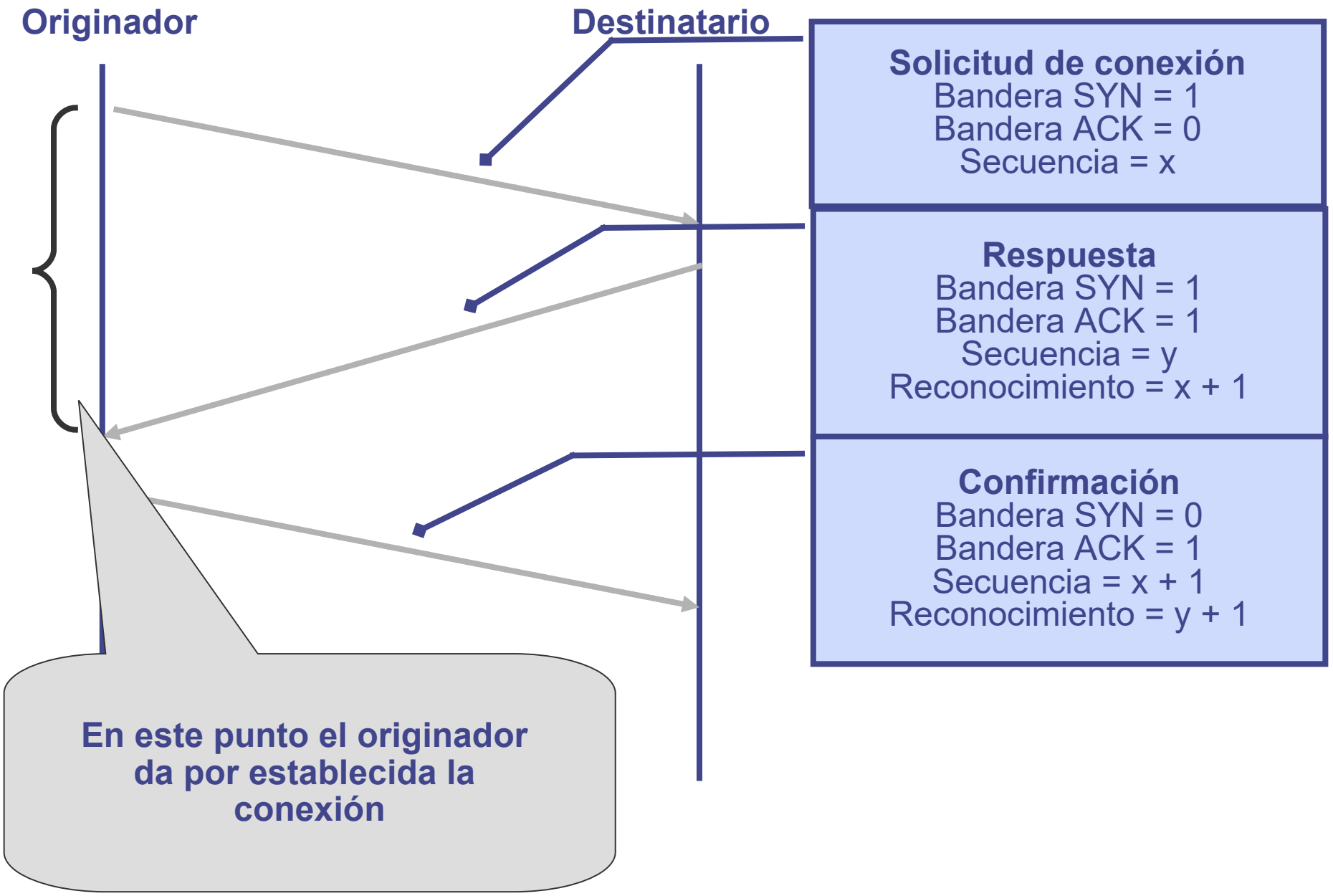
TCP – Inicio de Conexión en 3 vías



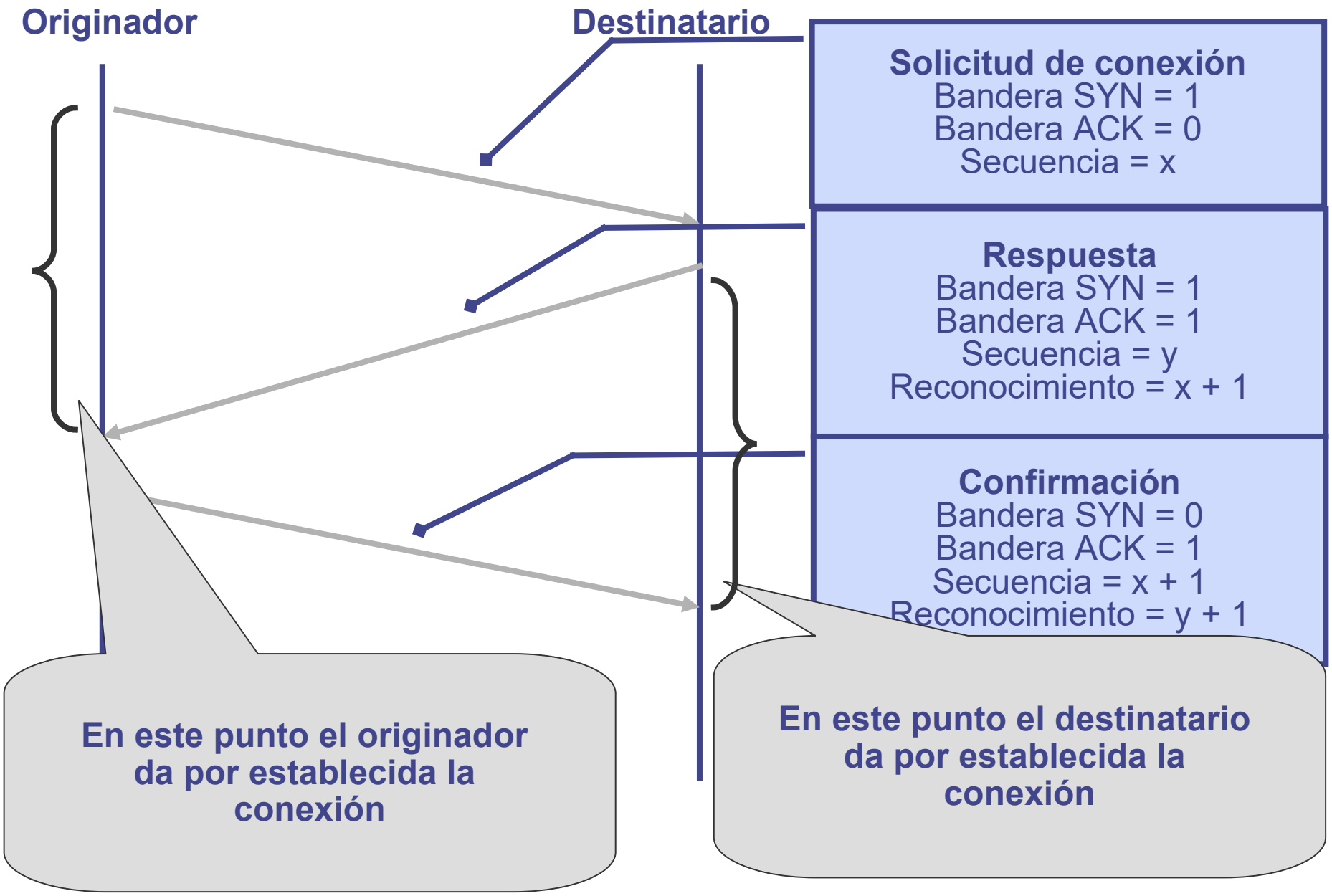
TCP – Inicio de Conexión en 3 vías



TCP – Inicio de Conexión en 3 vías



TCP – Inicio de Conexión en 3 vías



TCP – Fin de Conexión

- Terminar conexión:
  - Simétrica
    - Se intenta asegurar que no haya pérdida de datos, asegurando que ambos extremos estén de acuerdo en cerrar la conexión
    - se cierran separadamente ambos sentidos
    - dificultad “problema de los dos ejércitos”
  - Asimétrica
    - Uno de los extremos cierra la conexión unilateralmente
    - No garantiza que no haya pérdida de datos

CR = Connection Request
(Solicitud de Conexión)

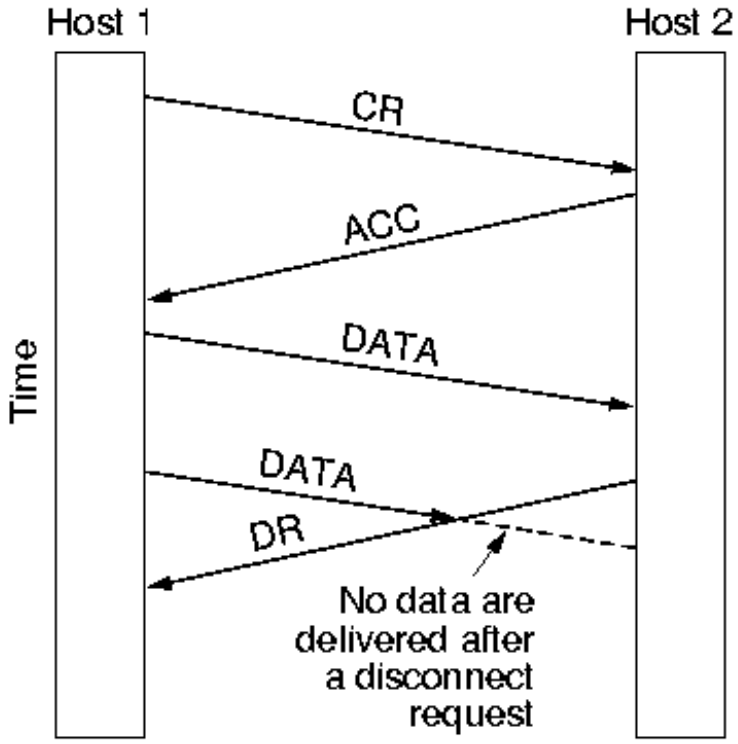
DR = Disconnection Request
(Solicitud de Desconexión)

TCP – Fin de Conexión

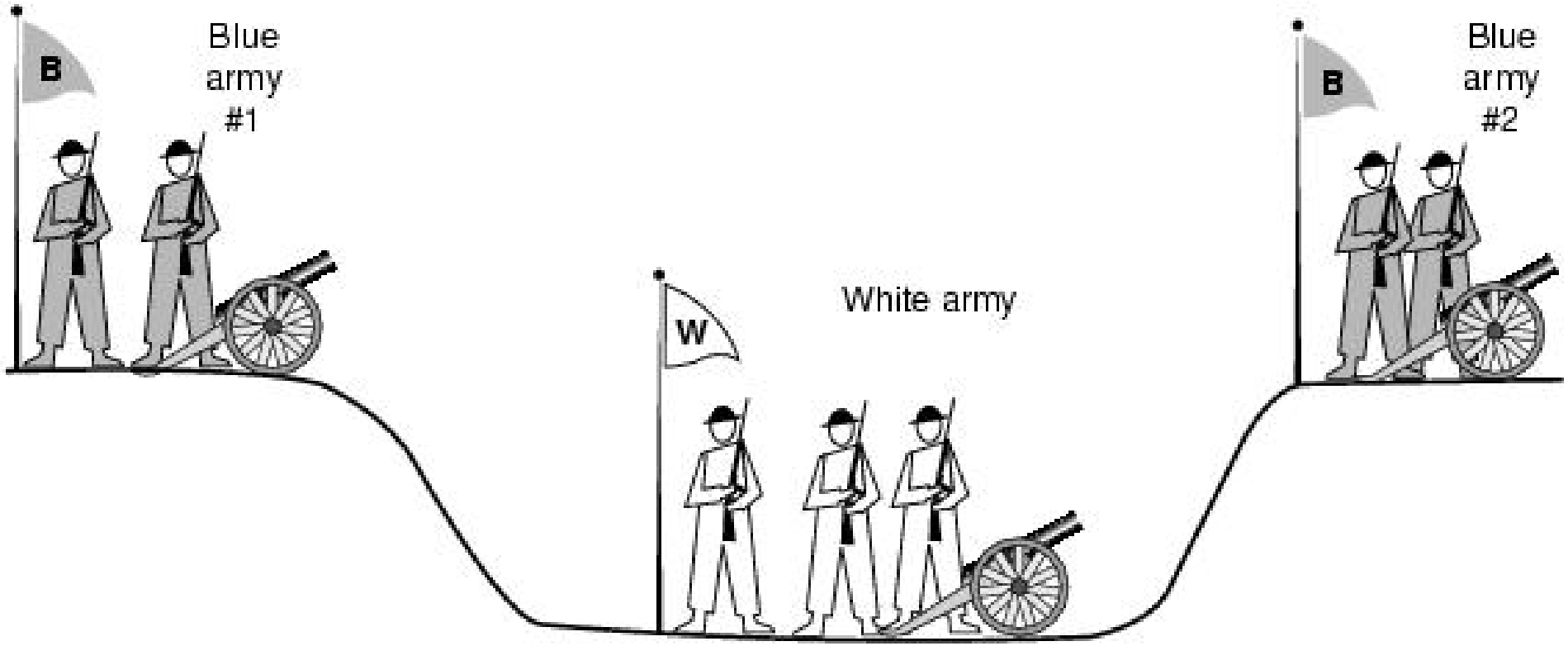
- Terminar conexión:
  - Simétrica
    - Se intenta asegurar que no haya pérdida de datos, asegurando que ambos extremos estén de acuerdo en cerrar la conexión
    - se cierran separadamente ambos sentidos
    - dificultad “problema de los dos ejércitos”
  - Asimétrica
    - Uno de los extremos cierra la conexión unilateralmente
    - No garantiza que no haya pérdida de datos

CR = Connection Request
(Solicitud de Conexión)

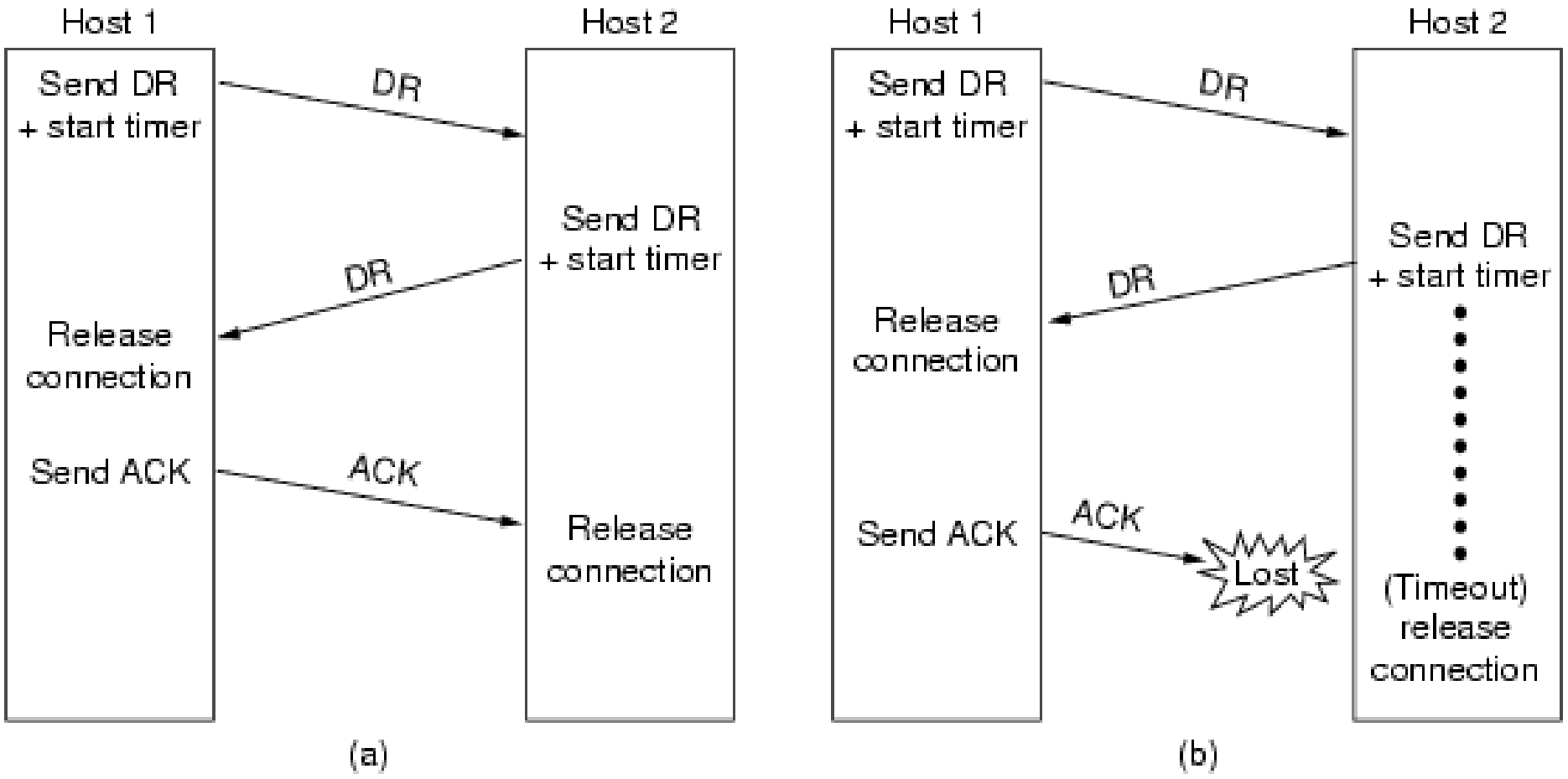
DR = Disconnection Request
(Solicitud de Desconexión)



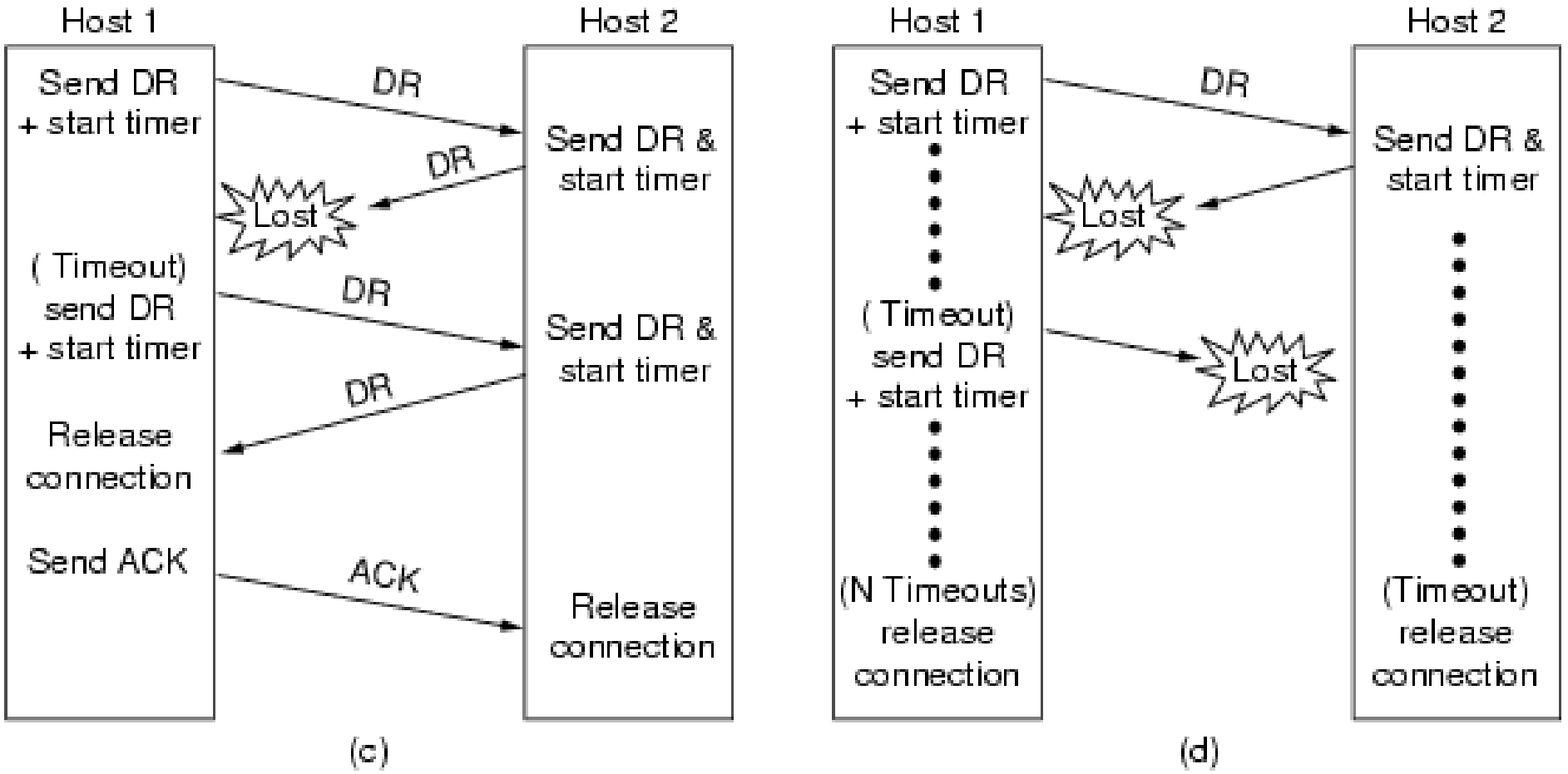
Fin de Conexión - Problema de los dos ejércitos



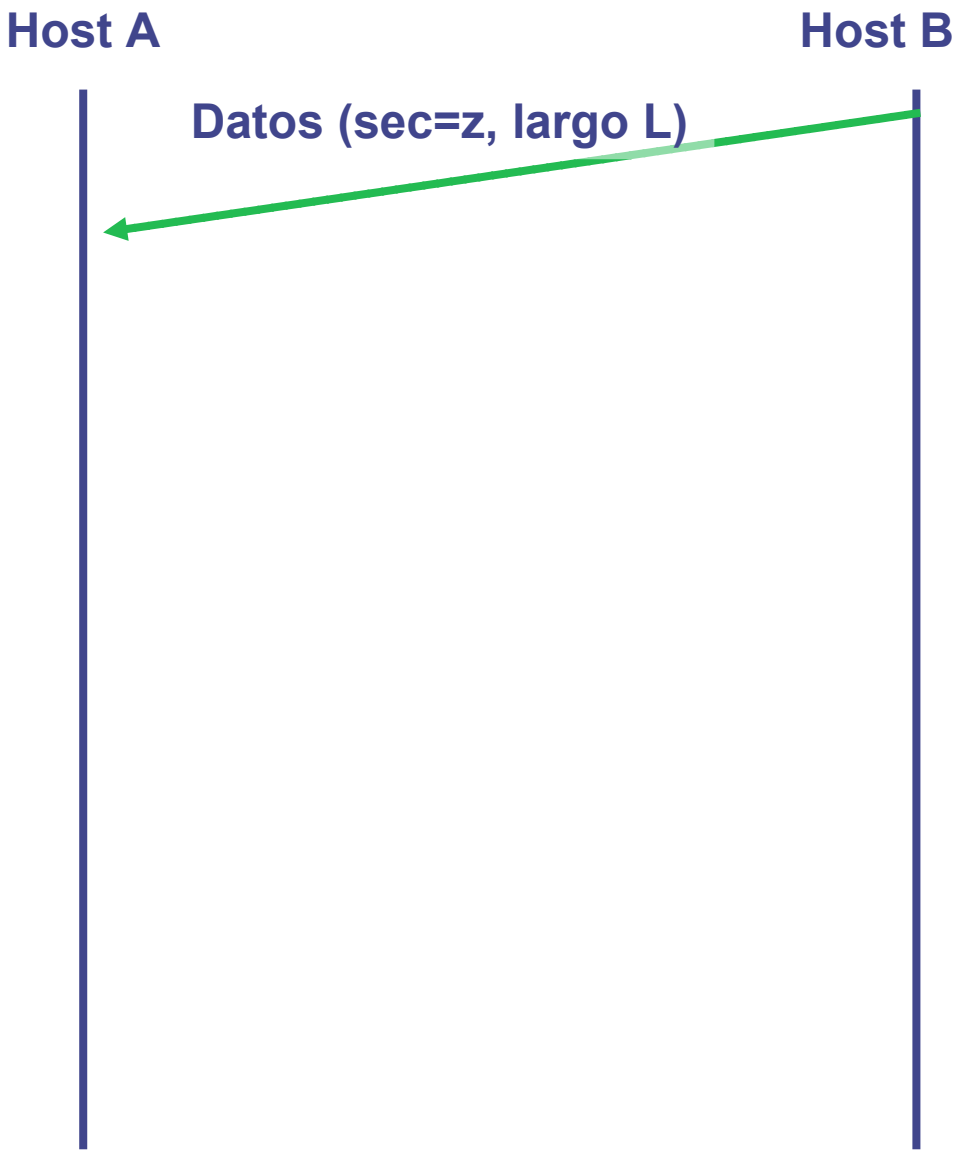
Fin de Conexión – Pérdida de algunos segmentos



Fin de Conexión – Pérdida de algunos segmentos

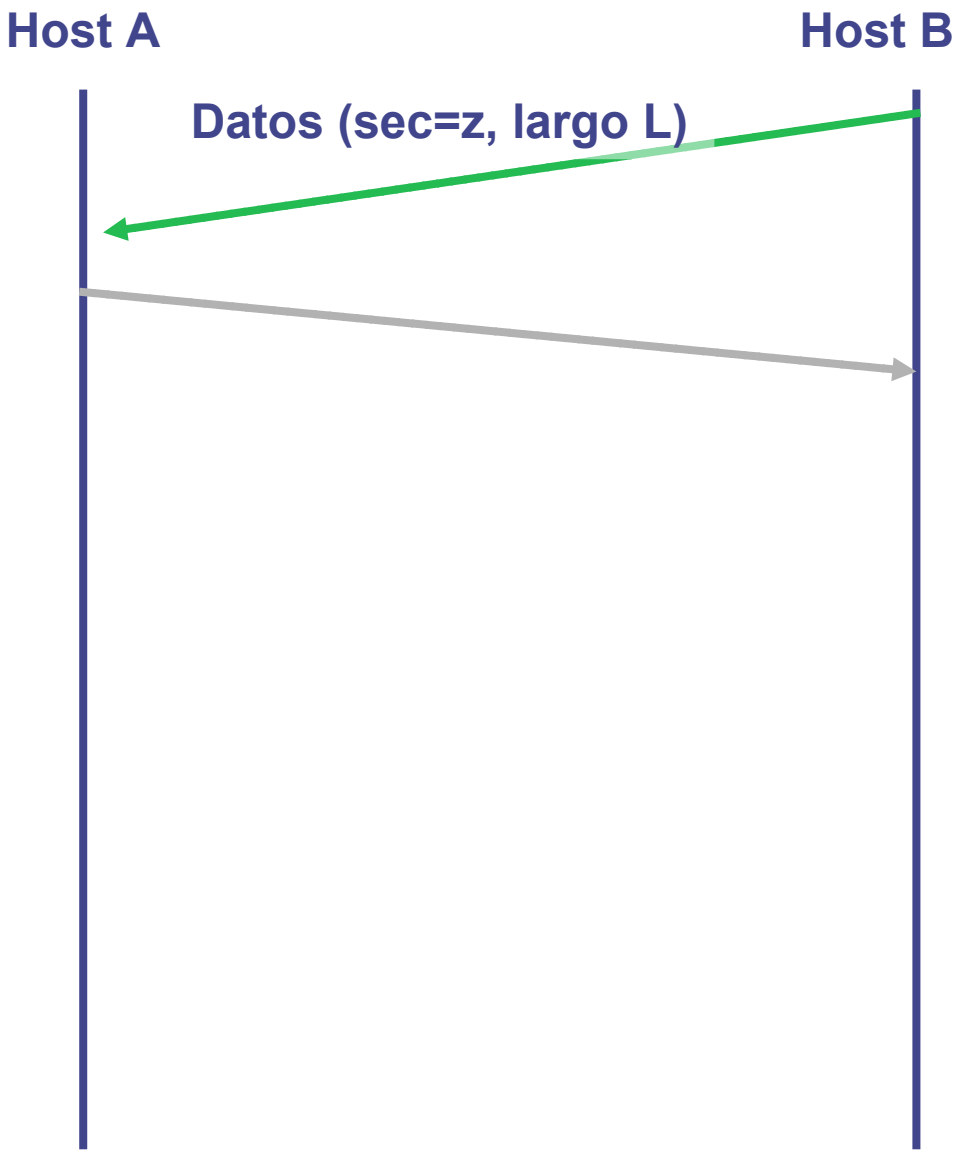


TCP - Fin de Conexión Simétrica



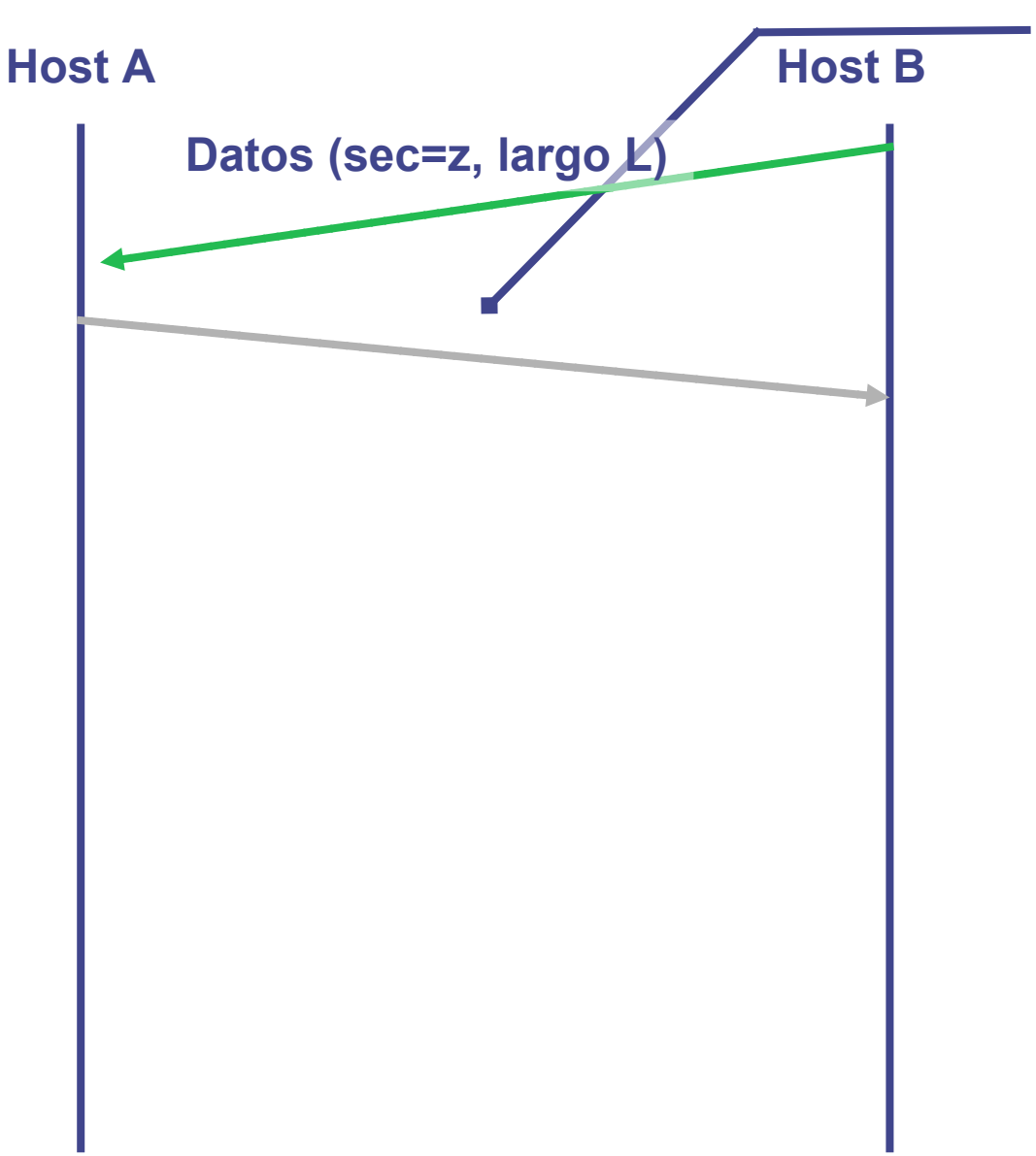
228

TCP - Fin de Conexión Simétrica



229

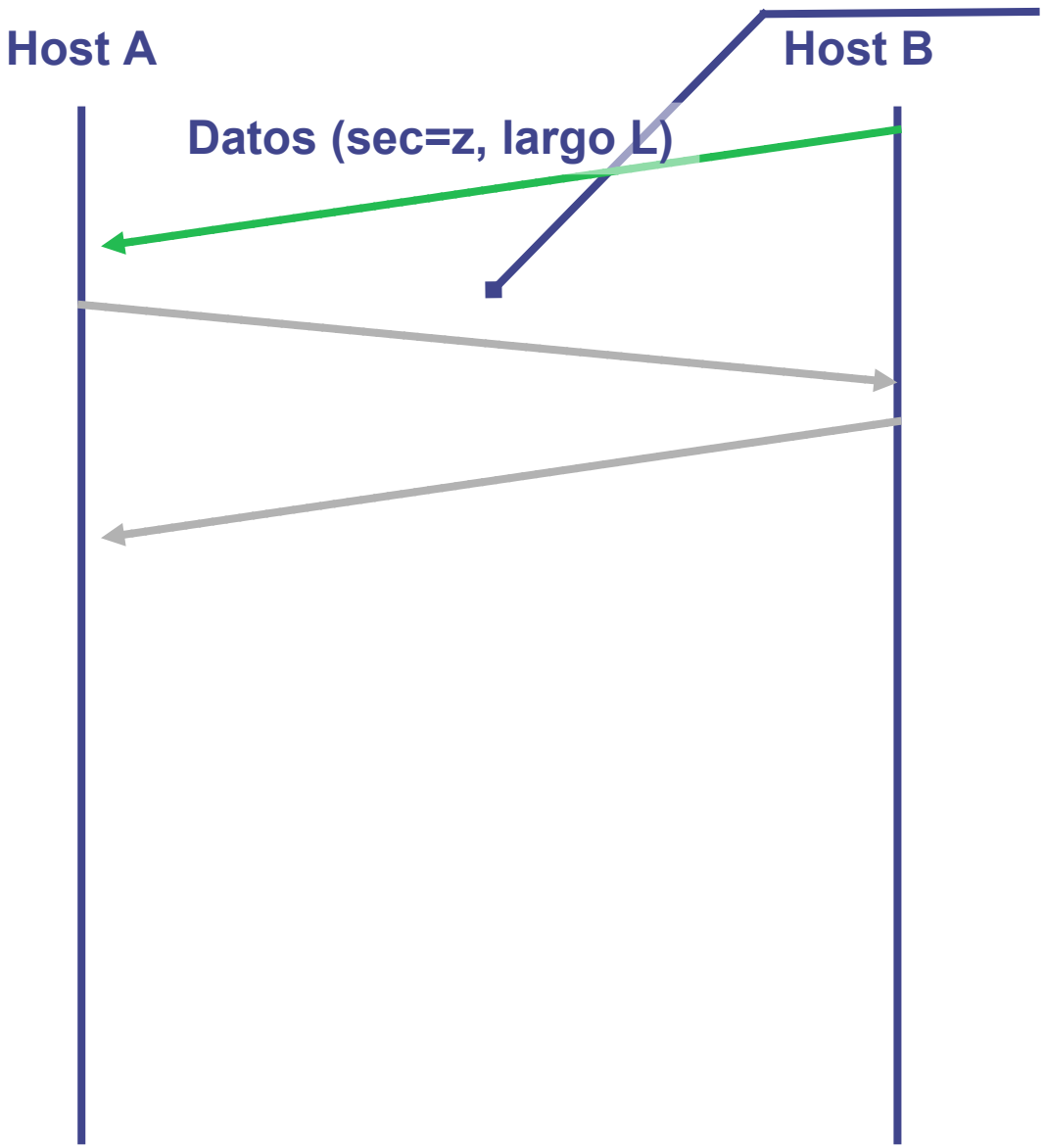
TCP - Fin de Conexión Simétrica



Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + L

230

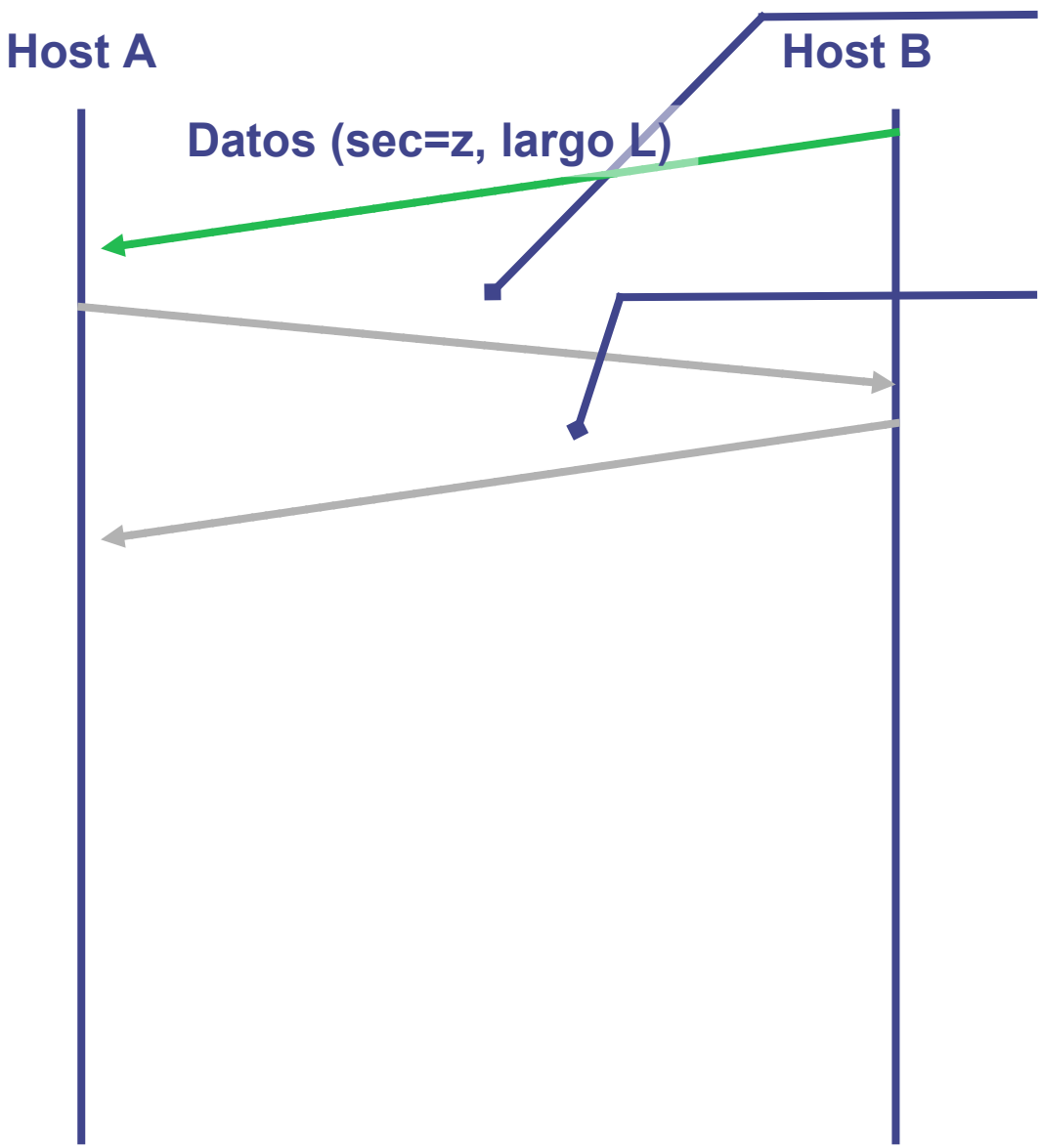
TCP - Fin de Conexión Simétrica



Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + L

231

TCP - Fin de Conexión Simétrica

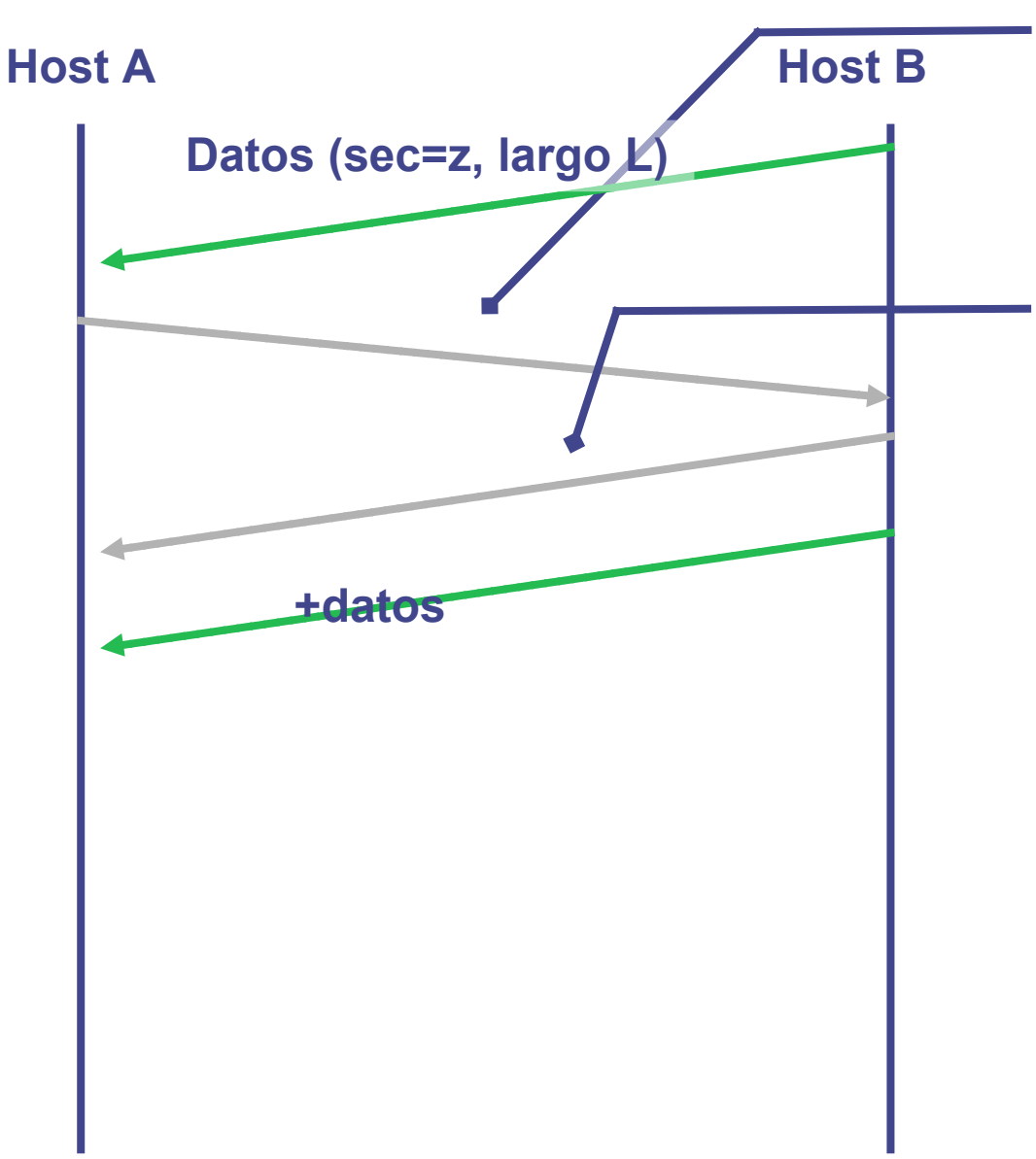


Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + L

Confirmación de corte
Bandera FIN = 0
Bandera ACK = 1
Secuencia = z + L
Reconocimiento = w + 1

232

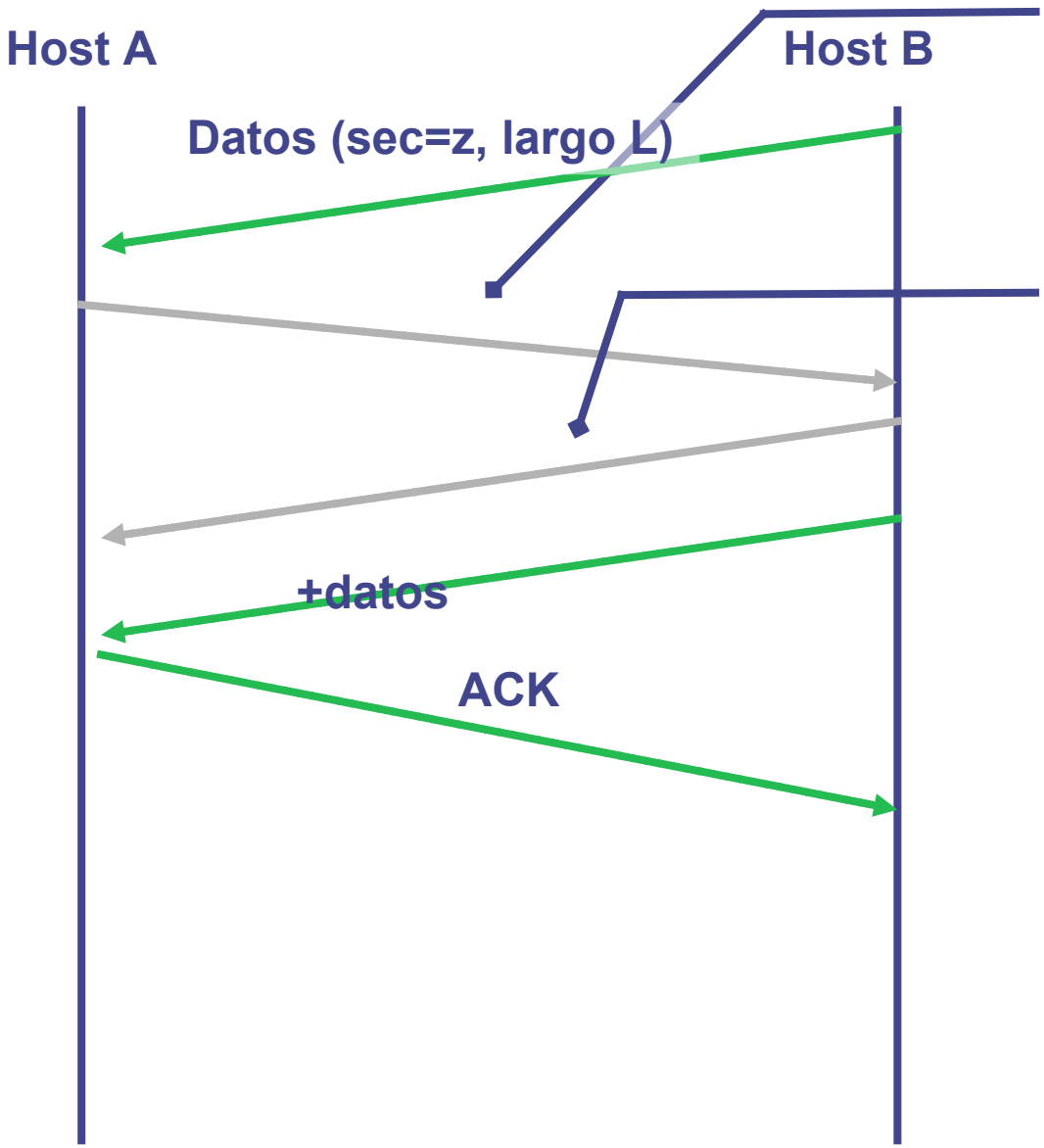
TCP - Fin de Conexión Simétrica



Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + L

Confirmación de corte
Bandera FIN = 0
Bandera ACK = 1
Secuencia = z + L
Reconocimiento = w + 1

