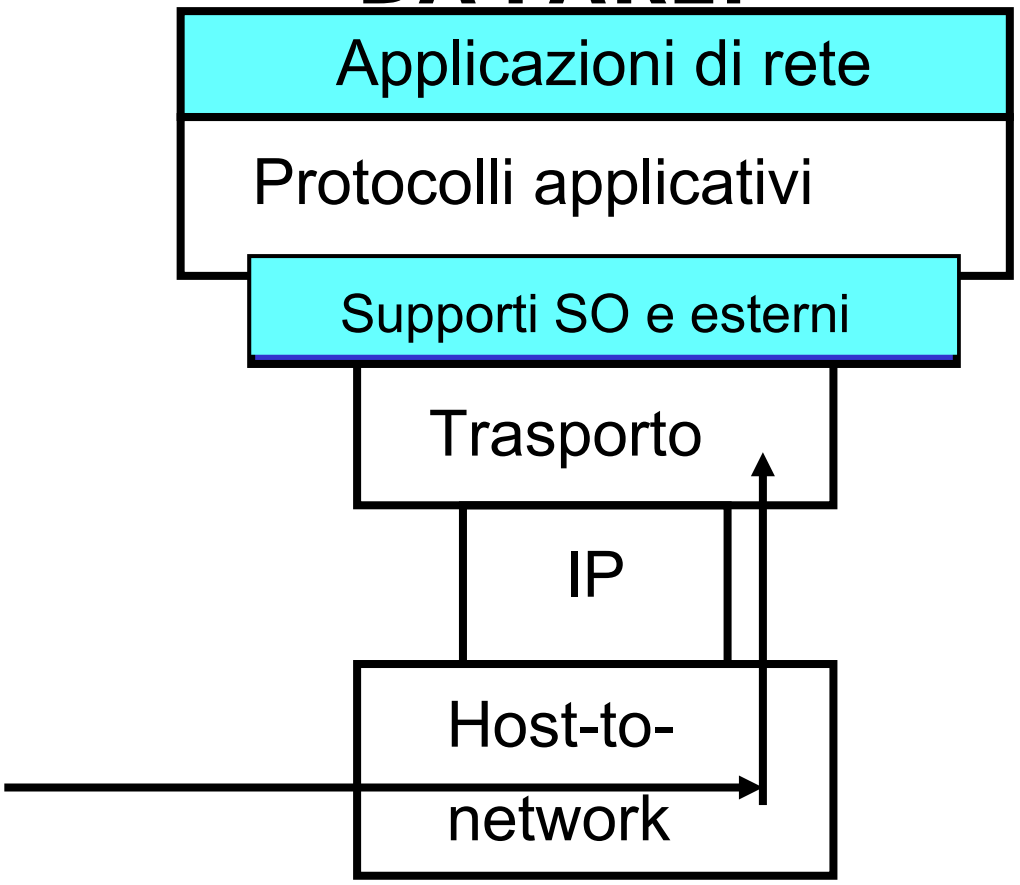
PARTE 5a

LIVELLO TRASPORTO

Dove ci troviamo?

- **Abbiamo introdotto i termini ed i concetti fondamentali del corso**
- **Abbiamo trattato il livello h2n**
- **Abbiamo trattato il livello IP**

DA FARE:



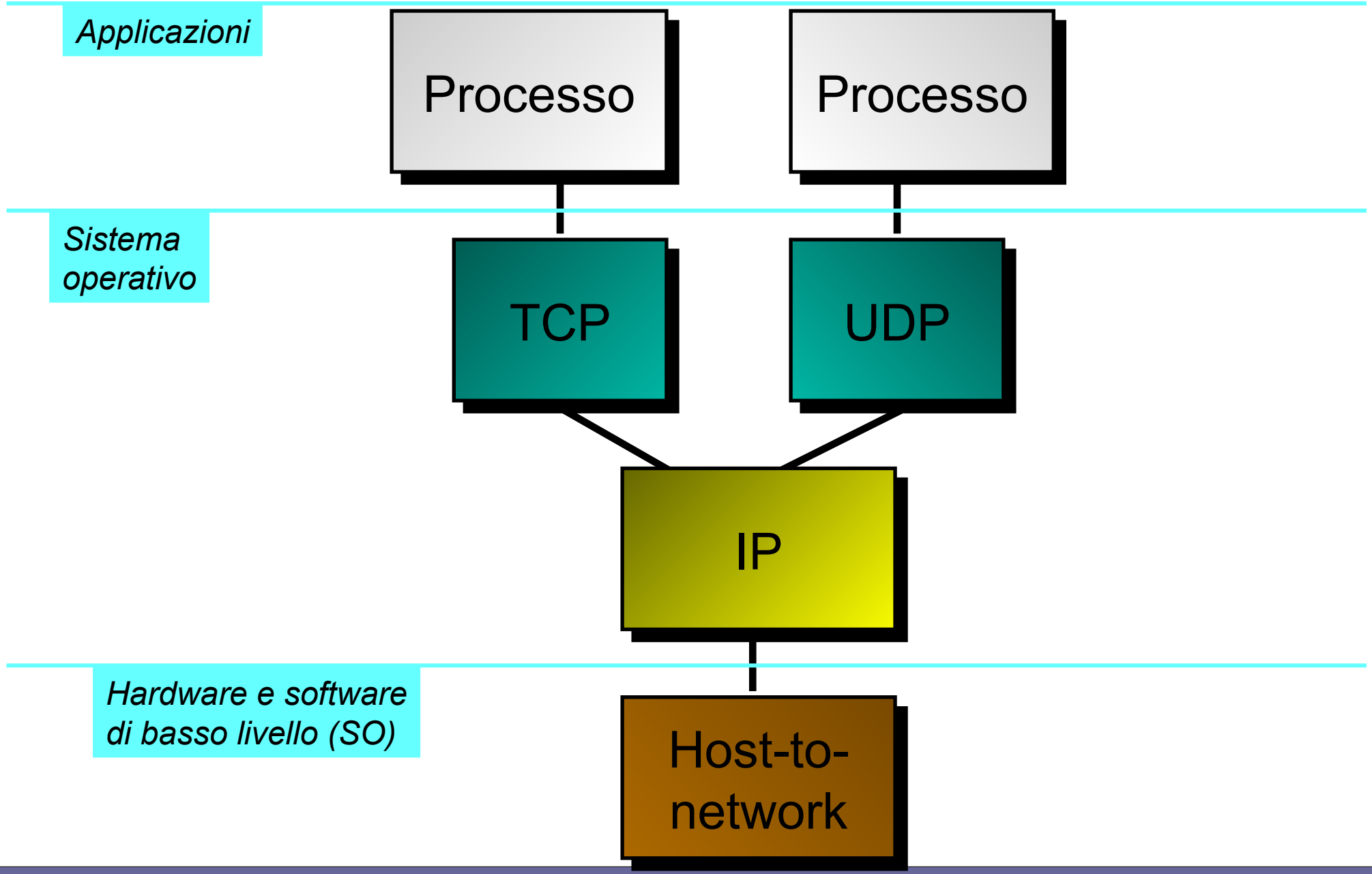
Livello 4 (transport)