TCP - Fin de Conexión Simétrica

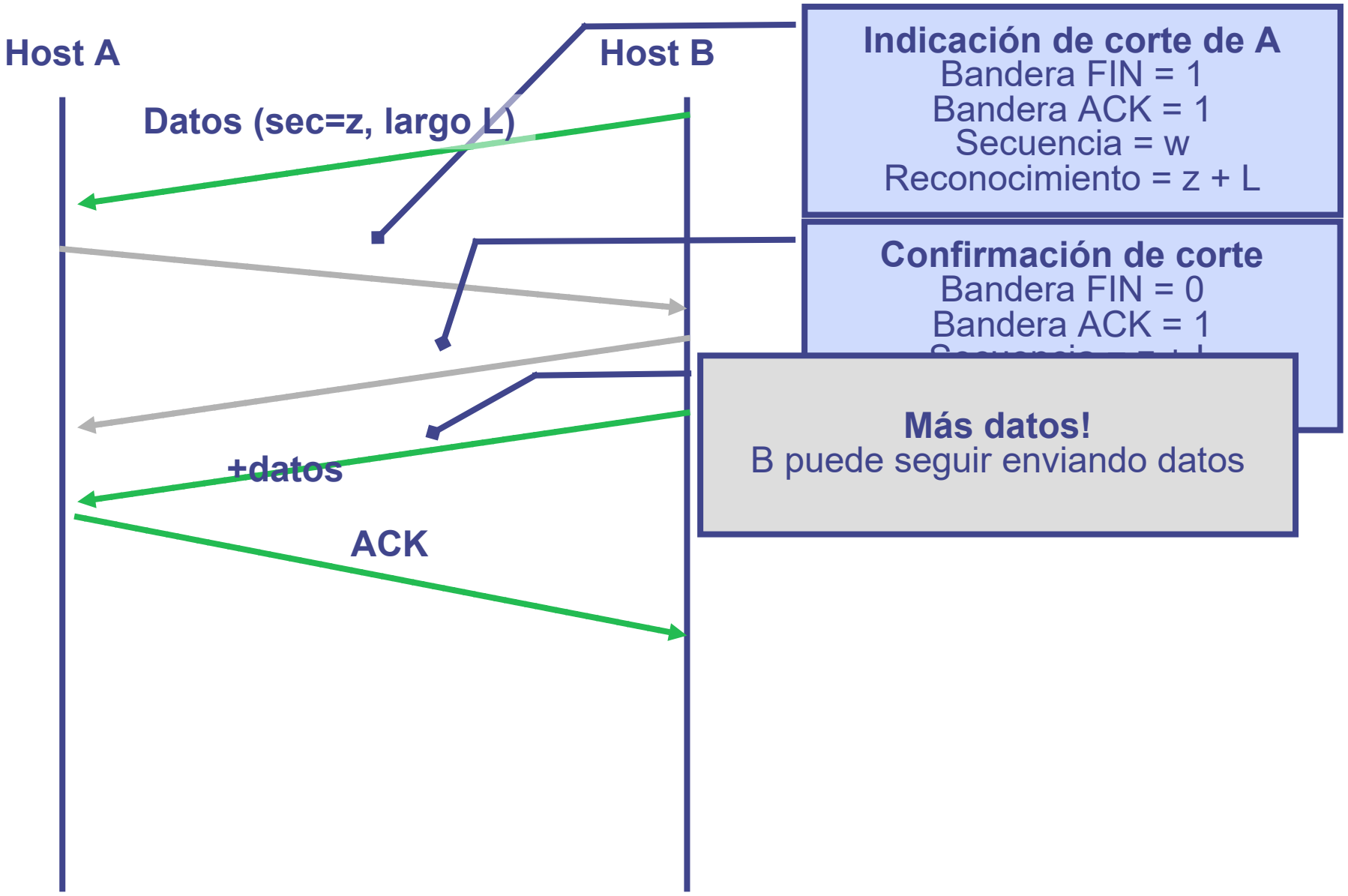


Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + L

Confirmación de corte
Bandera FIN = 0
Bandera ACK = 1
Secuencia = z + L
Reconocimiento = w + 1

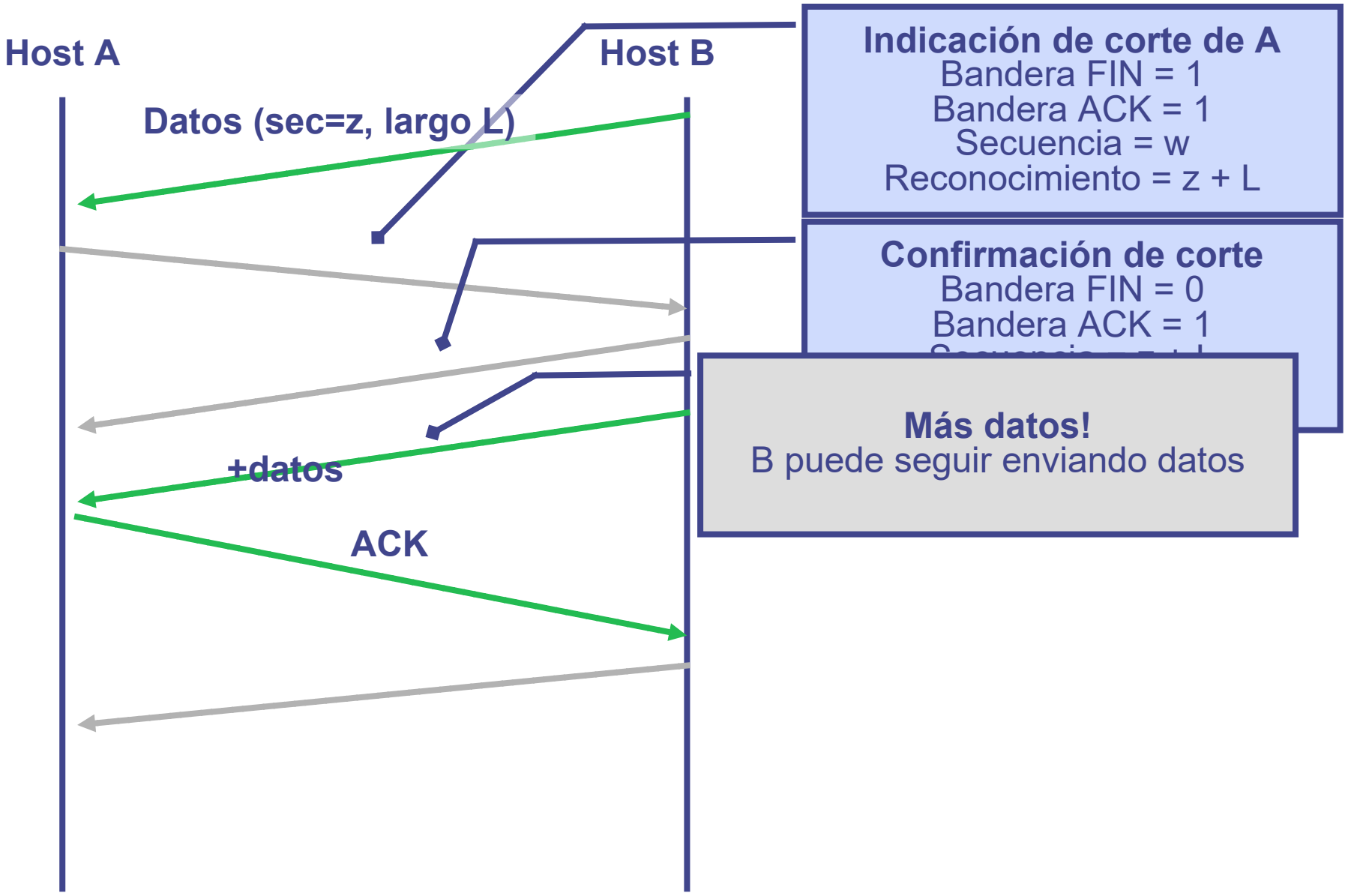
234

TCP - Fin de Conexión Simétrica



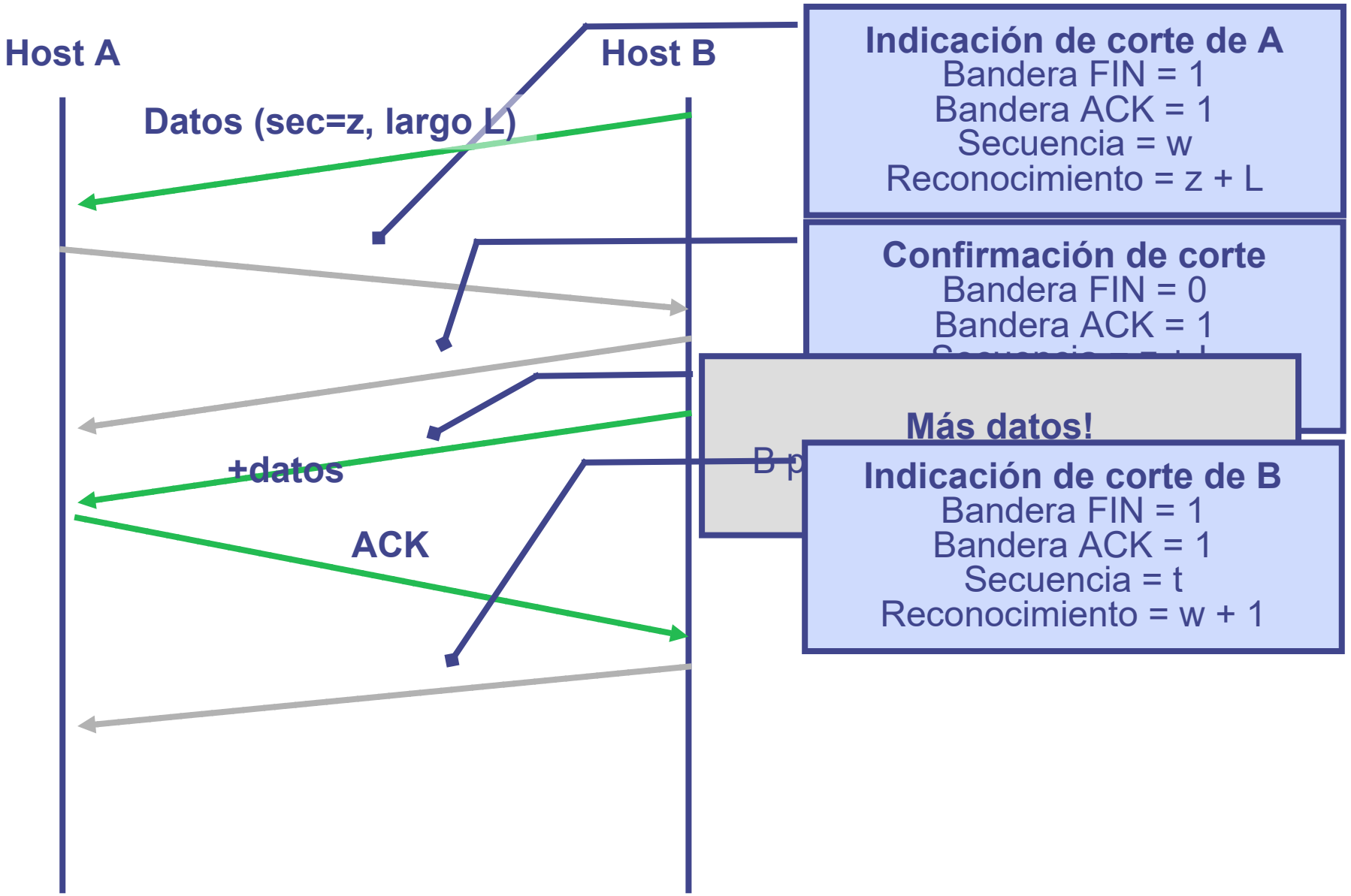
235

TCP - Fin de Conexión Simétrica

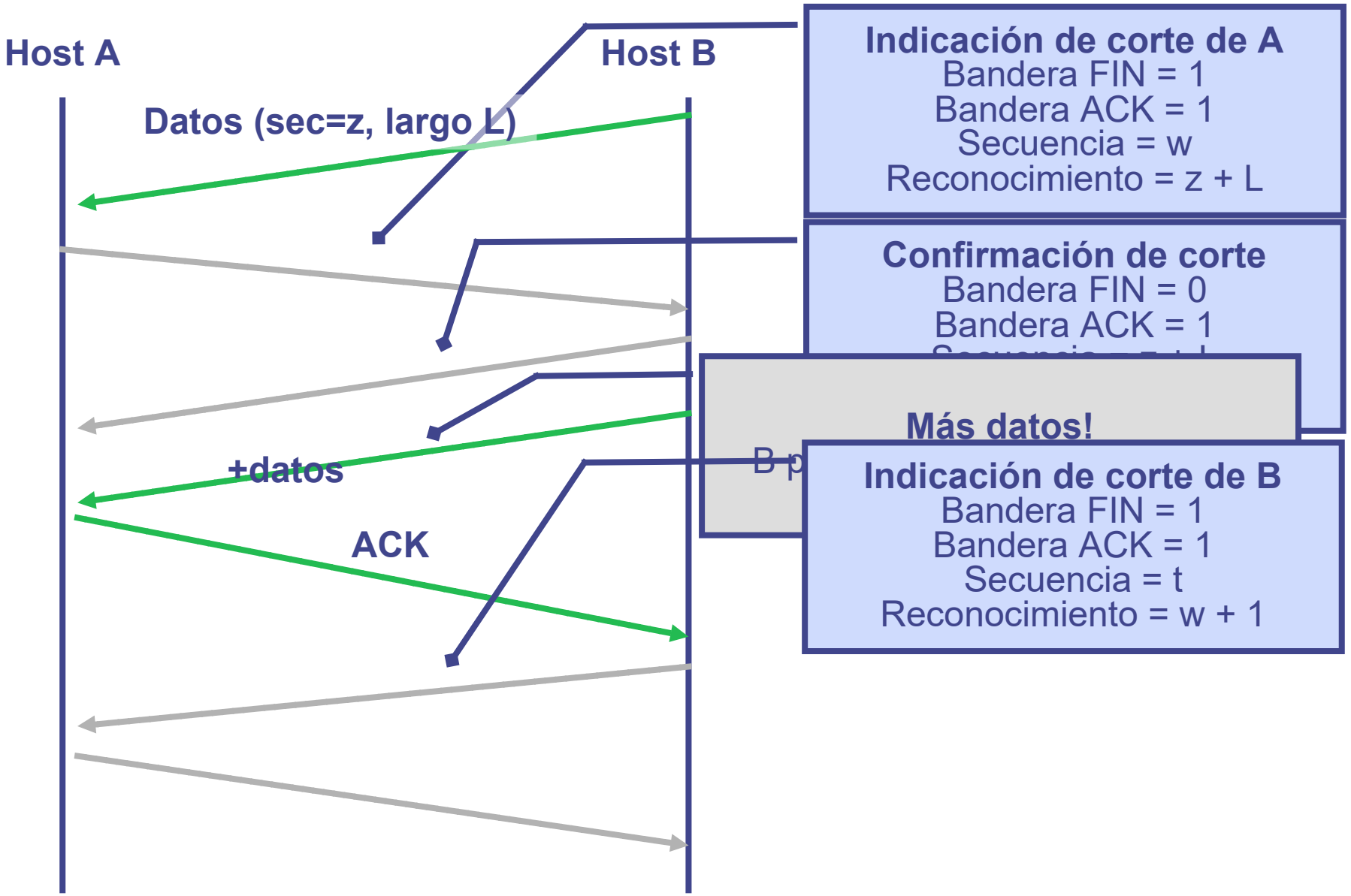


236

TCP - Fin de Conexión Simétrica

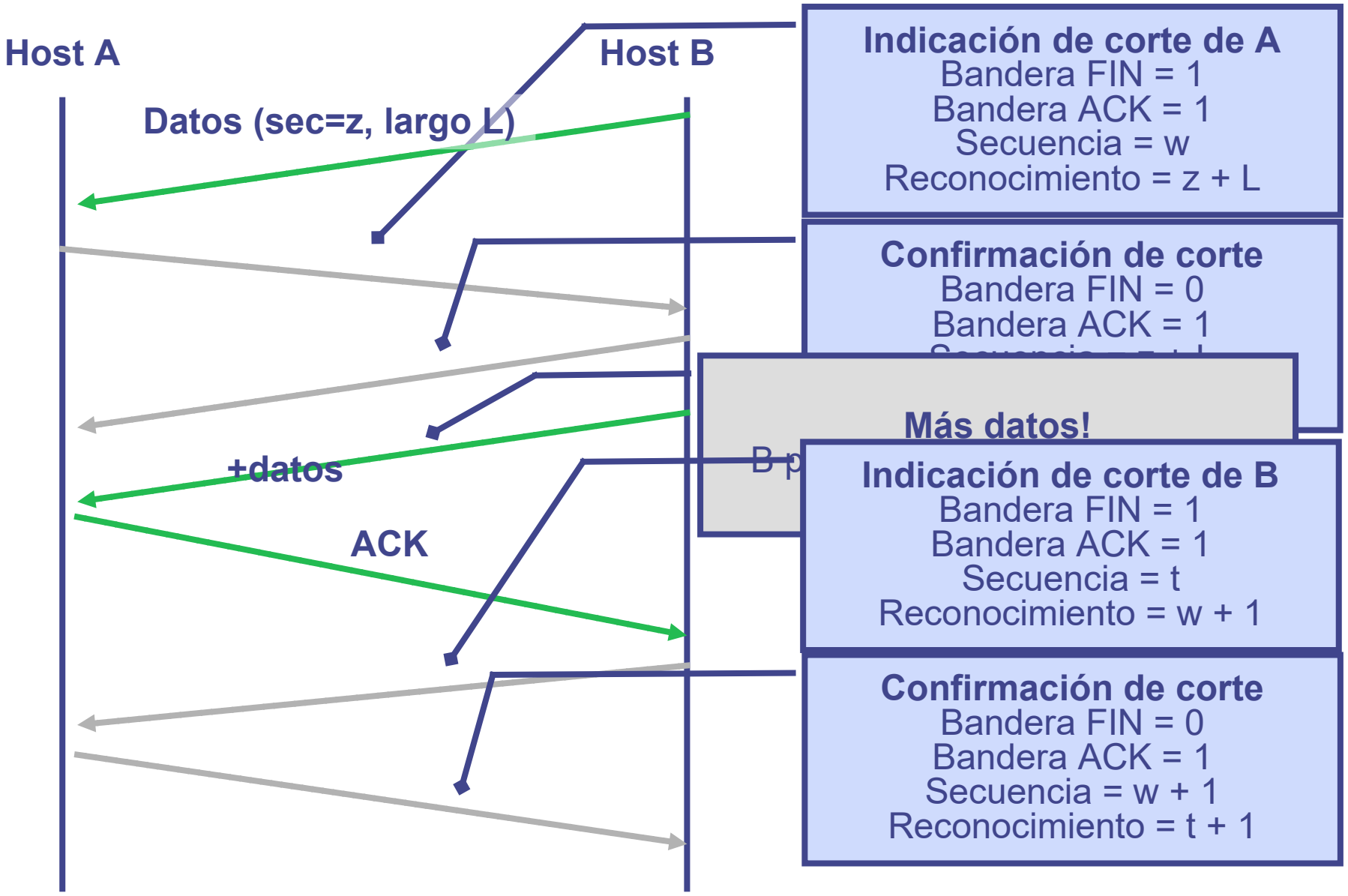


TCP - Fin de Conexión Simétrica



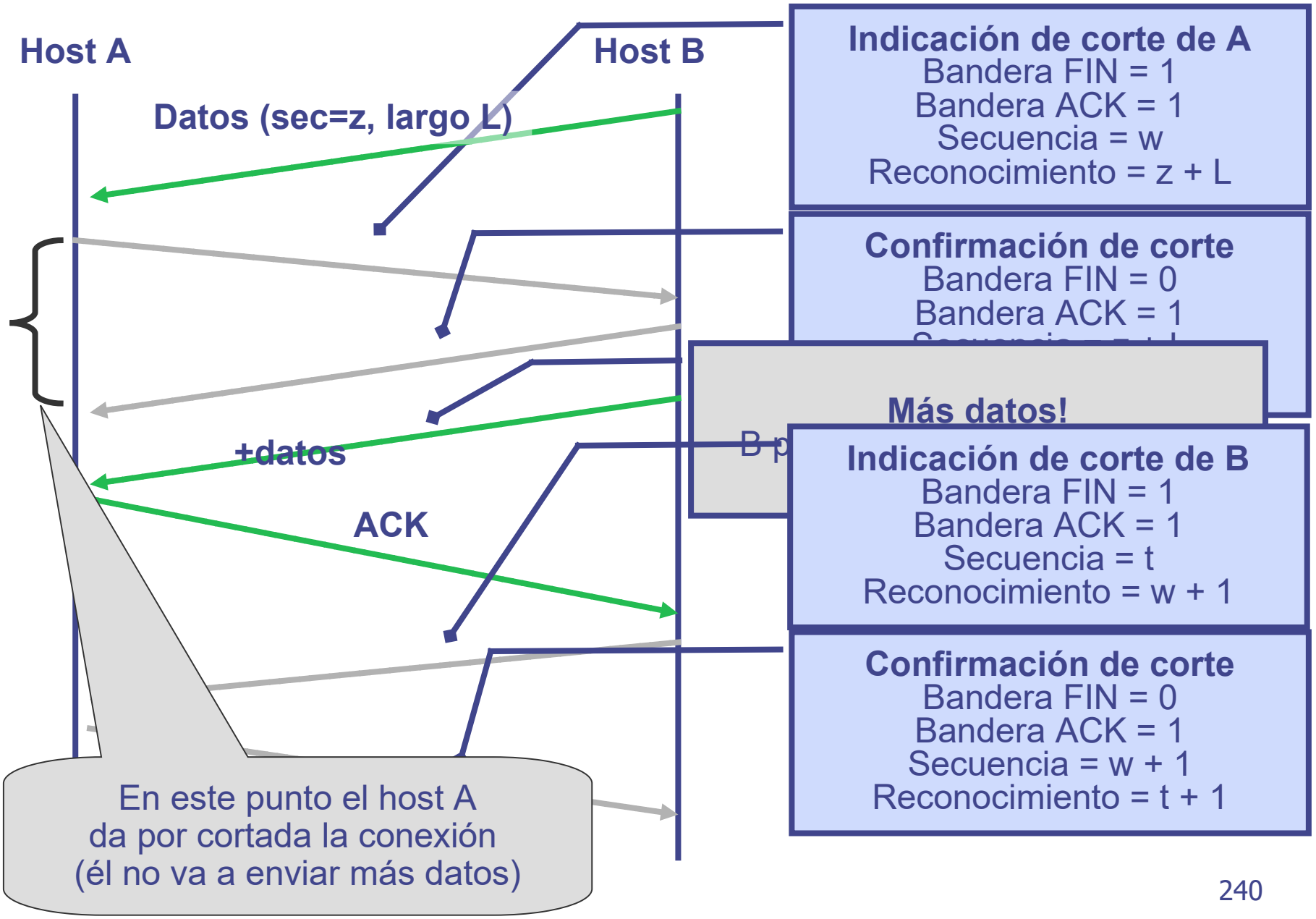
238

TCP - Fin de Conexión Simétrica



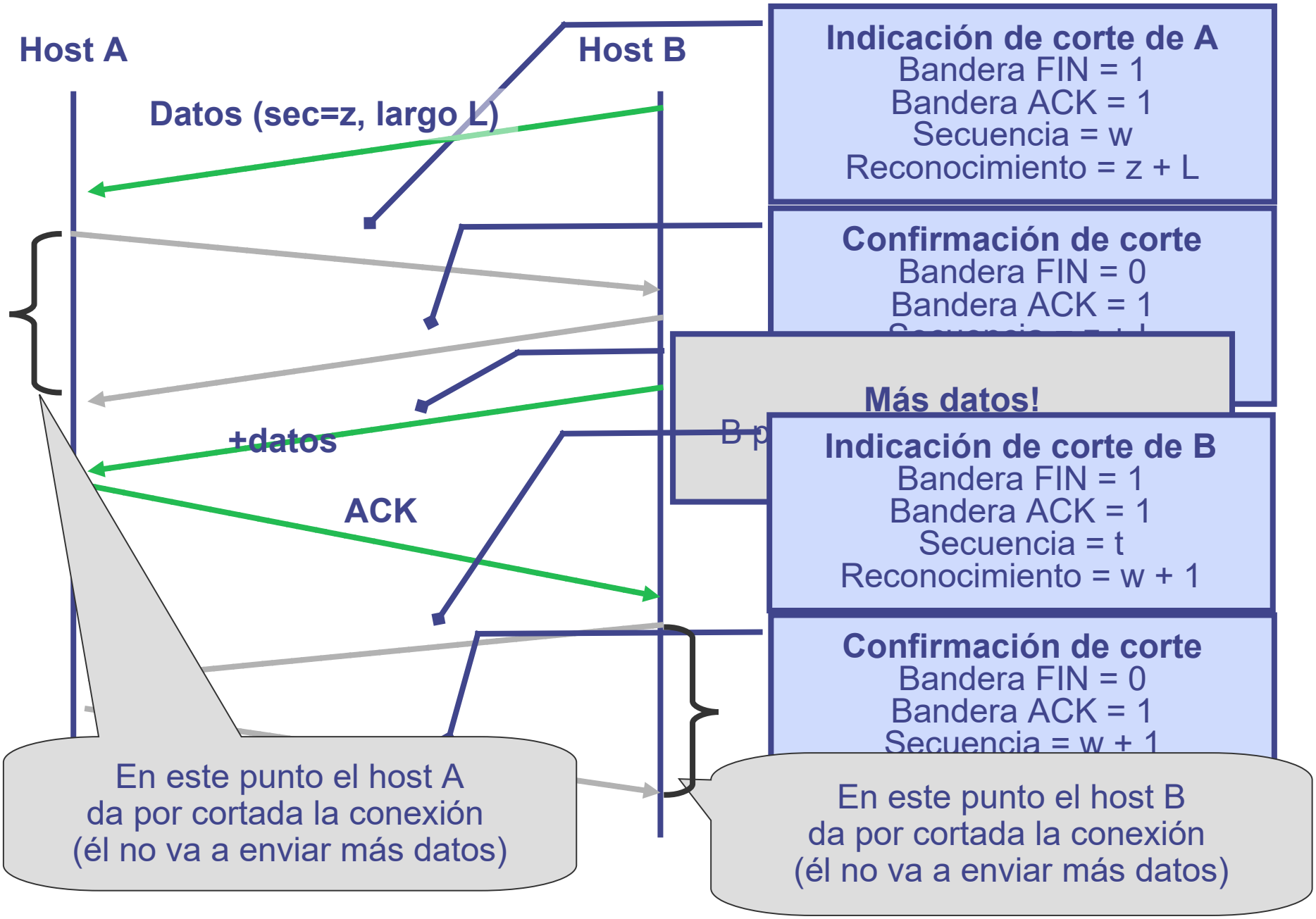
239

TCP - Fin de Conexión Simétrica

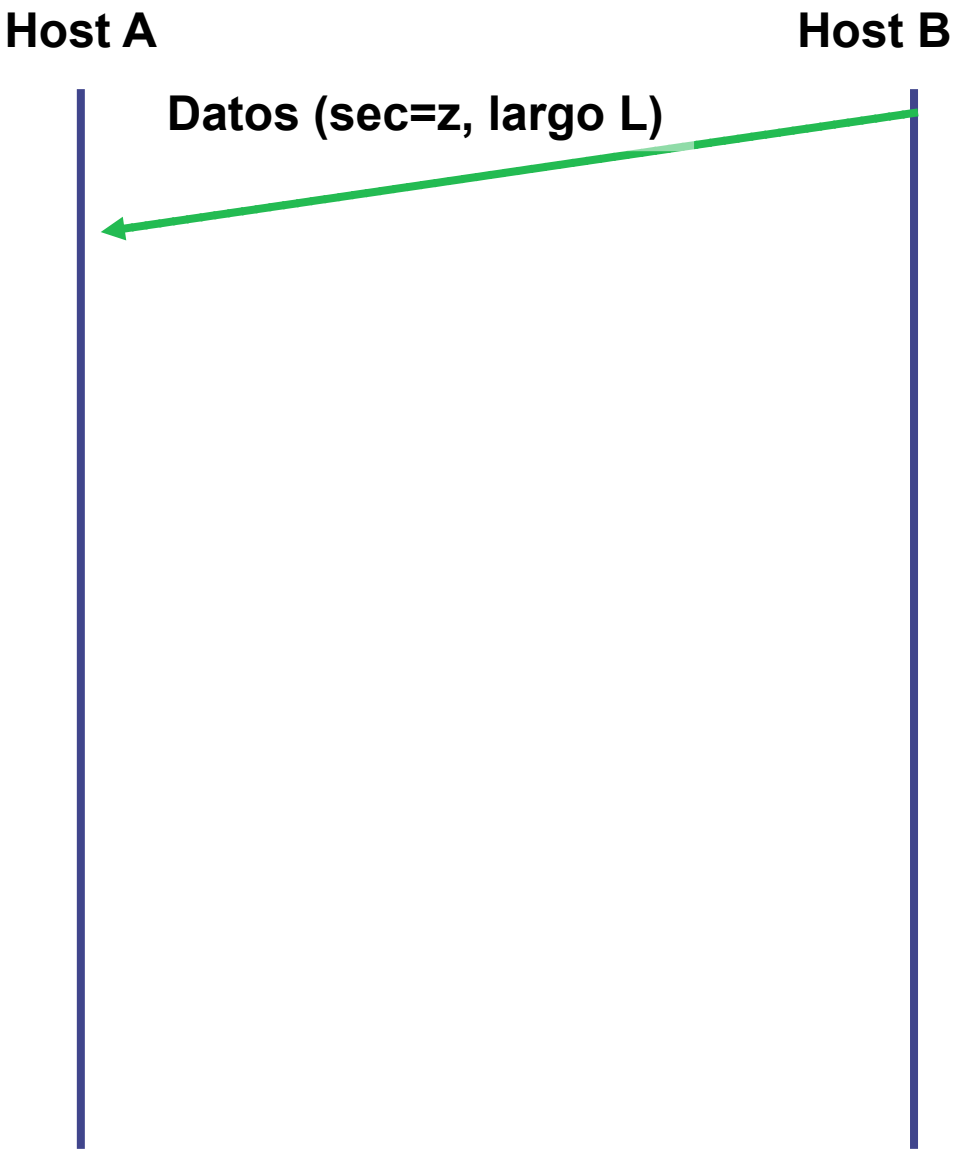


240

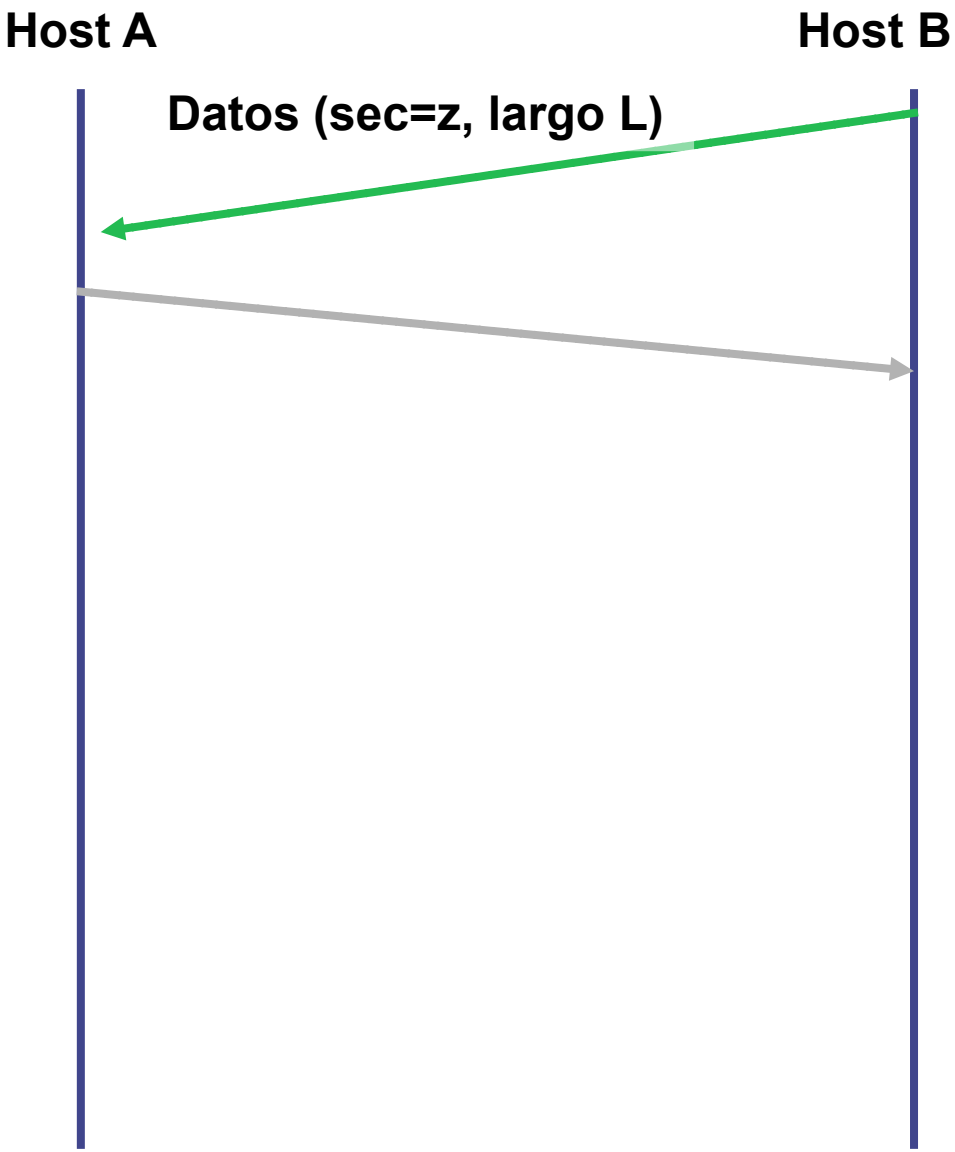
TCP - Fin de Conexión Simétrica



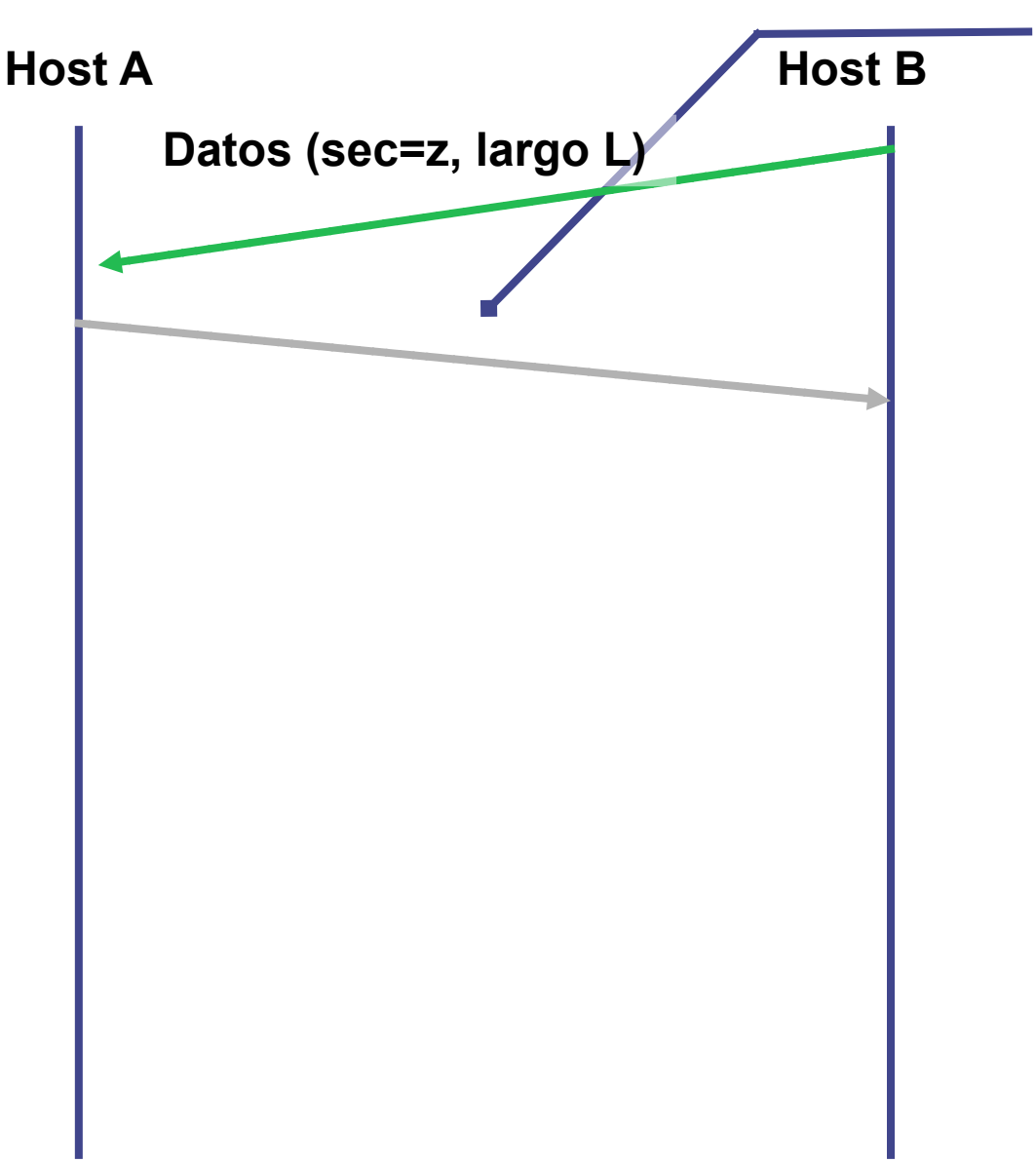
TCP - Fin de Conexión Asimétrica



TCP - Fin de Conexión Asimétrica

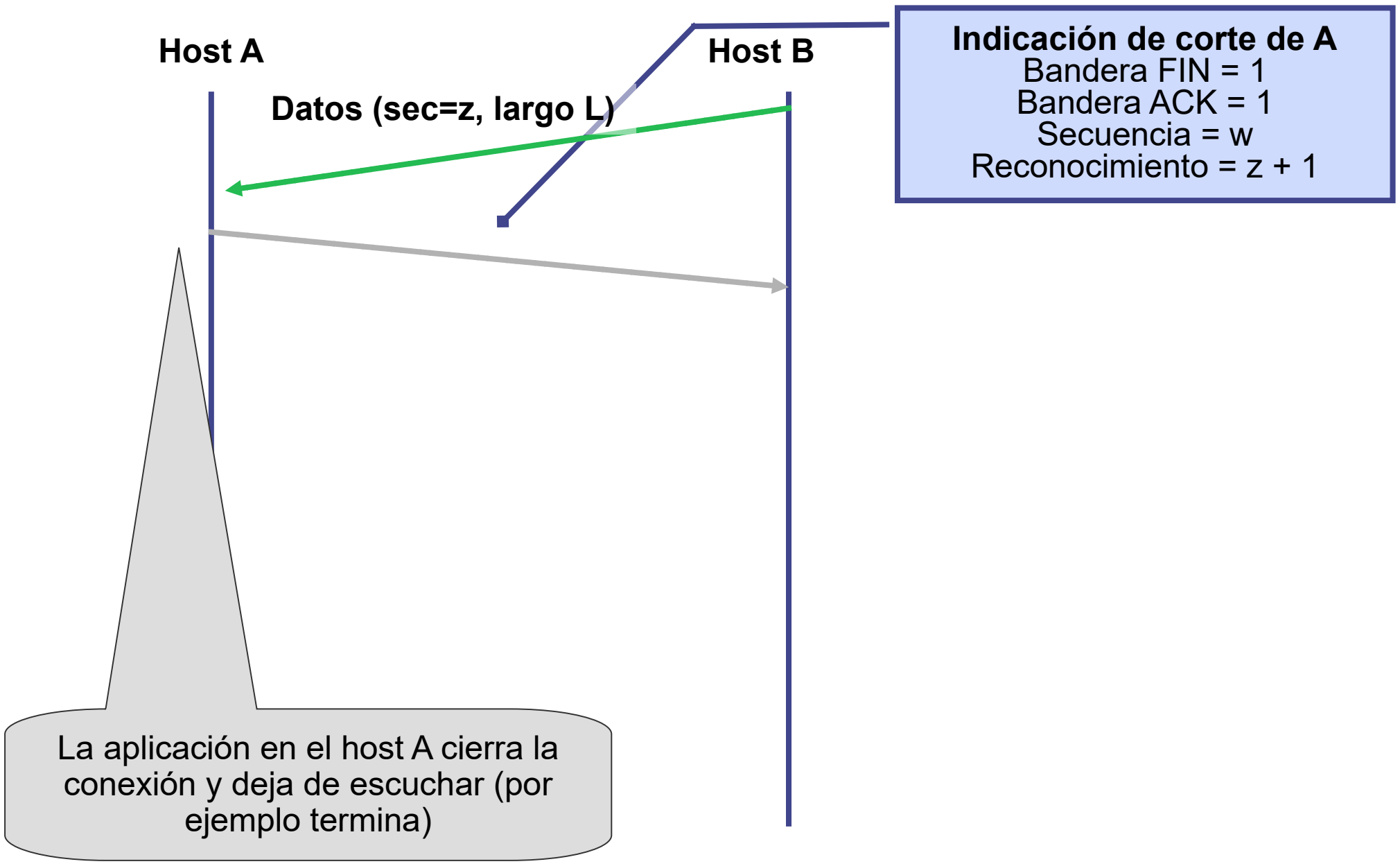


TCP - Fin de Conexión Asimétrica

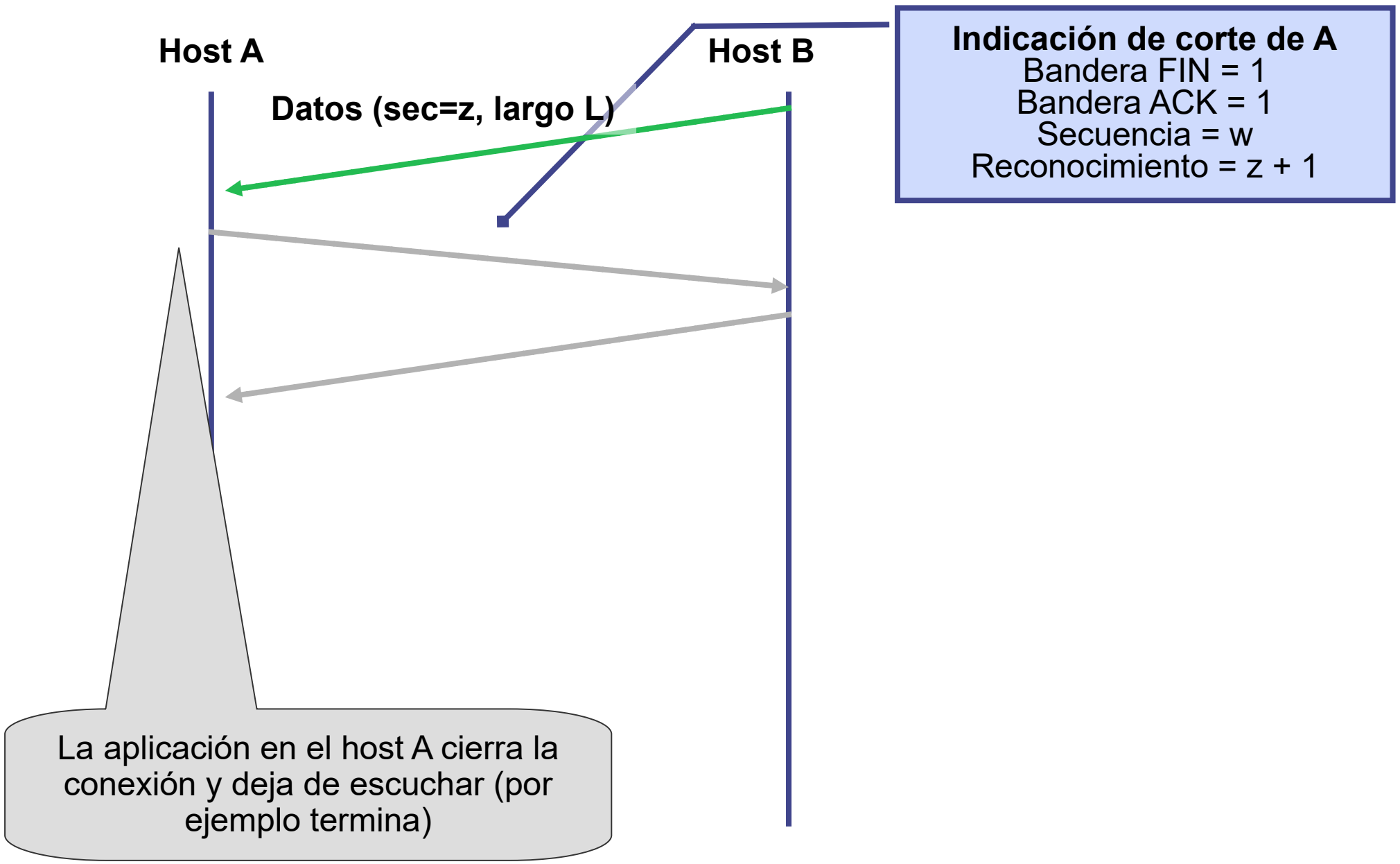


Indicación de corte de A
Bandera FIN = 1
Bandera ACK = 1
Secuencia = w
Reconocimiento = z + 1