- **Il livello trasporto estende il servizio di consegna con impegno proprio del protocollo IP tra due host terminali ad un servizio di consegna a due processi applicativi in esecuzione sugli host**
- **Se IP è il protocollo di rete, TCP sarà il protocollo di trasporto? → NO!**
- **TCP è solo un componente del livello di trasporto della suite TCP/IP**
- **L'altro componente è costituita dal protocollo UDP (User Datagram Protocol)**

Livello 4 (transport)

- **Il livello transport estende il servizio di consegna con impegno proprio del protocollo IP tra due host terminali ad un servizio di consegna a due processi applicativi in esecuzione sugli host**
- **Servizi aggiuntivi rispetto a IP**
  - multiplazione e demultiplazione messaggi tra processi
  - rilevamento dell'errore (mediante checksum)
- **Esempi di protocolli transport**
  - UDP (User Datagram Protocol)
  - TCP (Transmission Control Protocol): offre servizi aggiuntivi rispetto a UDP

Livelli dello stack TCP/IP: chi gestisce?



Servizi del livello di trasporto

Servizi comuni a UDP e TCP:

Estensione del servizio di consegna del protocollo IP tra due nodi terminali ad un servizio di consegna a due processi applicativi in esecuzione sui nodi terminali

- **multiplazione e demultiplazione**
- **rilevamento dell'errore (non correzione!)**

Servizi del livello di trasporto

Servizi aggiuntivi di TCP:

- **Trasferimento affidabile dei dati**
  - → controllo di flusso, numeri di sequenza, acknowledgement e timer
- **Controllo di congestione**
  - → regola il tasso di invio dei segmenti da parte del mittente

Modulo 1: Multiplazione e demultiplazione

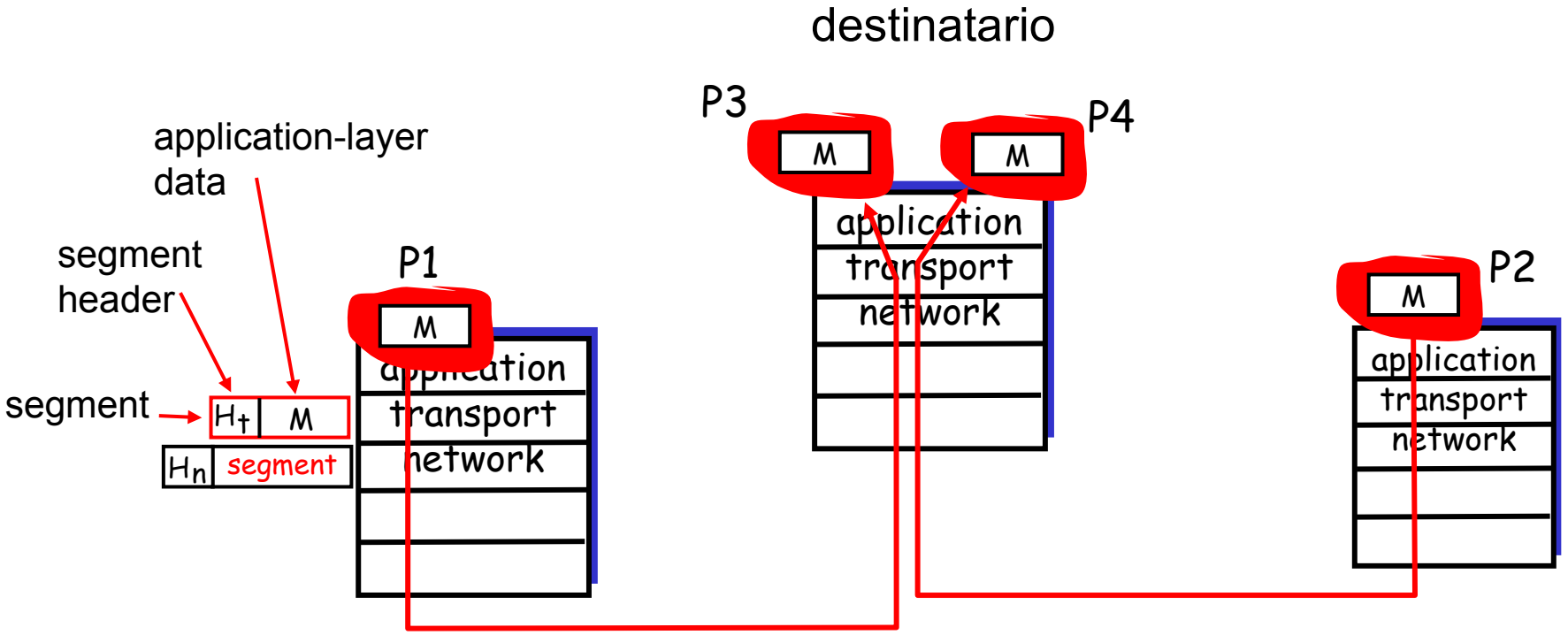
1. Multiplazione e demultiplazione

- **Il protocollo IP non consegna i dati tra processi applicativi in esecuzione sui nodi terminali (un indirizzo IP per identificare ogni interfaccia di un nodo terminale)**
  - → compito del protocollo di trasporto
- **Ogni segmento dello strato di trasporto possiede un campo contenente l'informazione usata per determinare a quale processo deve essere consegnato il segmento**
  - → demultiplazione
- **La demultiplazione avviene dal lato del nodo destinatario**

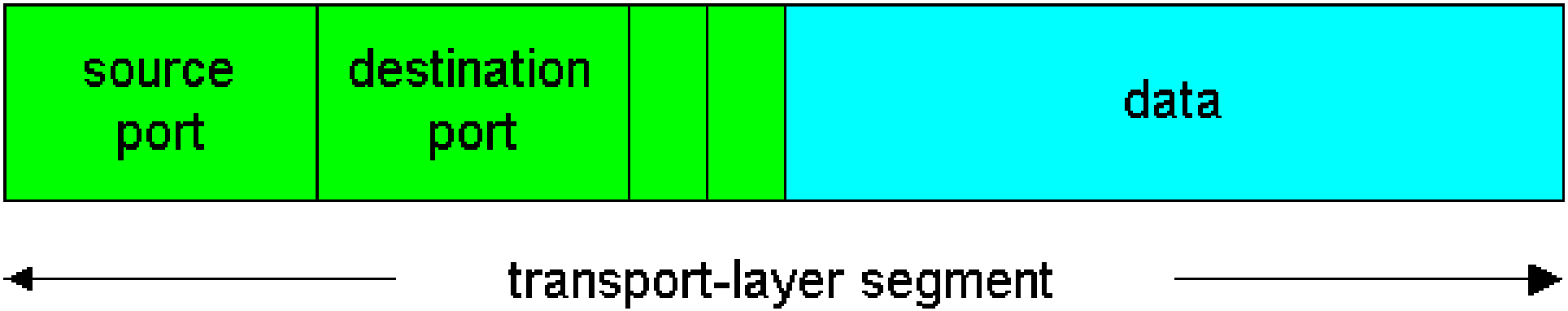
1. Multiplazione e demultiplazione

- **Creazione dei segmenti provenienti dai messaggi di diversi processi applicativi**
  - → multiplazione
- **La multiplazione avviene dal lato del nodo mittente**

Esempio


destinatario
P3
P4
P1
P2
application-layer data
segment header
segment
M
application
transport
network
H_t
H_n
segment

Multiplazione e demultiplazione



UDP e TCP attuano la multiplazione/demultiplazione includendo due campi speciali nell'header del segmento:

- **il numero di porta del mittente**
- **il numero di porta del destinatario**

Permettono di identificare in modo univoco i due processi applicativi, residenti su due nodi terminali e comunicanti tra loro

Multiplazione e demultiplazione

- **Numero di porta: numero di 16 bit compreso tra 0 e 65535**
- **Numeri di porta noti (well-known ports, assigned numbers in [RFC 1700]): tra 0 e 1023 riservati per protocolli applicativi noti (ad es., HTTP e FTP)**
  - HTTP: numero di porta 80
  - Telnet: numero di porta 23
  - SMTP: numero di porta 25
  - DNS: numero di porta 53

Multiplazione e demultiplazione

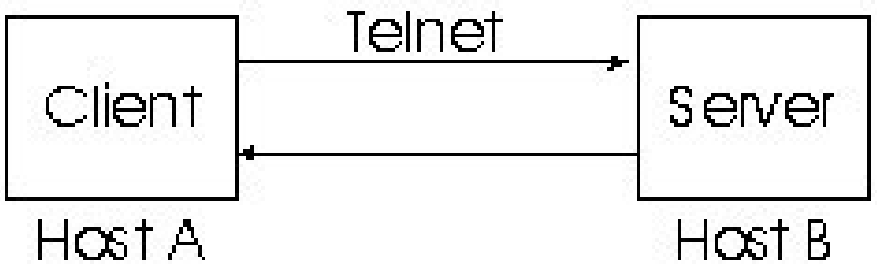
- **File /etc/services su sistemi Unix per conoscere i numeri di porta principali**
- **Quando si realizza un nuovo servizio di rete è necessario assegnargli un nuovo numero di porta**
- **Occorre sia il numero di porta del mittente sia quello del destinatario per distinguere processi dello stesso tipo ed in esecuzione negli stessi istanti**