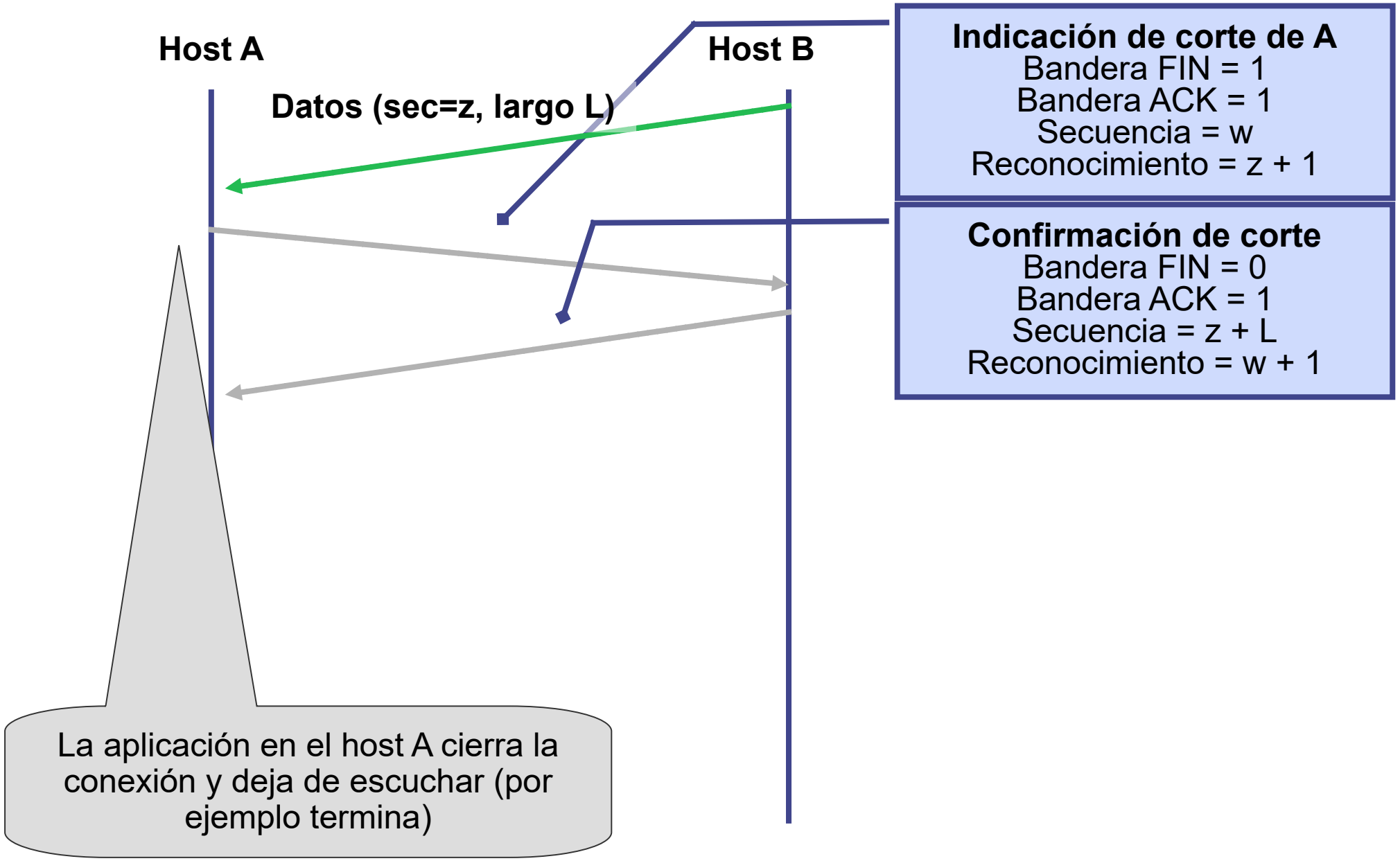
TCP - Fin de Conexión Asimétrica



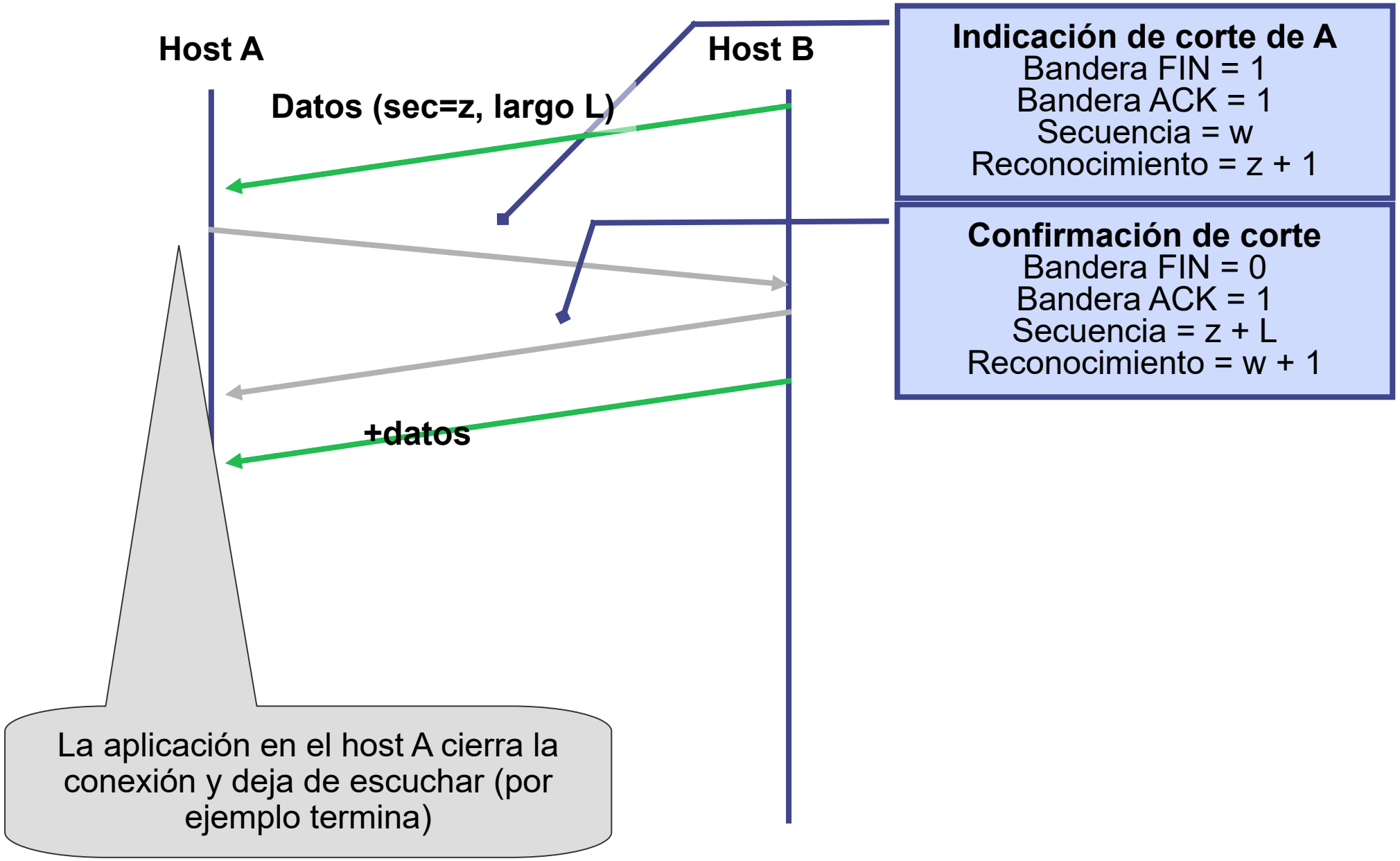
TCP - Fin de Conexión Asimétrica



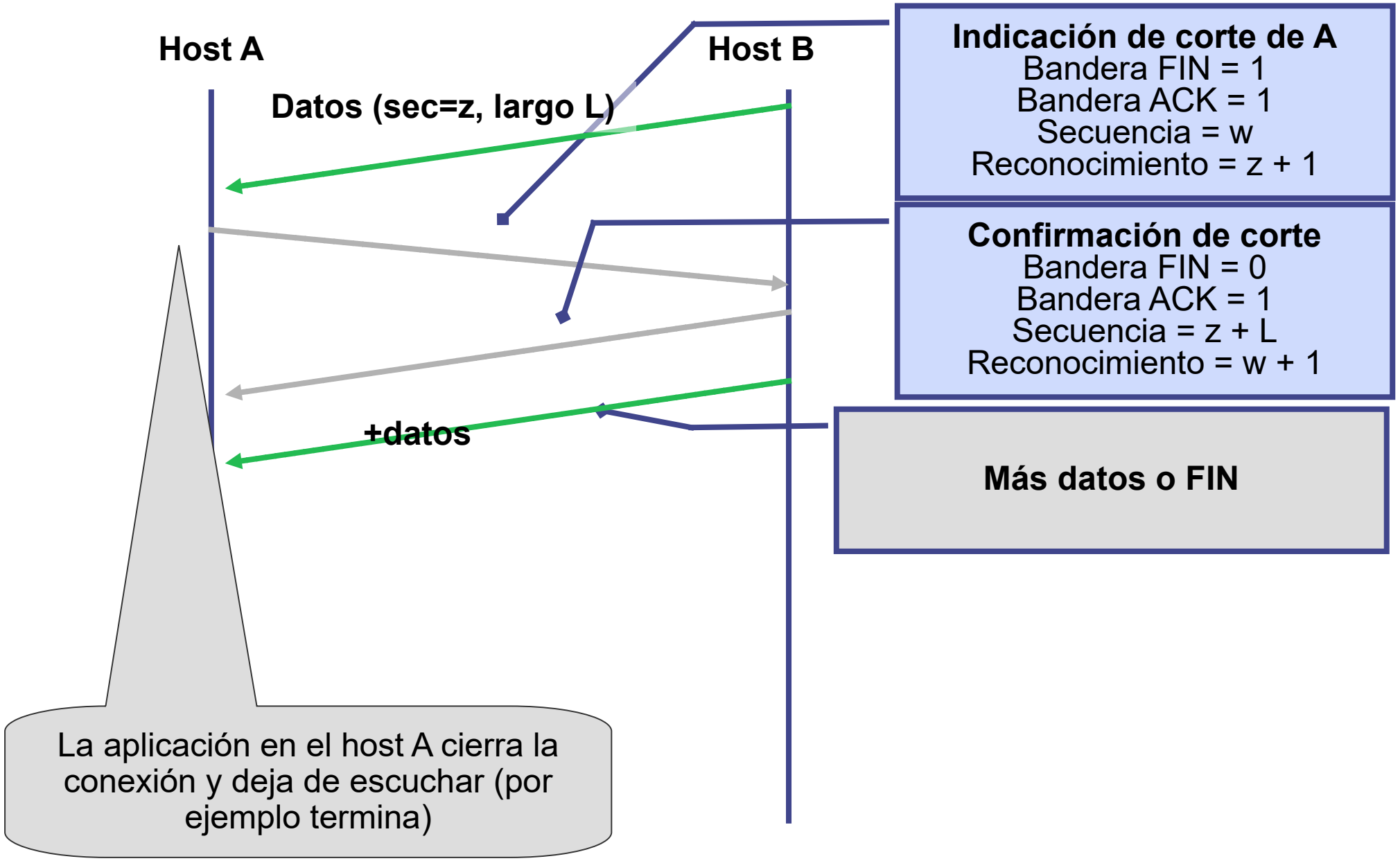
TCP - Fin de Conexión Asimétrica



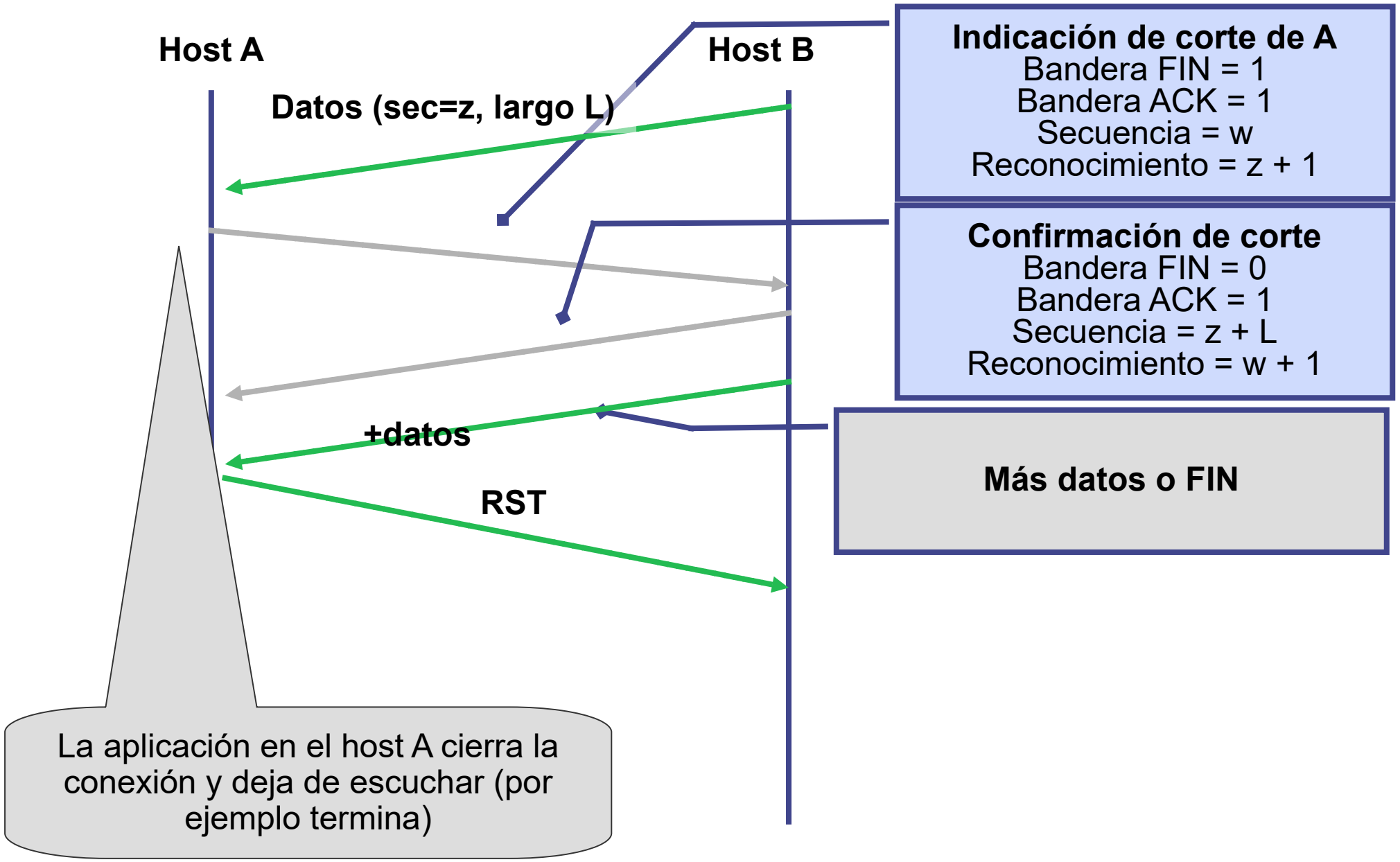
TCP - Fin de Conexión Asimétrica



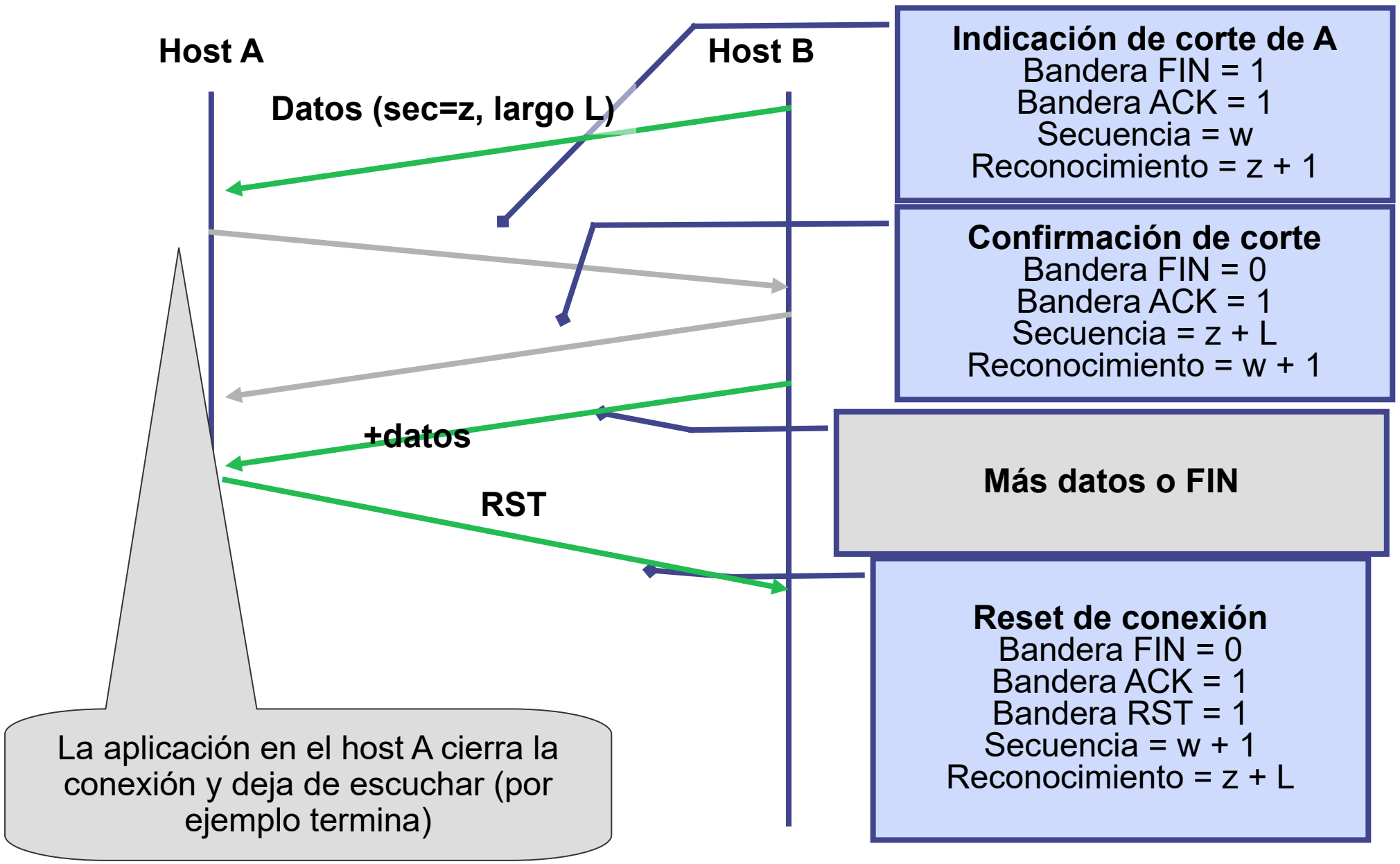
TCP - Fin de Conexión Asimétrica



TCP - Fin de Conexión Asimétrica



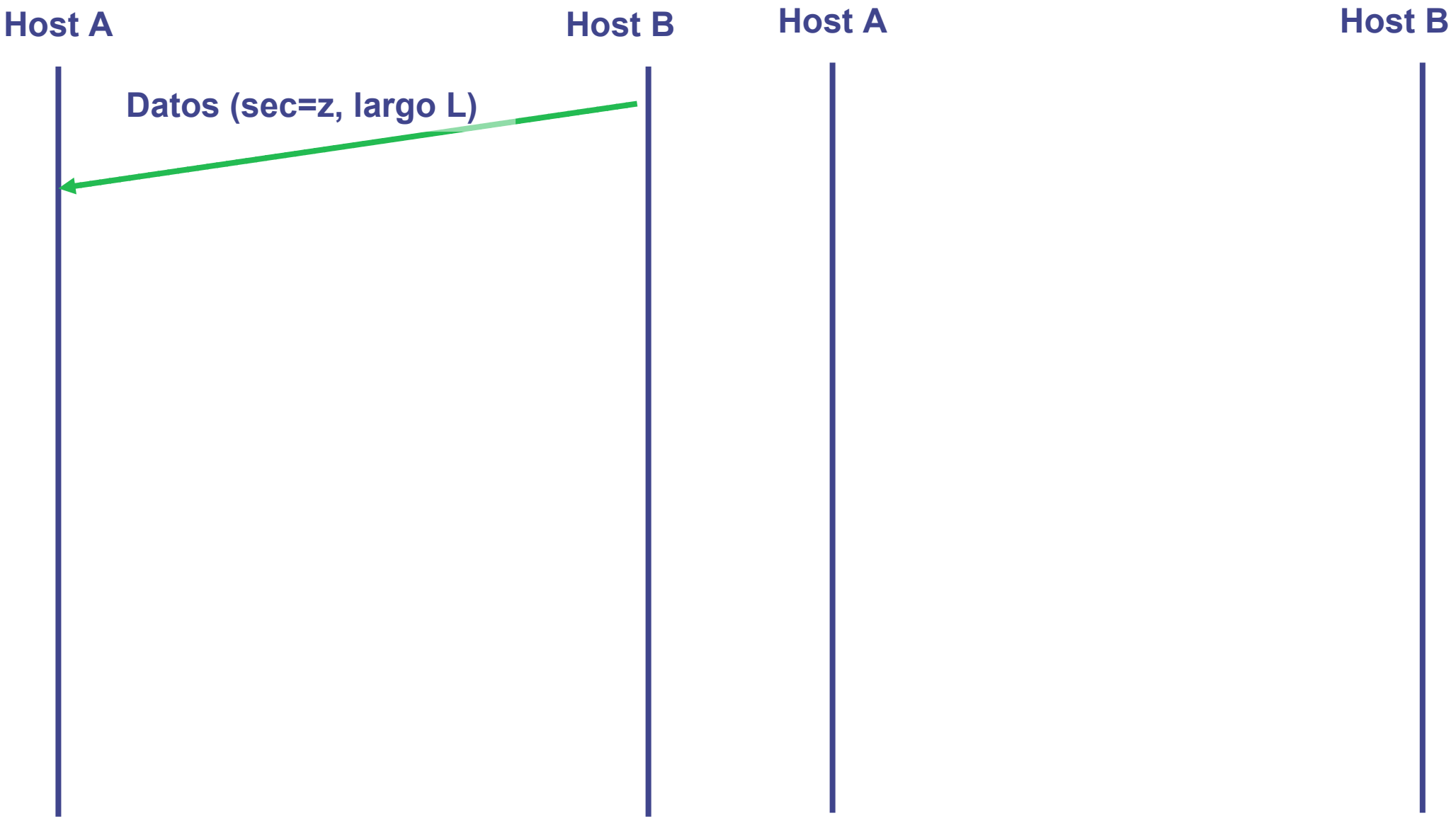
TCP - Fin de Conexión Asimétrica



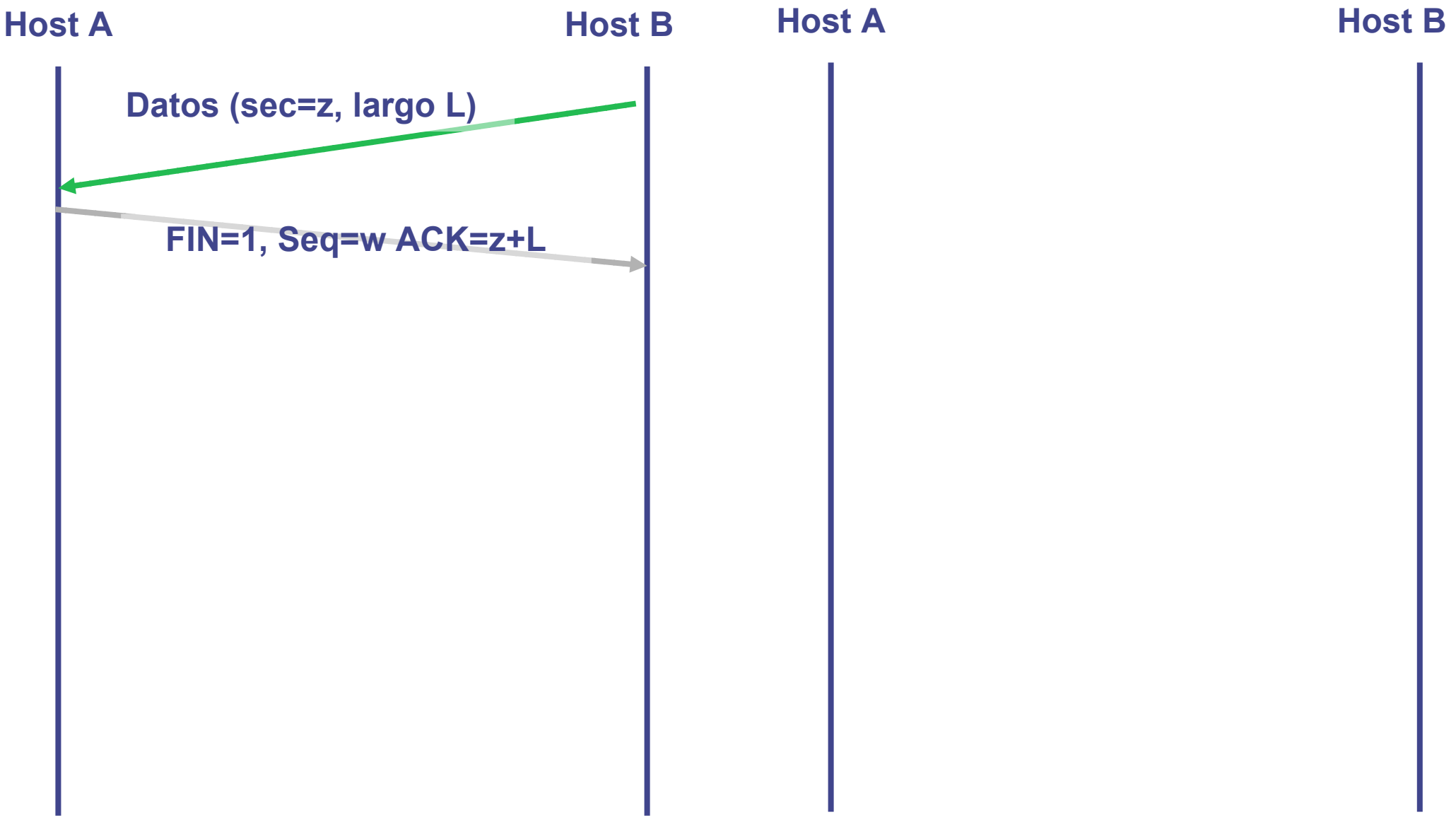
TCP - Fin de Conexión

- Simétrica:
  - Cuatro vías: cada sentido finaliza y se reconoce la finalización.
  - Tres vías: caso particular del anterior, al ACK el FIN, también notifico mi intención de finalizar la conexión (FIN).
- Asimétrica:
  - Flag RST
  - Los datos enviados luego de la confirmación de corte de B hacia A se pierden.
  - Los últimos datos enviados por B y reconocidos son hasta el byte “z”, cualquier otro dato posterior debe asumirse que se perdió (a menos que recibamos ACK)

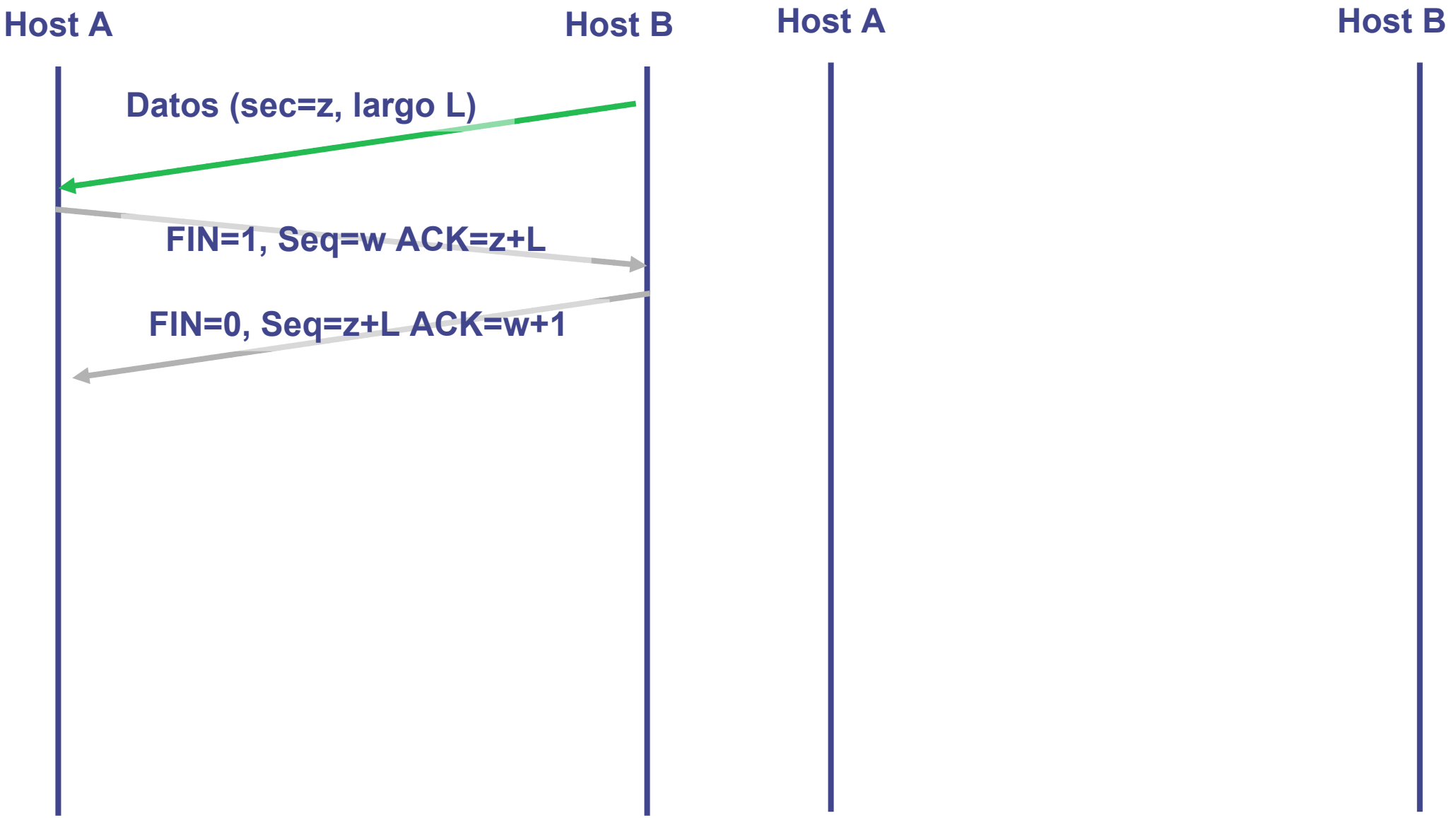
TCP - Fin de Conexión en 4 y 3 vías



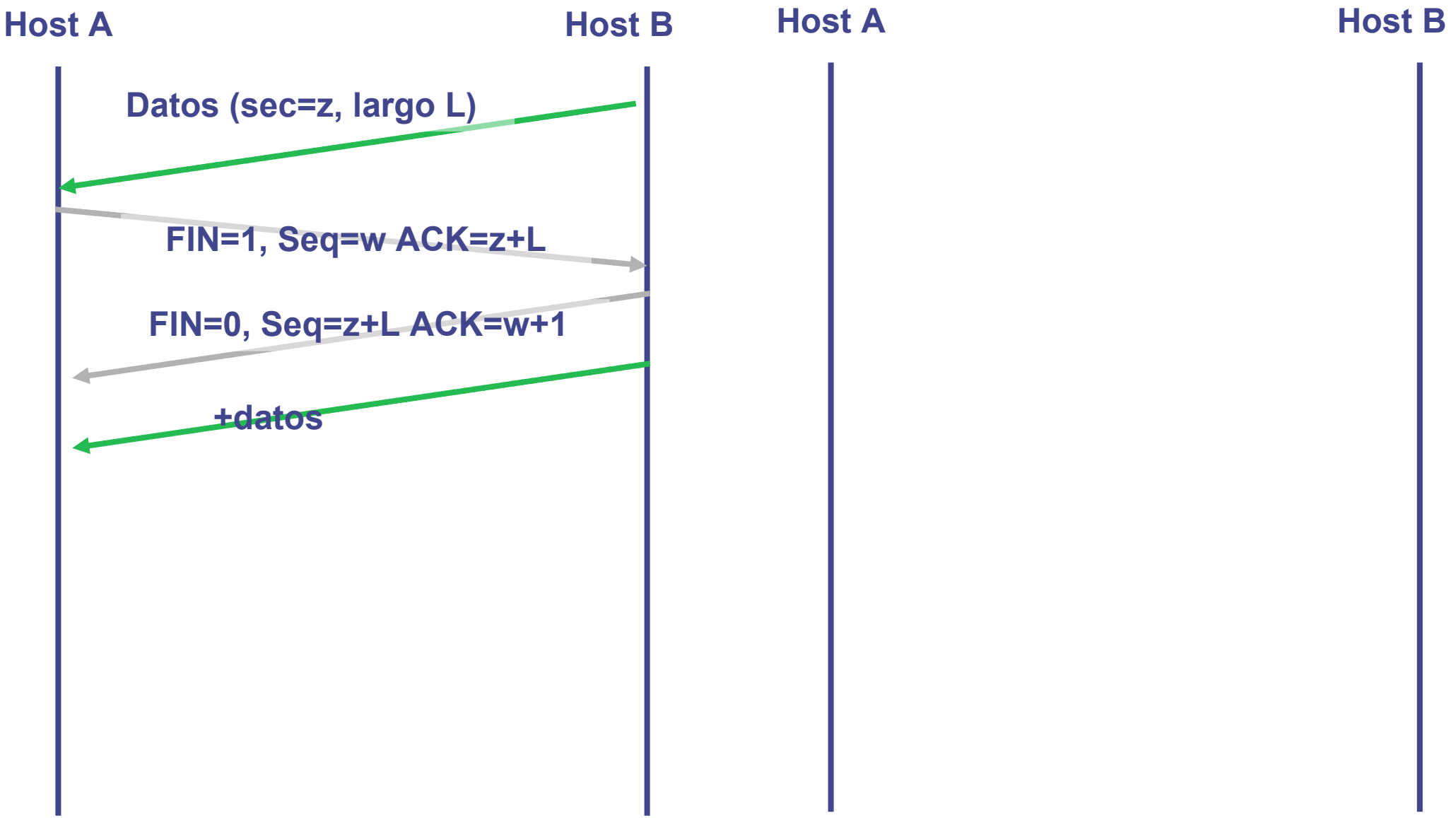
TCP - Fin de Conexión en 4 y 3 vías



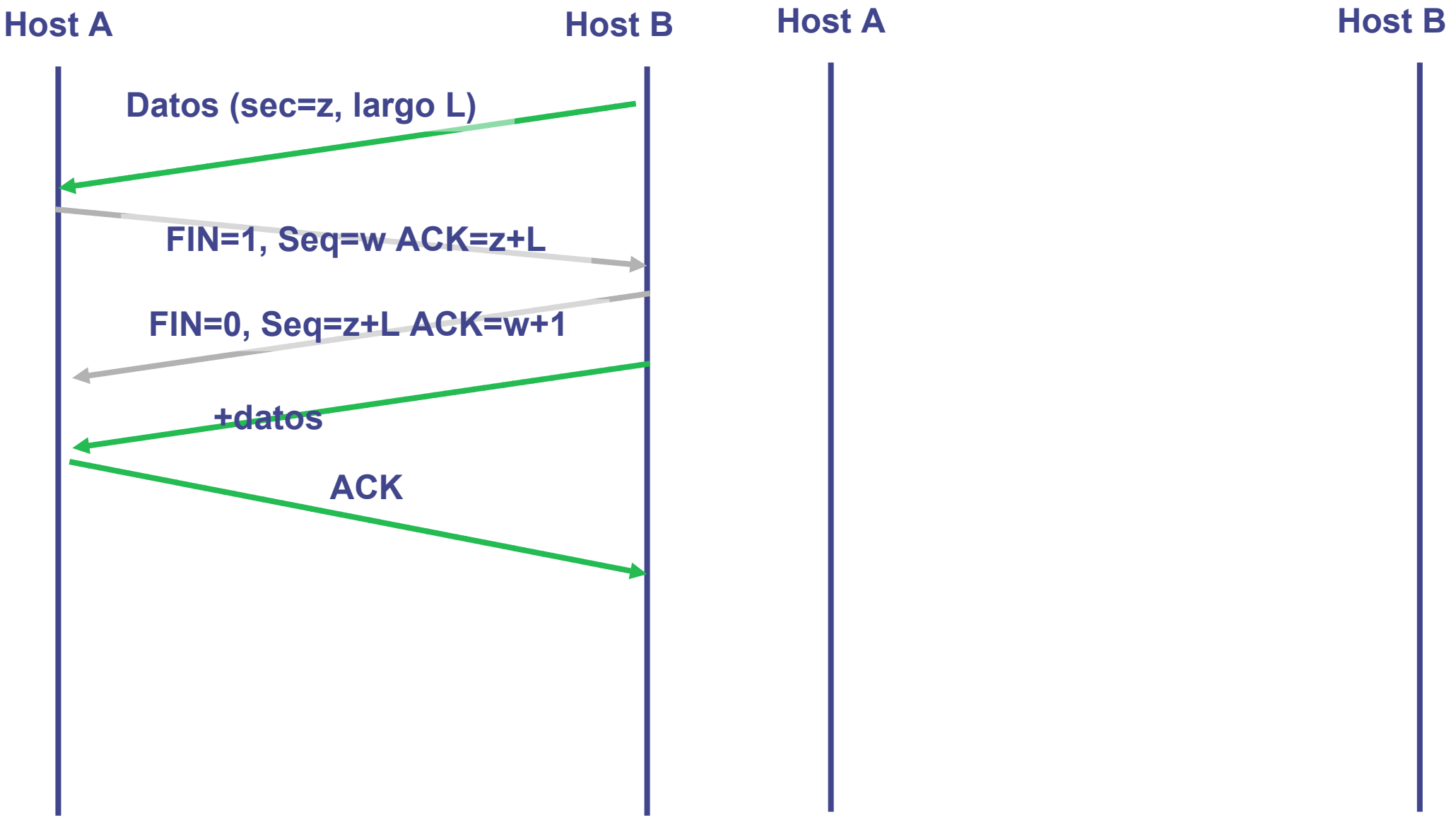
TCP - Fin de Conexión en 4 y 3 vías



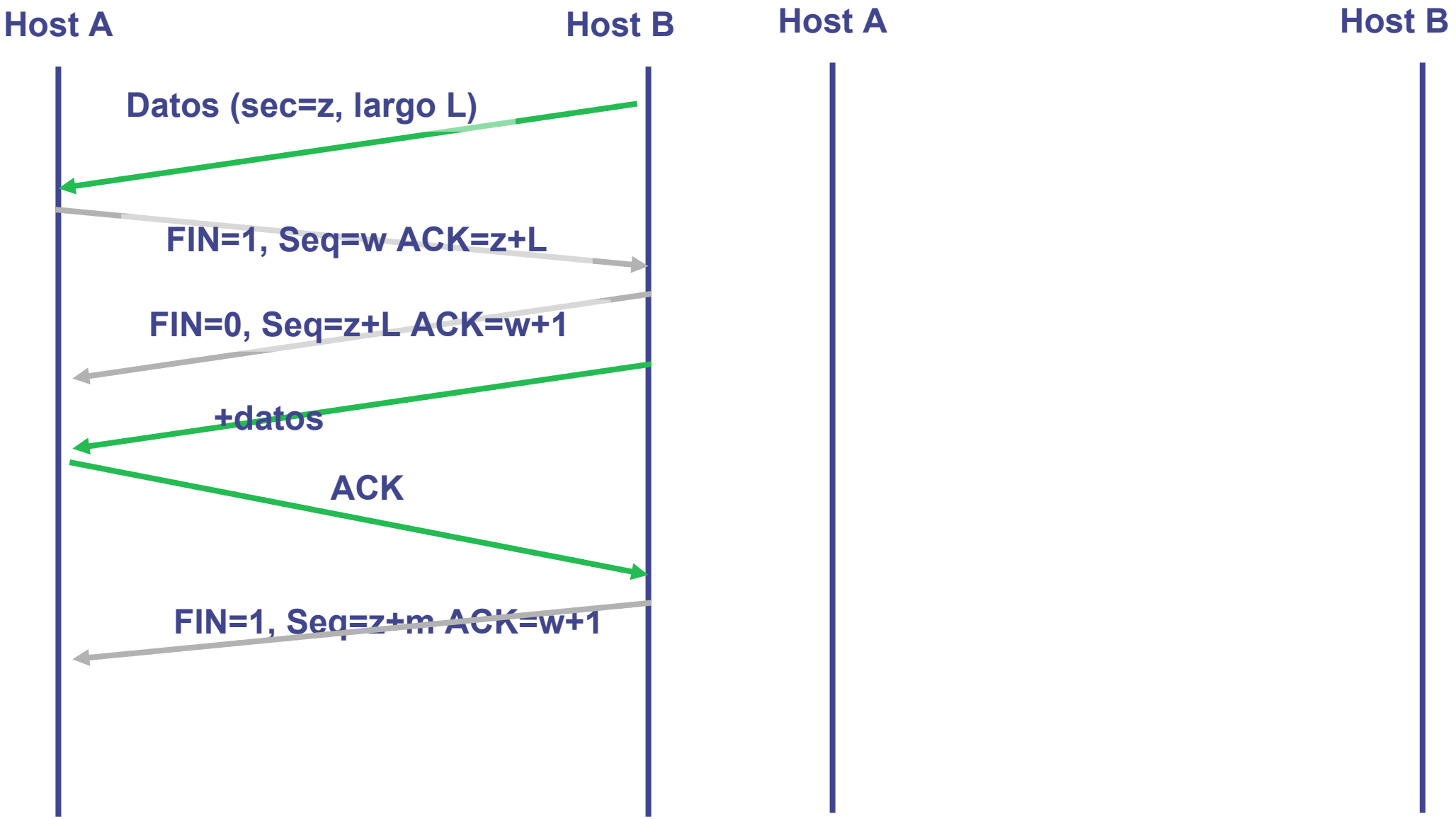
TCP - Fin de Conexión en 4 y 3 vías



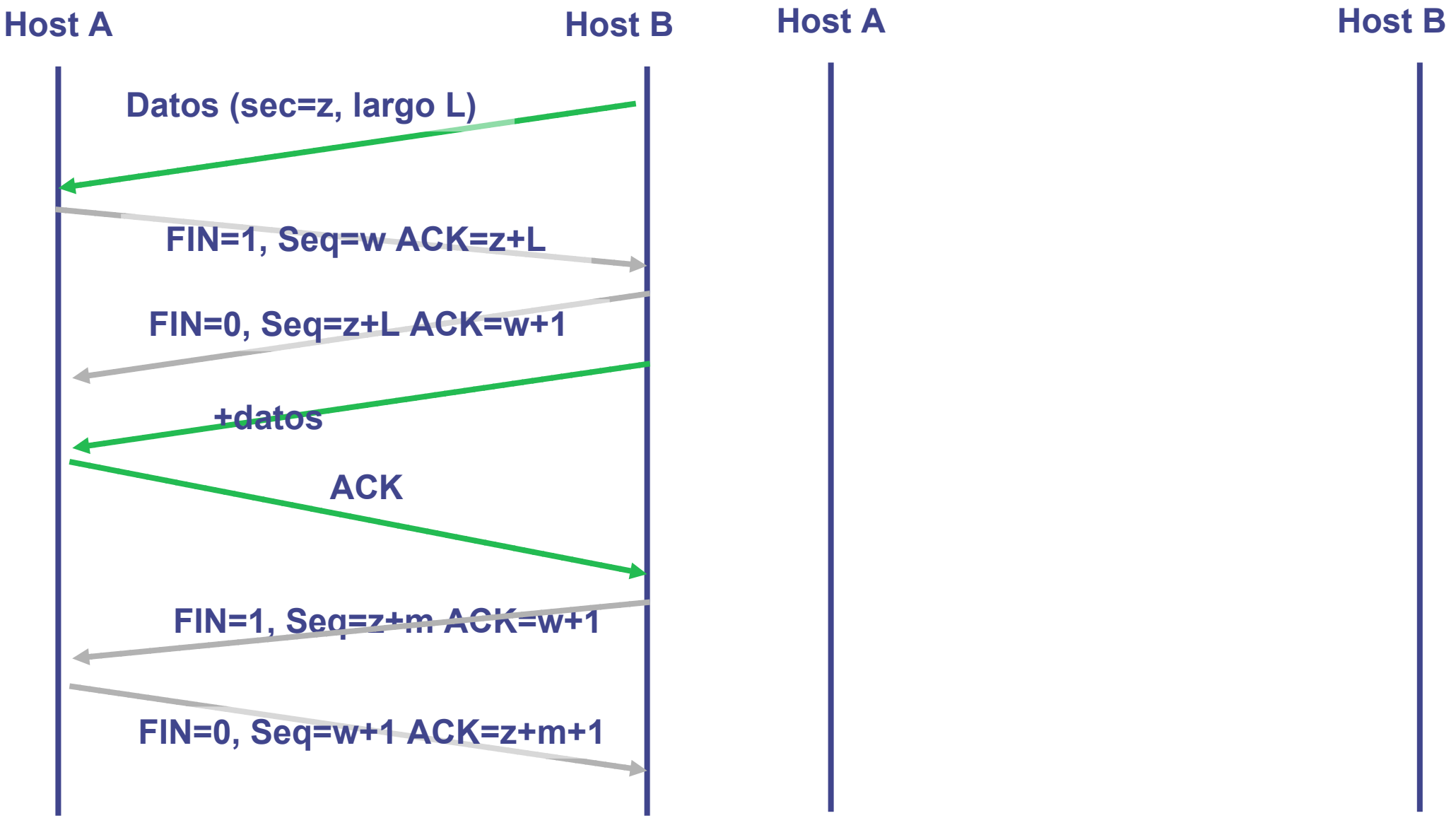
TCP - Fin de Conexión en 4 y 3 vías



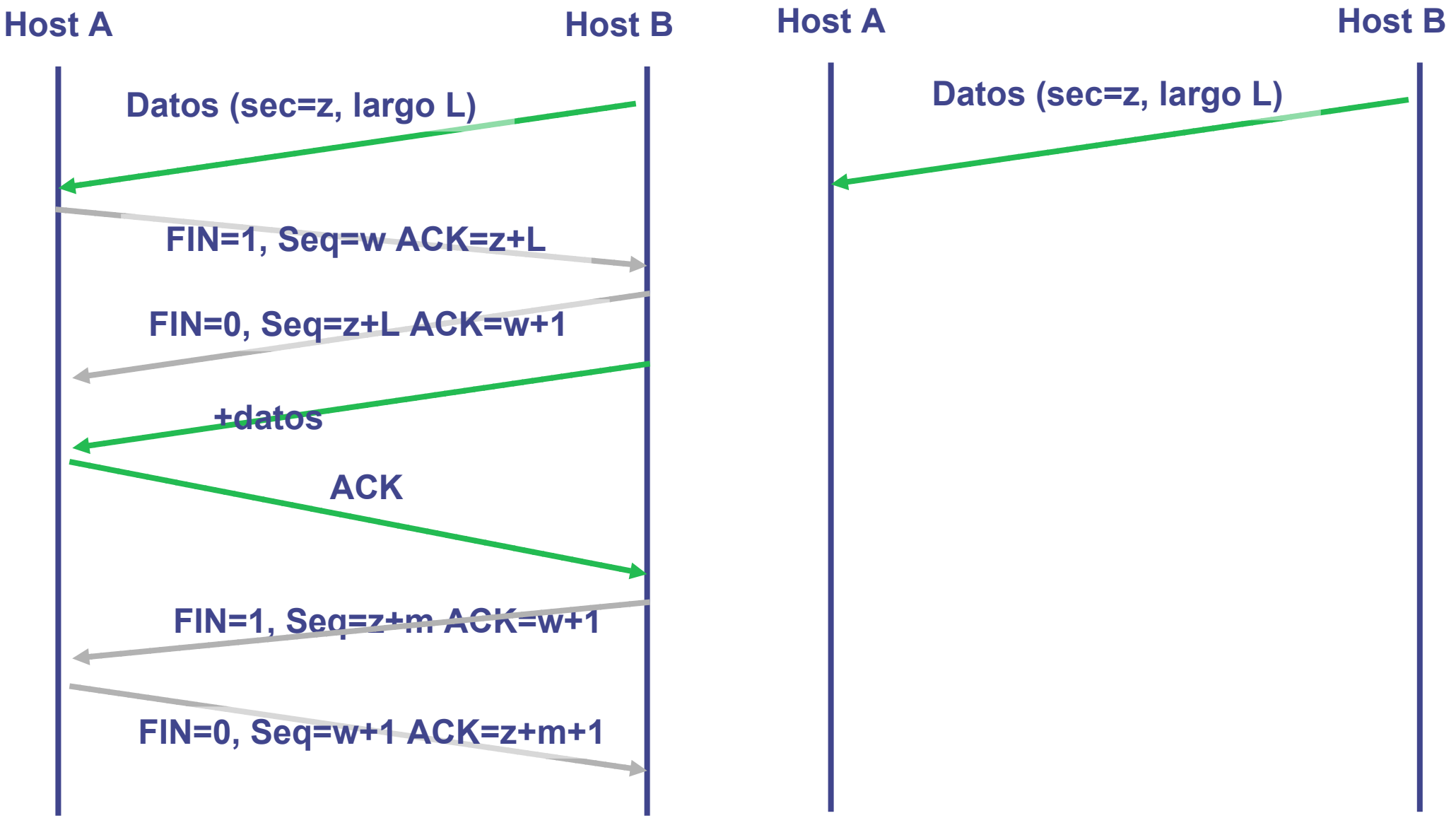
TCP - Fin de Conexión en 4 y 3 vías



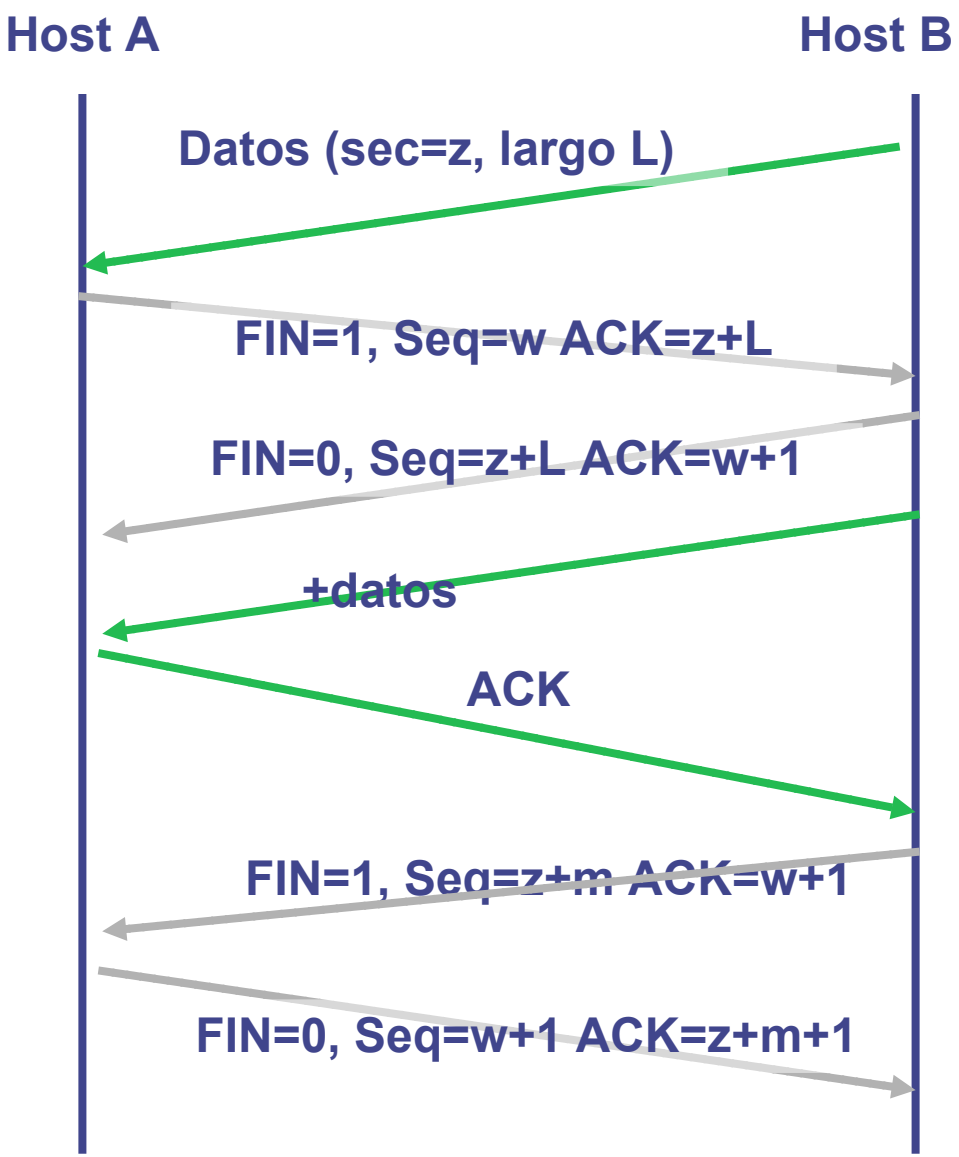
TCP - Fin de Conexión en 4 y 3 vías

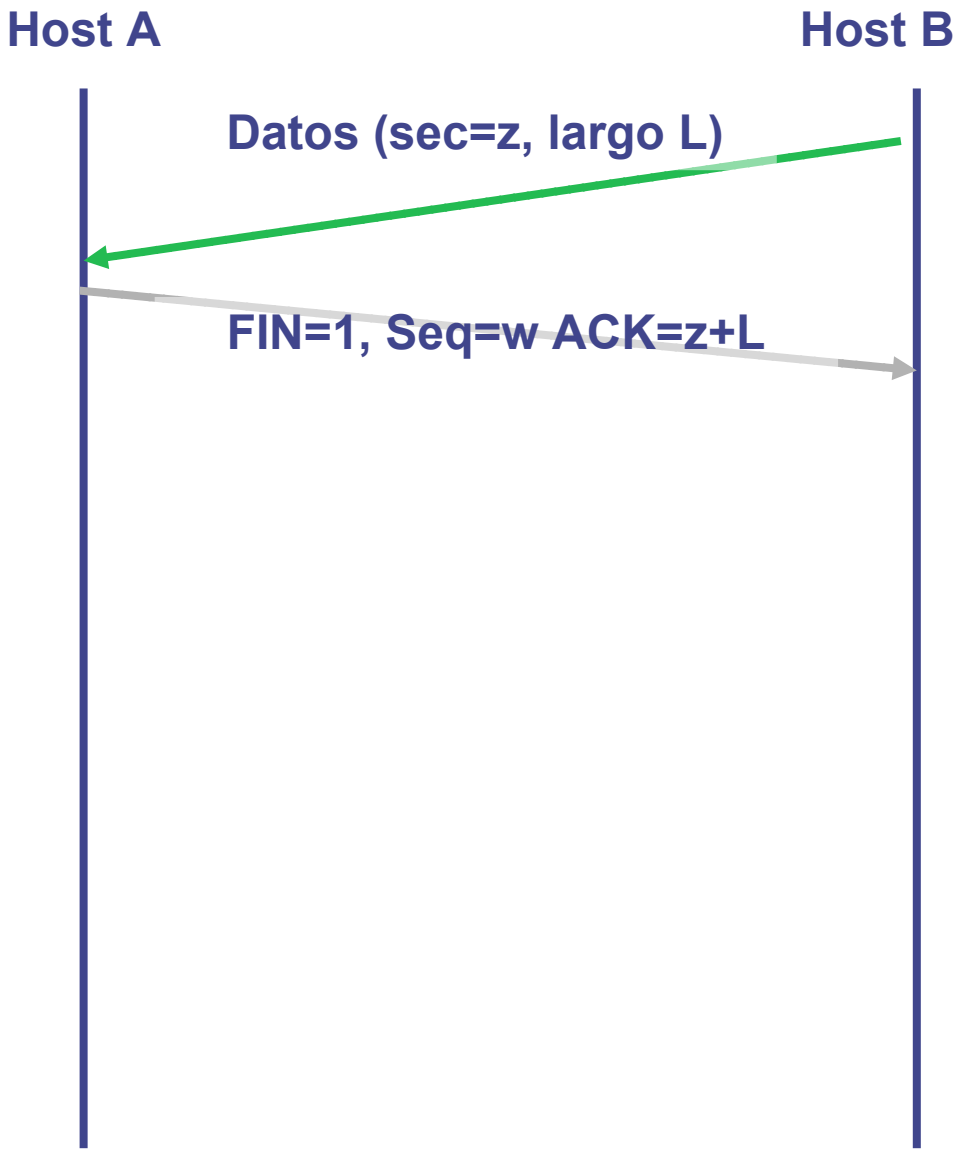


TCP - Fin de Conexión en 4 y 3 vías

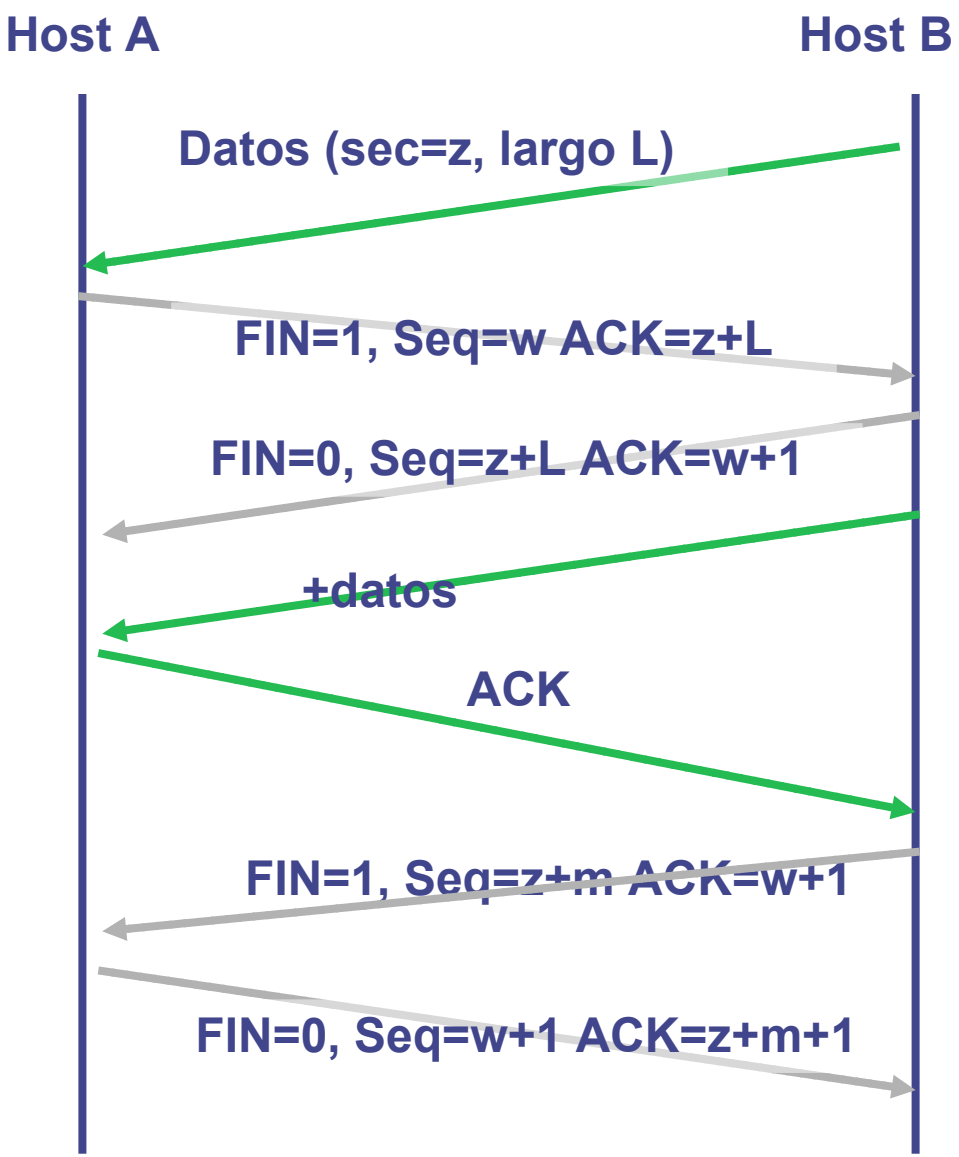


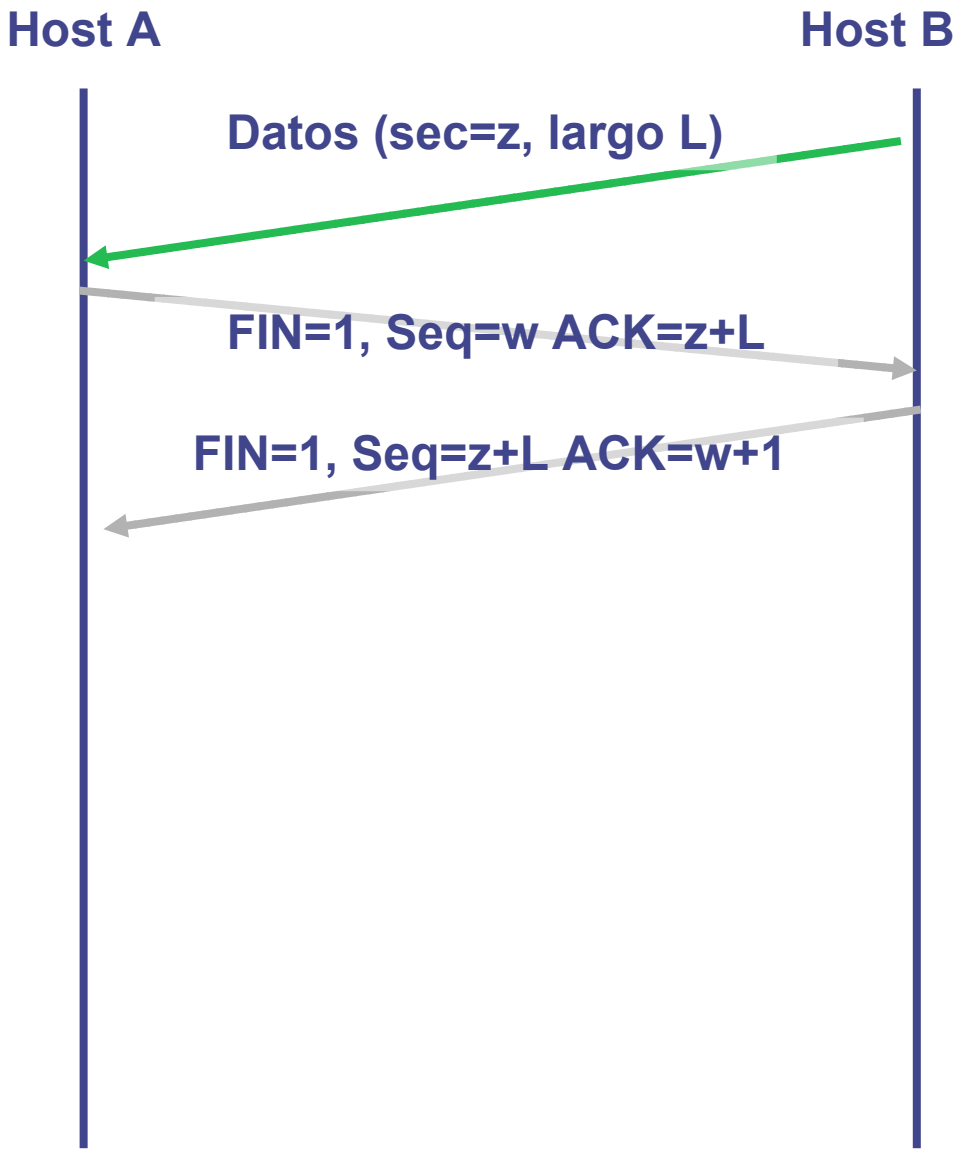
TCP - Fin de Conexión en 4 y 3 vías



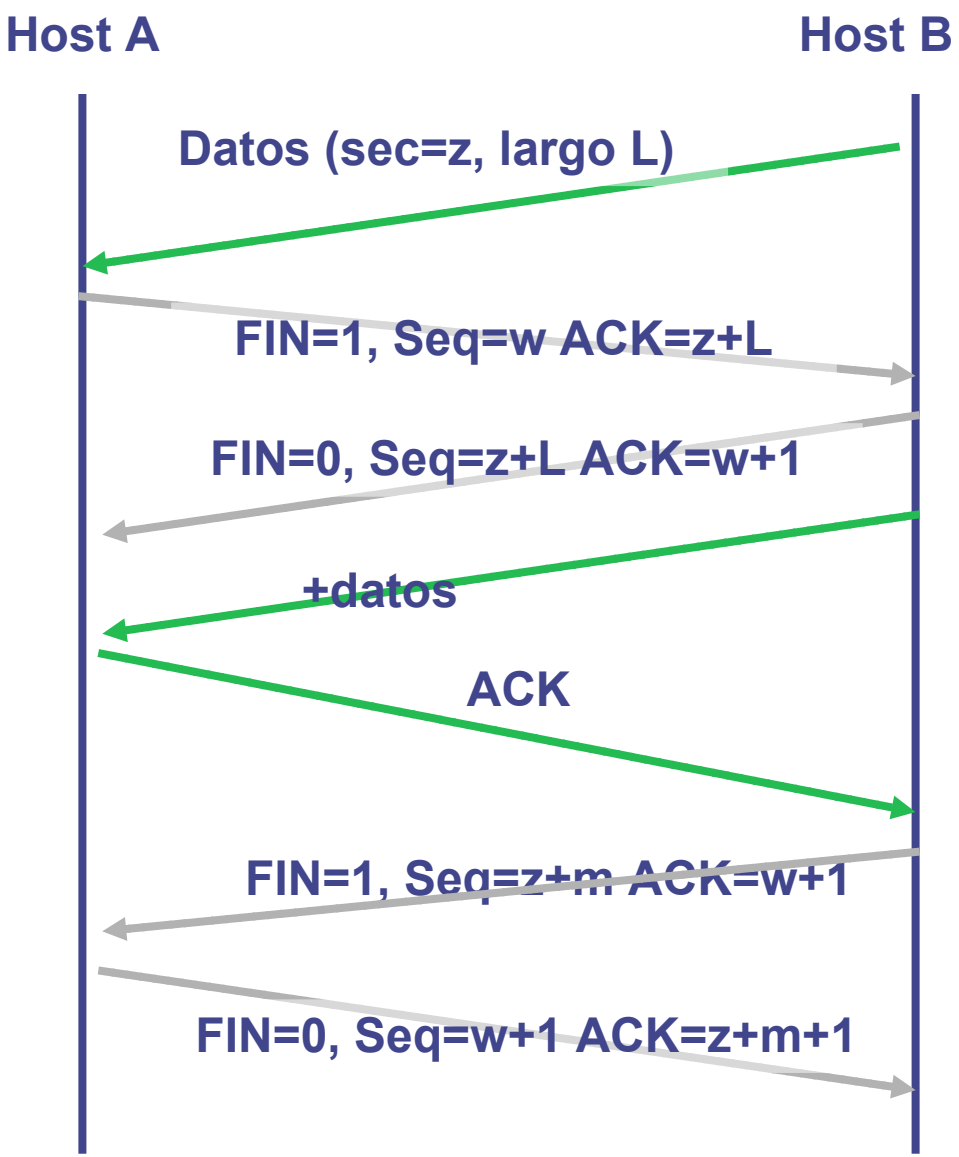


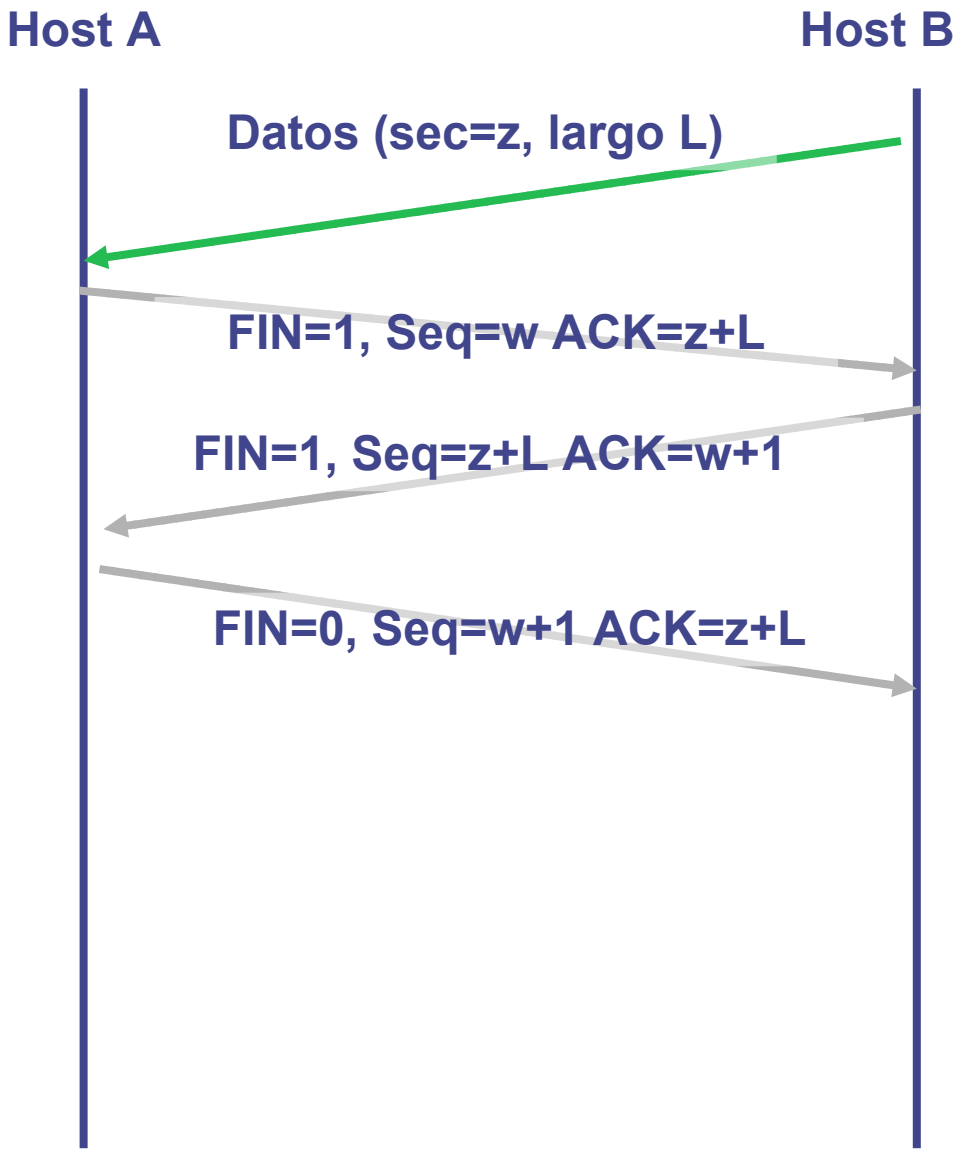
TCP - Fin de Conexión en 4 y 3 vías





TCP - Fin de Conexión en 4 y 3 vías





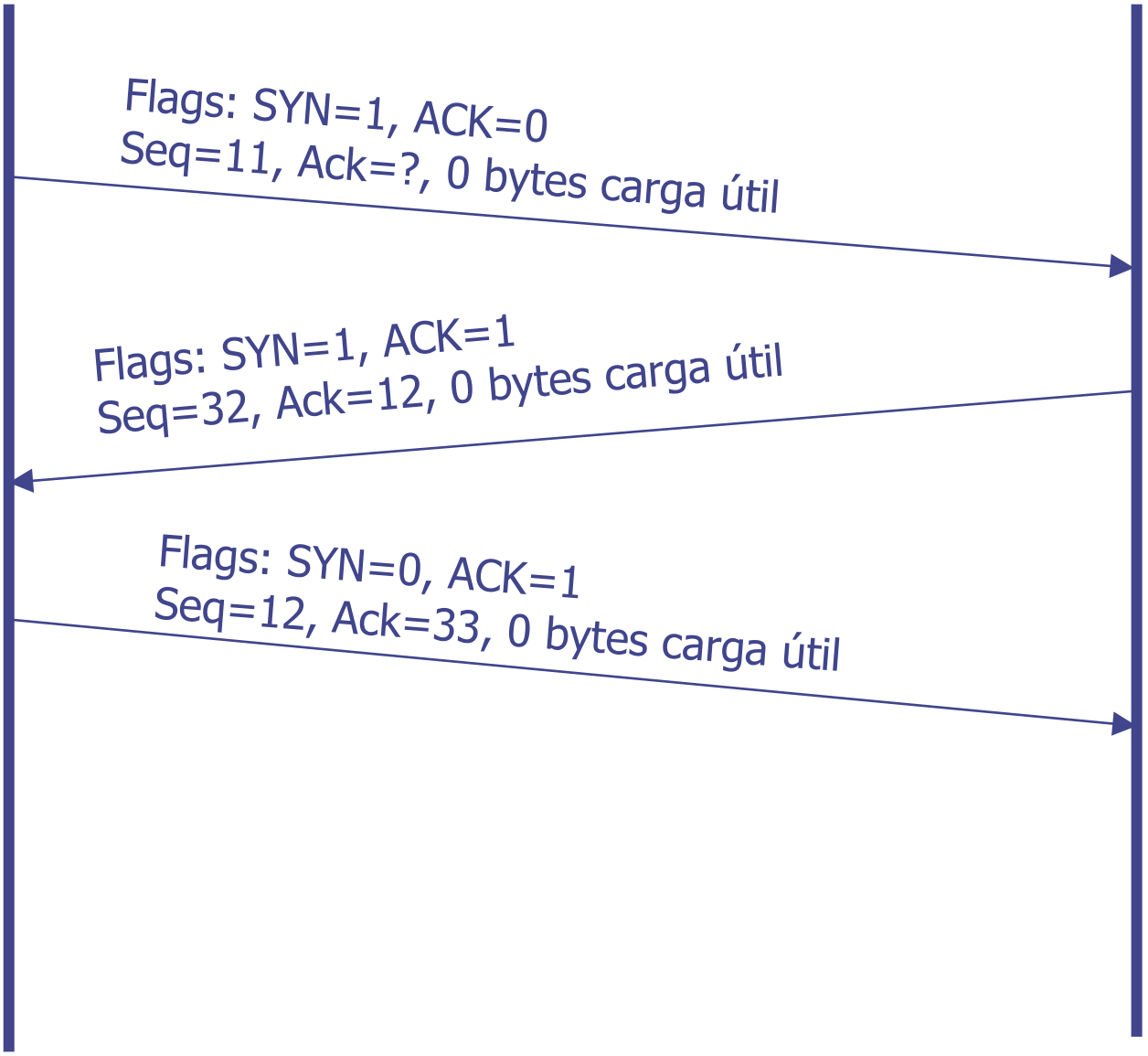
TCP – Números de Secuencia y Reconocimiento

- En TCP se numeran los bytes, no los segmentos
  - Un segmento sin datos no incrementa el número de secuencia
- En el reconocimiento se indica el próximo nº de secuencia esperado
- El establecimiento **(SYN)** y finalización de conexión **(FIN)** consumen 1 número de secuencia cada uno.
- Los reconocimientos son acumulativos **ACK(x)** => reconozco hasta el **x-1** y el siguiente que “espero ver” es el **x**

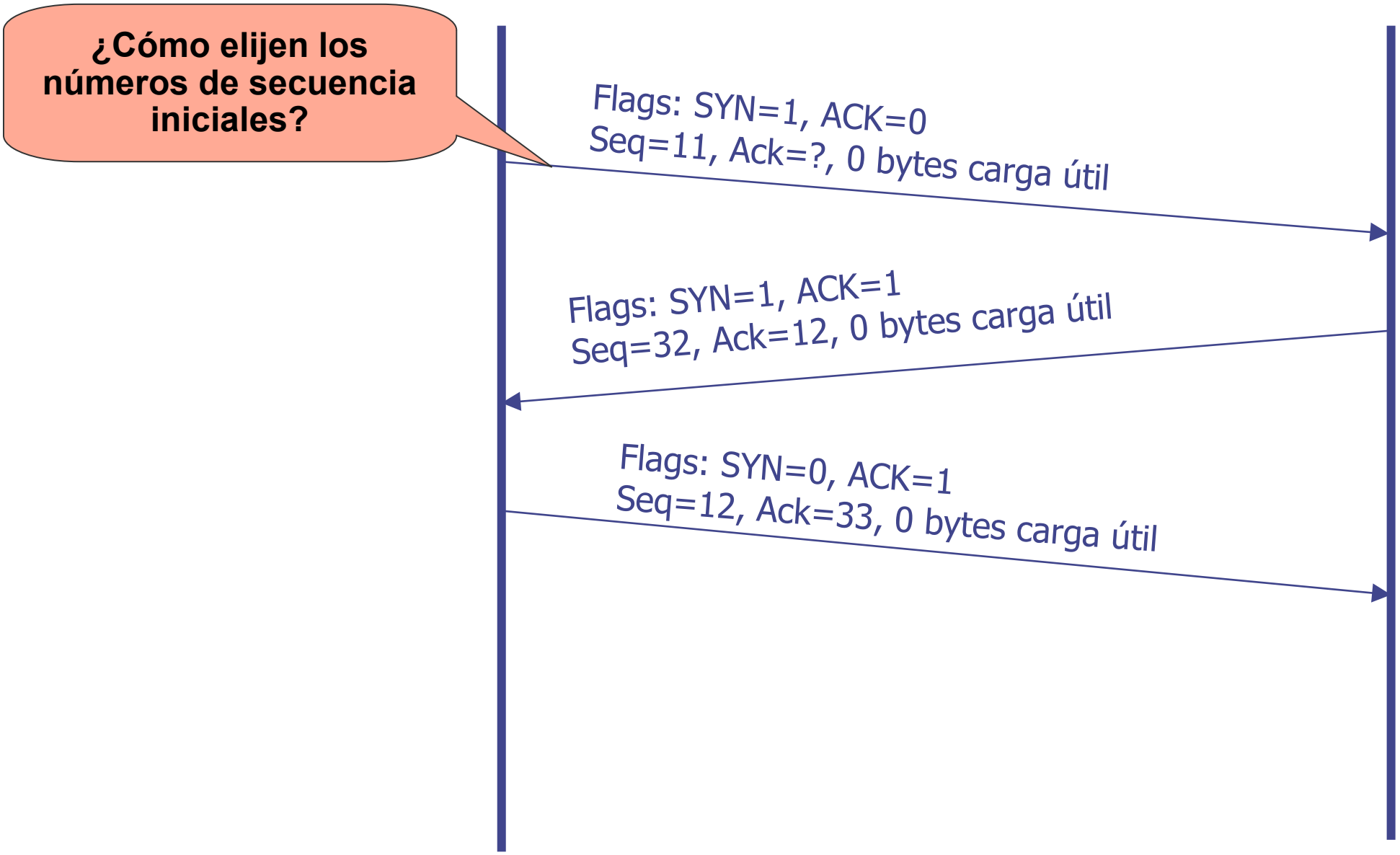
TCP

- Inicio y fin de conexión
- Manejo de números de secuencia
- Control de Flujo
- Control de Congestión
- Estados y temporizadores

TCP – Ejemplo de Establecimiento de Conexión



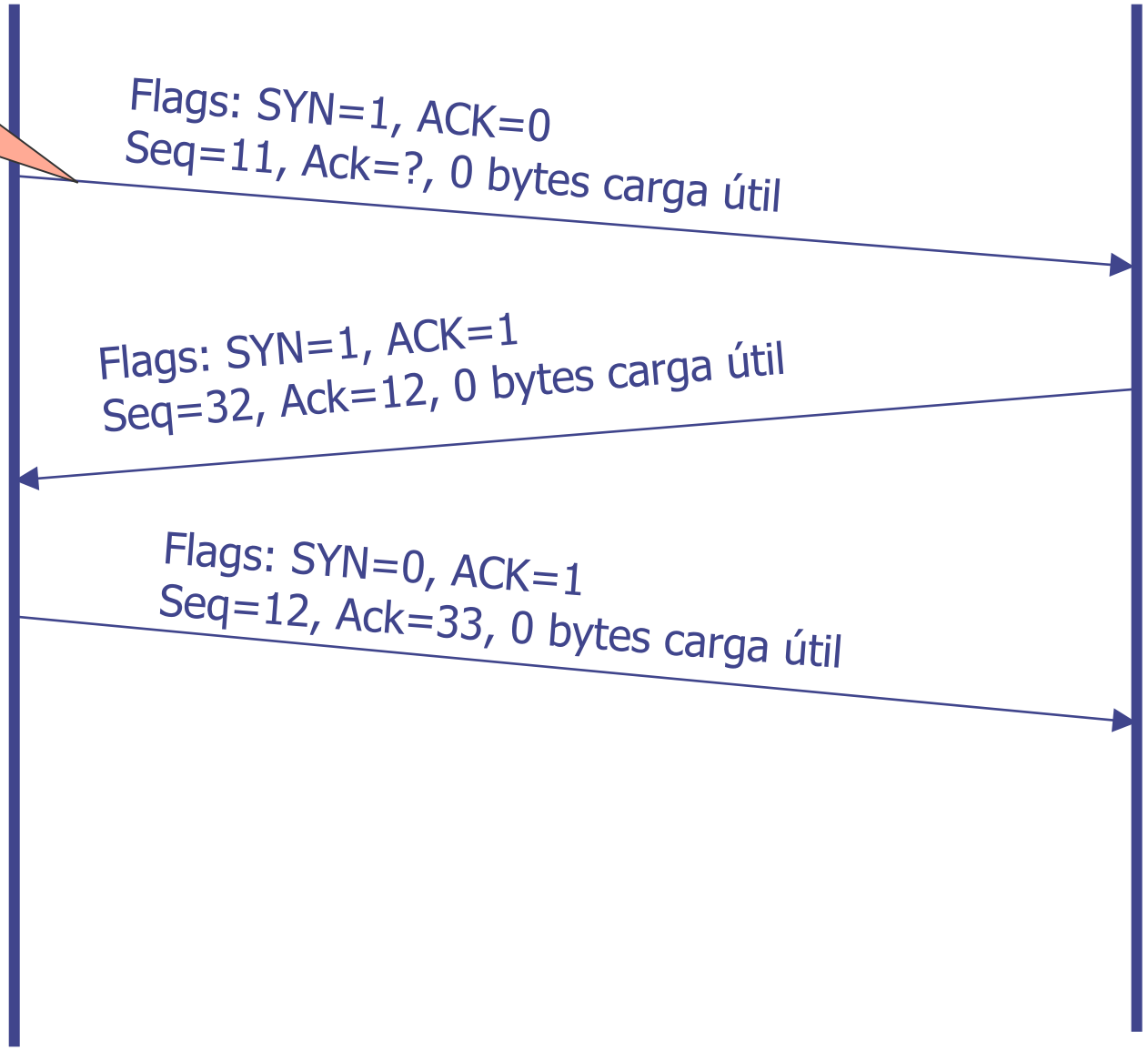
TCP – Ejemplo de Establecimiento de Conexión



TCP – Ejemplo de Establecimiento de Conexión

¿Cómo elijen los números de secuencia iniciales?

Si comienzan siempre desde cero la secuencia es predecible. Permite realizar ataques de Denegación de Servicio (DoS)

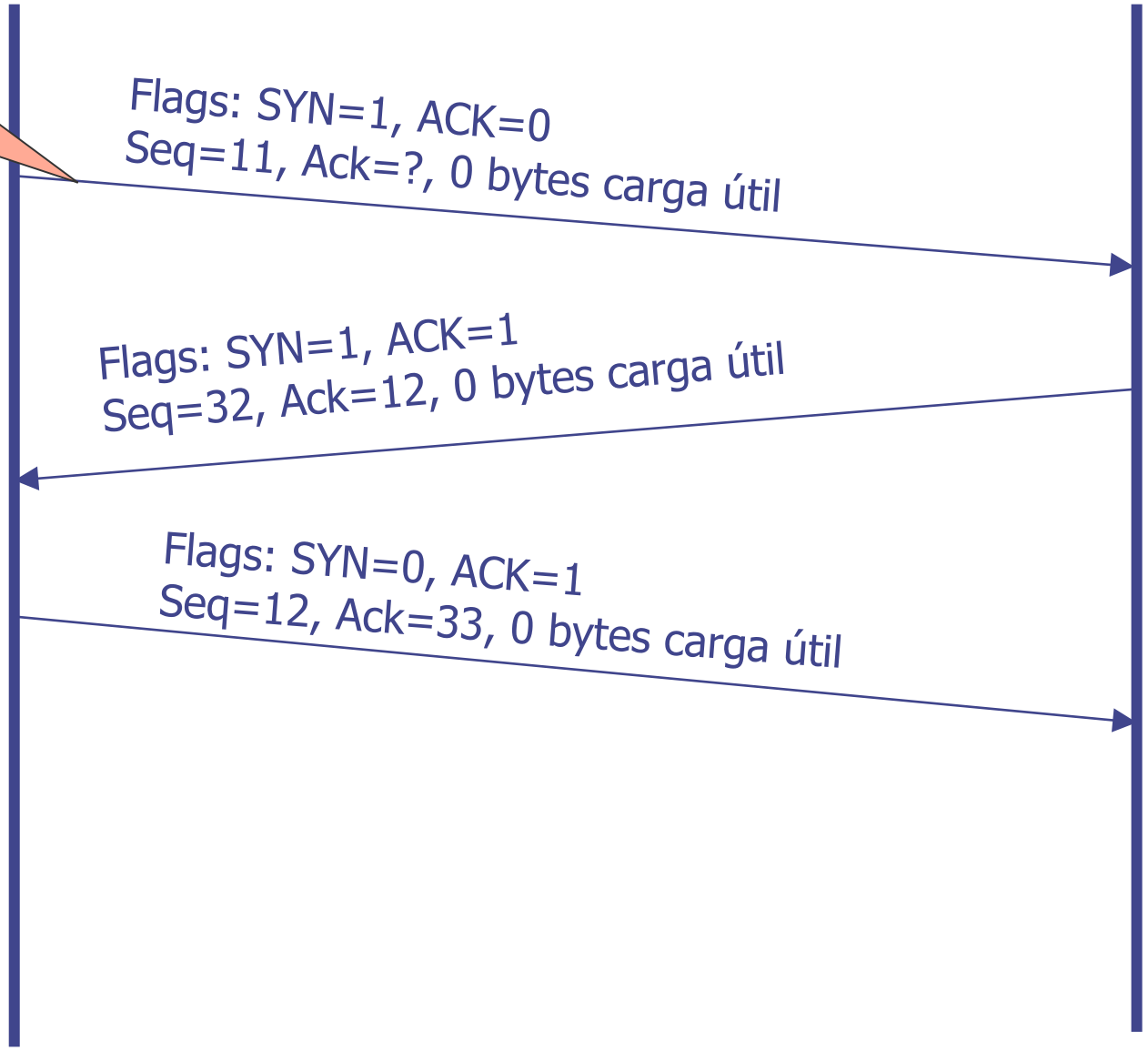


TCP – Ejemplo de Establecimiento de Conexión

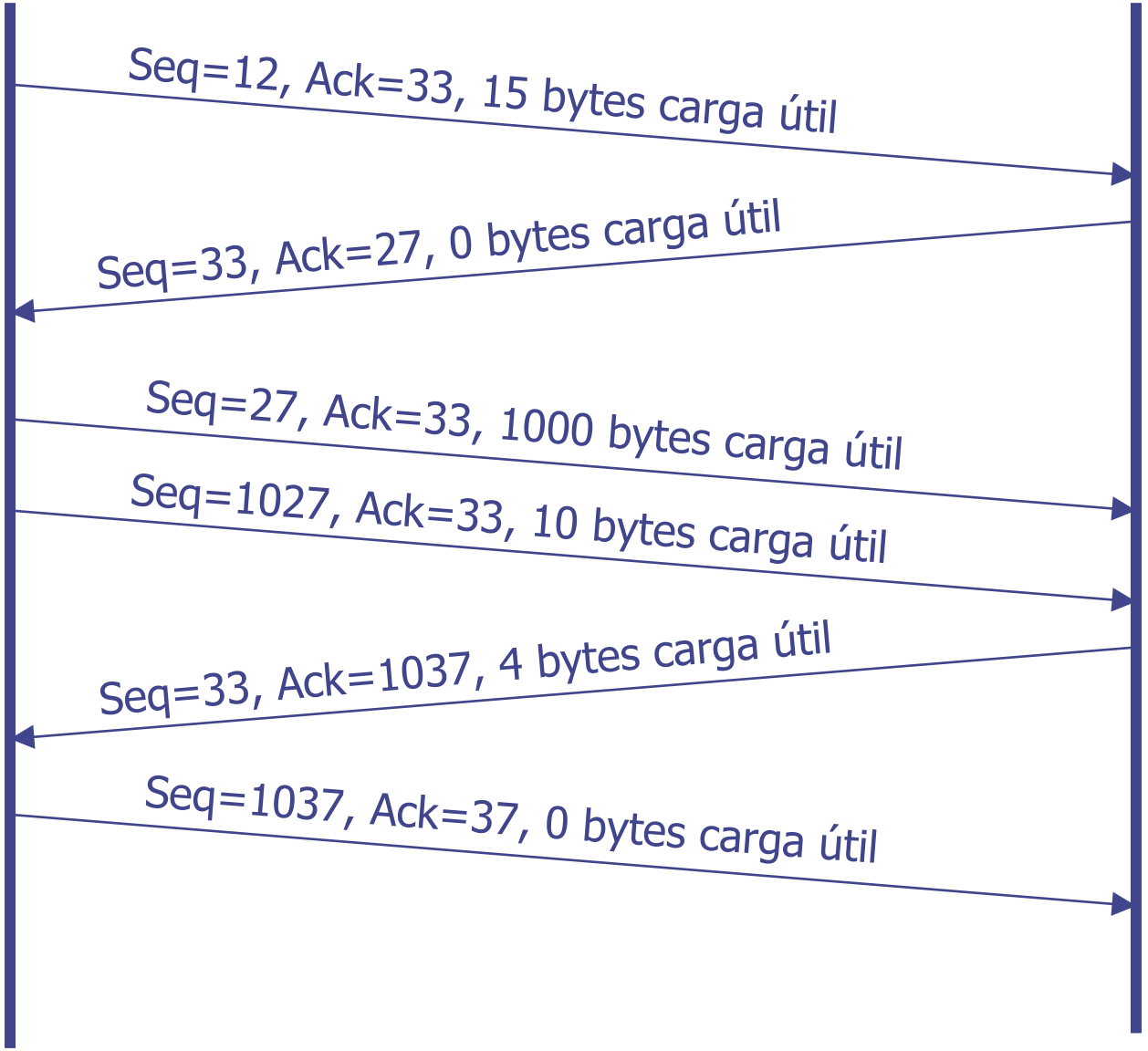
¿Cómo elijen los números de secuencia iniciales?

Si comienzan siempre desde cero la secuencia es predecible. Permite realizar ataques de Denegación de Servicio (DoS)

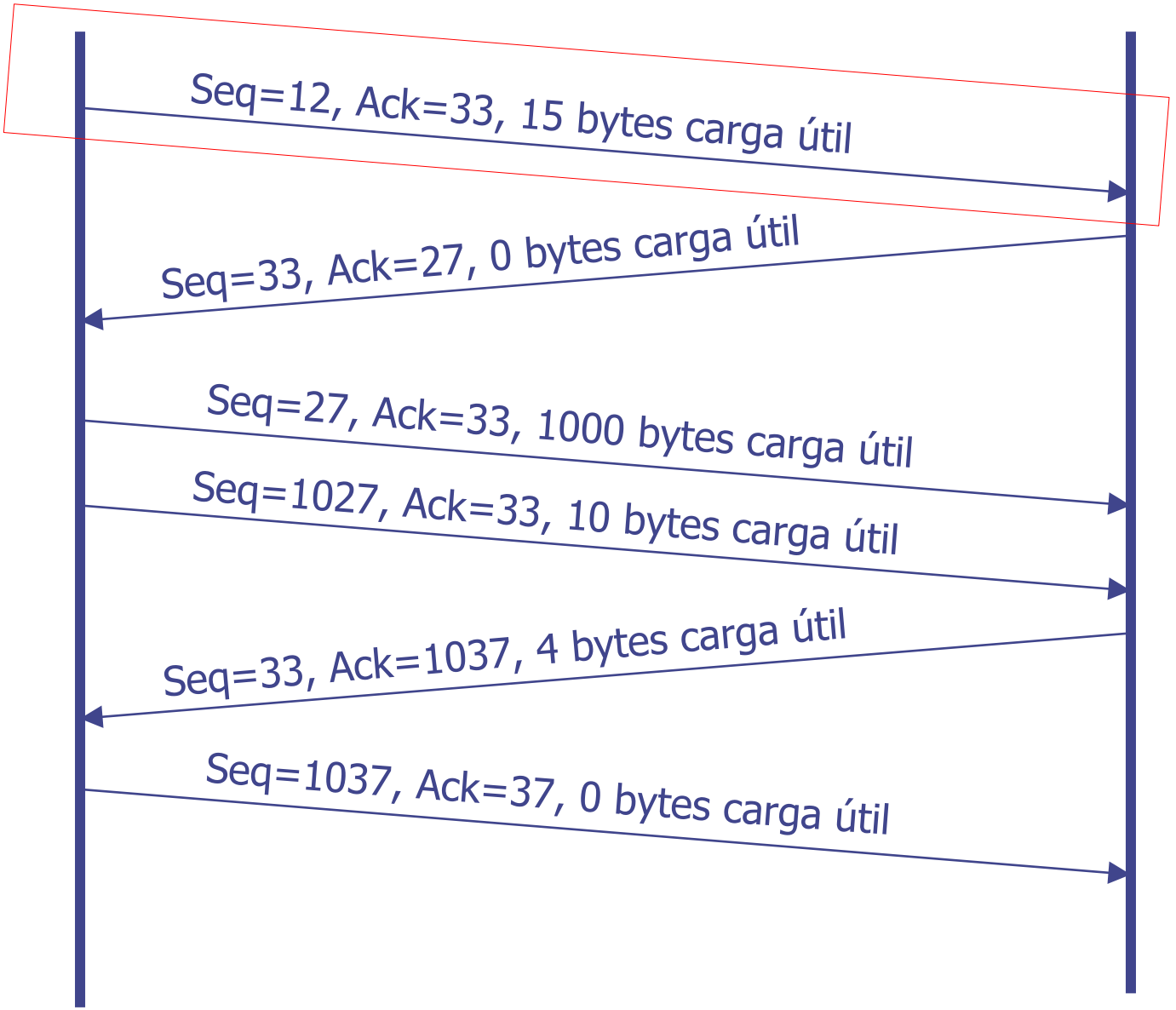
Los números iniciales en cada extremo de elijen de forma aleatoria e independientemente en extremos