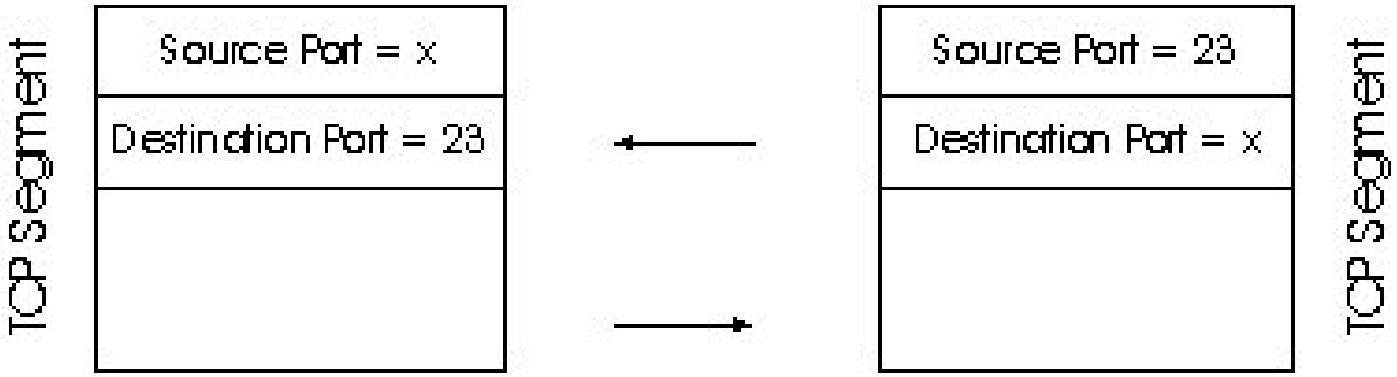
Categorie di numeri di porta

- **0-1023 → Well Known Ports**
  - NON DEVONO essere usate senza una precedente autorizzazione da IANA [RFC4340]. Nella maggior parte dei sistemi, possono essere usate solo da processi con privilegi di root o simili
- **1024-49151 → Registered Ports**
  - NON DEVONO essere usate senza una precedente autorizzazione da IANA [RFC4340]. Nella maggior parte dei sistemi, possono essere usate da qualsiasi processo
- **49152-65535 → Dynamic or Private Ports**

Esempio: telnet

Uso dei numeri di porta in un'applicazione client/server (es. Telnet, con numero di porta 23):





Assegnazione dei numeri di porta

Modello client/server

- **Numero di porta del destinatario nel segmento inviato dal client al server corrisponde al numero di porta del servizio richiesto (ad es., 80 per HTTP)**

- **Numero di porta del mittente nel segmento inviato dal client al server corrisponde ad un numero di porta scelto tra quelli non in uso sul client**

Assegnazione dei numeri di porta

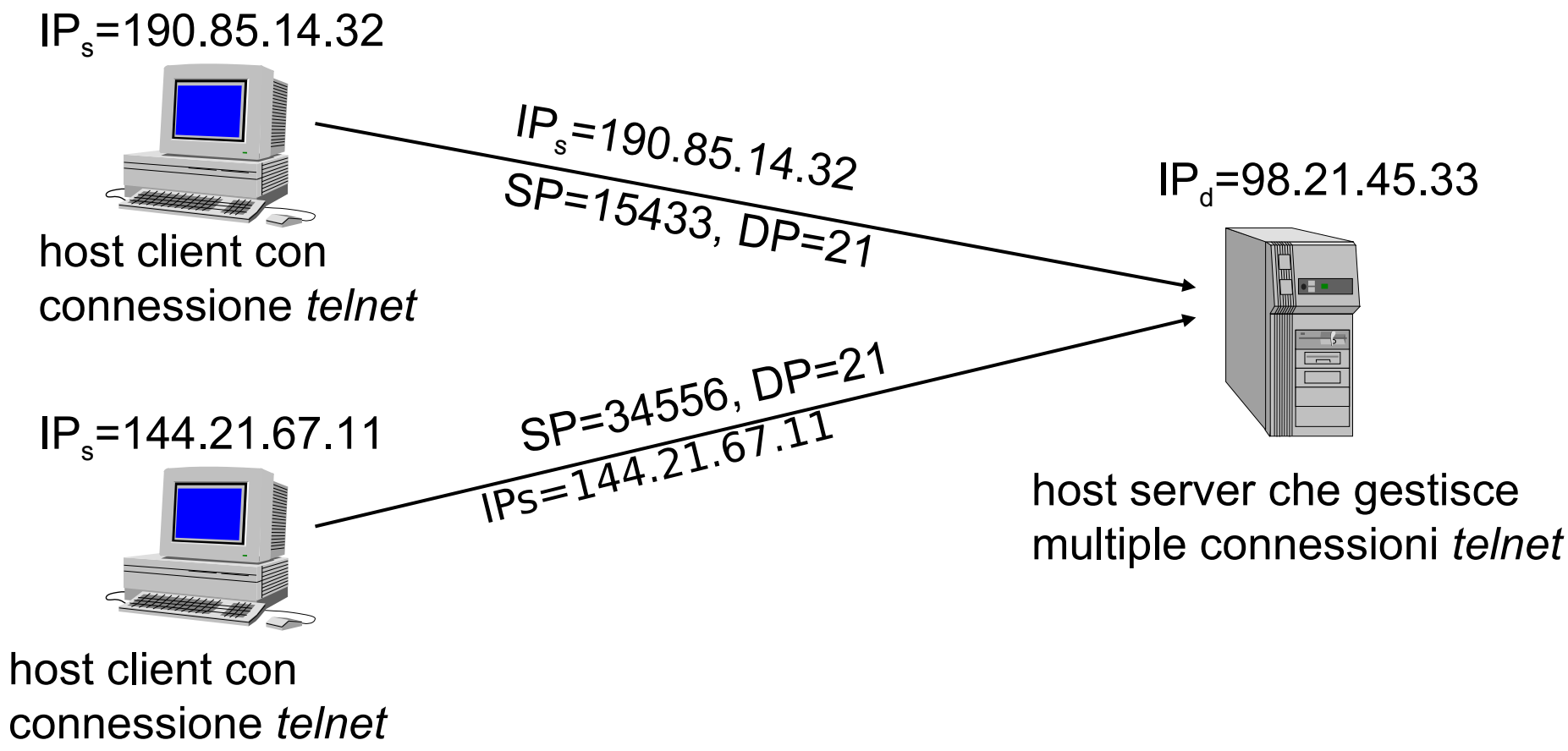
Modello client/server

- **Numero di porta del mittente nel segmento inviato dal server al client corrisponde al numero di porta del servizio richiesto (ad es., 80 per HTTP)**

- **Numero di porta del destinatario nel segmento inviato dal server al client corrisponde al numero di porta indicato dal client nel messaggio precedentemente inviato**