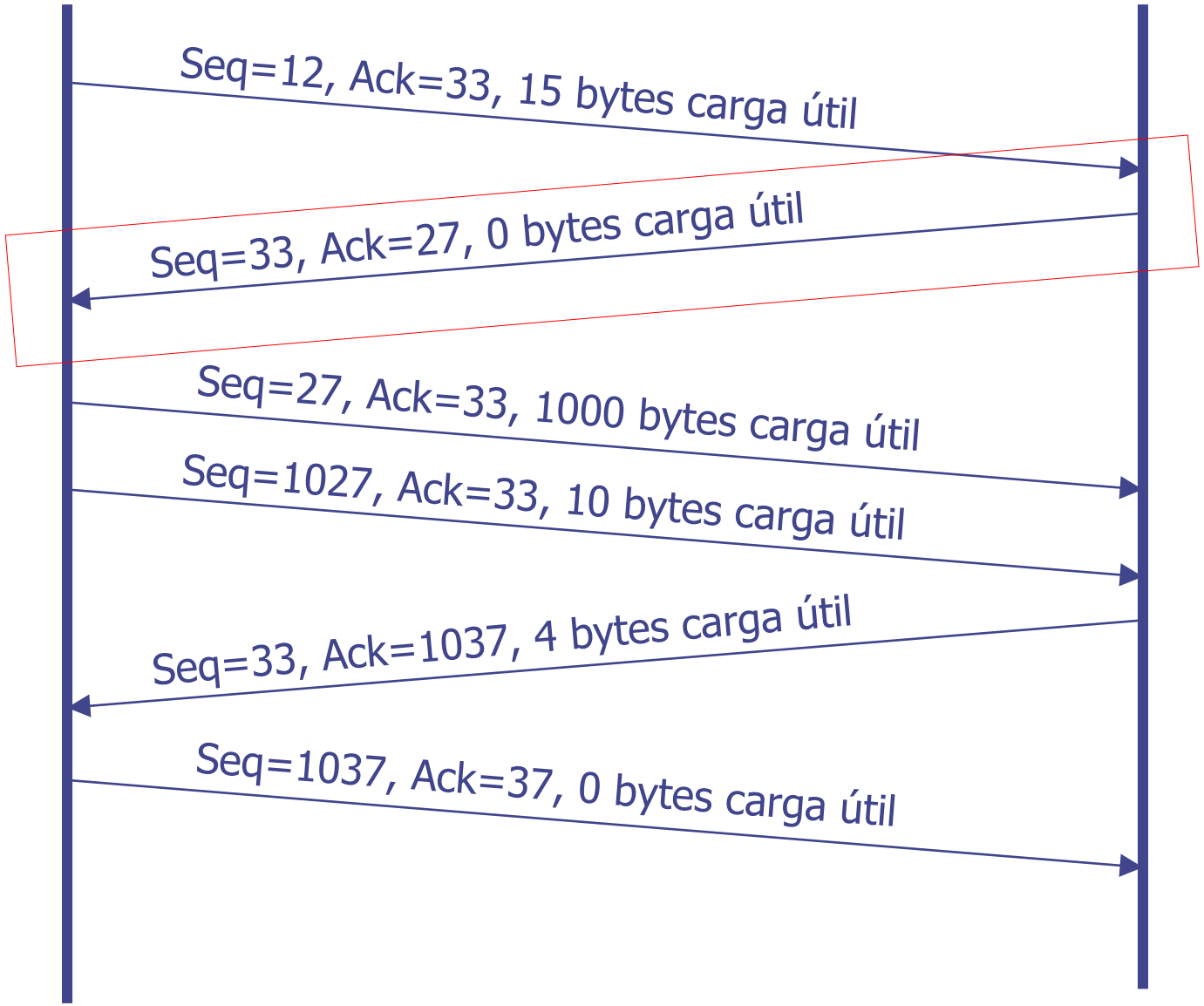
TCP – Ejemplo de Transmisión de Datos



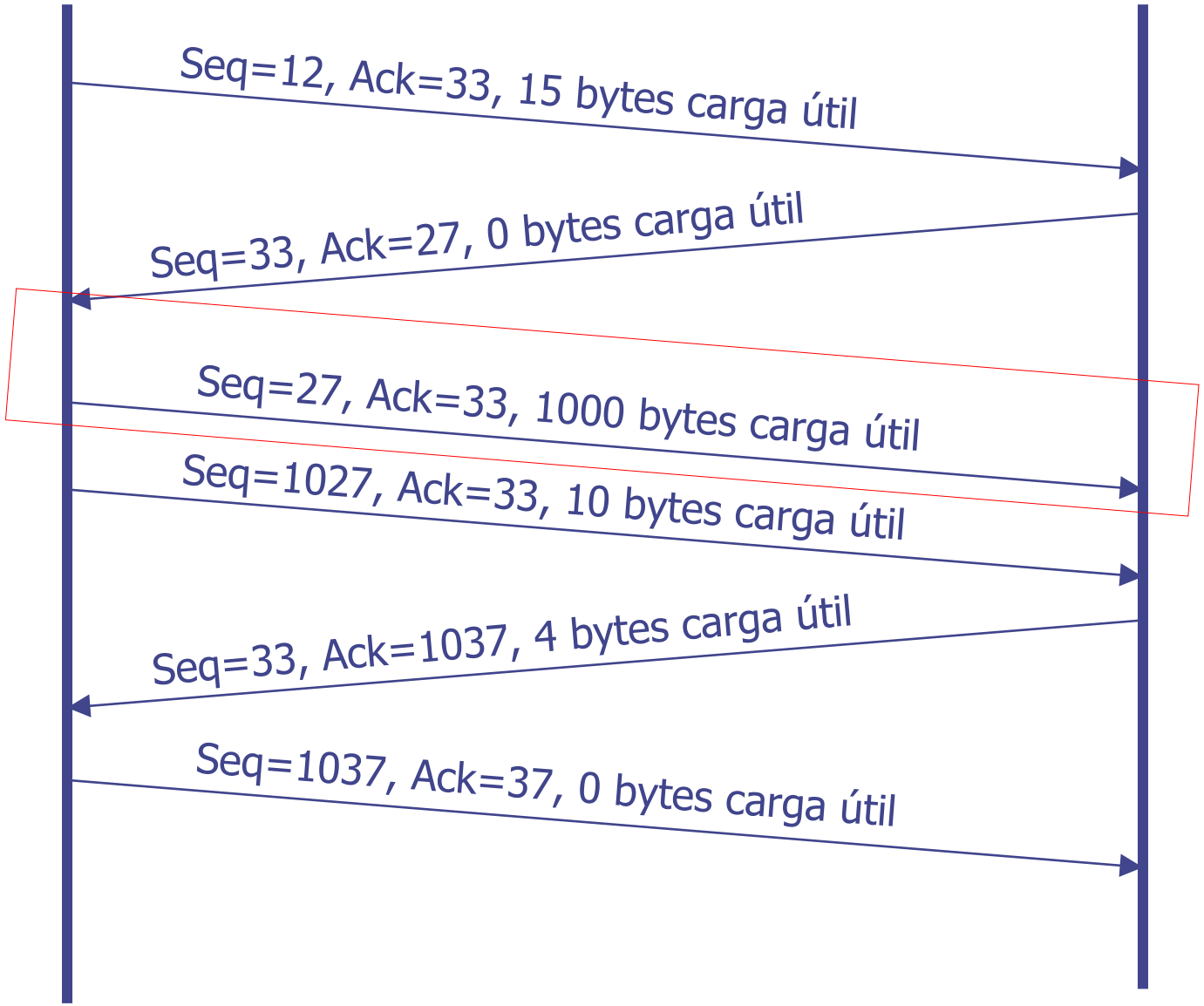
TCP – Ejemplo de Transmisión de Datos



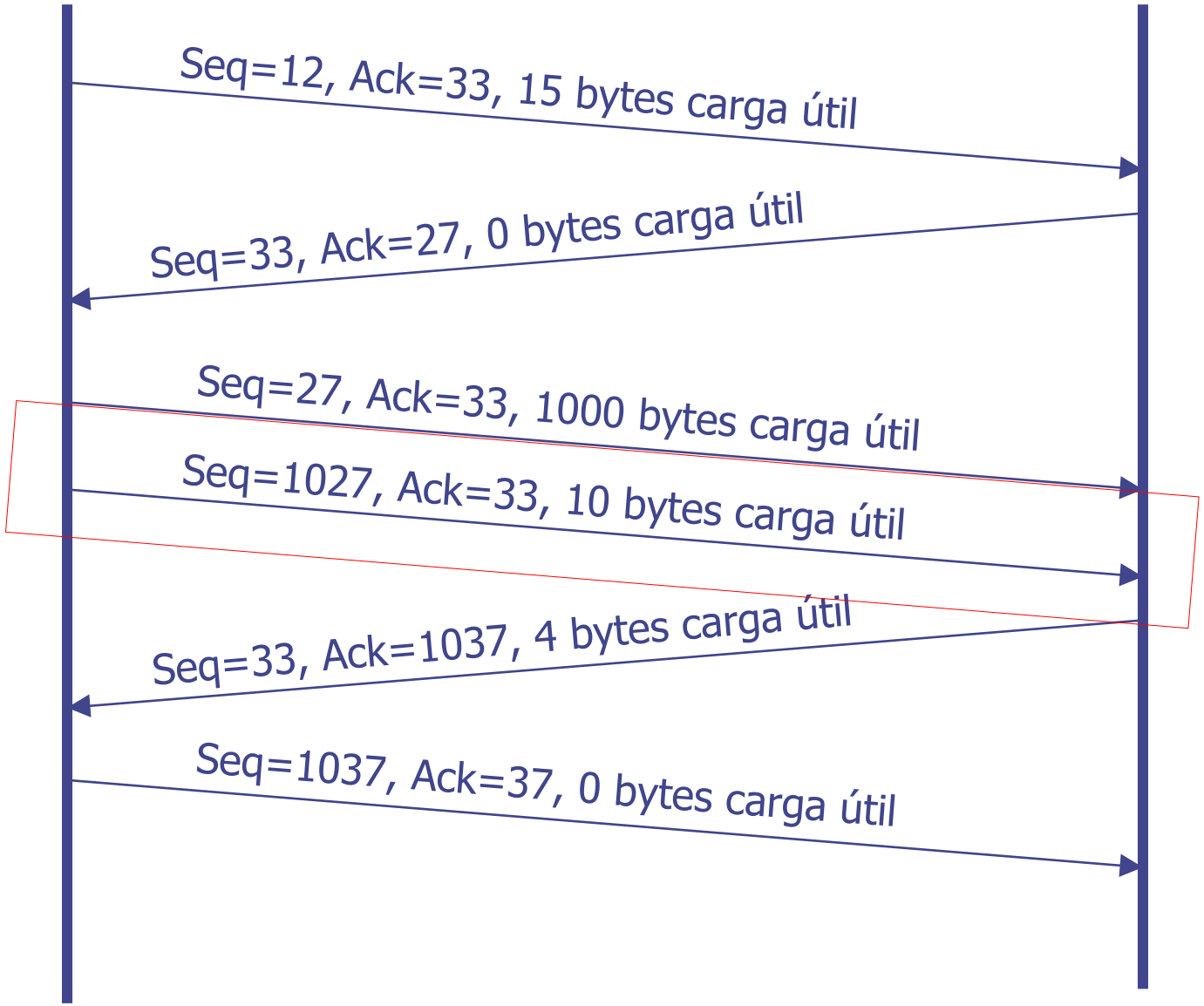
TCP – Ejemplo de Transmisión de Datos



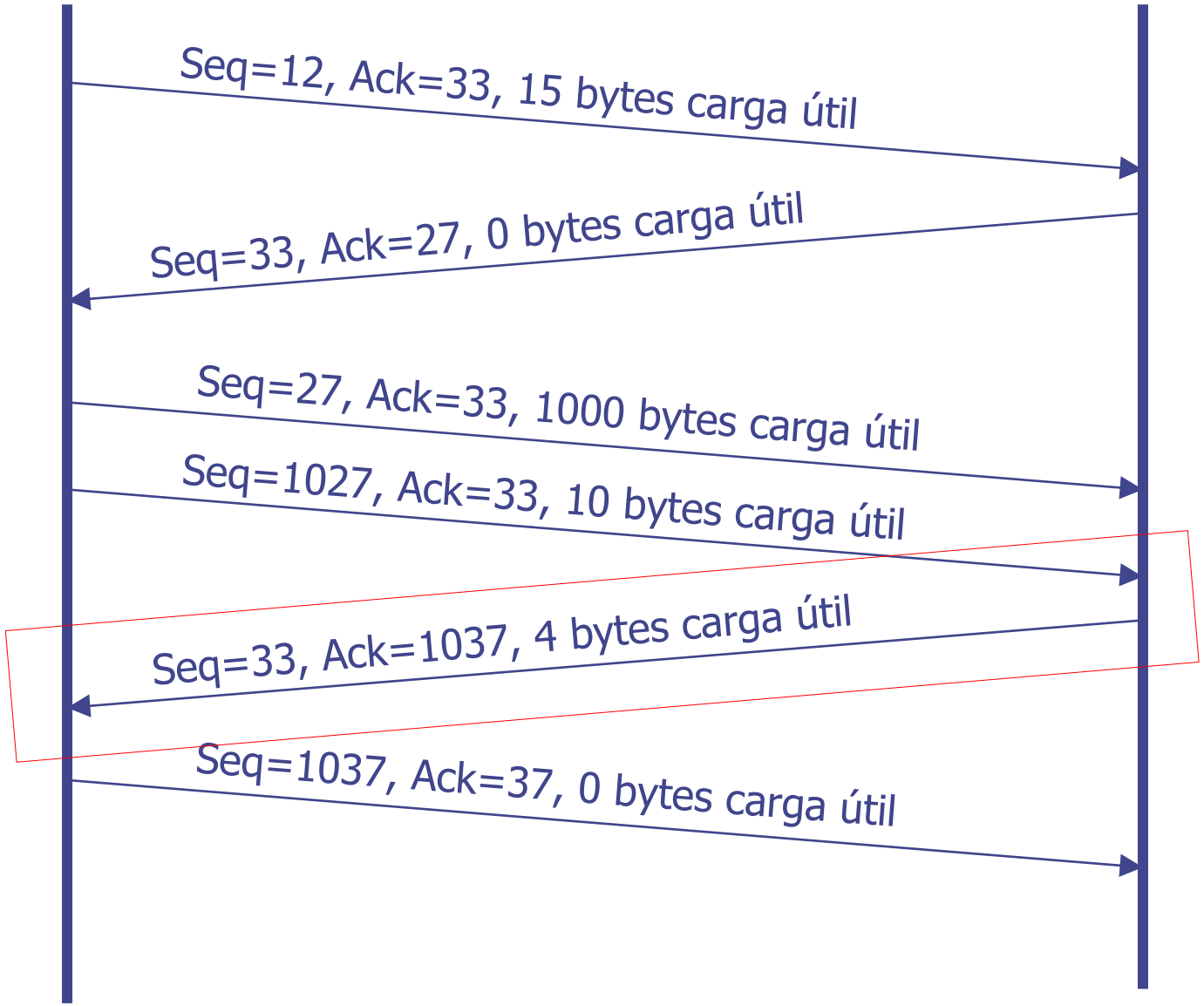
TCP – Ejemplo de Transmisión de Datos



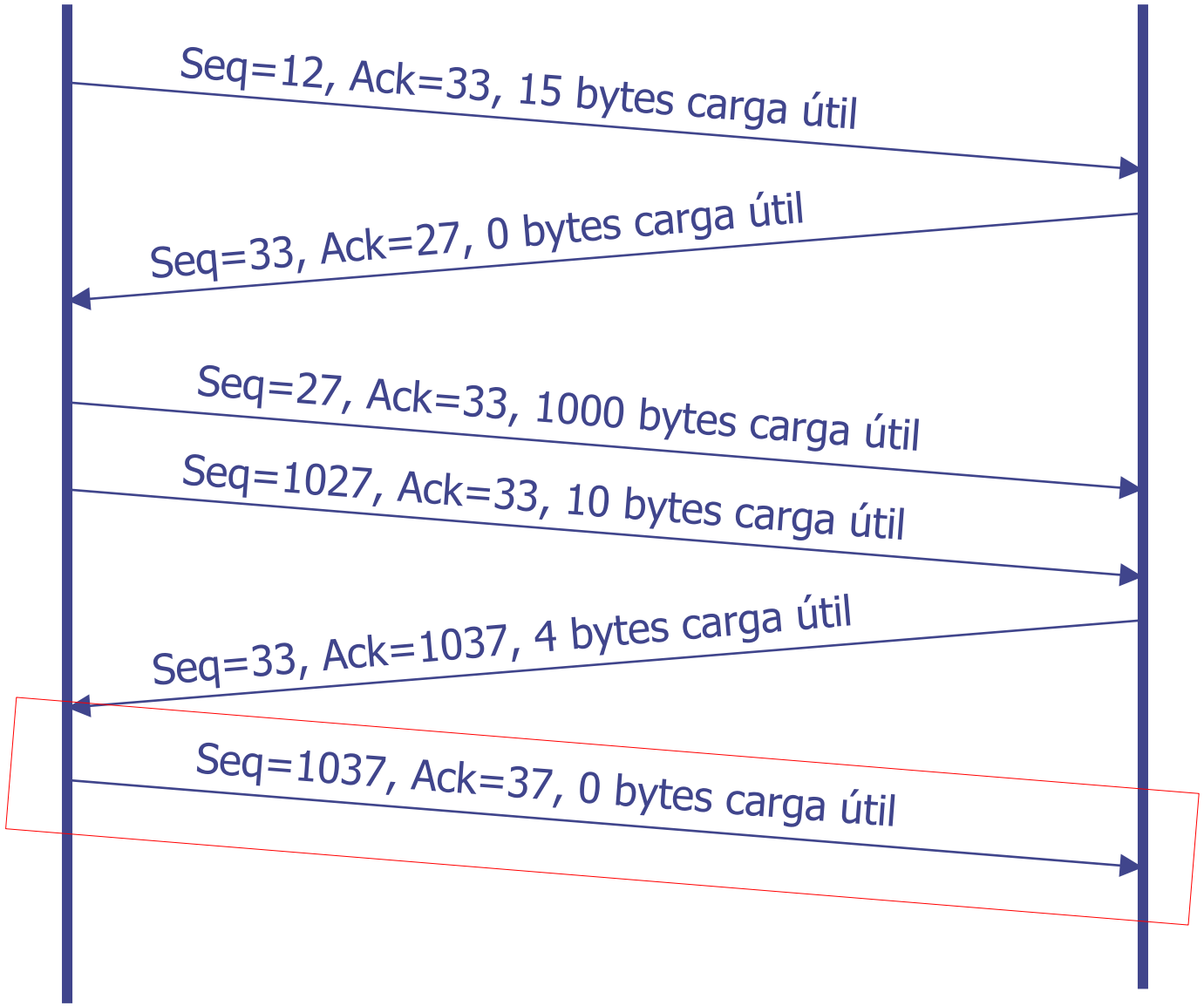
TCP – Ejemplo de Transmisión de Datos



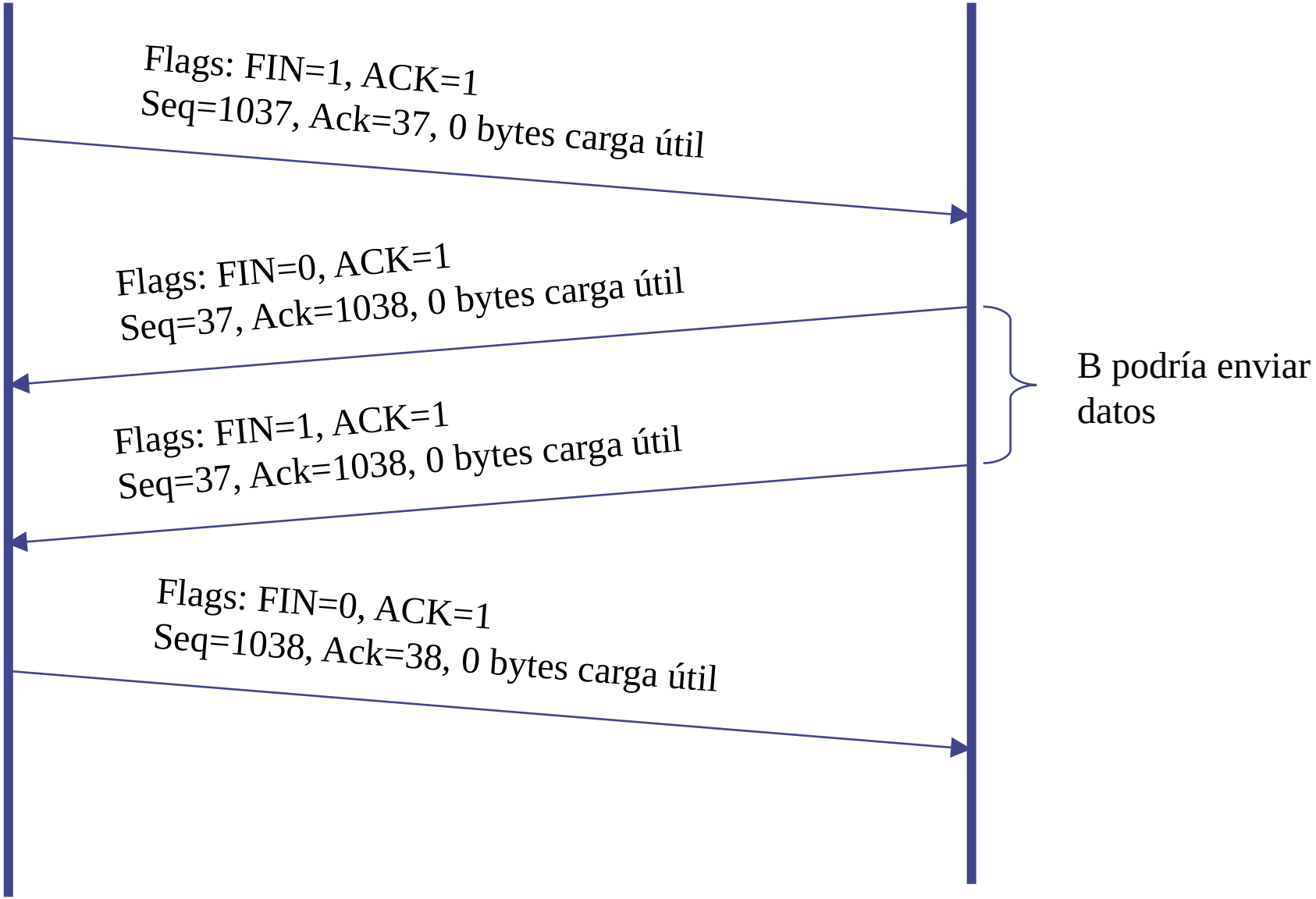
TCP – Ejemplo de Transmisión de Datos



TCP – Ejemplo de Transmisión de Datos



TCP – Ejemplo de Fin de Conexión Simétrico



TCP – Uso de banderas

- Bandera de SYN
  - Solamente está en 1 en los primeros dos segmentos de la conexión:
    - El segmento de solicitud de conexión de A
    - El segmento de respuesta de B
  - En todos los demás segmentos está en 0
- Bandera de FIN
  - Solamente está en 1 cuando cada extremo desea cortar su sentido de la conexión (no quiere enviar más datos al otro)
- Bandera de ACK
  - Solamente está apagada en el primer segmento de la conexión (no hay nada previo para reconocer)
  - En todos los restantes segmentos está en 1 ya que siempre estaré reconociendo cuál es el próximo número de secuencia esperado

TCP

- Inicio y fin de conexión
- Manejo de números de secuencia
- Control de Flujo
- Control de Congestión
- Estados y temporizadores

TCP - Control de Flujo

- **El objetivo del control de flujo es adaptar la velocidad de envío a la capacidad del receptor.**
  Evitar que un transmisor “rápido” sature a un receptor “lento” o sobrecargado.
- Usualmente se basa en limitar la cantidad de datos que puede enviar el transmisor.
  - El receptor informa sobre el tamaño de la ventana (tamaño de buffer disponible) en cada segmento

    Campo "Tamaño de ventana” en encabezado TCP

    (RWIN o WIN)
- El transmisor no puede tener en tránsito más que RWIN bytes a partir del último byte reconocido
  - **De esta manera, el receptor controla la cantidad máxima de datos que el transmisor puede enviarle en cada momento**
- Ejemplo: no enviar reconocimientos hasta tener buffers libres. Problema: fuerzo retransmisión por timeout

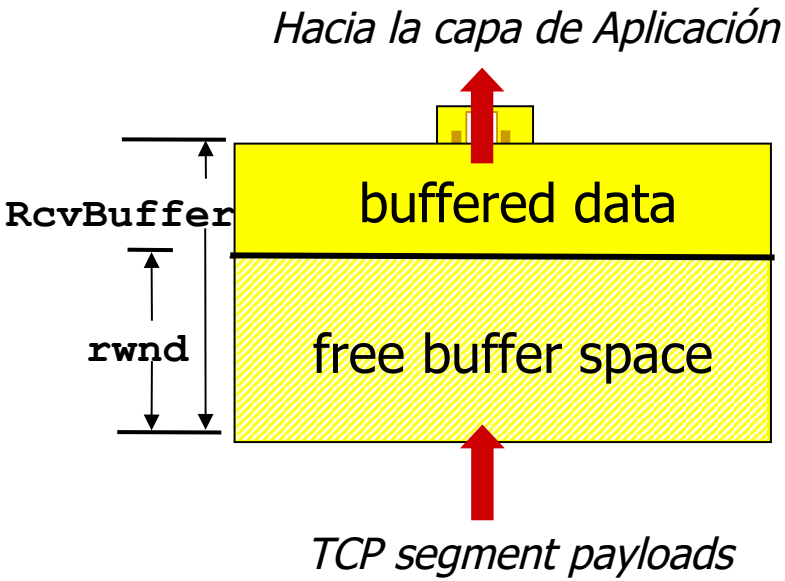
TCP - Buffers y Control de Flujo

- Las máquinas pueden manejar varias conexiones TCP simultáneas, por lo que se requiere manejo dinámico de buffers.
  - Cada conexión entre entidades de transporte tiene su respectiva $\mathbf{W_{Tx}}$, $\mathbf{W_{Rx}}$ y los buffer correspondientes.
- El uso de buffers dinámicos hace conveniente separar el asentimiento (ACK) del segmento, de la señalización del tamaño de la ventana.
  - Notifico la recepción de los datos previos pero no tengo espacio para recibir más datos. Enviar ACK para evitar las retransmisiones
  - Cuando la aplicación lee del buffer y libera espacio, al tener espacio disponible, actualizo el tamaño de ventana

TCP - Buffers y Control de Flujo

Recordar:

- Número de buffers del receptor = ventana de recepción (Windows Size) y no la cantidad de números de secuencia
- Es posible utilizar de un espacio de números de secuencia grande, e igual trabajar con equipos con poca capacidad de memoria



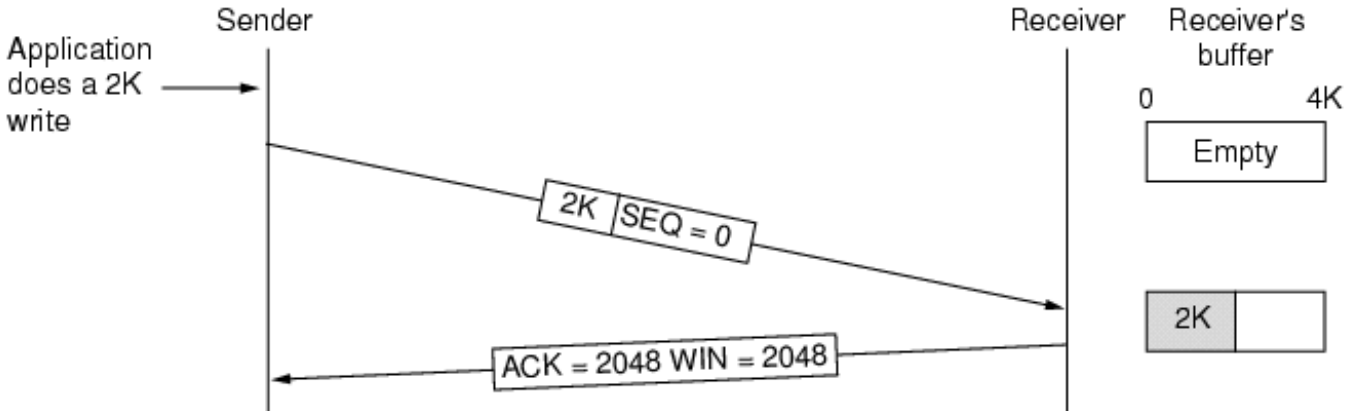
Receptor buffering

TCP – Gestión de Ventana del Receptor

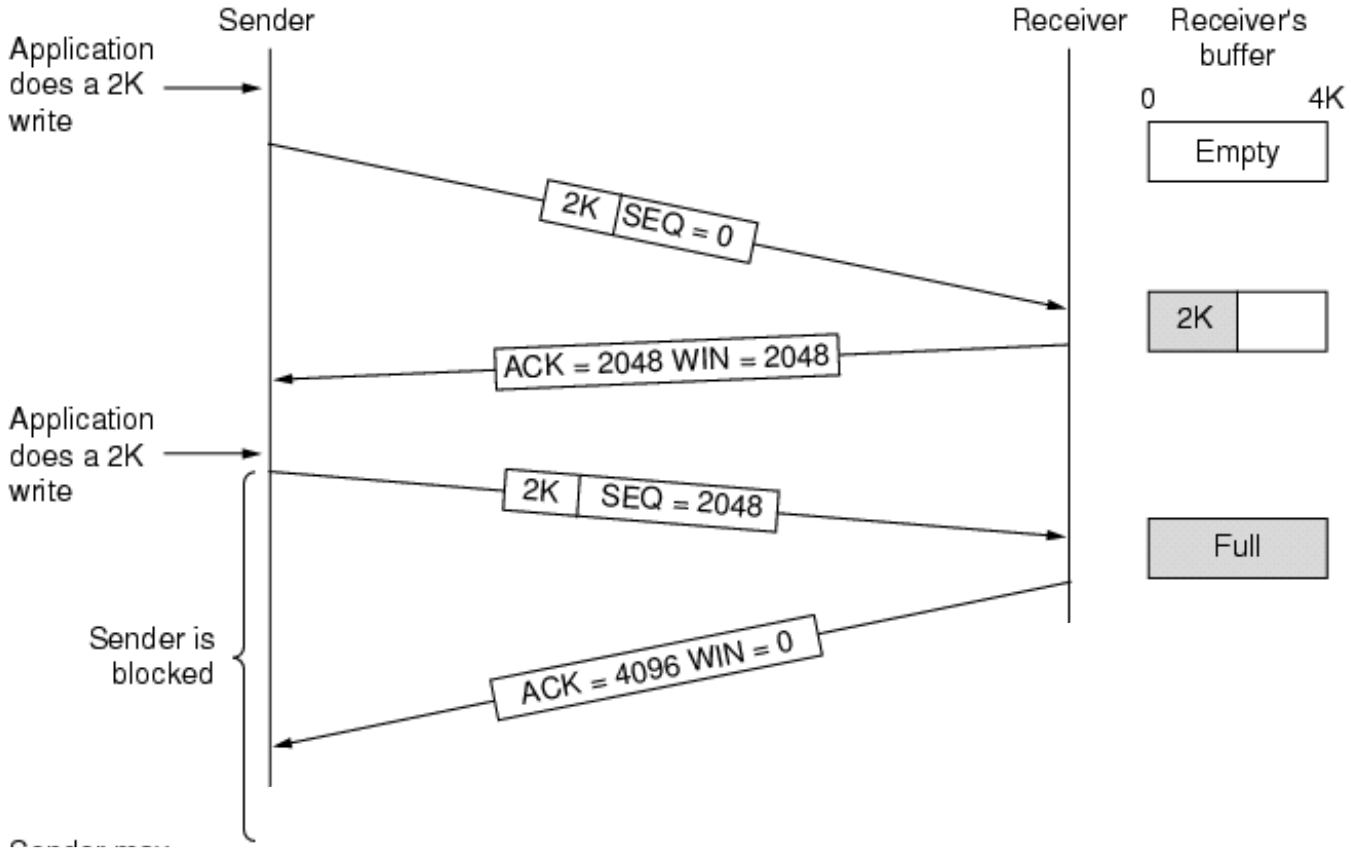
- Podría destinarse buffer fijo en el receptor
  - La aplicación puede solicitarlo
- Muchos sistemas operativos adaptan automáticamente el tamaño del buffer
- Un buffer muy chico puede limitar la tasa de transmisión (¿cómo?)
- Si la aplicación no lee los datos suficientemente rápido, la ventana del receptor decrecerá
  - Puede llegar a “0” si se llena todo el buffer

TCP – Gestión de Ventana

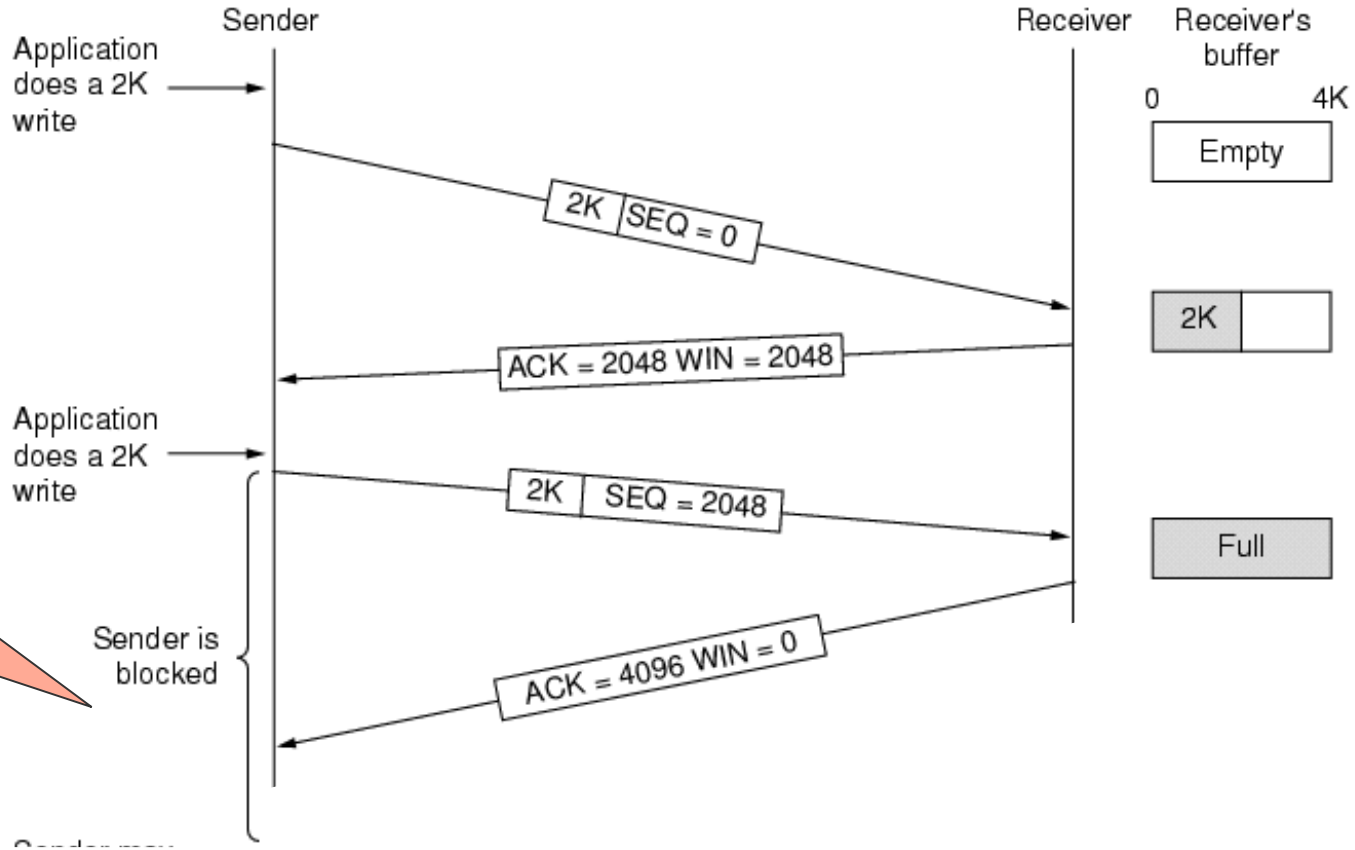
TCP – Gestión de Ventana



TCP – Gestión de Ventana



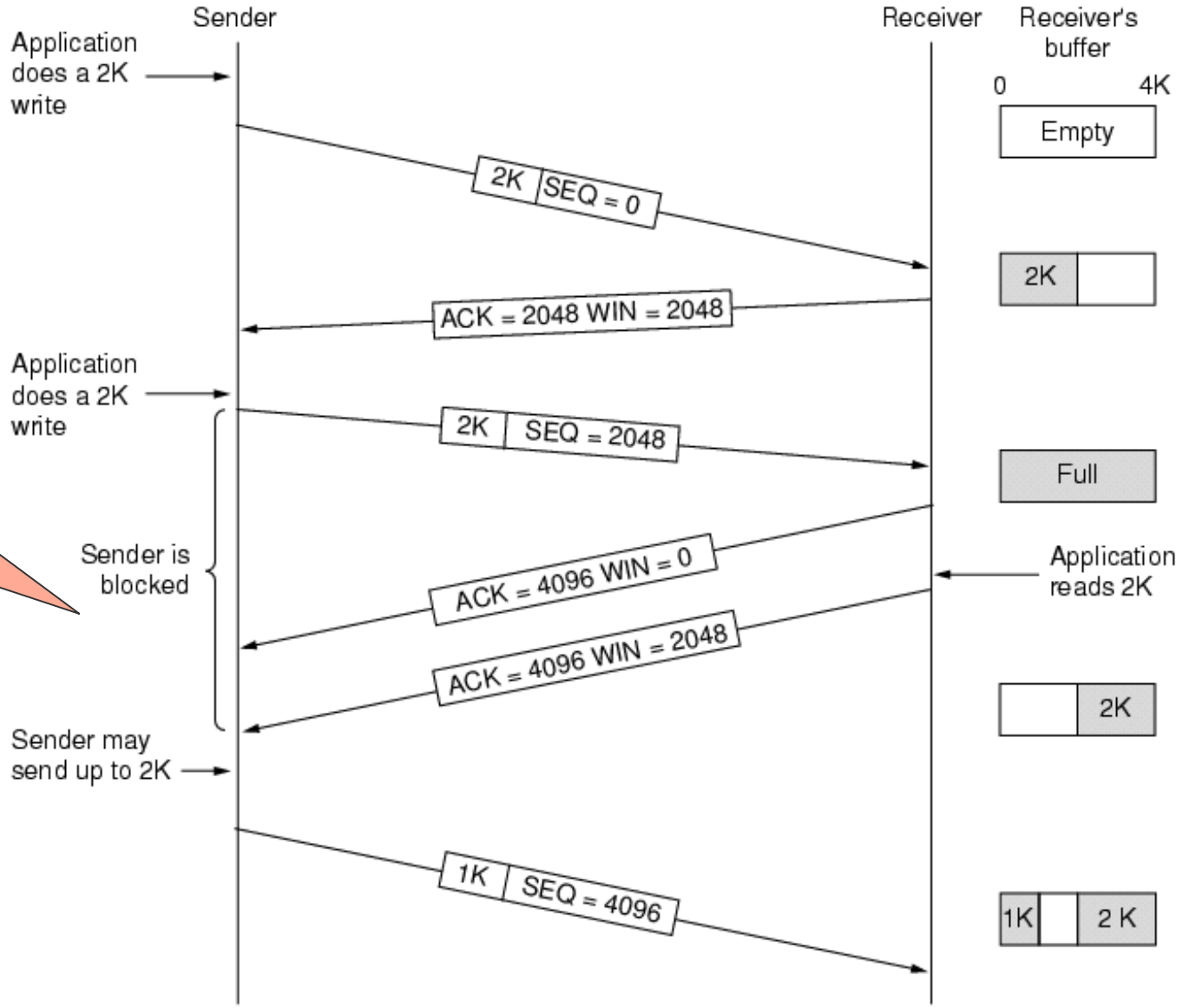
TCP – Gestión de Ventana



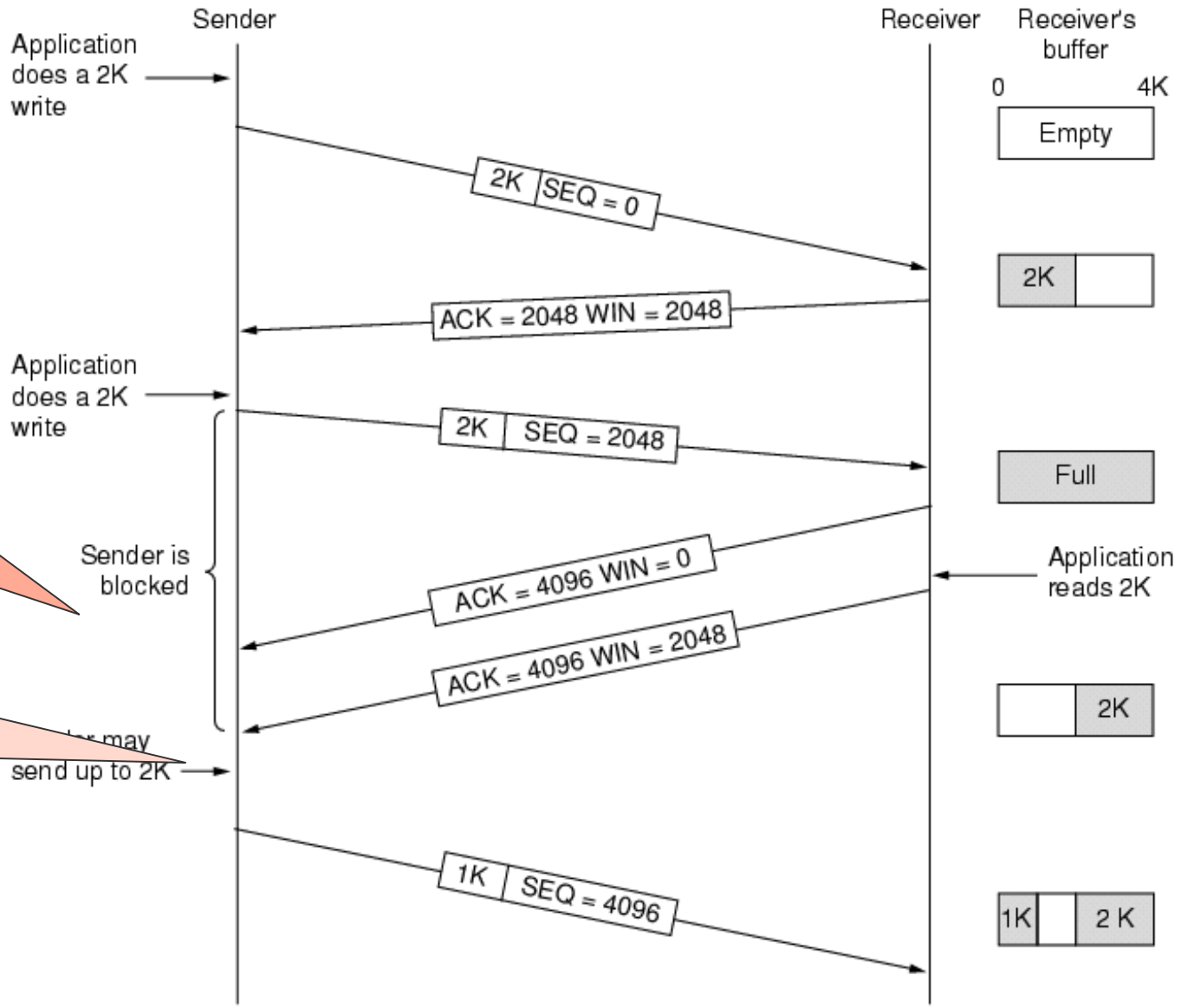
BLOQUEO: el Rx no tiene espacio para recibir nuevos datos

TCP – Gestión de Ventana

BLOQUEO: el Rx no tiene espacio para recibir nuevos datos



TCP – Gestión de Ventana



BLOQUEO: el Rx no tiene espacio para recibir nuevos datos

¿Qué sucede si se pierde la actualización de WIN?

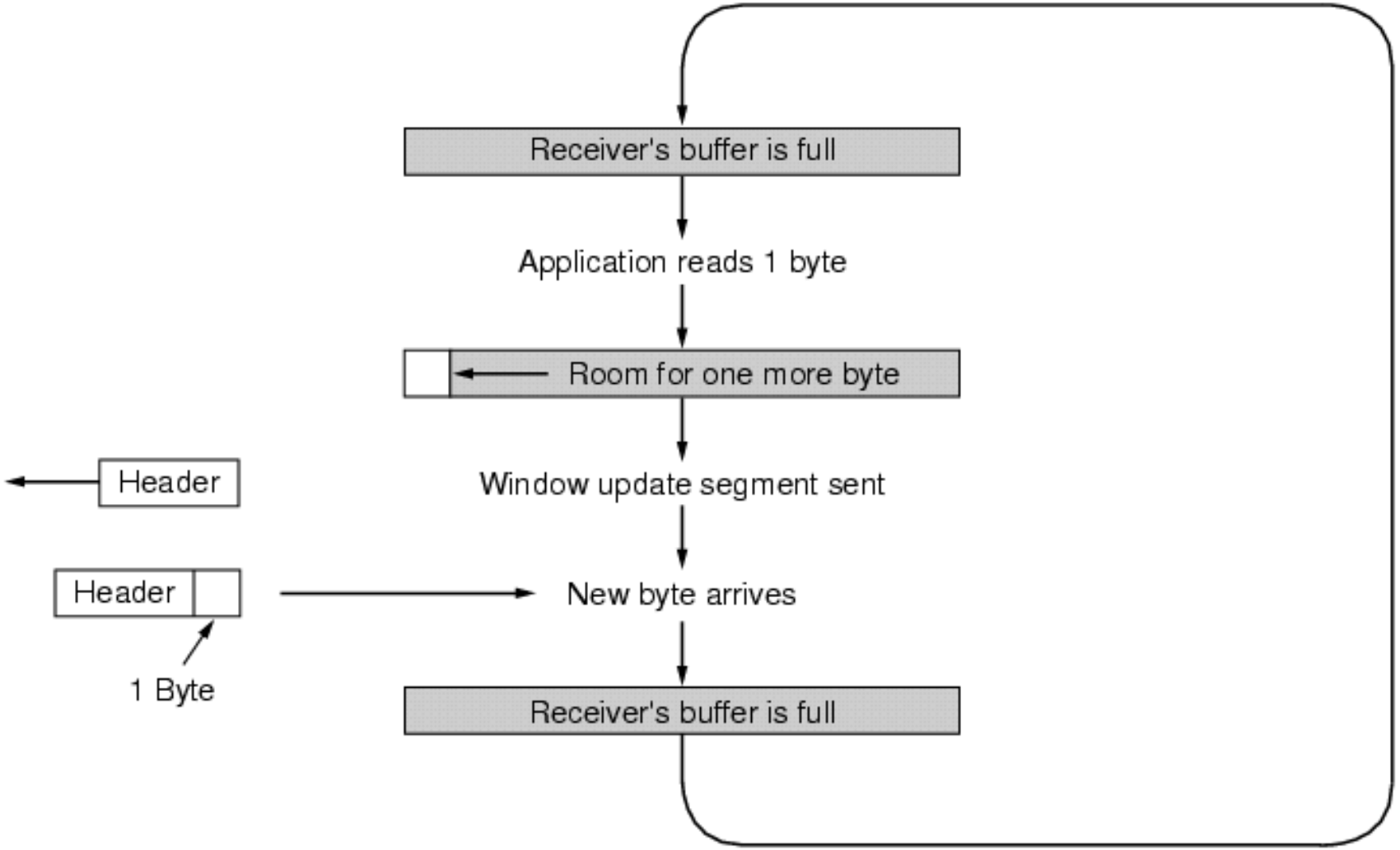
TCP – Anuncios de Ventana

- Problema: Se anuncia ventana 0, y siguiente anuncio se pierde
  - Debe evitarse el bloqueo
  - Para ello, el emisor enviará un segmento que fuerce una respuesta del receptor (prueba para forzar re-anuncio de ventana):
    - Enviar un segmento con número de secuencia menor al actual (y sin datos)
    - O enviar el próximo byte (que podría ser descartado)

TCP – Problemas de Performance

- Objetivo: evitar muchos segmentos “chicos”
- Posibilidad de retardar el envío de reconocimiento (hasta 500 ms) para esperar a tener datos para transmitir (piggybacking)
- Algoritmo de Nagle (en transmisor)
  - esperar el ack del primer segmento y mientras almacenar en buffer
  - se puede enviar también cuando se llena media ventana o el tamaño máximo del segmento
  - Malo en aplicaciones interactivas remotas (mouse)
- Síndrome de la ventana tonta (solución de Clark)
  - Aviso de ventana de 1 byte (o muy pequeña)
  - Clark: No avisar disponibilidad de ventana hasta tener libre segmento máximo o mitad del buffer

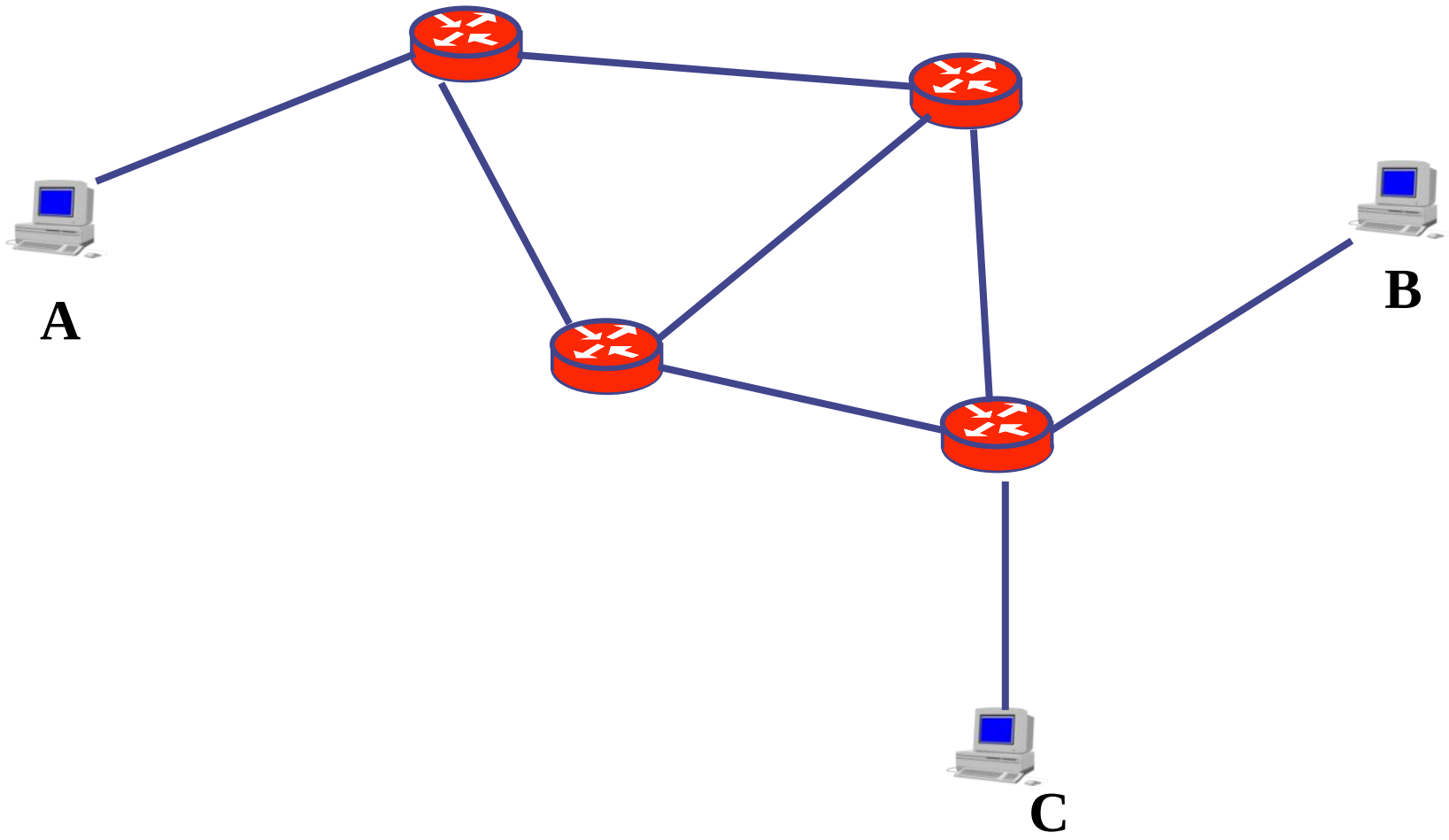
TCP – Sindrome de Ventana Tonta



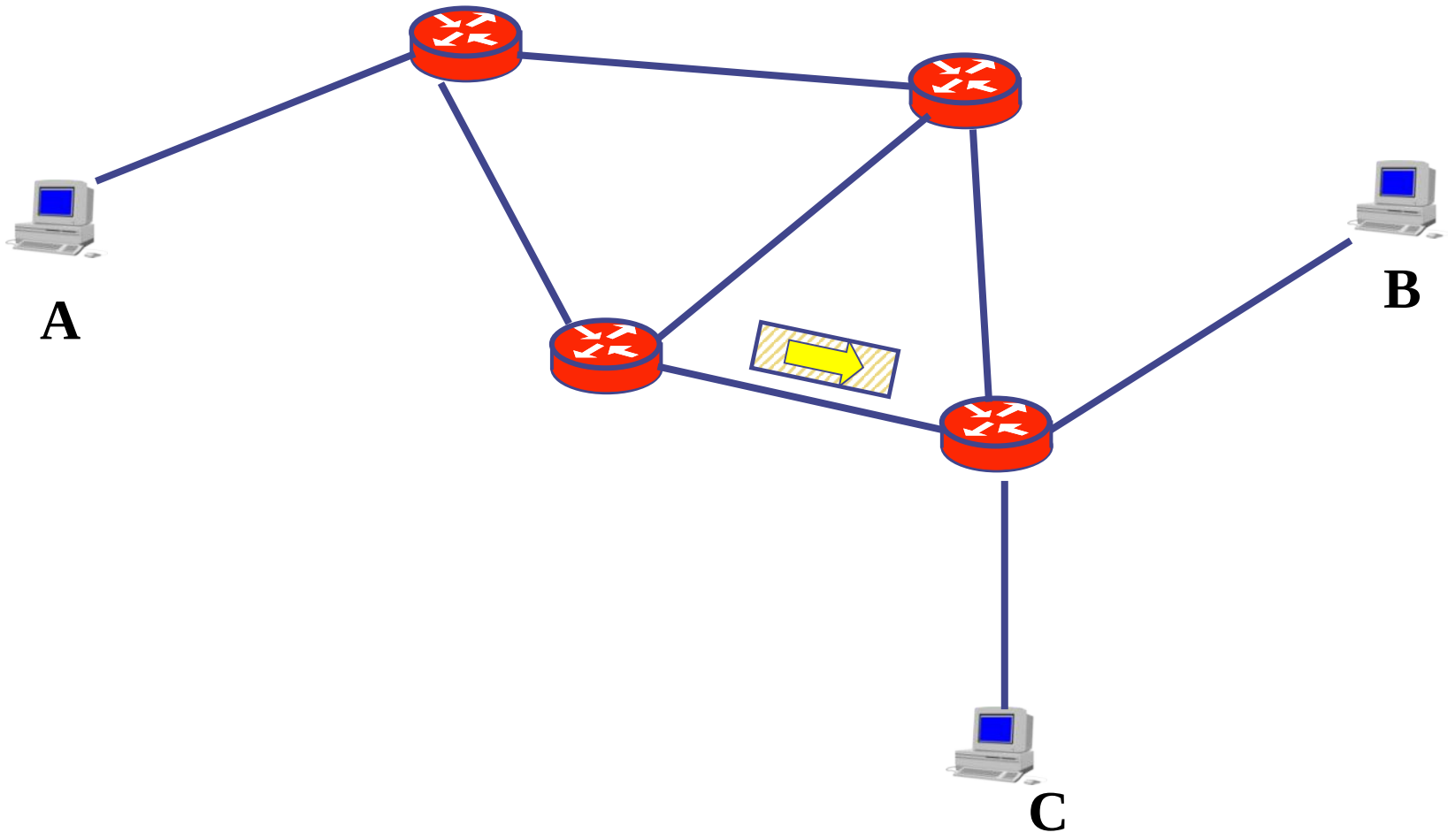
TCP

- Inicio y fin de conexión
- Manejo de números de secuencia
- Control de Flujo
- Control de Congestión
- Estados y temporizadores

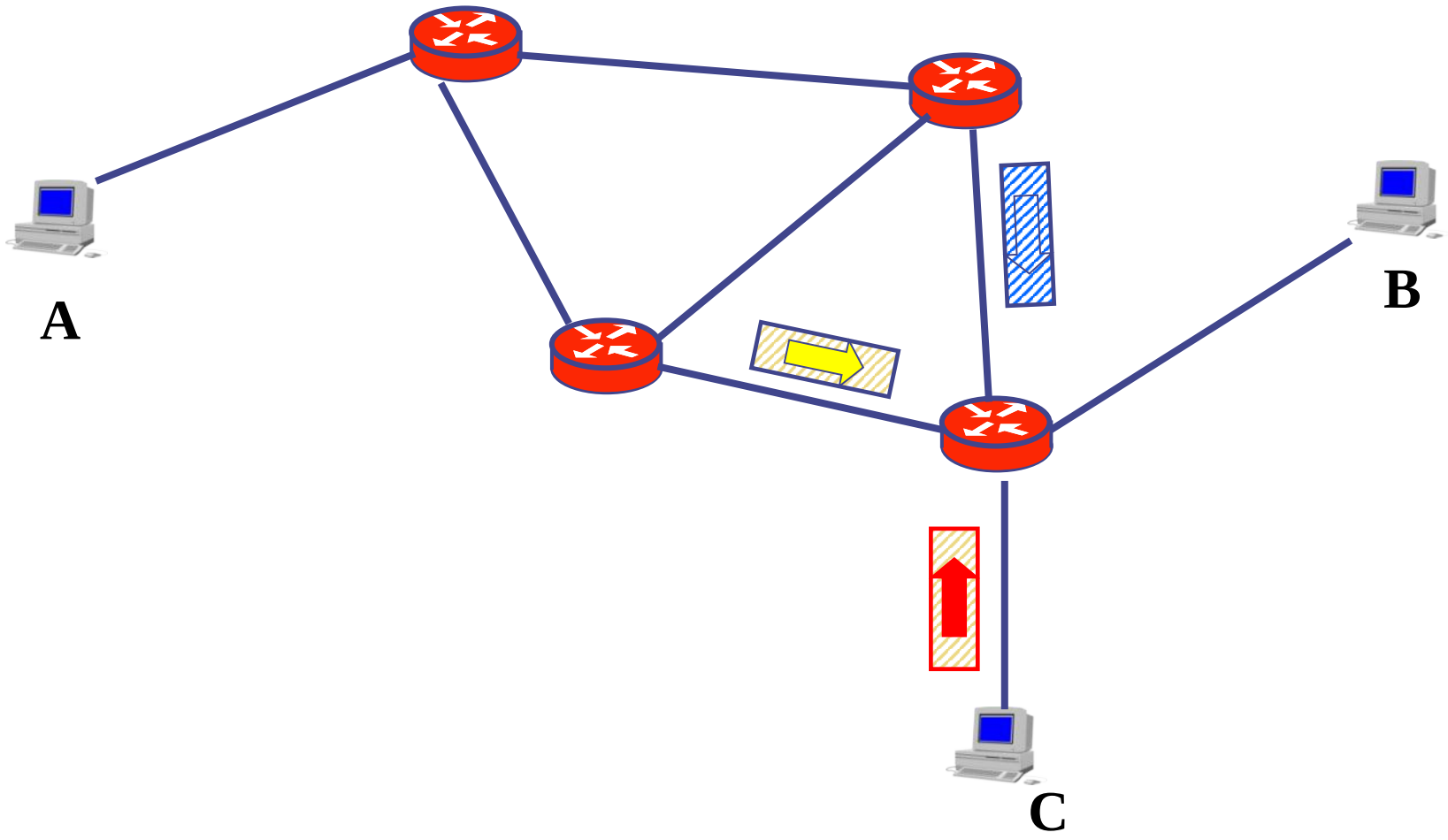
Congestión – Motivación del Problema



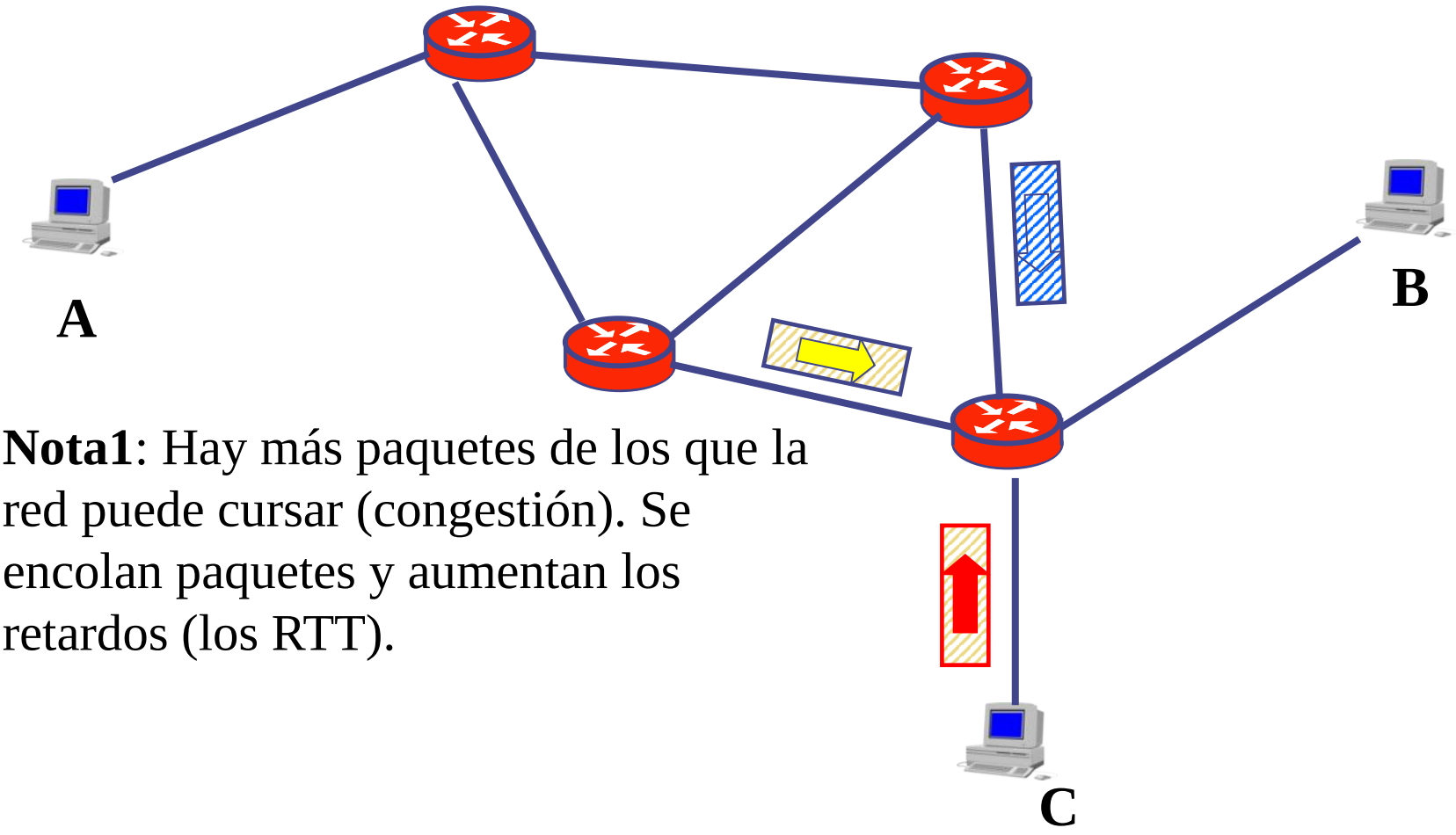
Congestión – Motivación del Problema



Congestión – Motivación del Problema

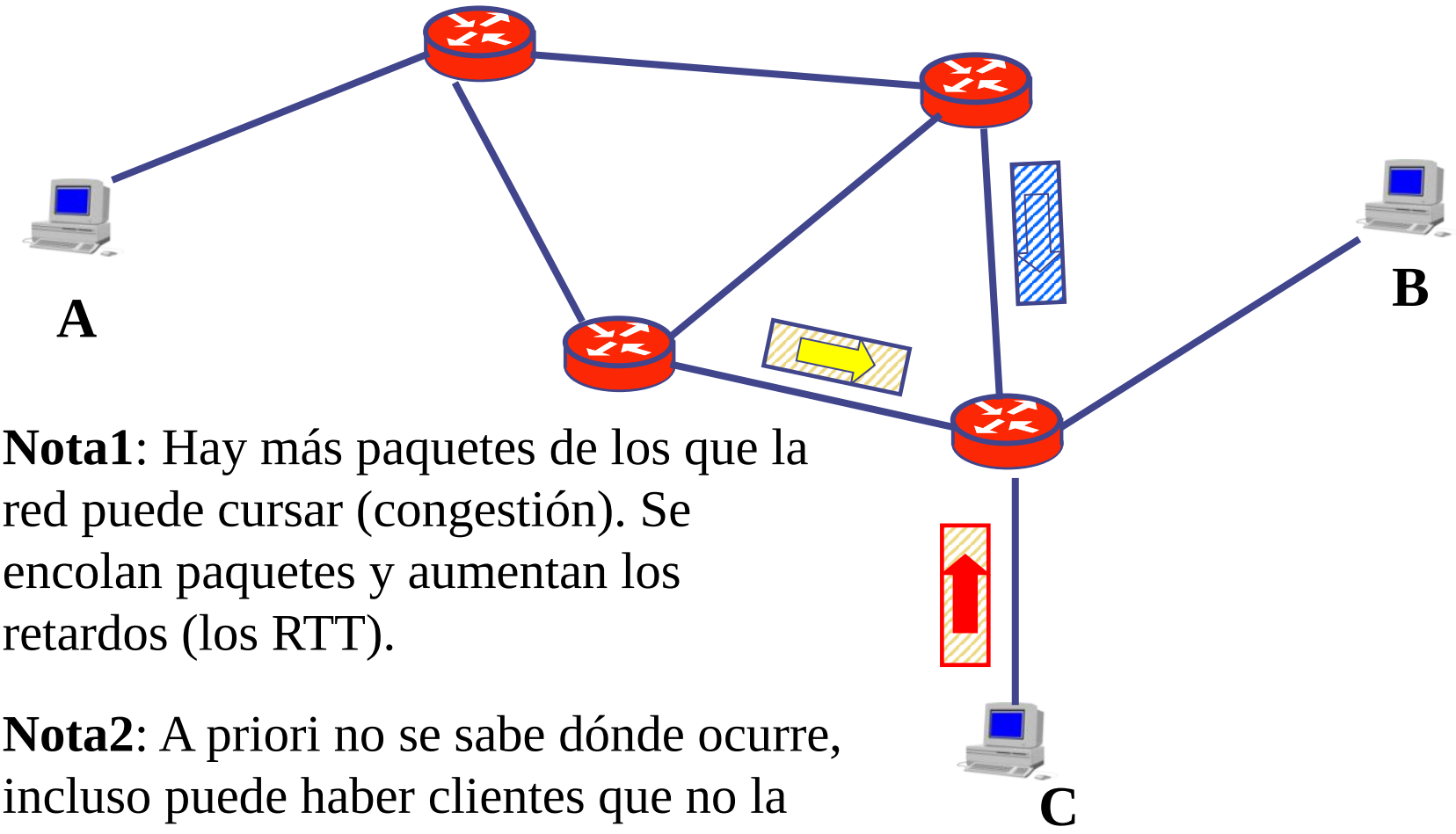


Congestión – Motivación del Problema



Nota1: Hay más paquetes de los que la red puede cursar (congestión). Se encolan paquetes y aumentan los retardos (los RTT).

Congestión – Motivación del Problema

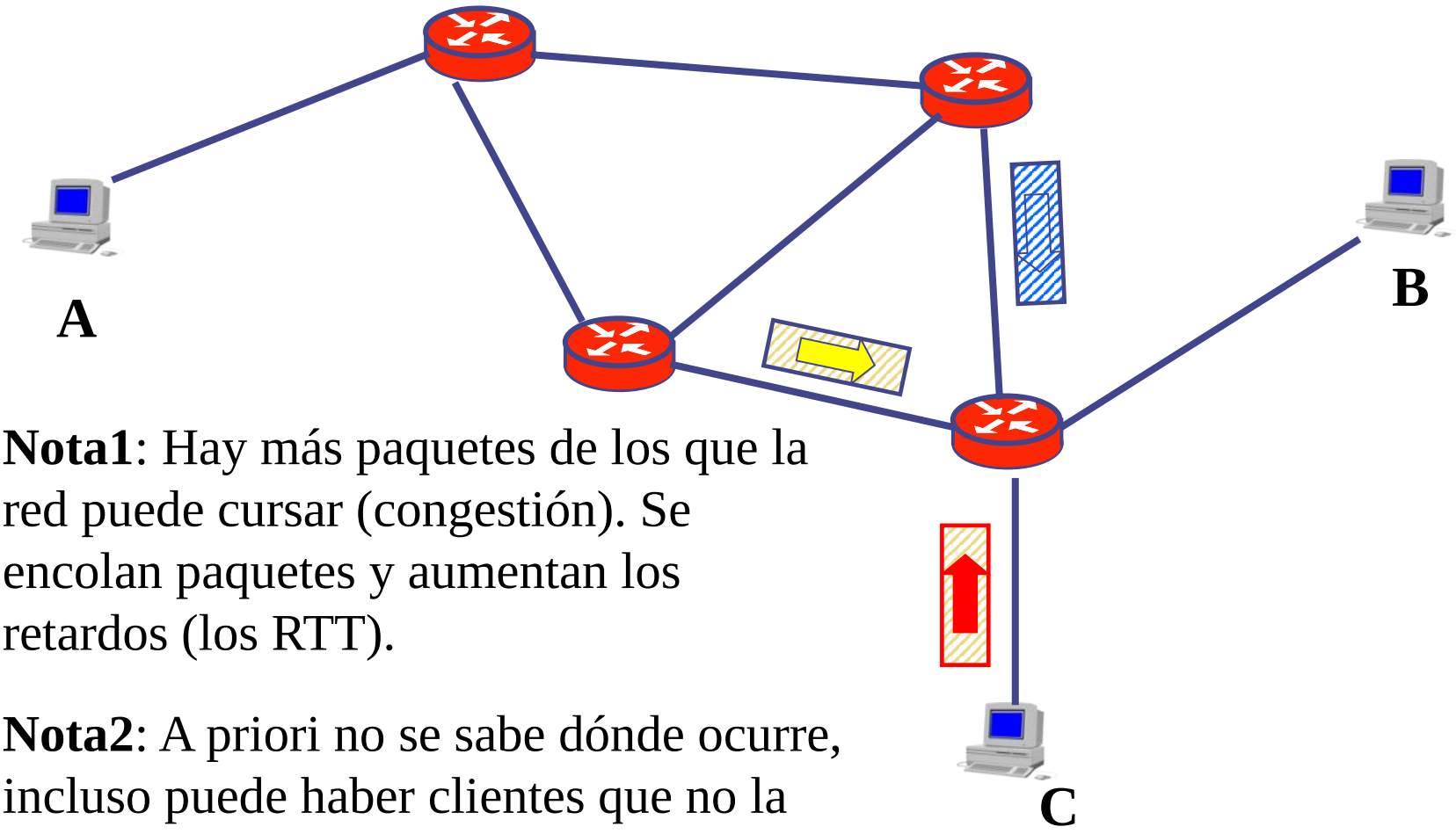


Nota1: Hay más paquetes de los que la red puede cursar (congestión). Se encolan paquetes y aumentan los retardos (los RTT).

Nota2: A priori no se sabe dónde ocurre, incluso puede haber clientes que no la experimenten.

Congestión – Motivación del Problema

Nota3: Cuando ocurre, si no disminuyo la tasa de envío puede ser "inútil" enviar, empeoramos la situación (llegan menos).

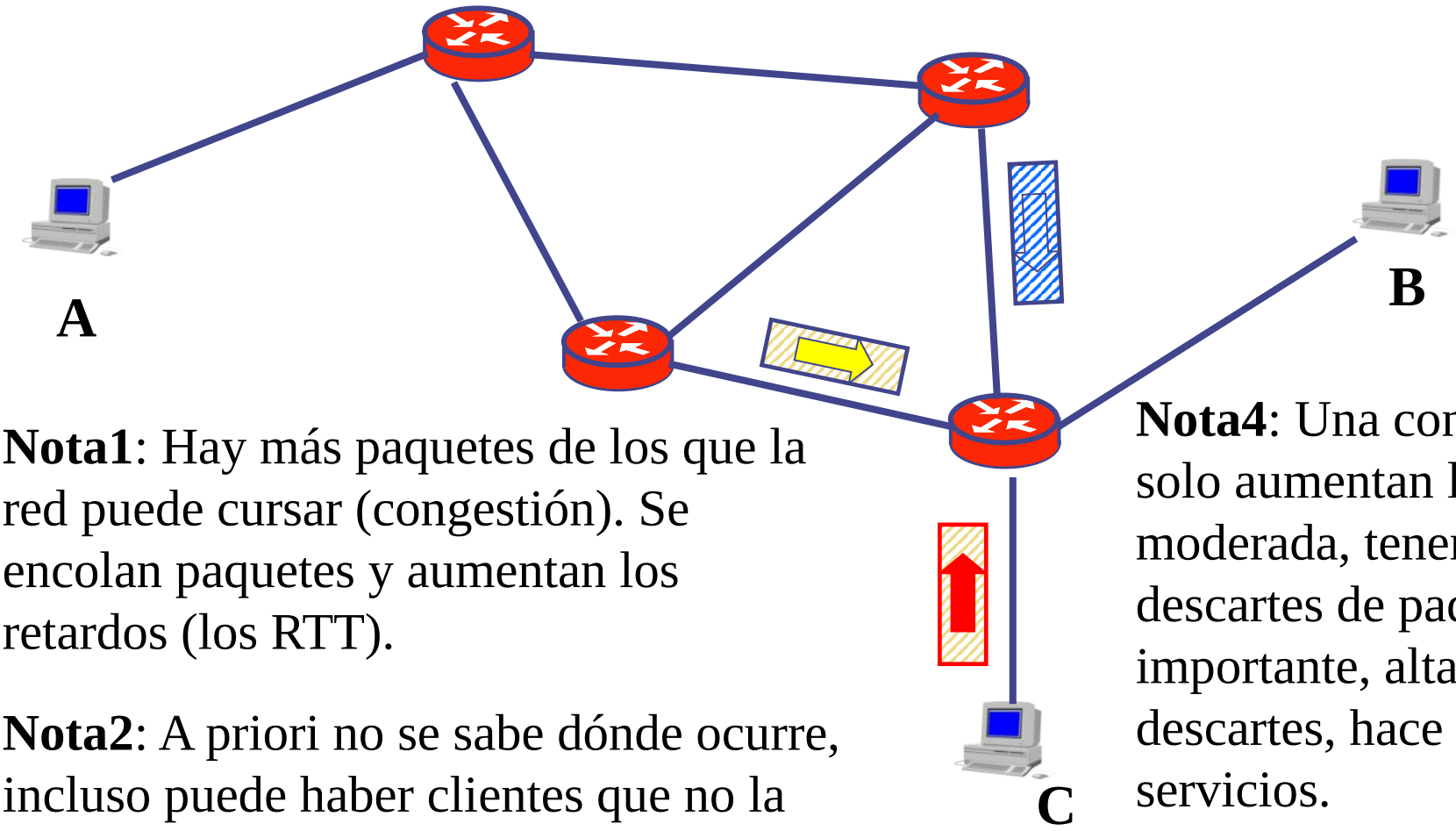


Nota1: Hay más paquetes de los que la red puede cursar (congestión). Se encolan paquetes y aumentan los retardos (los RTT).

Nota2: A priori no se sabe dónde ocurre, incluso puede haber clientes que no la experimenten.

Congestión – Motivación del Problema

Nota3: Cuando ocurre, si no disminuyo la tasa de envío puede ser "inútil" enviar, empeoramos la situación (llegan menos).



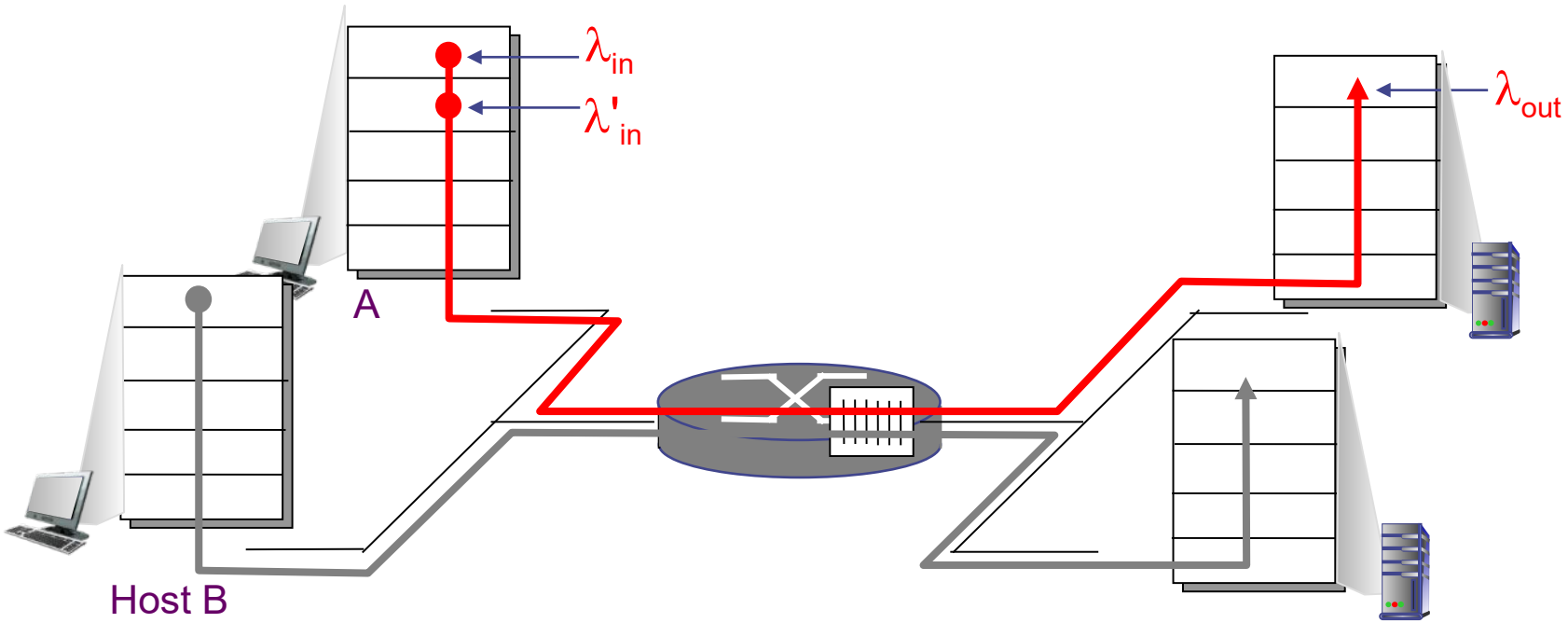
Nota1: Hay más paquetes de los que la red puede cursar (congestión). Se encolan paquetes y aumentan los retardos (los RTT).

Nota2: A priori no se sabe dónde ocurre, incluso puede haber clientes que no la experimenten.

Nota4: Una congestión leve, solo aumentan los retardos; una moderada, tenemos algunos descartes de paquetes; una importante, alta tasa de descartes, hace inviable varios servicios.

CONGESTIÓN

Dos transmisiones en simultáneo



λ_{in}- bytes/segundo versiones “originales”

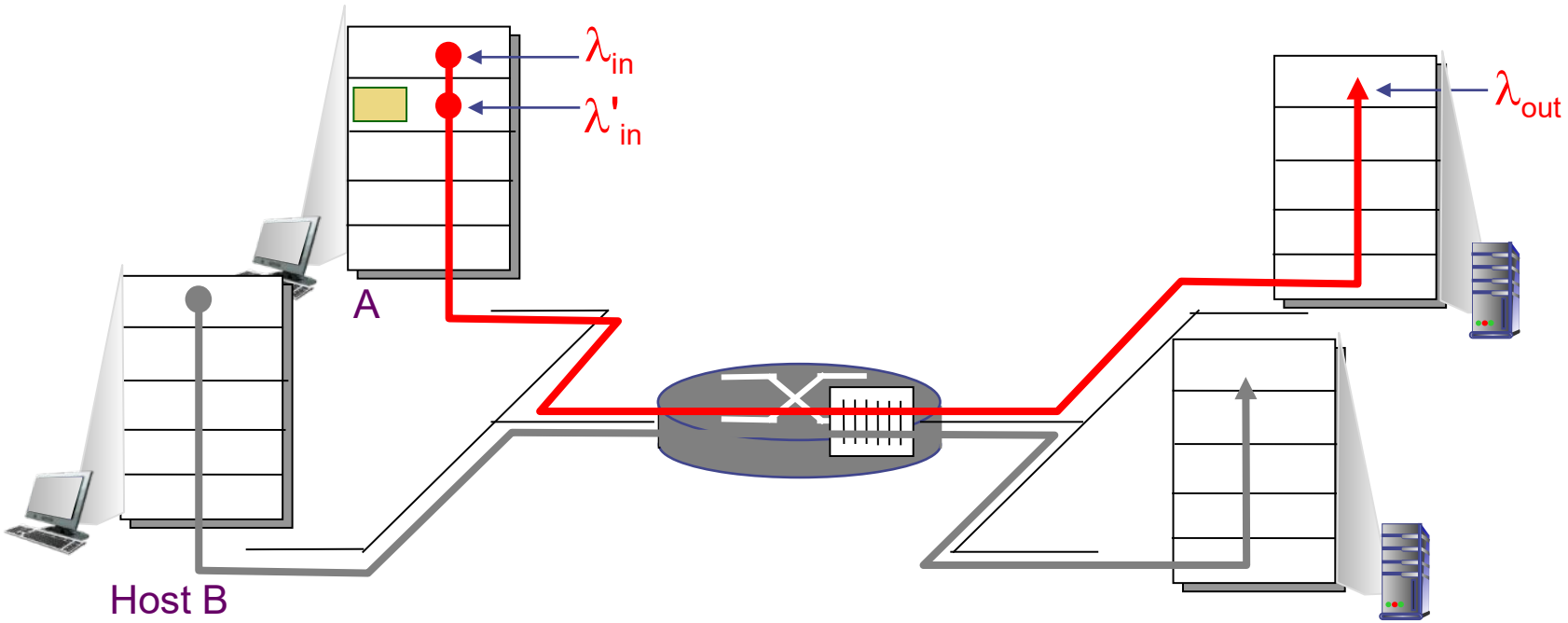
λ'_{in} - bytes/segundo versiones “originales” + retransmisiones por timeout

λ_{out} – bytes/segundo tasa “efectiva” de aplicación

R – Capacidad del Link

CONGESTIÓN

Dos transmisiones en simultáneo



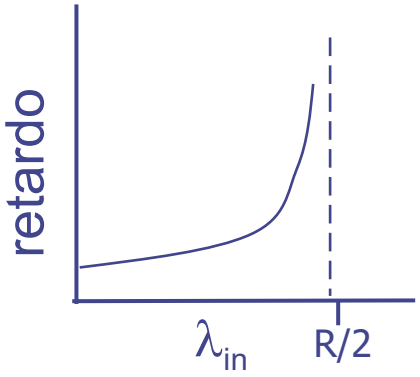
λ_{in} - bytes/segundo versiones “originales”

λ'_{in} - bytes/segundo versiones “originales” + retransmisiones por timeout

λ_{out} – bytes/segundo tasa “efectiva” de aplicación

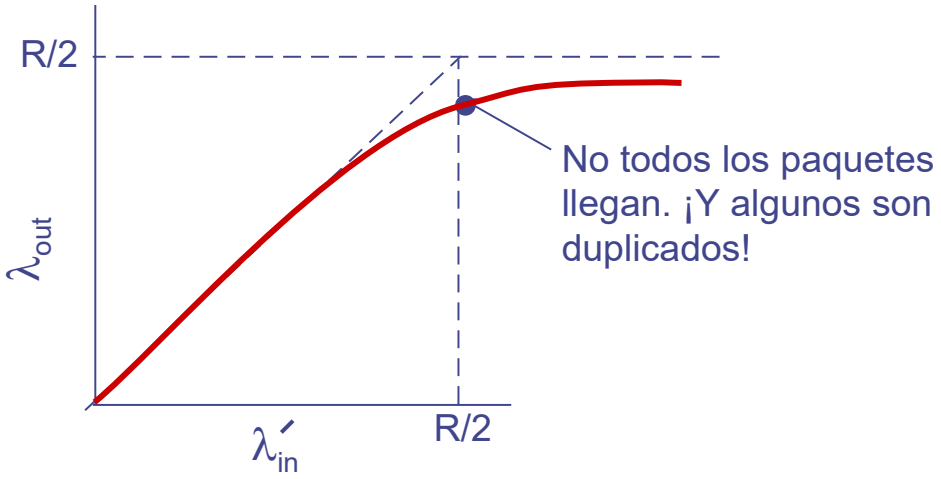
R – Capacidad del Link

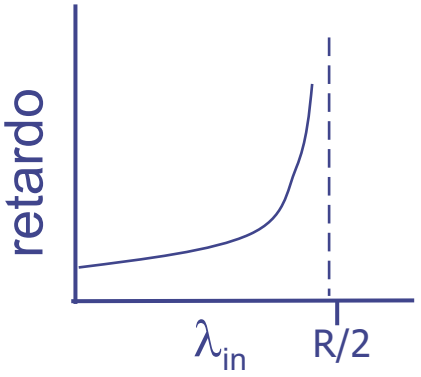
CONGESTIÓN



- El retardo crece
- Se presentan pérdidas
- Se retransmiten segmentos innecesariamente (por timeout debido al retardo)

CONGESTIÓN





- El retardo crece
- Se presentan pérdidas
- Se retransmiten segmentos innecesariamente (por timeout debido al retardo)

CONGESTIÓN

- La congestión ocurre cuando se sobrepasa la capacidad de algún elemento en la red
  - Típicamente algún enlace o la CPU de algún enrutador
- Los efectos observados incluyen aumento de retardo (RTT) y pérdidas (retransmisiones)
  - Si no se controla, lleva a un uso muy ineficiente de la red
- Puede participar la capa de transporte, la capa de red, o ambas.
- **Concepto de Multiplexado Estadístico de Tráfico.**
- Requerimos “alguna” realimentación sobre el estado de carga de la red
- **Objetivo:** enviar exactamente la cantidad de datos que la red puede transmitir
- **Dificultad:** no conocemos la capacidad instantánea disponible

Control de Congestión

- No enviar más datos que los que la red puede aceptar
  - Complementario al control de flujo
- Idea: “Entubado”
  - Si tenemos disponible un ancho de banda B, y el retardo de ida y vuelta es 2T, queremos mandar:

    V = B*2T
- **Problema:** Cómo conocer B

TCP – Control de CONGESTIÓN

- **Hipótesis:** las pérdidas de paquetes son por congestión (los enlaces son buenos)
  - Problema en enlaces inalámbricos con muchas pérdidas
- El **transmisor** utiliza una nueva ventana llamada **ventana de congestión**, que actualiza dinámicamente de acuerdo a las condiciones de la red
  - Mantiene también un valor "Umbral"
- El transmisor no permite que haya en tránsito más bytes que los que indica la ventana de congestión.

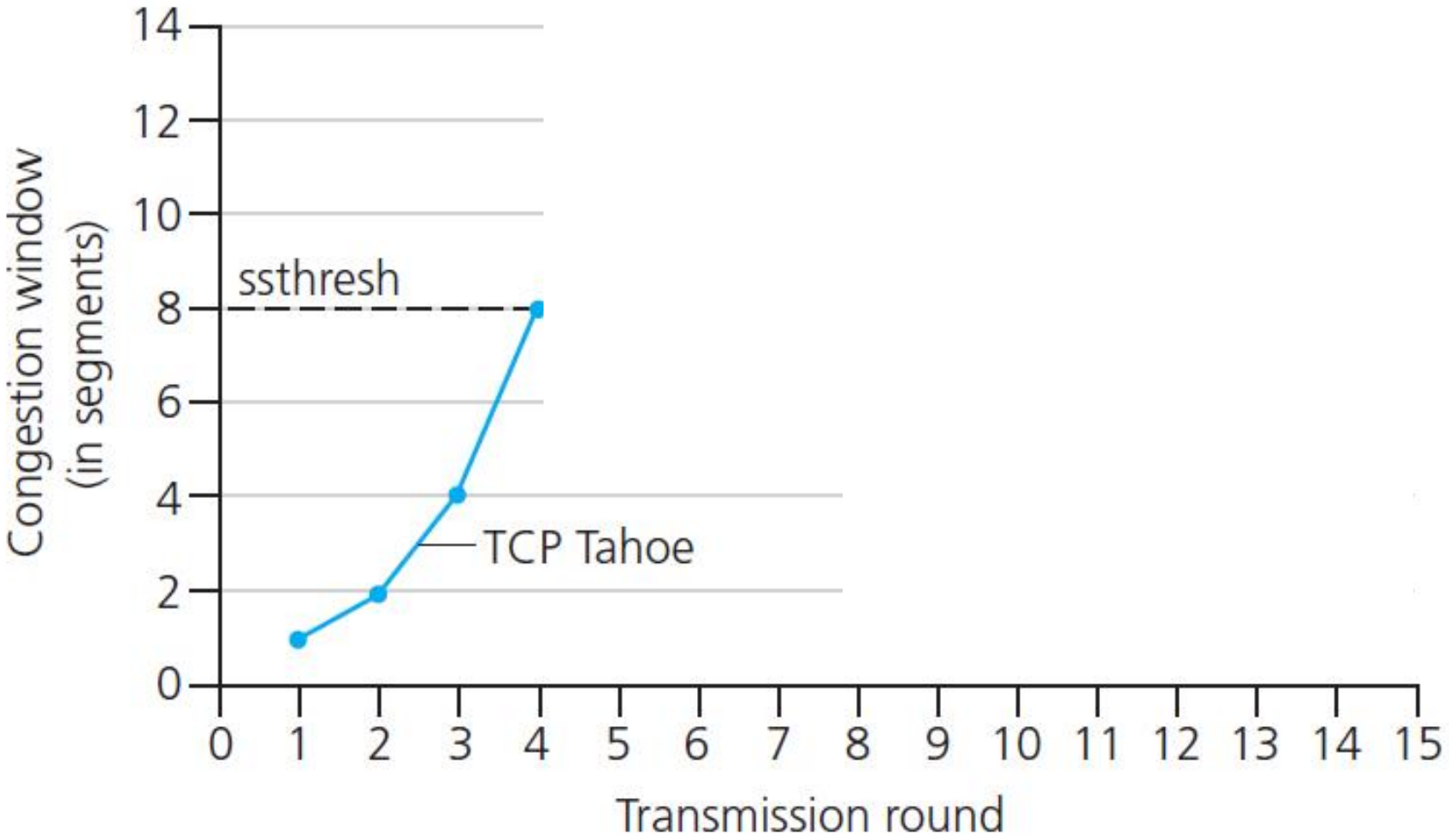
TCP – Control de CONGESTIÓN

- Llamamos **cwnd** al tamaño de la ventana de congestión
- Se define también un tamaño de umbral (**ssthresh**)
- Iremos variando el tamaño cwnd
- Se identifican 2 períodos bien definidos:
  - Slow start: al comienzo, al no conocer la capacidad disponible, se comienza con una ventana chica pero se hace crecer rápidamente
  - Congestion Avoidance: al (posiblemente) acercarnos a la congestión, hacemos crecer despacio la ventana
- Ante pérdidas, disminuimos drásticamente la tasa de transmisión (disminuyendo la ventana)

TCP – Ventana de Congestión (cwnd) – Slow Start

- Valor inicial de **cwnd = 1 MSS**
  - Versiones modernas suelen comenzar con un tamaño un poco más grande
- Por cada byte reconocido, agrega 1 byte a la ventana (a cwnd)
  - Al reconocerse toda la ventana enviada, esta se duplica
- Si tenemos pérdidas, bajamos el valor de cwnd
- Comportamiento se mantiene hasta que cwnd alcanza al valor del umbral (**ssthresh**)

TCP – Ventana de Congestión (cwnd) – Slow Start

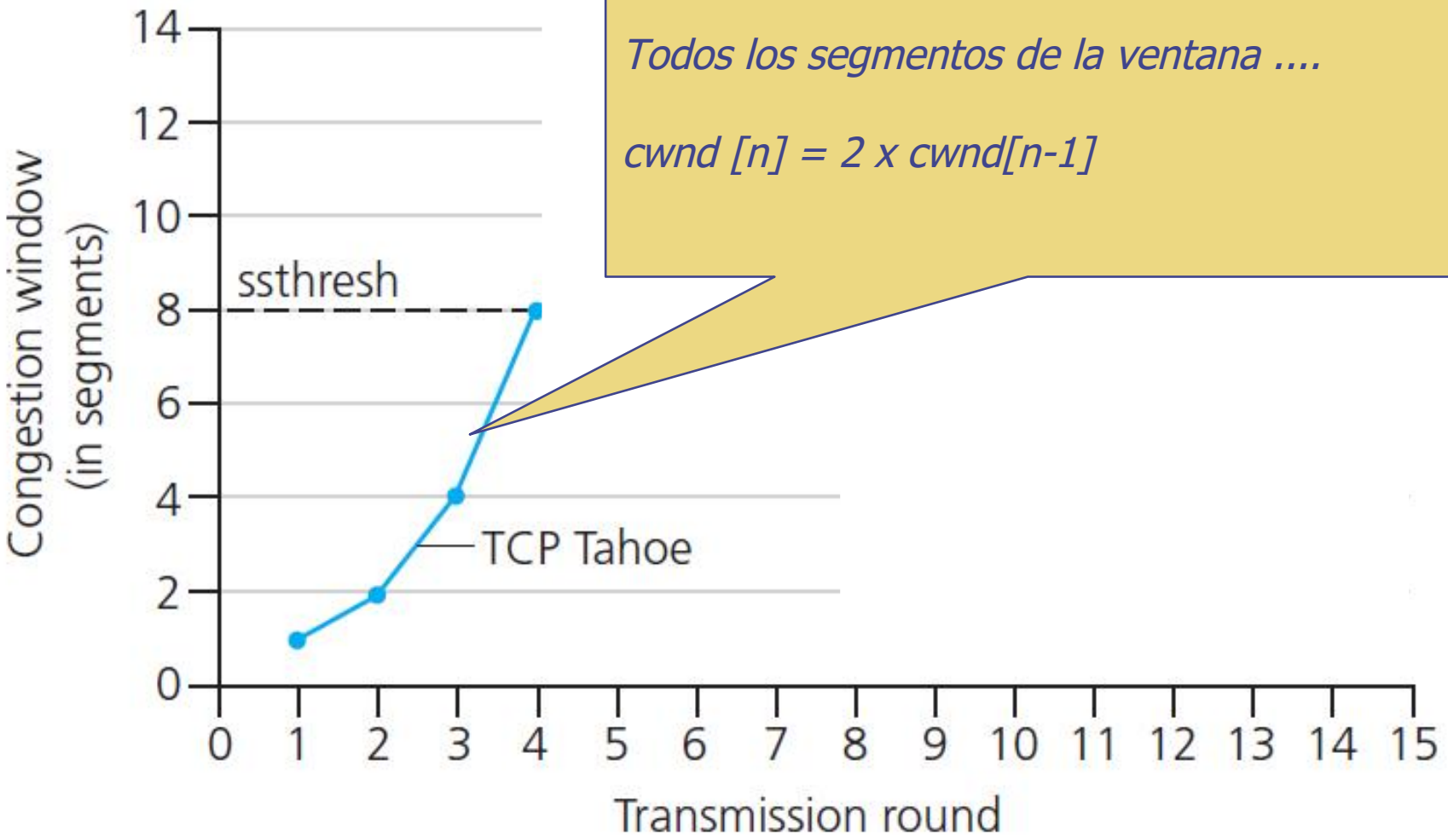


TCP – Ventana de Congestión (cwnd) – Slow Start

cwnd[n] = cwnd [n-1] + [Número de bytes ACK]

Todos los segmentos de la ventana ....

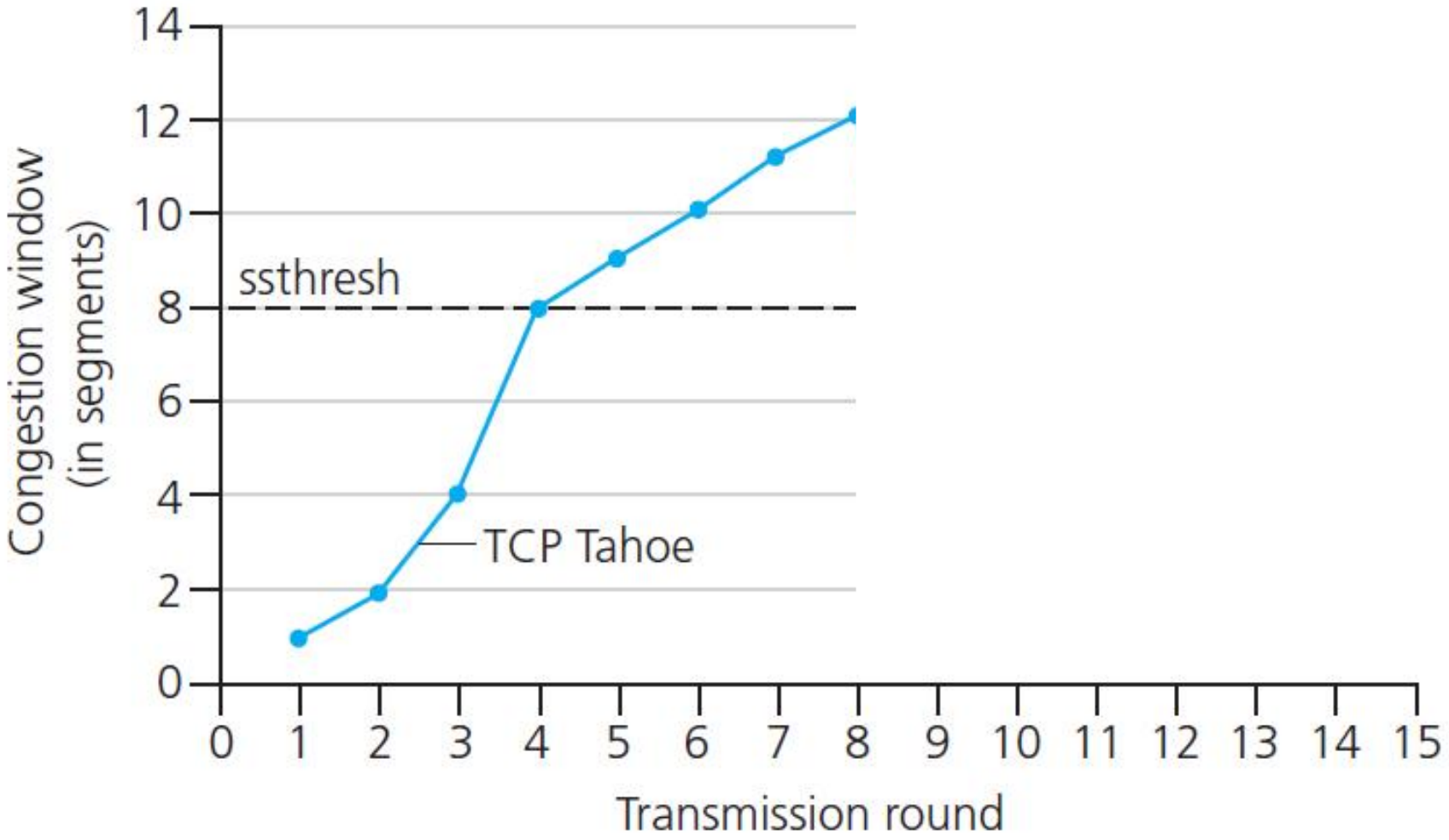
cwnd [n] = 2 x cwnd[n-1]



TCP – Ventana de Congestión – Congestion Avoidance

- A partir del umbral se asume que podemos estar más cerca de la congestión
  - Se incrementa cwnd de forma lineal
  - Se aumenta en 1 segmento por cada ventana completa reconocida
- Si no hay pérdidas, podríamos crecer indefinidamente
  - Usualmente nos termina limitando la ventana del receptor, también debe seguir respetando el control de flujo.