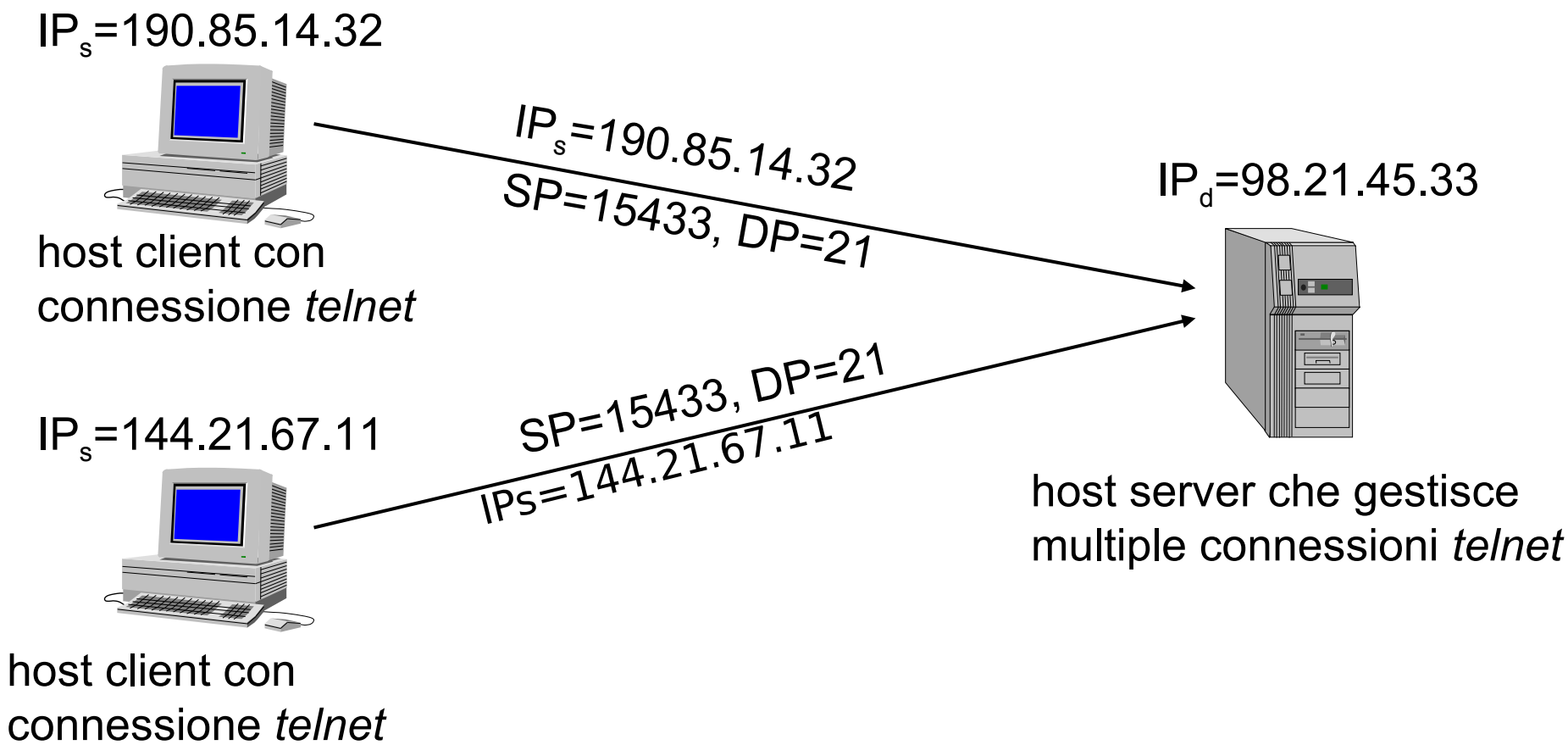
Indirizzo IP e numero di porta

Due processi client, residenti su host diversi per comunicare con lo stesso servizio applicativo sono sempre distinti in base al loro indirizzo IP (sorgente) e possono essere distinti in base al numero di porta sorgente (SP):



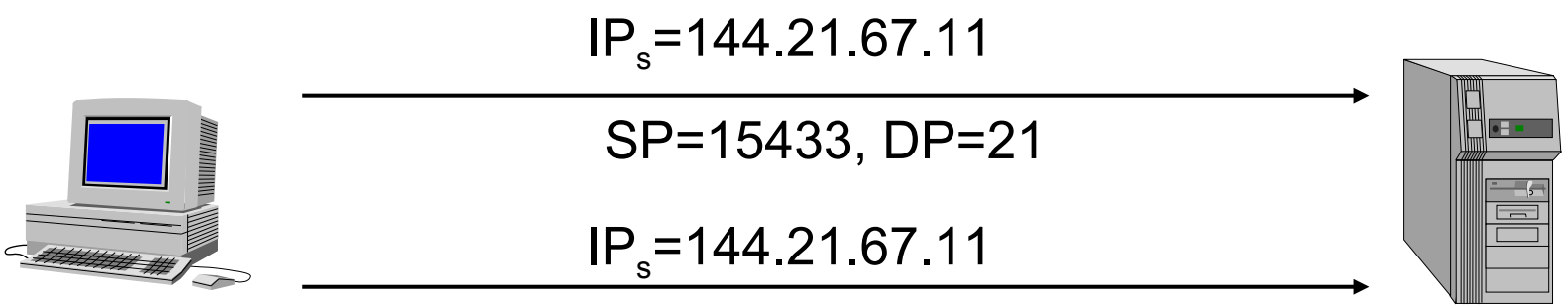
Indirizzo IP e numero di porta

Due processi client, residenti su host diversi e che, per eventualità, usano lo stesso numero di porta sorgente (SP) per comunicare con lo stesso servizio applicativo, sono distinti in base al loro indirizzo IP:



Indirizzo IP e numero di porta

Due processi client, residenti sullo stesso host per comunicare con lo stesso servizio applicativo, non essendo distinti in base al loro indirizzo IP, useranno diversi numeri di porta sorgente (SP) grazie al sistema operativo:



host client con due connessioni *telnet*

host server che gestisce multiple connessioni *telnet*

Indirizzi

- **A livello network (IP)**
  [indirizzo IP1, indirizzo IP2]

- **A livello trasporto (UDP, TCP)**
  [(ind. IP1, porta1), (ind. IP2, porta2)]

Modulo 2: Protocollo UDP

Caratteristiche protocollo UDP (cosa ha)

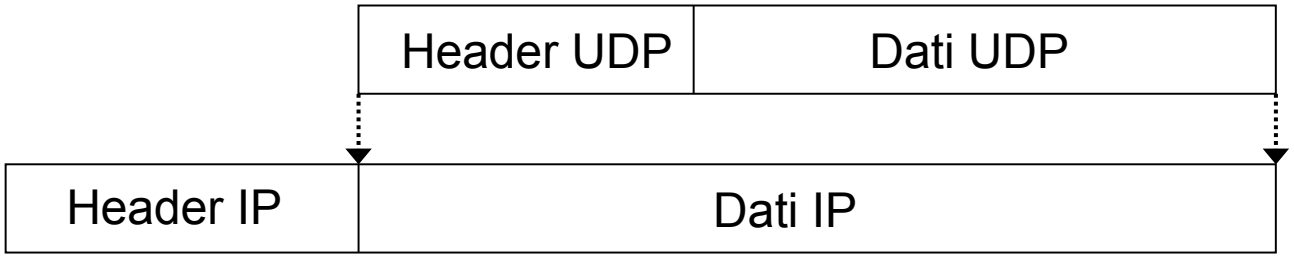
- **User Data Protocol (UDP), definito in**
- **[RFC 768], è un protocollo di trasporto leggero, ovvero dotato delle funzionalità minime del trasporto:**
- **Servizio di multiplazione/demultiplazione**
  - UDP aggiunge al messaggio proveniente dal livello applicativo il numero di porta del mittente e del destinatario
- **Controllo di errore**
  - UDP include nell'header un campo di checksum

Caratteristiche protocollo UDP (cosa non ha)

- **Servizio di consegna non garantito, ma solo di tipo best effort**
  - i segmenti UDP possono essere persi, duplicati, consegnati senza ordine
- **Servizio senza connessione (connectionless)**
  - non vi è handshaking tra mittente e destinatario del segmento UDP
  - ogni segmento UDP è trattato in modo indipendente dagli altri

Formato segmento UDP

Segmento UDP (o *user datagram*) incapsulato in un datagramma IP



Segmento UDP

32 bit

numero porta mittente	numero porta destinatario
lunghezza	checksum
dati dell'applicazione (messaggio)	

Campi del segmento UDP

- **numero di porta del mittente (16 bit)**
- **numero di porta del destinatario (16 bit)**
- **lunghezza (16 bit): dimensione in byte del segmento**
  - lunghezza = header + dati
  - header: dimensione pari a 8 byte

Campi del segmento UDP

- **checksum**
  - non è detto che tutti i link forniscano un servizio di livello 2 per rilevare errori
  - checksum a livello IP limitato all'header del datagram IP
  - non c'è recupero dell'errore (in alcune implementazioni il segmento viene scartato, in altre viene consegnato all'applicazione segnalando l'errore)
- **dati: contiene il messaggio fornito dal livello applicativo**

Checksum UDP

Scopo: individuare “errori” (es., bit modificati) nel segmento trasmesso

Mittente

- **Tratta i contenuti del segmento come sequenza di interi a 16 bit**
- **Checksum=somma dei contenuti dei segmenti con complemento a 1**
- **Il mittente invia il valore del checksum nel campo checksum del segmento UDP**

Destinatario

- **Calcola il checksum del segmento ricevuto**
- **Controlla se il valore del checksum calcolato è uguale al valore del campo checksum:**
  - **NO** → errore
  - **SI** → non si individua errore. Ci può essere lo stesso un errore?

Checksum UDP

Calcolato usando un maggior numero di informazioni di quelle presenti nell'header UDP → definizione di uno *pseudo-header* UDP

- *zero padding*: dimensione dello pseudo-header (multiplo di 32 bit)
- *protocollo*: campo protocollo del datagram IP
- pseudo-header anteposto al segmento UDP
- checksum calcolato su pseudo-header e intero segmento UDP
- lo pseudo-header *non* è trasmesso dal mittente

Pseudo header UDP e UDP6

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv4 Address																															
4	32	Destination IPv4 Address																															
8	64	Zeroes								Protocol								UDP Length															
12	96	Source Port																Destination Port															
16	128	Length																Checksum															

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															

Checksum UDP

Calcolato usando il complemento ad 1 della somma di tutti i campi dello pseudo-header e del segmento UDP

Esempio

3 parole da 16 bit l'una

0110011001100110
0101010101010101
0000111100001111

Somma delle 3 parole

0110011001100110
0101010101010101
0000111100001111
1100101011001010

- complemento ad 1di 1100101011001010 → 0011010100110101
- campo checksum nel segmento UDP trasmesso = 0011010100110101
- il destinatario calcola il suo checksum su pseudo-header e segmento UDP ricevuto (senza calcolare il complemento a 1)
- checksum dest. + checksum UDP = 1111111111111111 → no errore
- checksum dest. + checksum UDP ≠ 1111111111111111 → errore

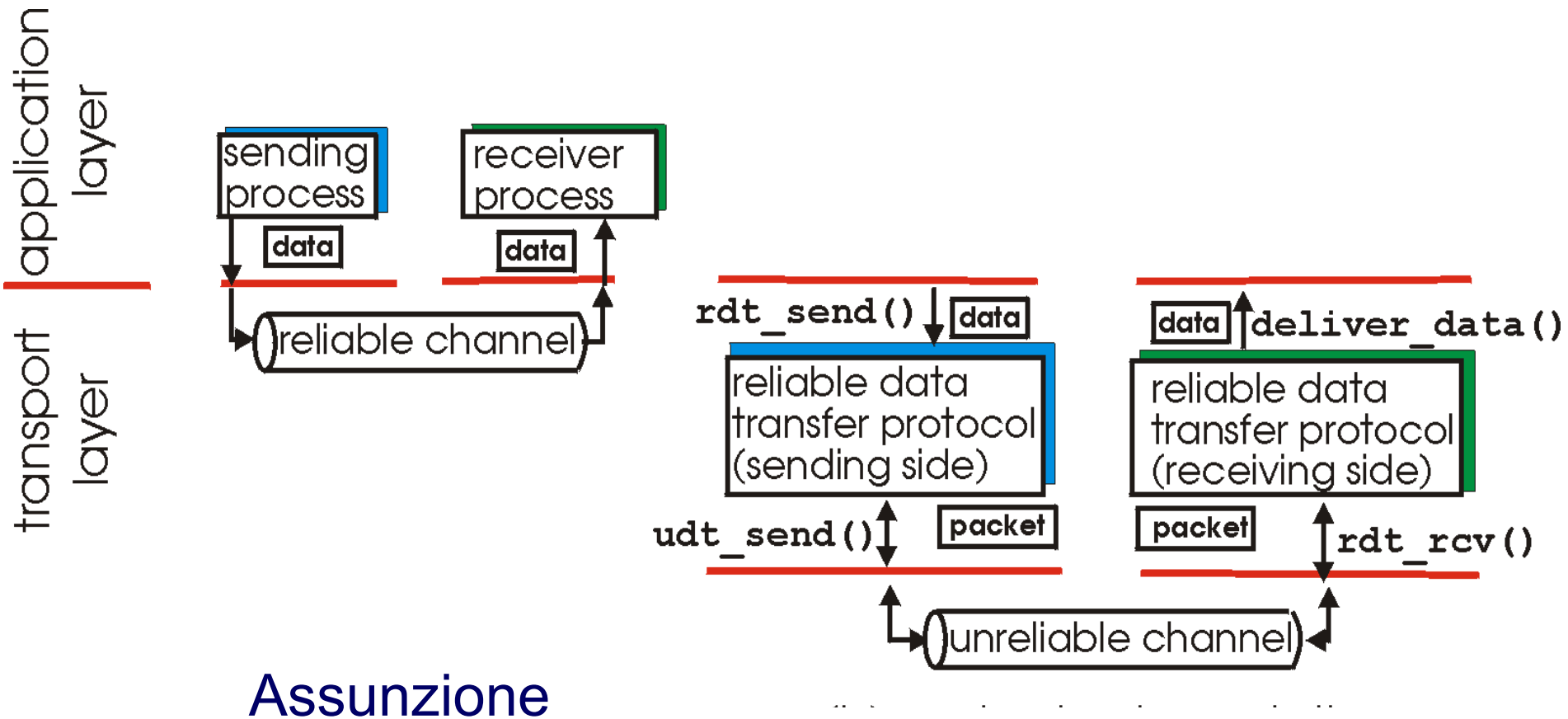
Calcolo del checksum UDP

- **Conoscenza dell'indirizzo IP del mittente e del destinatario**
- **il processo mittente a livello UDP non può acquisire l'indirizzo IP del destinatario dall'applicazione di livello superiore**
- **il processo mittente a livello UDP chiede al livello IP di costruire lo pseudo-header, calcolare il checksum UDP ed eliminare lo pseudo-header**

→ Violazione del principio di indipendenza funzionale per protocolli appartenenti a livelli diversi

Modulo 3:
Protocolli su canale affidabile

Applicazioni e trasporto



Assunzione delle applicazioni

Realtà implementativa

Problema del trasferimento affidabile

- **Il problema del trasferimento affidabile di dati su infrastruttura inaffidabile riguarda un contesto più generale del livello TCP**
  - E’ uno dei problemi principali del networking!
  - Esiste anche a livello 2 e a livello 7 (applicazioni)
  - Il livello di inaffidabilità del canale di comunicazione determina la complessità del protocollo che deve gestire tale inaffidabilità

Diversi protocolli per diversi canali

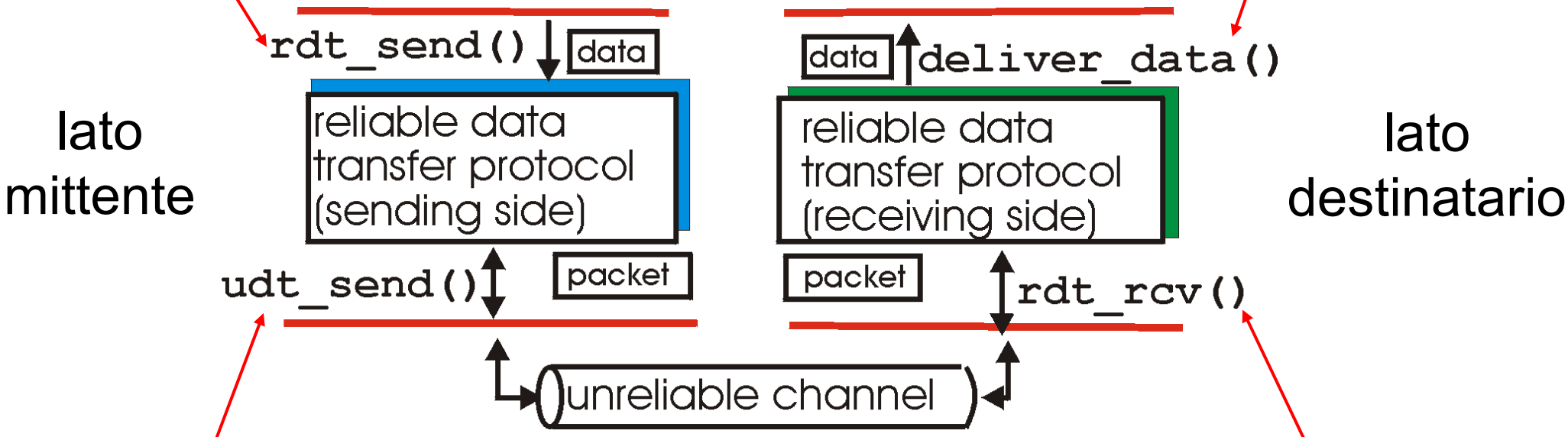
- **Protocollo rdt1.0: Trasferimento su canale completamente affidabile**
- **Protocollo rdt2.x: Trasferimento su un canale con errore sui bit**
  - 2.0 - versione base
  - 2.1 - risolve i problemi di duplicazione
  - 2.2 - evita due tipi di ACK
- **Protocollo rdt3.0: Trasferimento su un canale con errore sui bit e perdita di pacchetti**