TCP – Ventana de Congestión – Congestion Avoidance

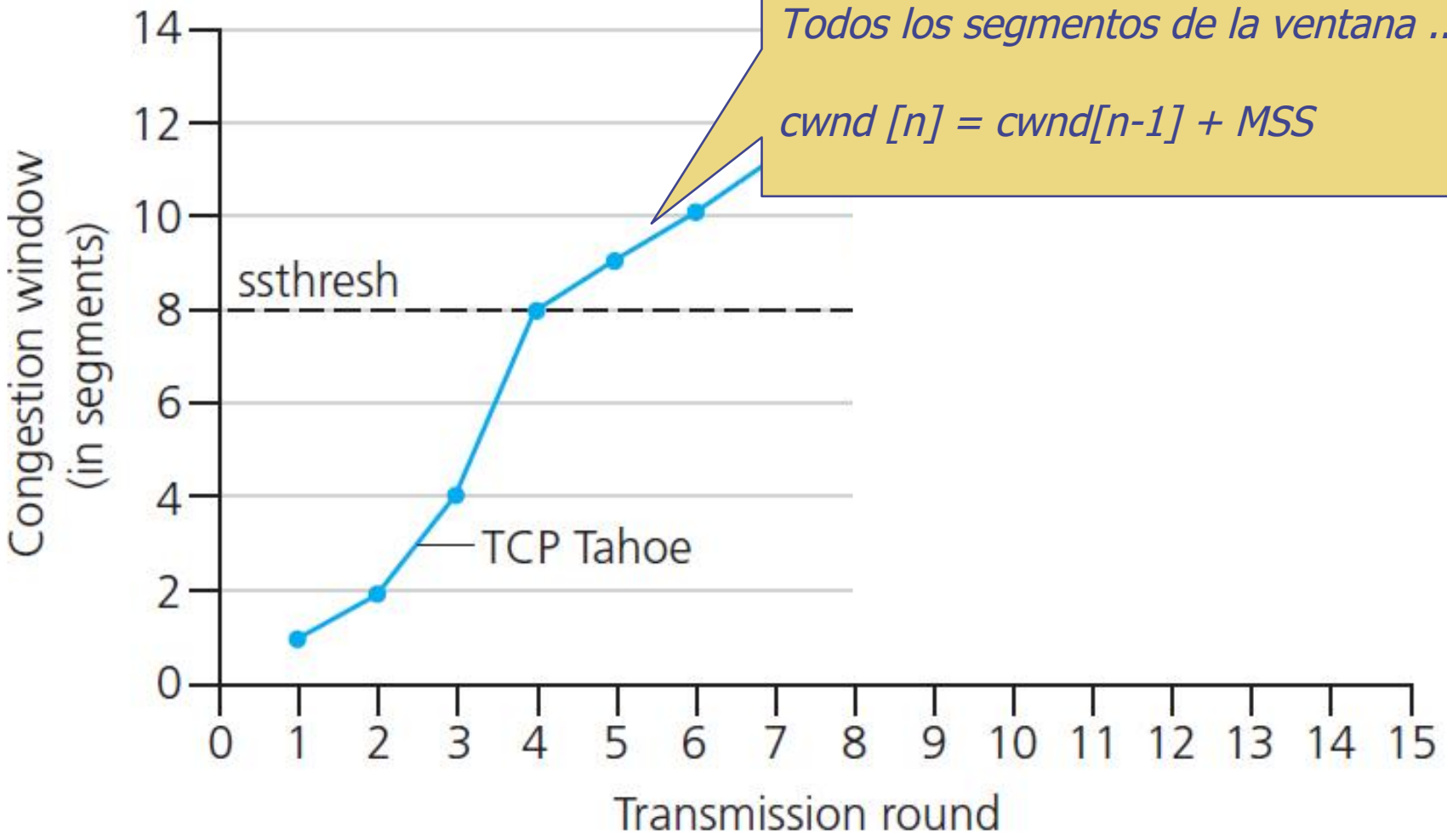


TCP – Ventana de Congestión – Congestion Avoidance

cwnd [n] = cwnd [n-1] + (MSS/cwnd[n-1]) x [Número de bytes ACK]

Todos los segmentos de la ventana ....

cwnd [n] = cwnd[n-1] + MSS



TCP – Comportamiento ante pérdidas

- Timeout = Congestión
  - Bajar la tasa de transmisión
  - Bajar la cwnd

TCP – Comportamiento ante pérdidas

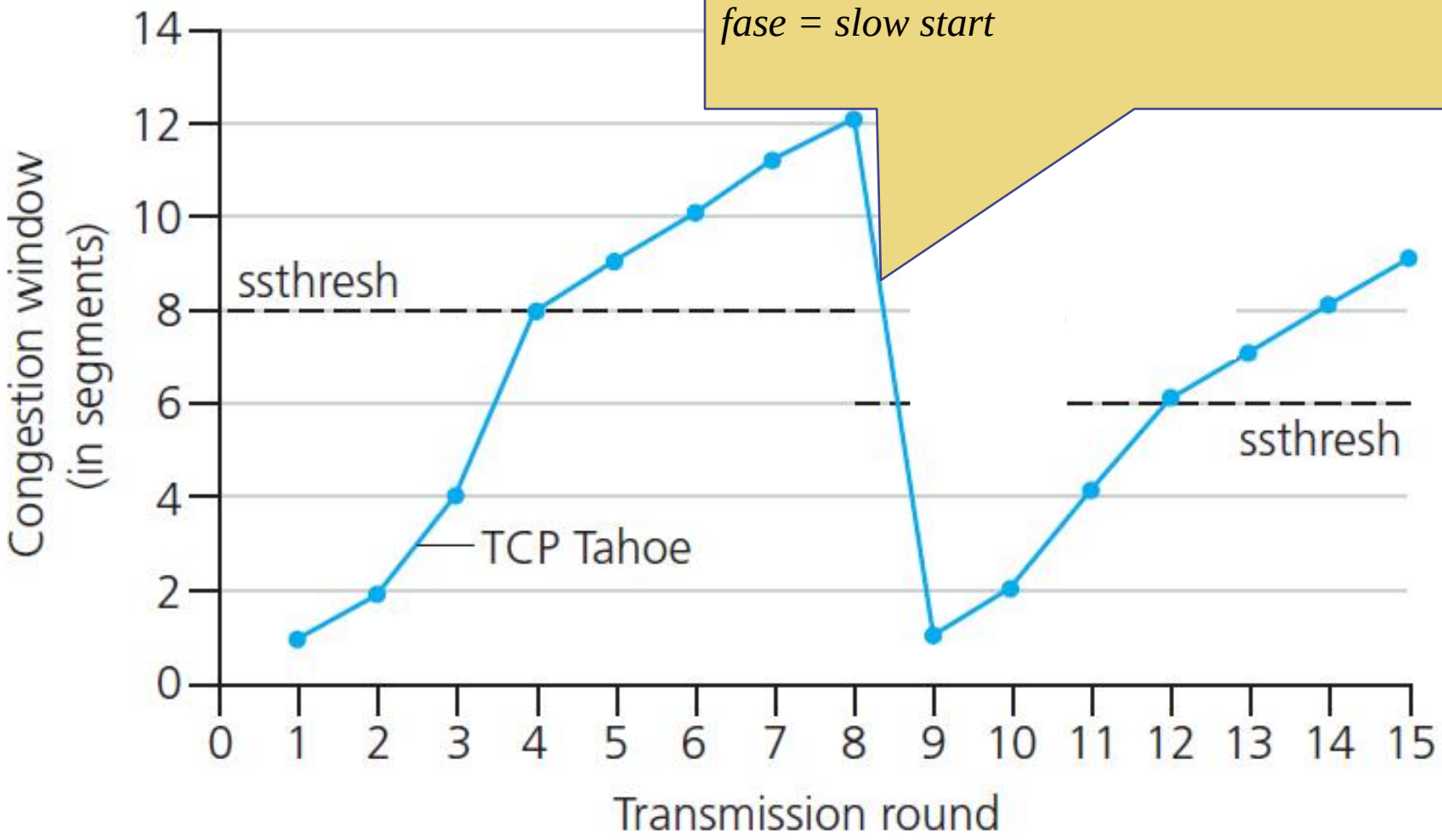
- Timeout = Congestión
  - Bajar la tasa de transmisión
  - Bajar la cwnd

- Timeout:
  - fijar umbral (ssthresh) a la mitad del valor de la ventana actual (NO a la mitad del umbral actual)
  - Bajar la ventana (cwnd) a 1 MSS
  - Retomar en fase slow start

TCP – Ventana de Congestión – Timeout

cwnd = 1 MSS
ssthresh[n] = cnwd [n-1] /2 (“la última ventana”)

fase = slow start

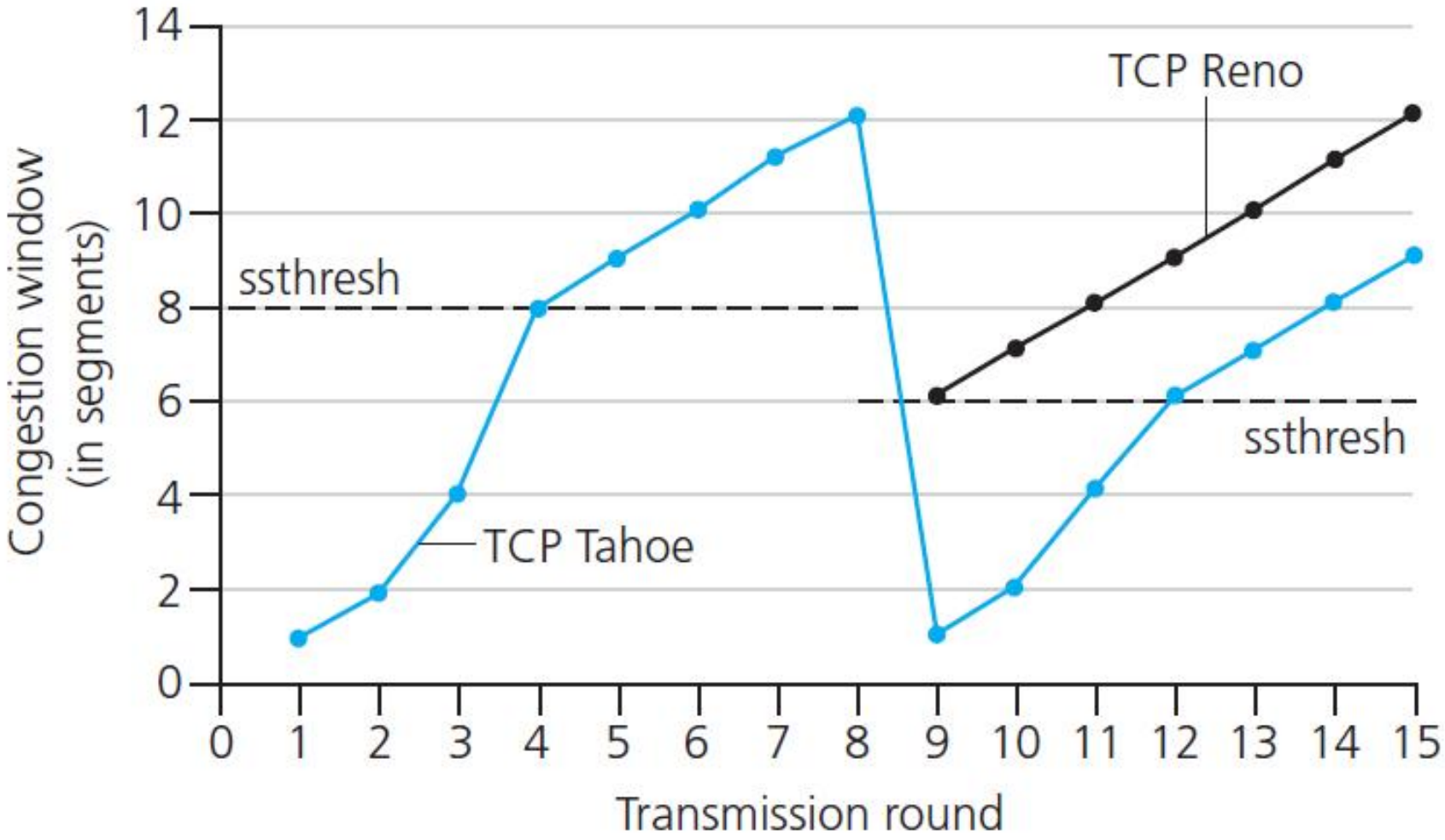


TCP – Comportamiento ante pérdidas

- No todos los escenarios de pérdidas son similares.
- Reconocimientos repetidos:
  - Recordemos el caso en que envío 4 segmentos y se pérdio solo el primero. Se repite el ACK(Z) tres veces.
  - Fijar umbral a la mitad del valor de la ventana actual
  - Bajar cwnd al valor del umbral (TCP Reno)

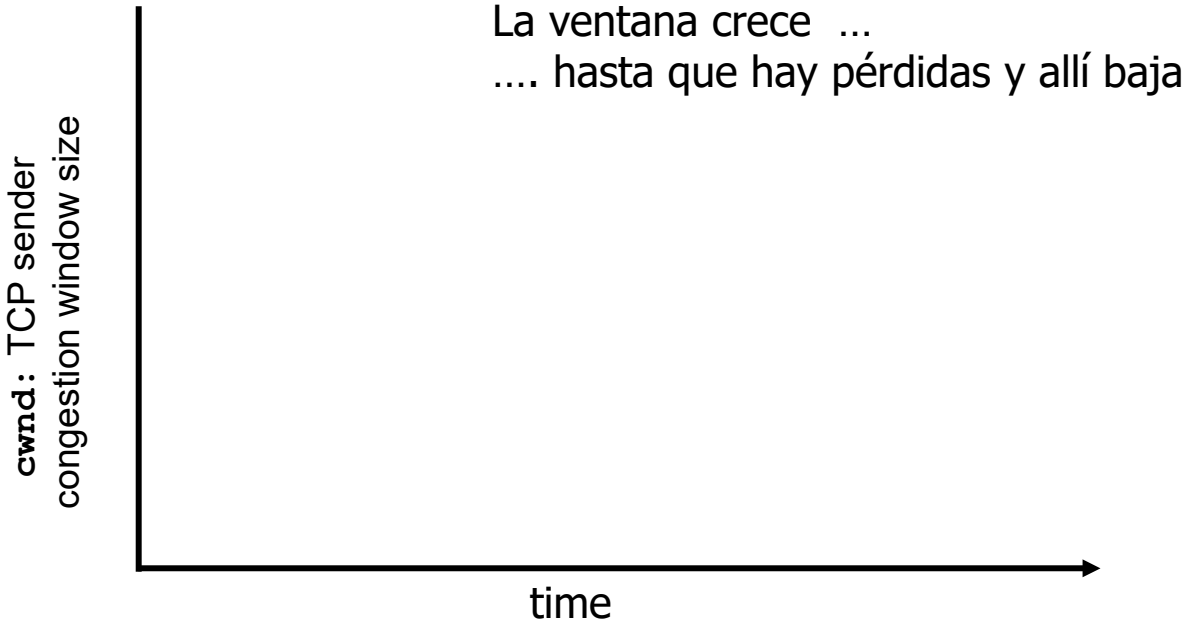
    Se siguen recibiendo segmentos, así que se asume que la congestión no es tan importante

TCP – Comportamiento ante pérdidas



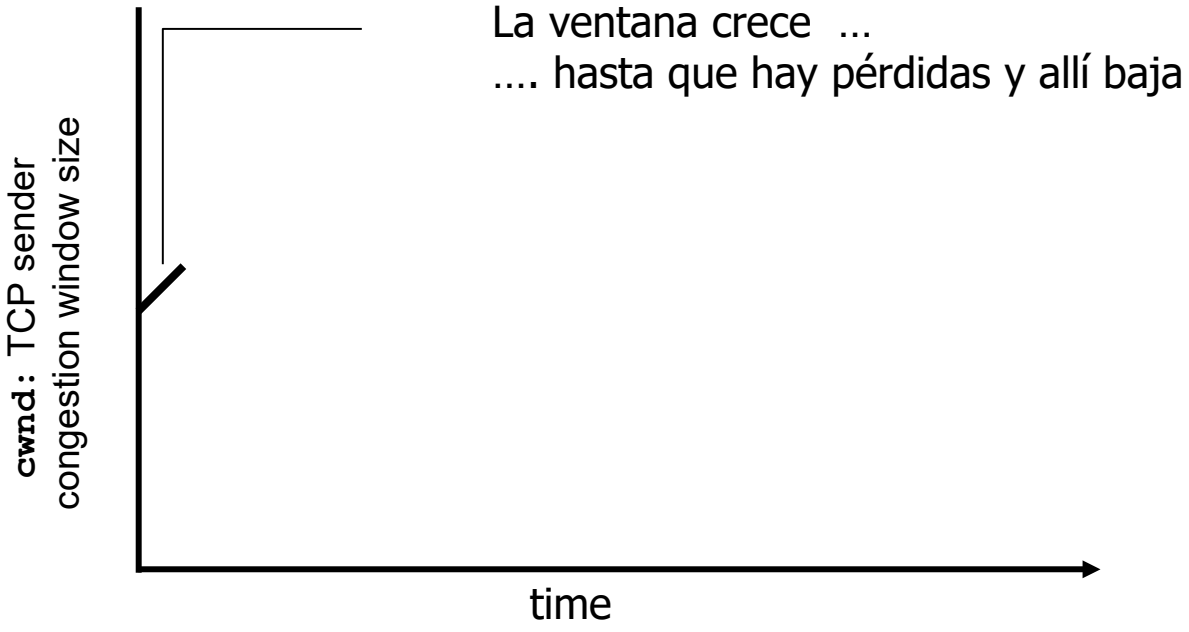
TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



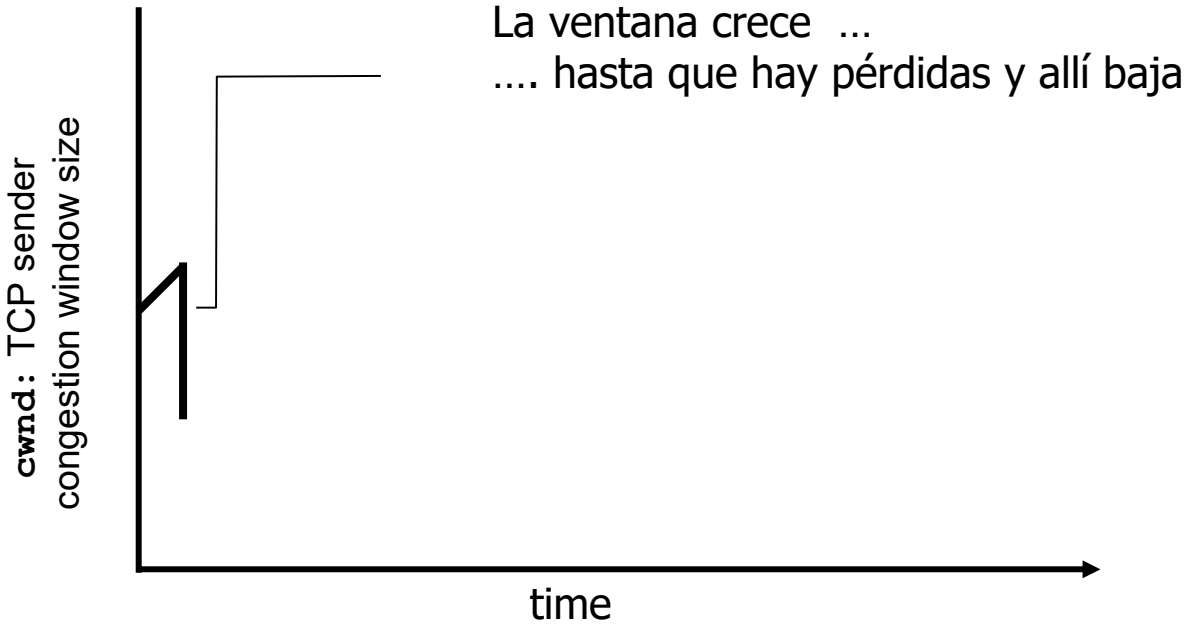
TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



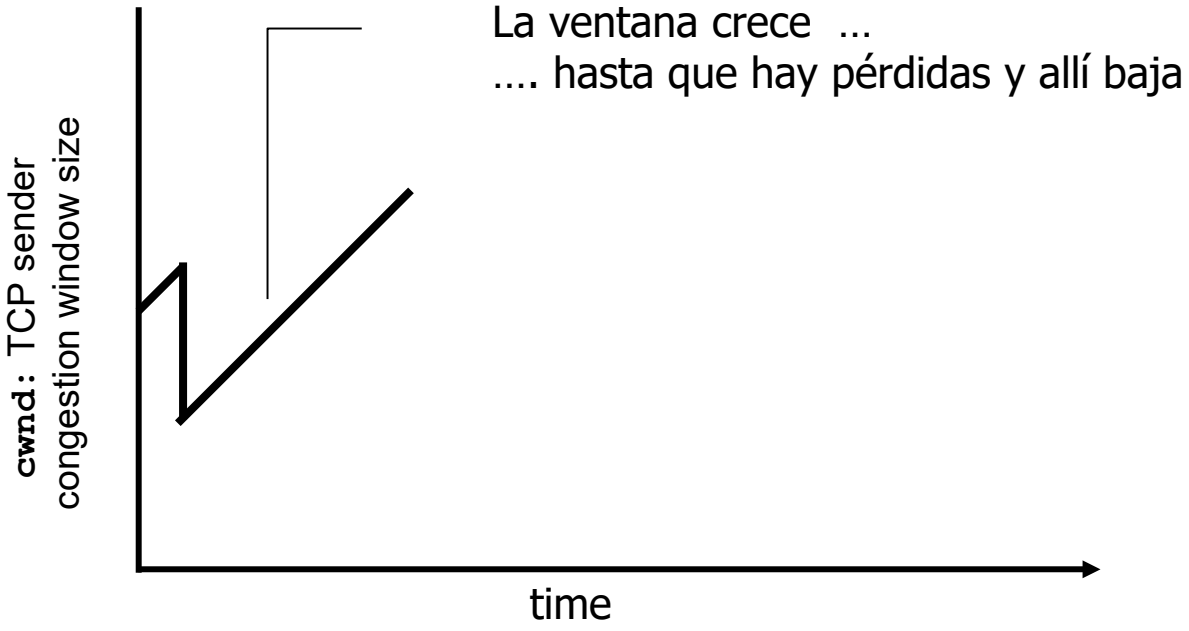
TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



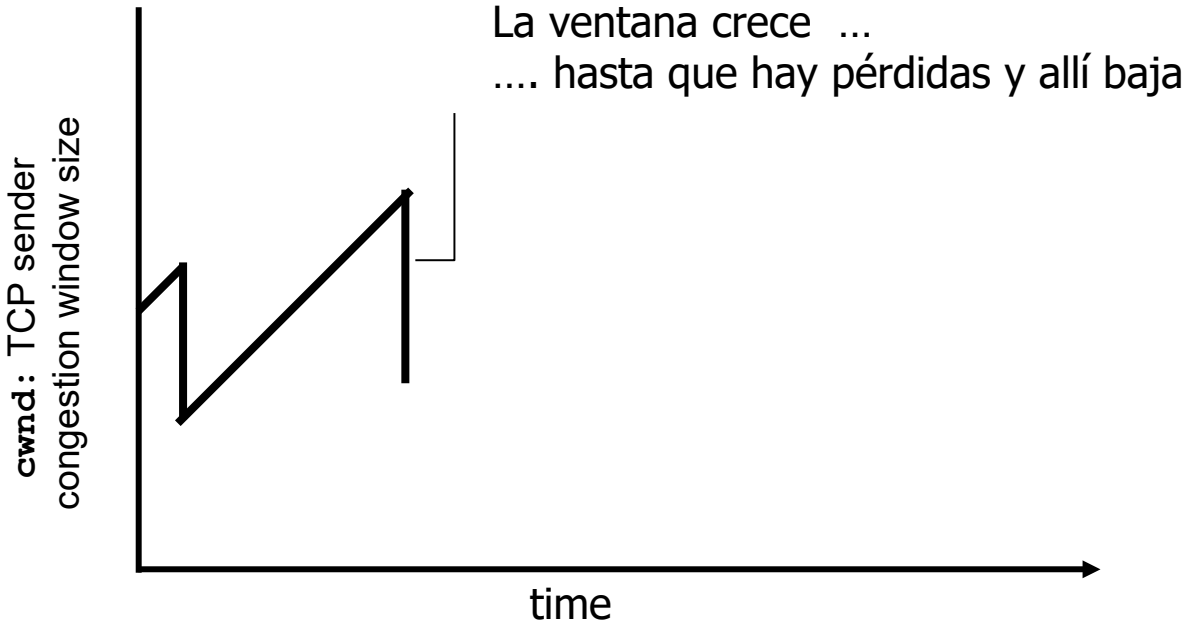
TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



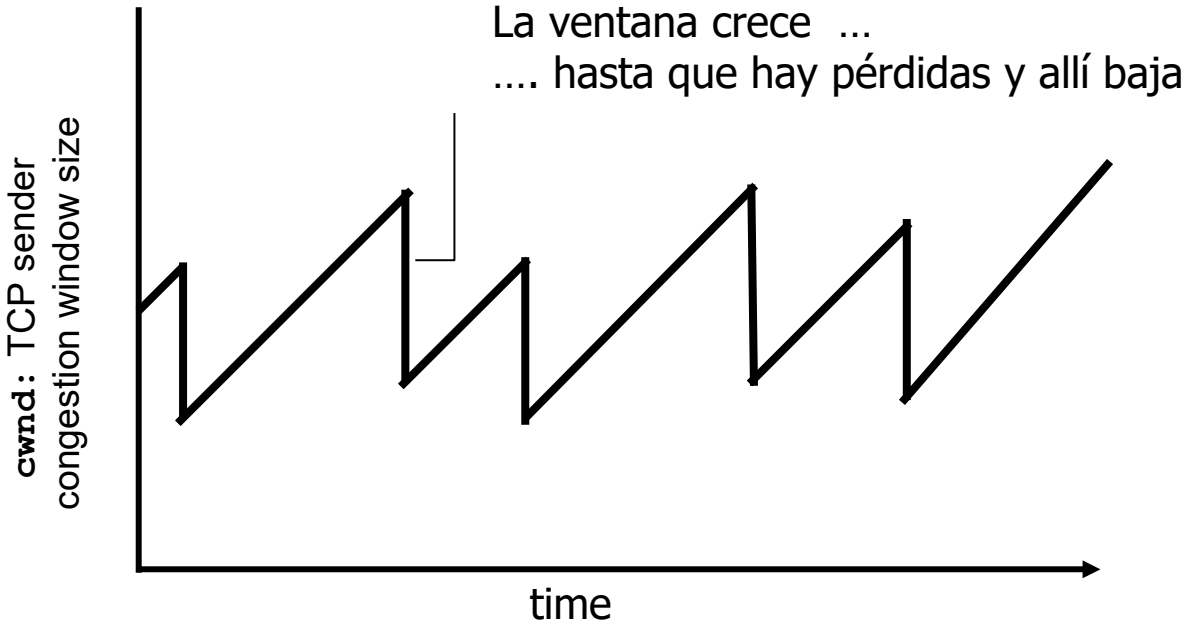
TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



TCP – Control de Congestión y diente de sierra

- El mecanismo de prueba para encontrar la congestión, aumentar el envío hasta detectar una pérdida.
- Si consideramos la fase de “congestion avoidance”:
  - Incremento aditivo de cwnd en 1 MSS cada RTT
  - Decremento multiplicativo, cwnd la mitad de cwnd antes de la pérdida.
  - Produce un efecto de **diente de sierra**, que puede “sincronizarse” entre los transmisores de segmentos que compartan el punto de congestión.



TCP – Ventana de Congestión

- El gráfico es una versión simplificada
  - “Todo los segmentos” de forma instantánea (idealización cuando el tiempo de serialización es mucho menor que el valor de RTT).
  - Observar que el comportamiento esta basado en RTT, que varía dinámicamente.
  - ¿Cómo elijo el valor de los temporizadores para los timeout?

TCP

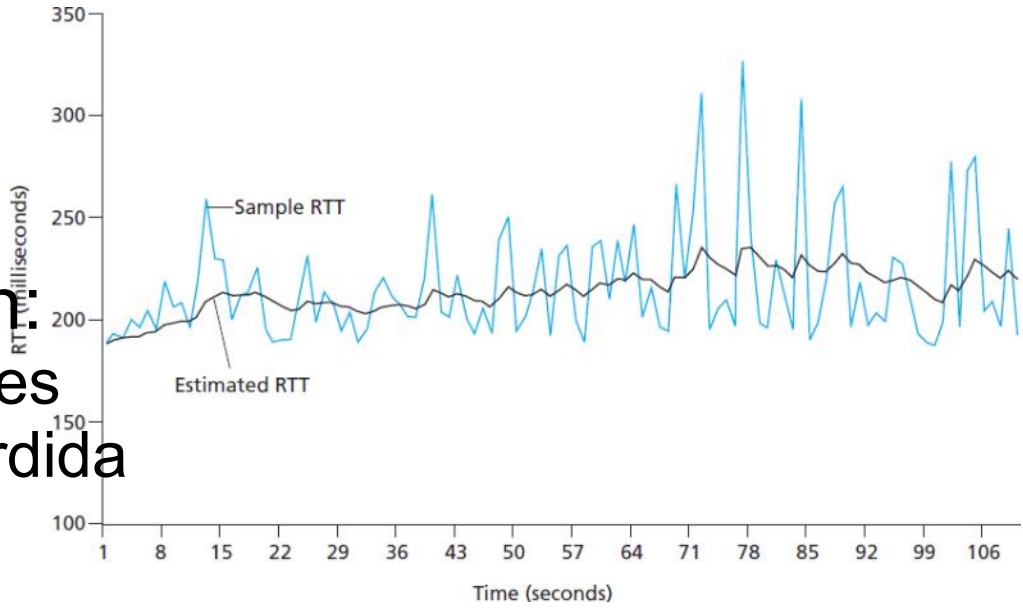
- Inicio y fin de conexión
- Manejo de números de secuencia
- Control de Flujo
- Control de Congestión
- Estados y temporizadores

TCP – Gestión de Temporizadores

- Varios temporizadores
  - el más importante es el de retransmisión
- **Jacobson:**
  - RTT[n] = a RTT[n-1] + (1 - a) M[n] a = 7/8
  - D[n] = b D[n-1] + (1 – b) |RTT[n-1] – M[n]| b = 3/4
  - Timeout = RTT + 4 * D
- Variantes agregadas por Karn:
  - No calcular sobre retransmisiones
  - Se duplica el timeout a cada pérdida
- Moderno: se utiliza la opción Timestamp para mejorar el cálculo del RTT
- Hay varios temporizadores más.

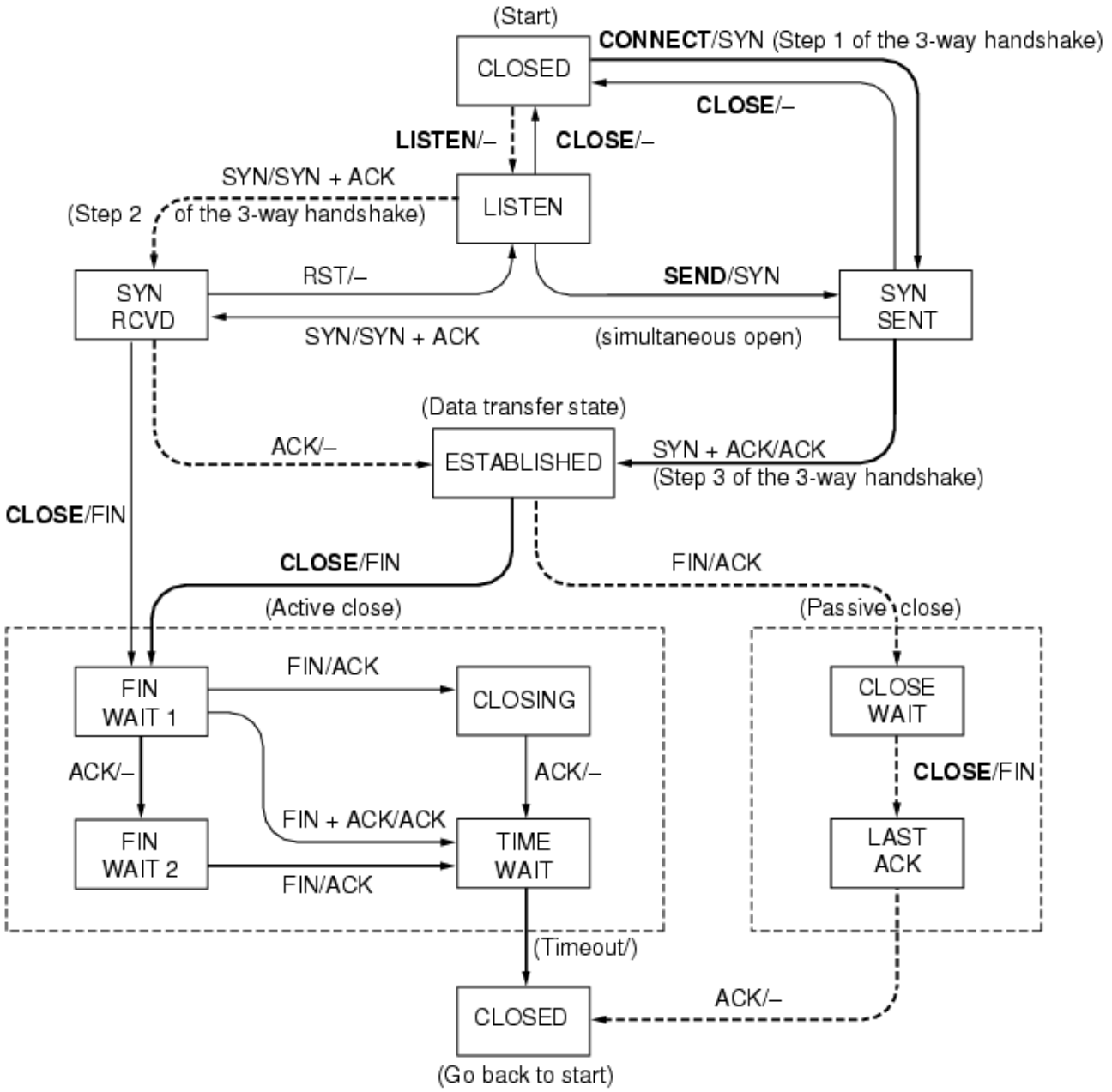
TCP – Gestión de Temporizadores

- Varios temporizadores
  - el más importante es el de retransmisión
- **Jacobson:**
  - RTT[n] = a RTT[n-1] + (1 - a) M[n] a = 7/8
  - D[n] = b D[n-1] + (1 – b) |RTT[n-1] – M[n]| b = 3/4
  - Timeout = RTT + 4 * D



- Variantes agregadas por Karn:
  - No calcular sobre retransmisiones
  - Se duplica el timeout a cada pérdida
- Moderno: se utiliza la opción Timestamp para mejorar el cálculo del RTT
- Hay varios temporizadores más.

TCP – Estados


(Start)
CLOSED
CONNECT/SYN (Step 1 of the 3-way handshake)
CLOSE/–
LISTEN/–
CLOSE/–
SYN/SYN + ACK
(Step 2 of the 3-way handshake)
LISTEN
SYN RCVD
RST/–
SEND/SYN
SYN SENT
SYN/SYN + ACK
(simultaneous open)
(Data transfer state)
ACK/–
ESTABLISHED
SYN + ACK/ACK
(Step 3 of the 3-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
(Active close)
(Passive close)
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/–
ACK/–
CLOSE/FIN
FIN + ACK/ACK
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
(Timeout/)
CLOSED
ACK/–
(Go back to start)

TCP – Significado de los Estados

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

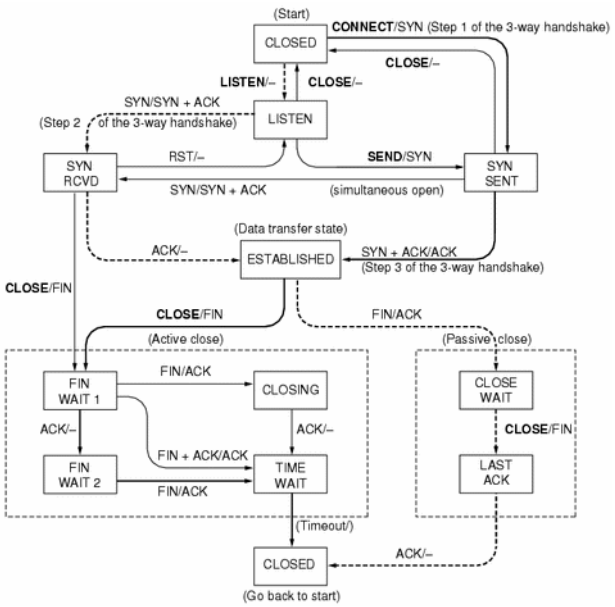
TCP – Transisión de estados a conexión Establecida

client state

CLOSED

server state

LISTEN



TCP – Transisión de estados a conexión Establecida

client state

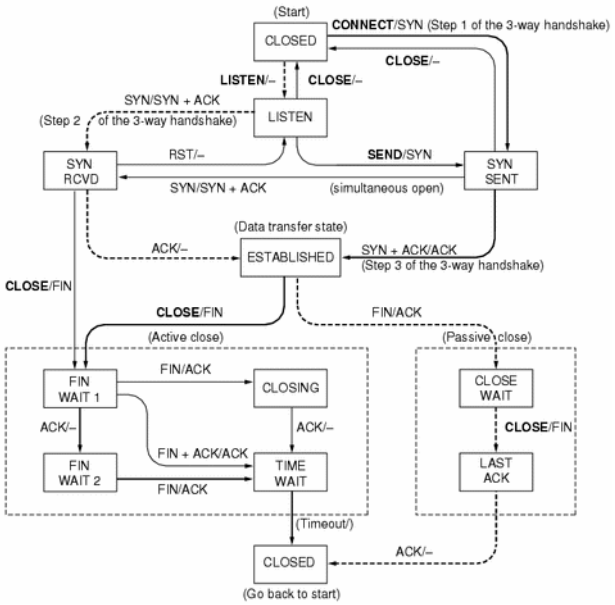
CLOSED

server state

LISTEN

choose init seq num, x
send TCP SYN msg

SYNbit=1, Seq=x



TCP – Transisión de estados a conexión Establecida

client state

CLOSED

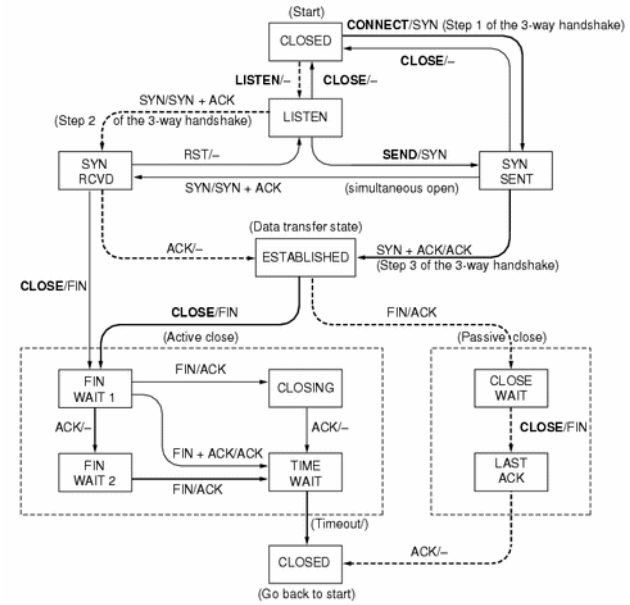
choose init seq num, x
send TCP SYN msg

SYN SENT

SYNbit=1, Seq=x

server state

LISTEN



TCP – Transisión de estados a conexión Establecida

client state

CLOSED

choose init seq num, x
send TCP SYN msg

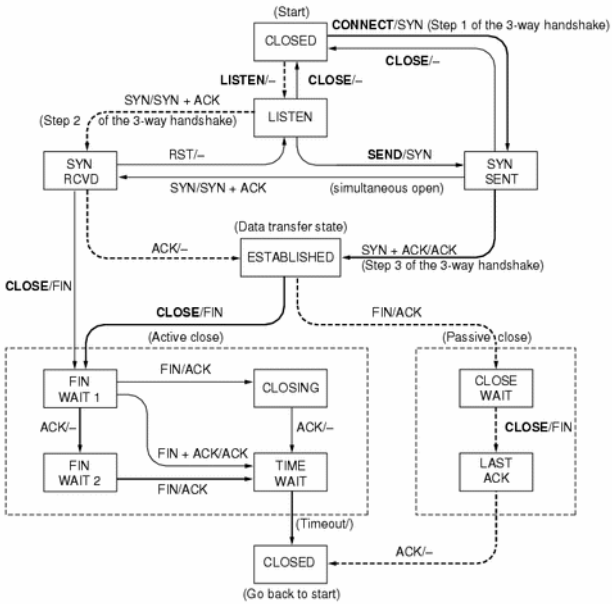
SYN SENT

SYNbit=1, Seq=x

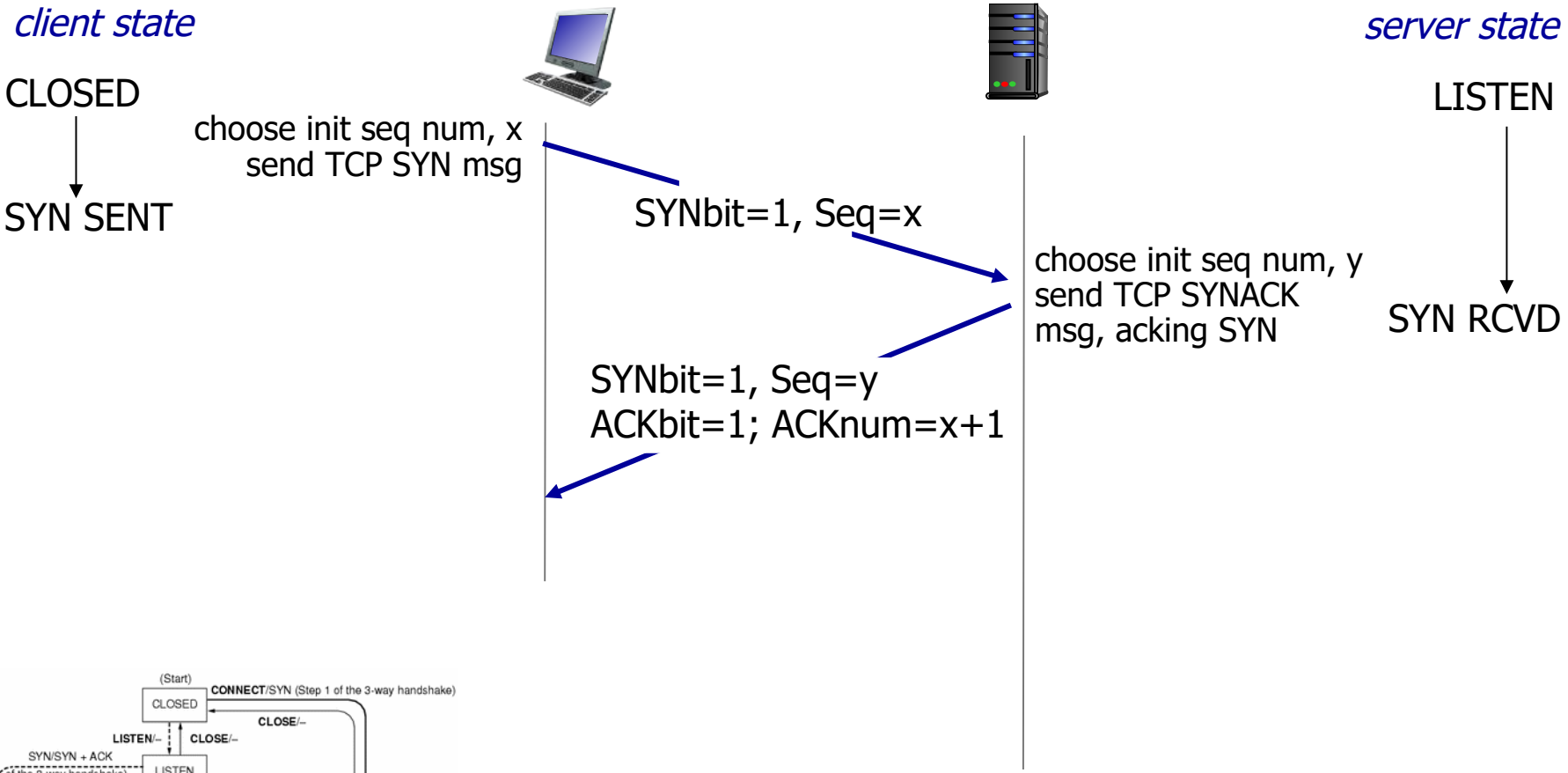
server state

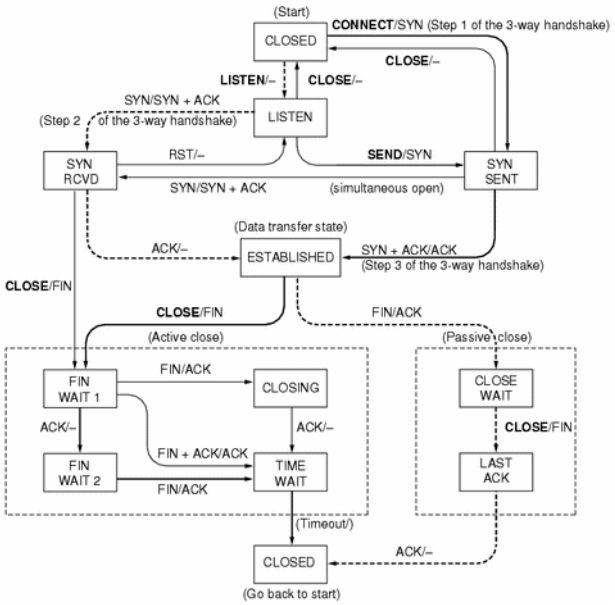
LISTEN

SYN RCVD

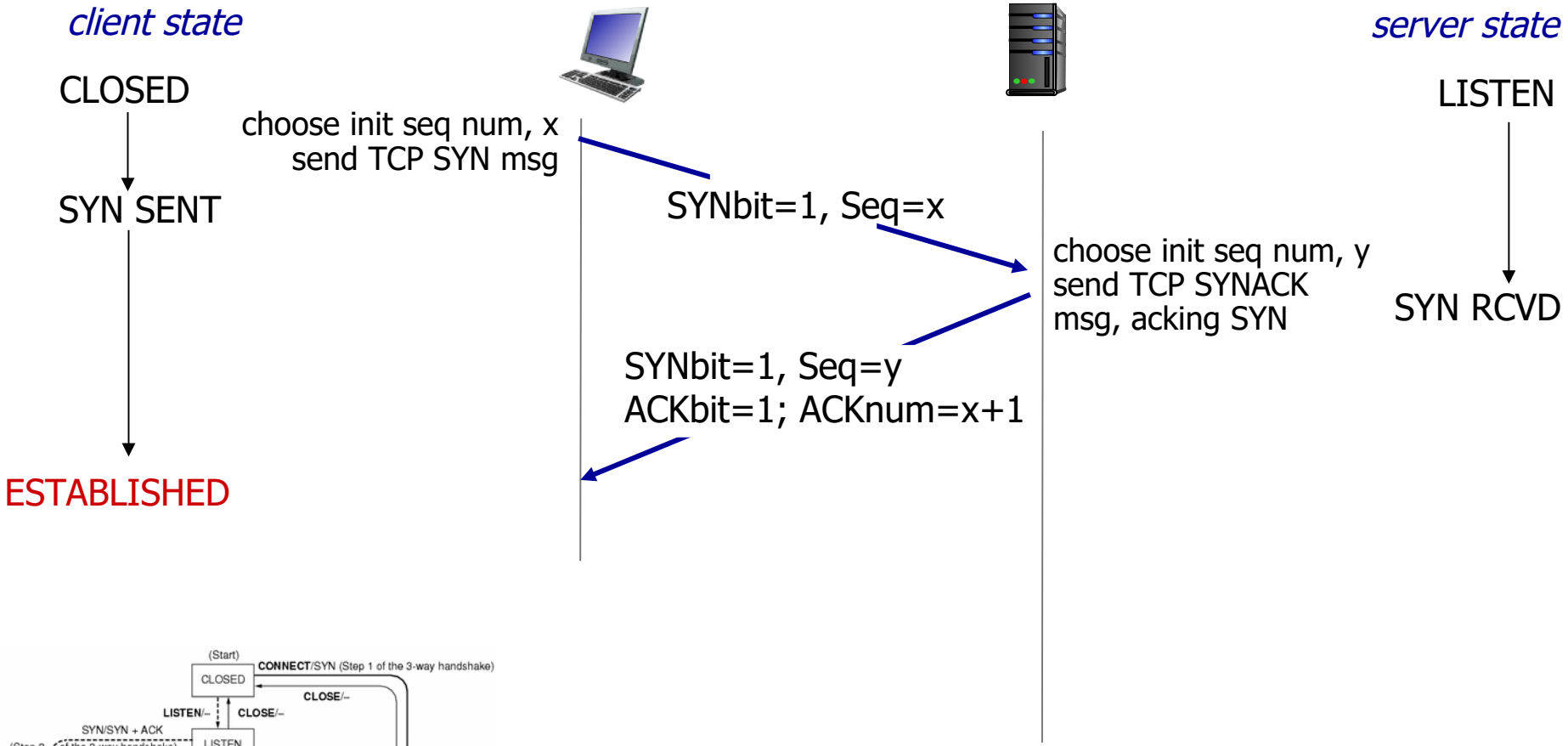


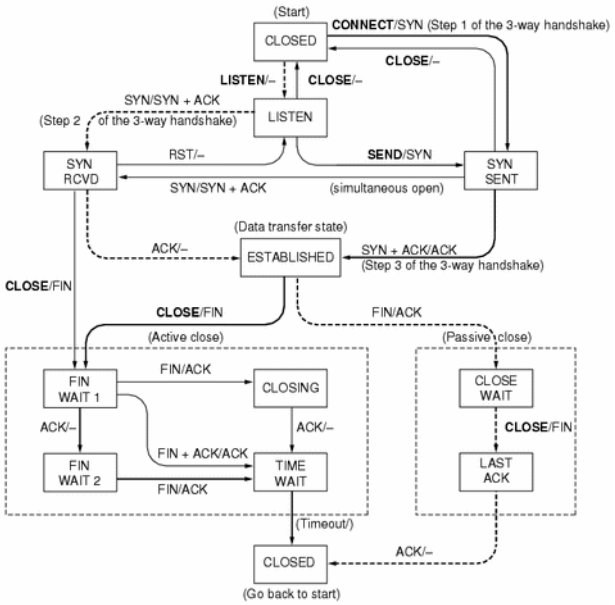
TCP – Transisión de estados a conexión Establecida





TCP – Transisión de estados a conexión Establecida





TCP – Transisión de estados a conexión Establecida

client state

CLOSED

choose init seq num, x
send TCP SYN msg

SYN SENT

SYNbit=1, Seq=x

choose init seq num, y
send TCP SYNACK
msg, acking SYN

server state

LISTEN

SYN RCVD

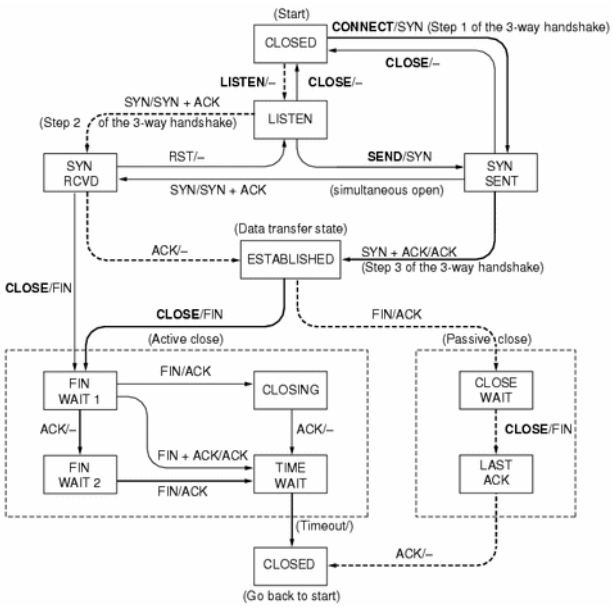
SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

ESTABLISHED

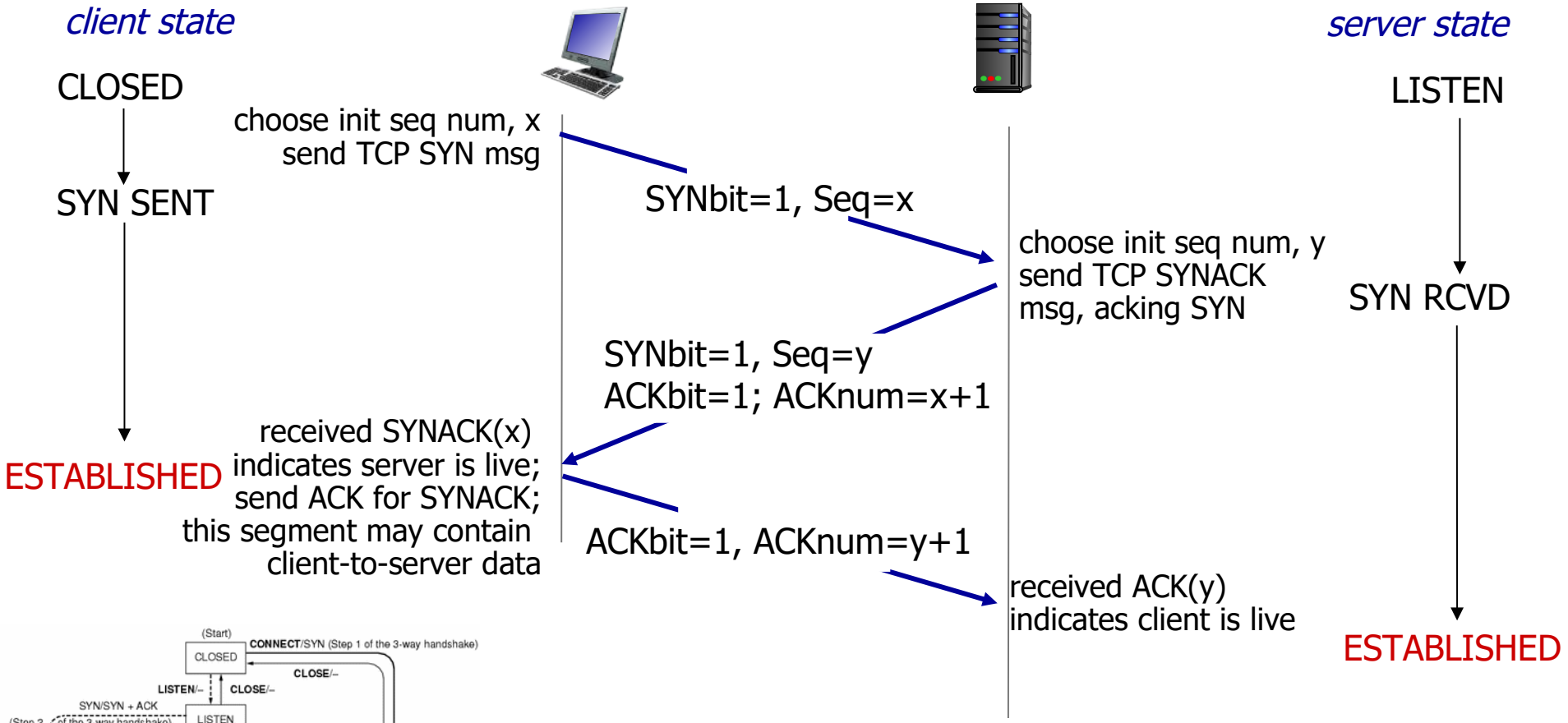
received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

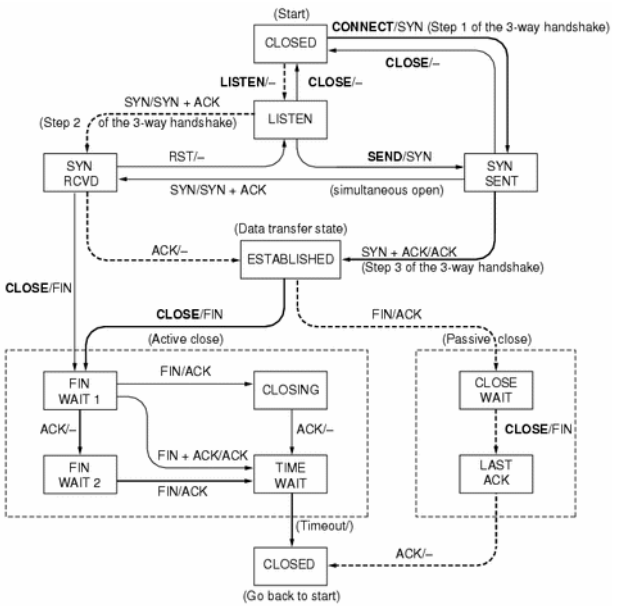
ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live



TCP – Transisión de estados a conexión Establecida





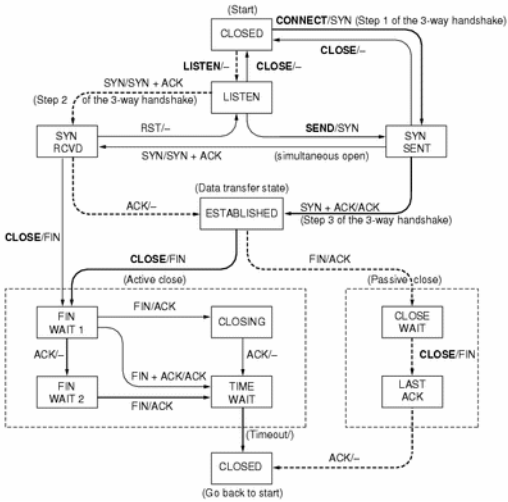
TCP – Transisión de estados a conexión Cerrada (Closed)

client state

ESTABLISHED

server state

ESTABLISHED



TCP – Transisión de estados a conexión Cerrada (Closed)

client state

ESTABLISHED

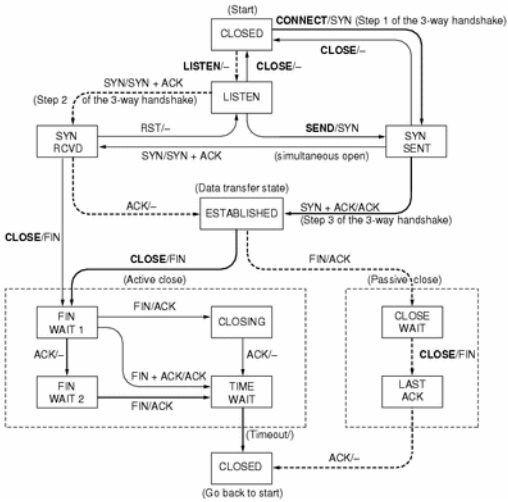
server state

ESTABLISHED

`clientSocket.close()`

can no longer send but can receive data

FINbit=1, seq=x



TCP – Transisión de estados a conexión Cerrada (Closed)

client state

ESTABLISHED

`clientSocket.close()`

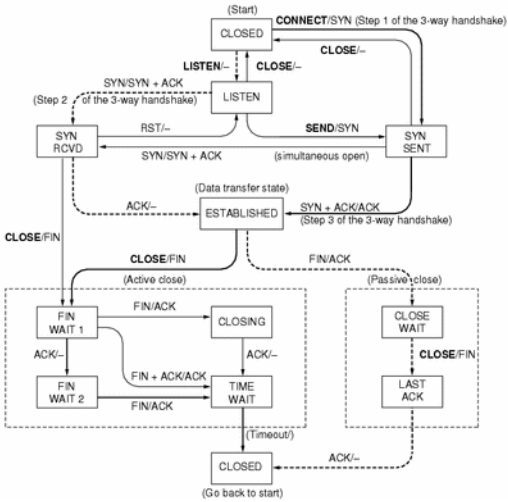
FIN_WAIT_1

can no longer send but can receive data

FINbit=1, seq=x

server state

ESTABLISHED



TCP – Transisión de estados a conexión Cerrada (Closed)

client state

ESTABLISHED

`clientSocket.close()`

FIN_WAIT_1

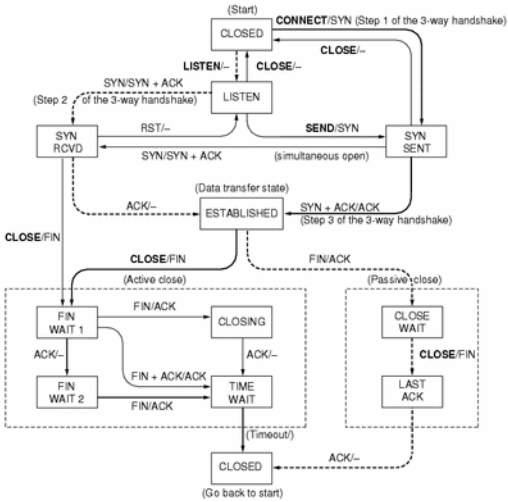
can no longer send but can receive data

FINbit=1, seq=x

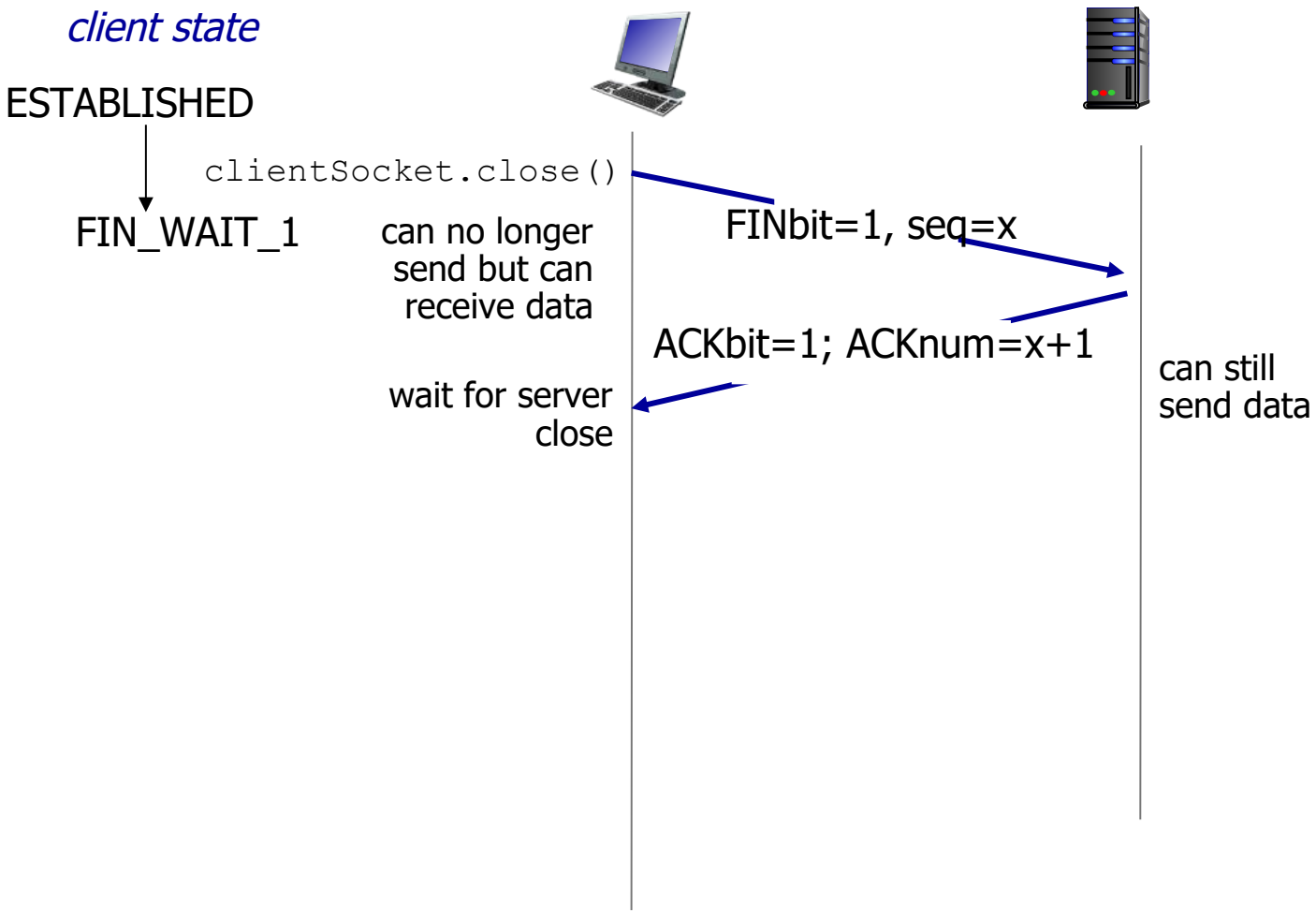
server state

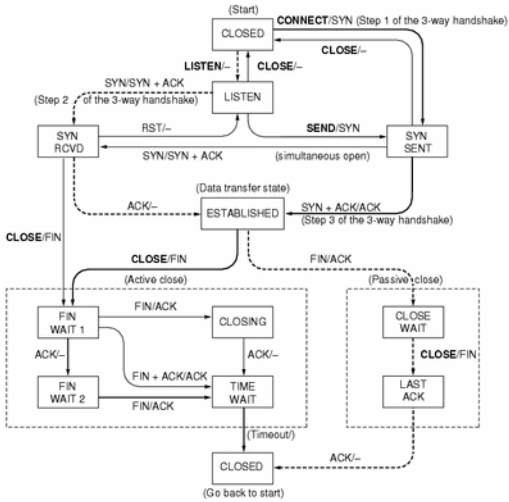
ESTABLISHED

CLOSE_WAIT

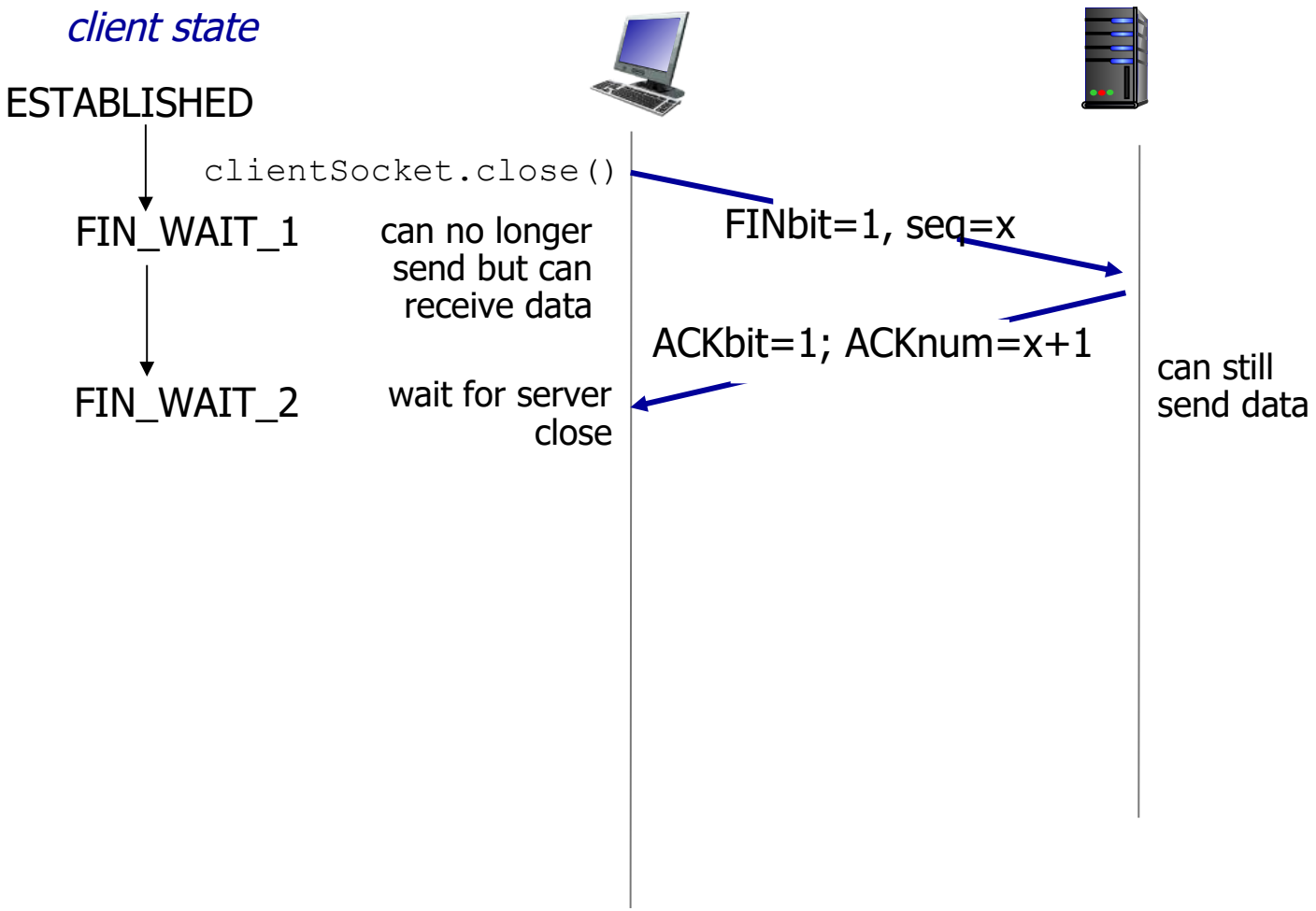


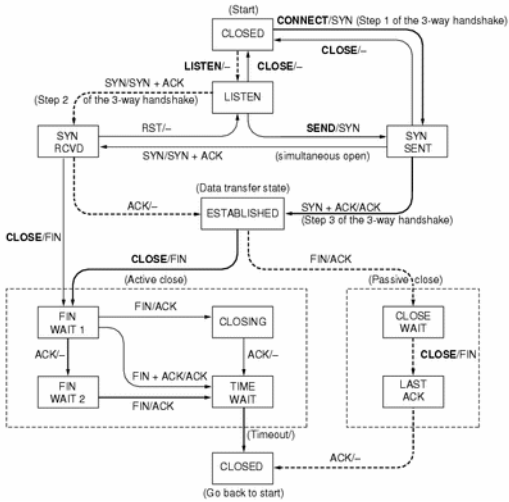
TCP – Transisión de estados a conexión Cerrada (Closed)





TCP – Transisión de estados a conexión Cerrada (Closed)





TCP – Transisión de estados a conexión Cerrada (Closed)

client state

ESTABLISHED

`clientSocket.close()`

FIN_WAIT_1

can no longer send but can receive data

FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

FIN_WAIT_2

wait for server close

can still send data

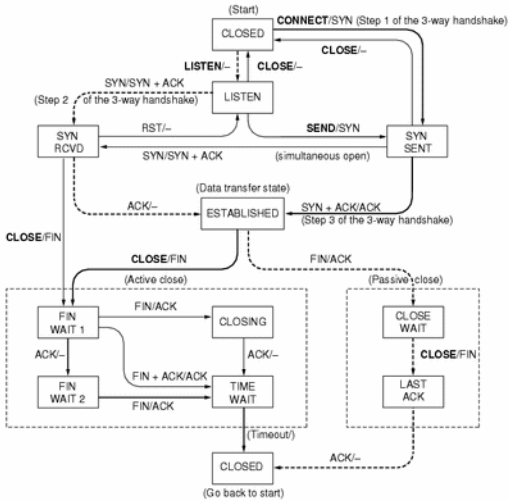
server state

ESTABLISHED

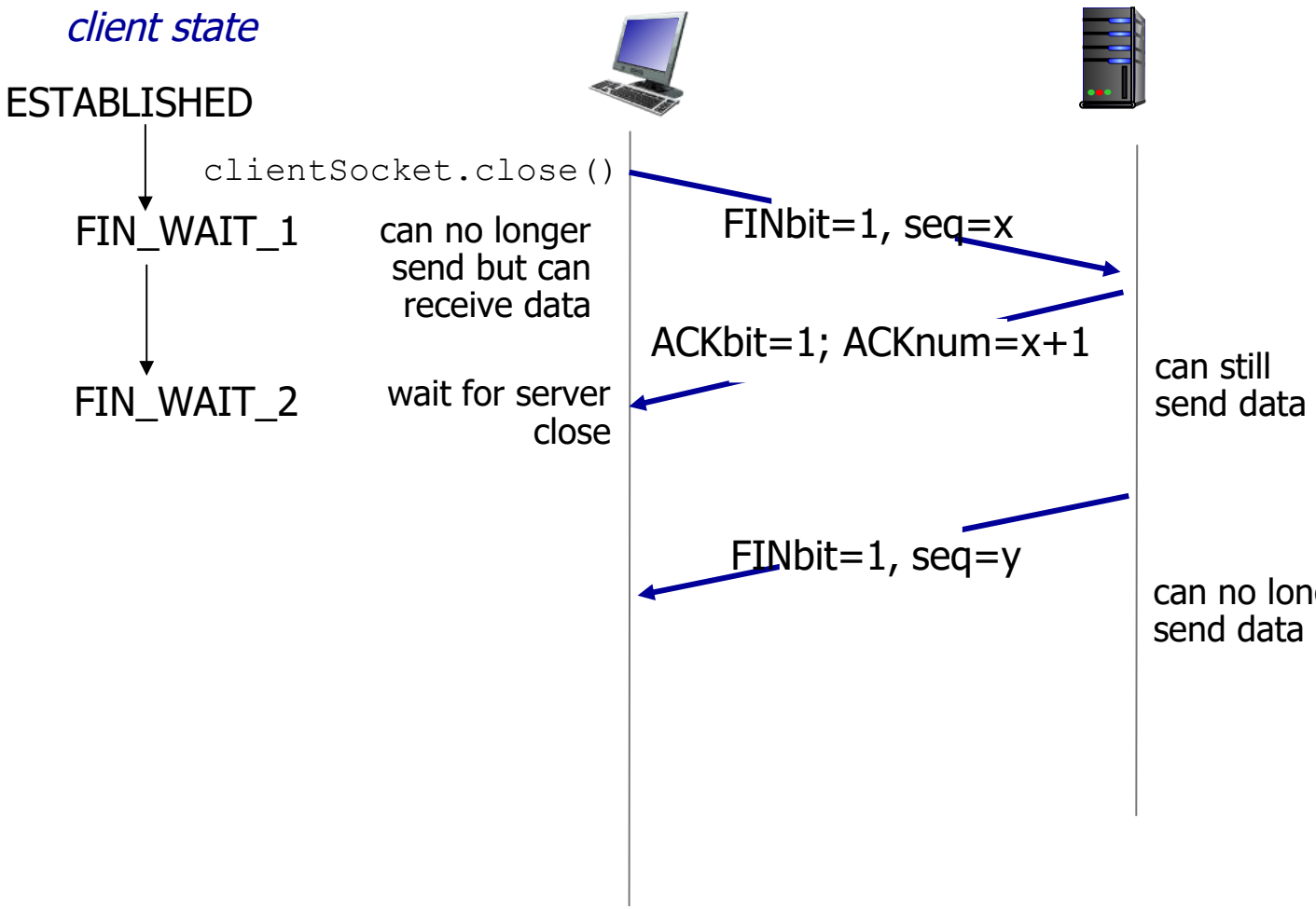
CLOSE_WAIT

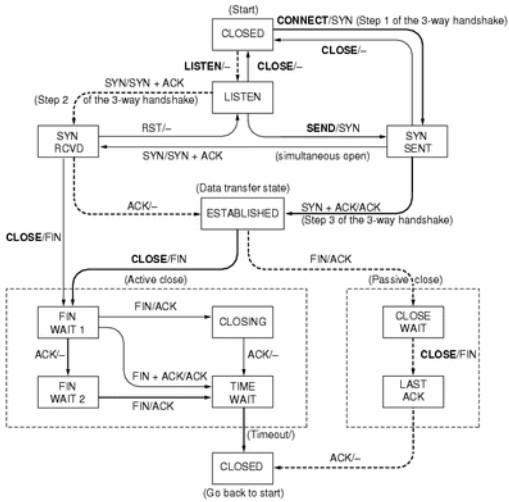
LAST_ACK

can no longer send data

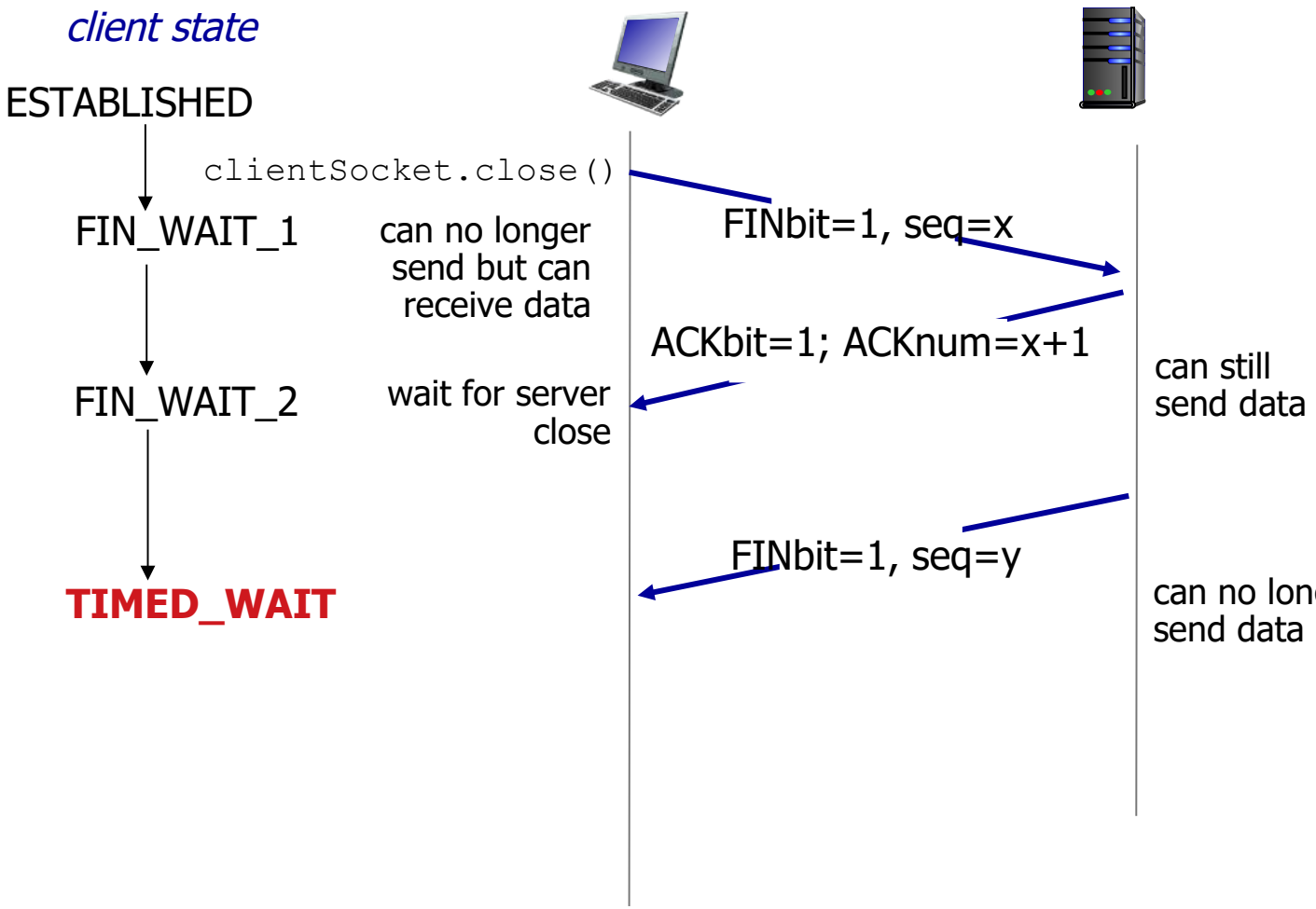


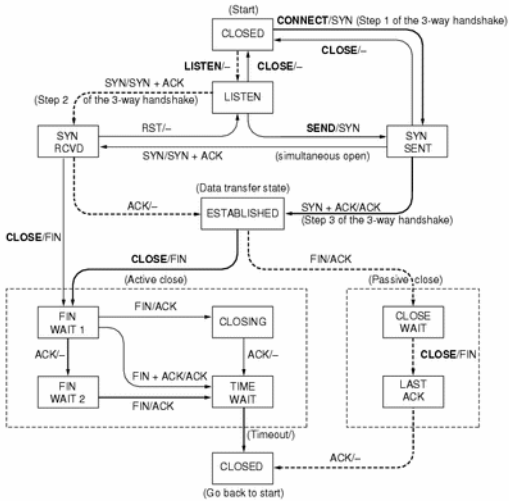
TCP – Transisión de estados a conexión Cerrada (Closed)



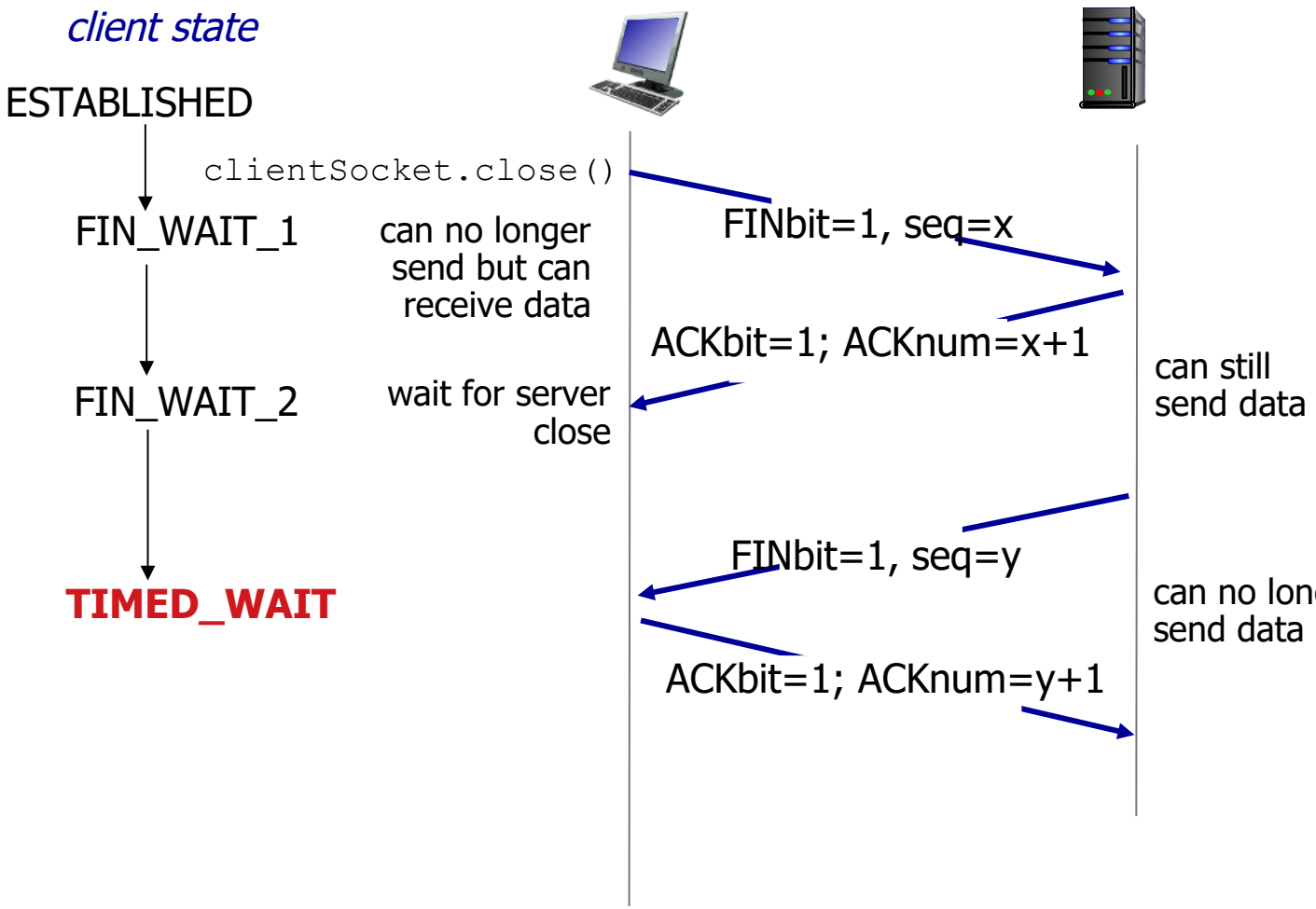


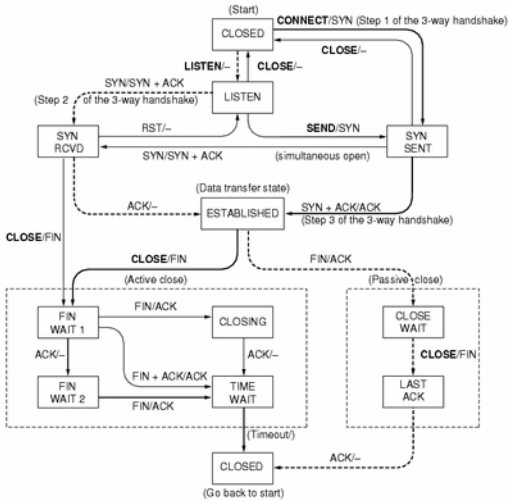
TCP – Transisión de estados a conexión Cerrada (Closed)



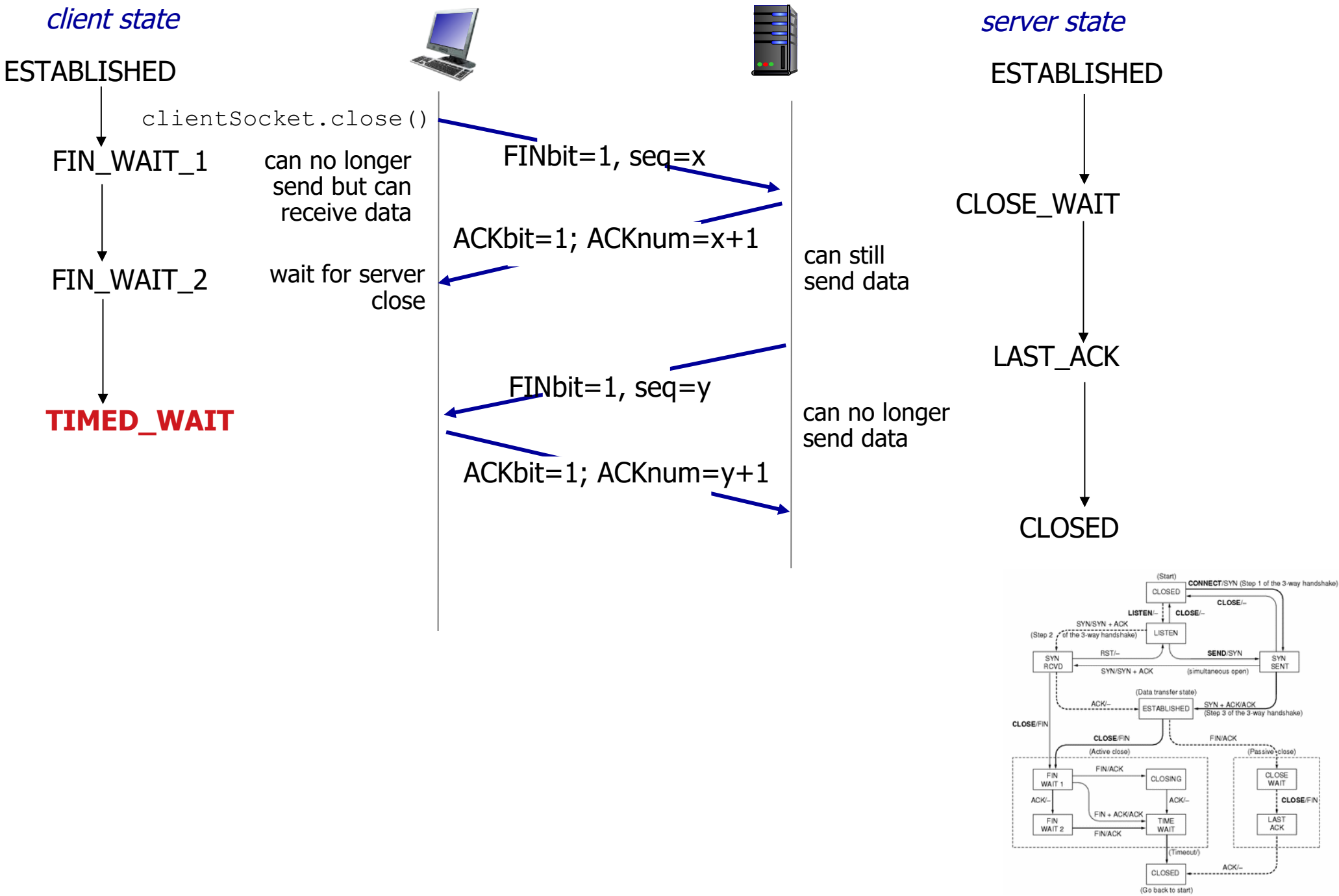


TCP – Transisión de estados a conexión Cerrada (Closed)

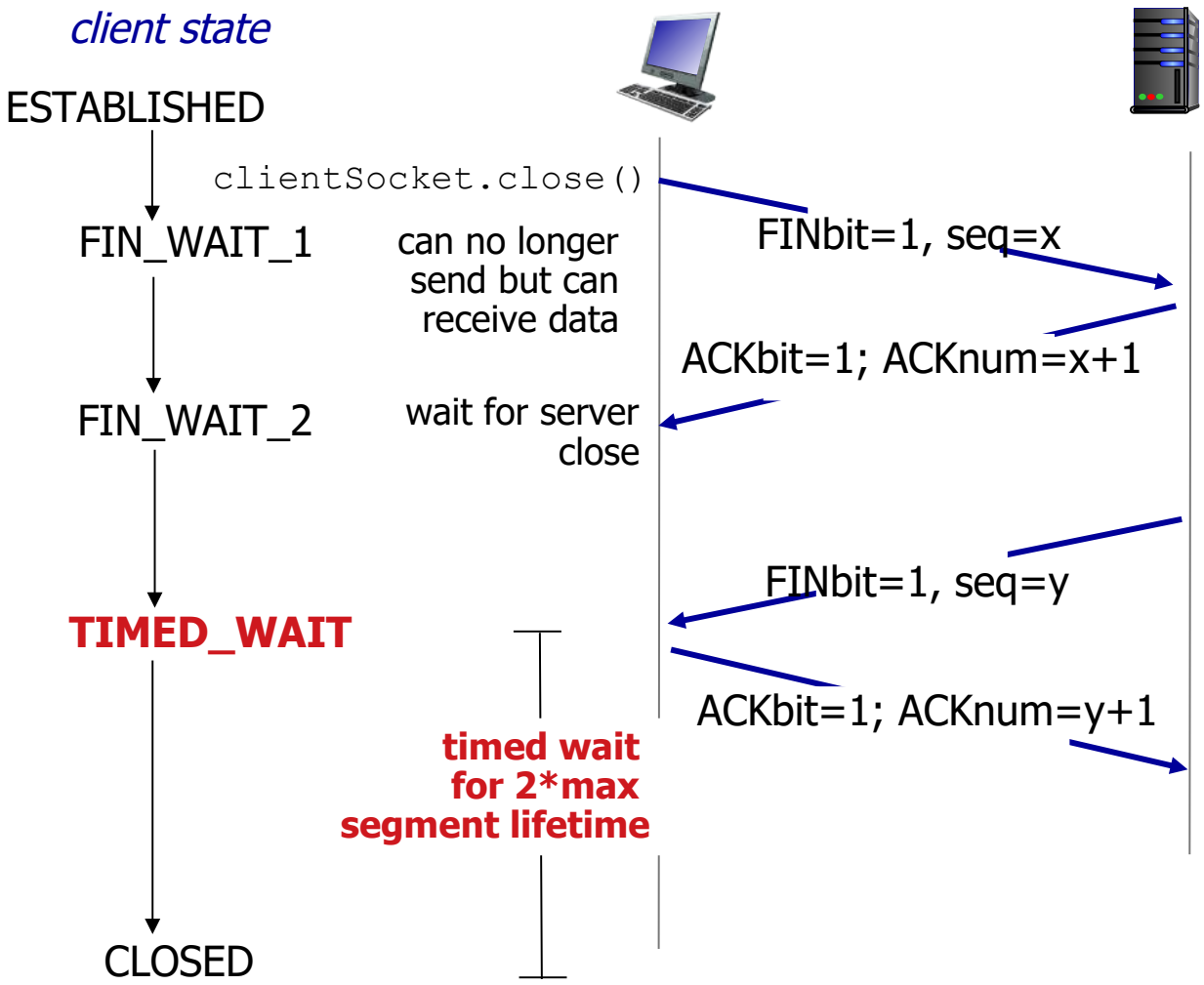


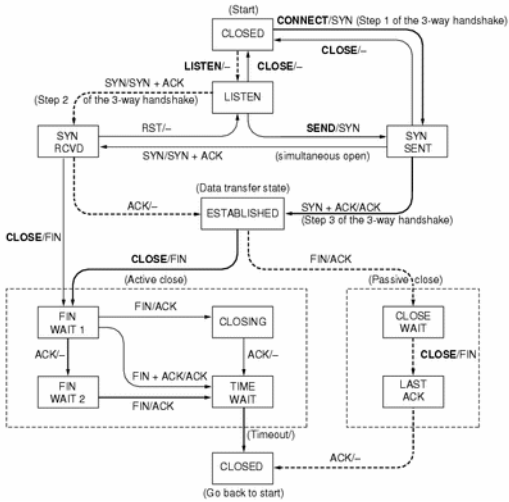


TCP – Transisión de estados a conexión Cerrada (Closed)



TCP – Transisión de estados a conexión Cerrada (Closed)





TCP – Modificaciones

- Window Scale: ventana máxima de 64 KB (representada en 16 bits) “quedó chica”
  - Calculen la velocidad máxima de transmisión en un enlace de 1Gbps y 100ms de RTT con una ventana de receptor de 64 KB
  - Opción "escala de ventana“: indica cuantos bits "0" agregar a la derecha del valor de buffer
- SACK (Reconocimiento selectivo)
  - Permite indicar qué segmentos se recibieron correctamente posteriores al indicado en el campo ACK del segmento

TCP – Modificaciones

- Enlace de 56 Kbps -> los números de secuencia se reutilizan en 3.6 días
- 1 Gbps -> se reutilizan en 17 segundos!!
  - Tiempo de vida máximo asumido por TCP de un paquete en la red: 120 Segundos!
- Solución: Se utiliza el número de secuencia junto con una opción Timestamp para detectar duplicados viejos
  - Receptor debe verificar que el valor de timestamp sea monótono creciente, descartando segmentos antiguos
  - Timestamp también sirve para mejorar el cálculo del RTT