Trasferimento affidabile su canale affidabile

1) Trasferimento su canale completamente affidabile (prot. *rdt1.0*)

rdt_send(): chiamata da sopra, (es., da applicazione di rete), passa i dati da consegnare al destinatario

deliver_data(): chiamata da rdt per consegnare i dati



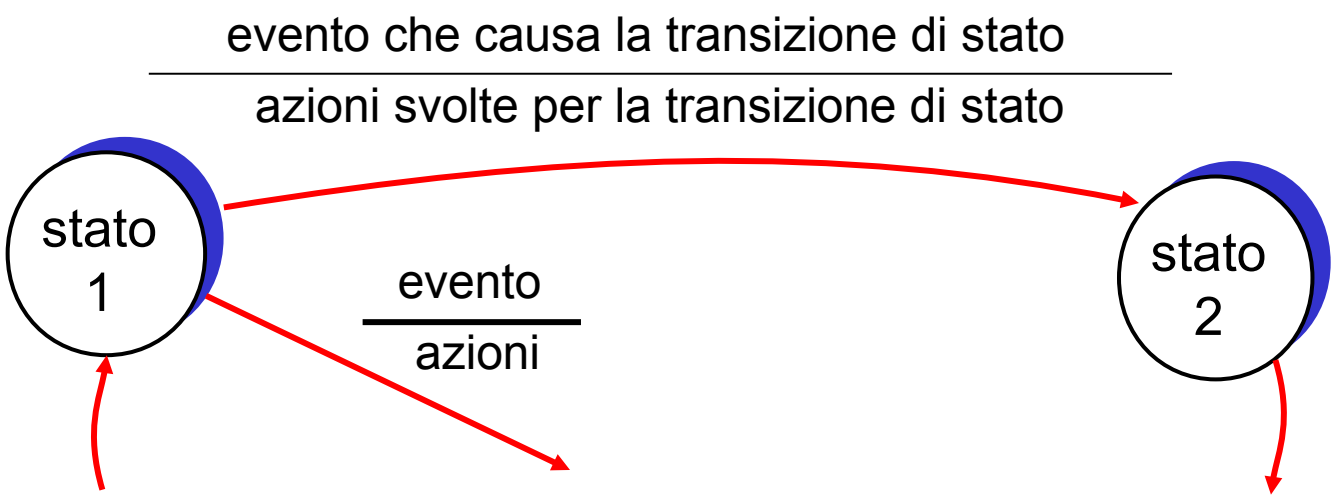
udt_send(): chiamata da rdt, trasferisce i pacchetti sul canale non affidabile

rdt_rcv(): chiamata all'arrivo del pacchetto al destinatario

Formalismo: FS;

Uso del formalismo di *macchine a stati finiti* (FSM) per modellare il comportamento del mittente e del destinatario

Stato: lo stato successivo è determinato in modo unico dallo stato attuale e dall'evento che occorre



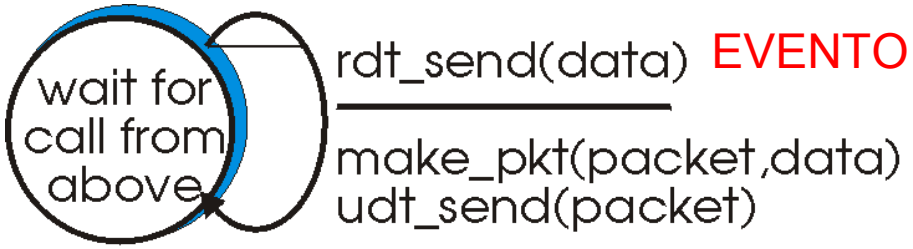
Sviluppo incrementale delle funzioni svolte dal mittente e dal destinatario

Ipotesi

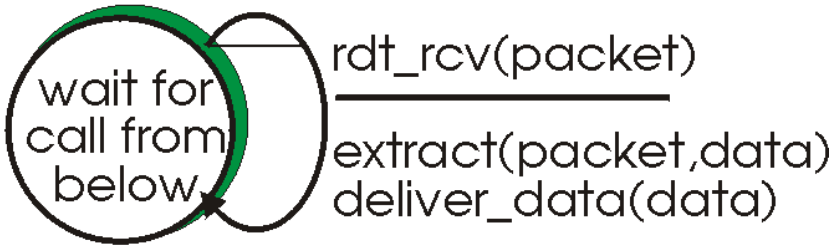
- **Trasferimento dei dati unidirezionale, anche se le informazioni per la gestione della comunicazione possono viaggiare in modo bidirezionale**
- **Meccanismo STOP-AND-WAIT**
  - Il mittente invia un pacchetto e si mette in attesa della risposta prima di inviare un altro pacchetto
- **TCP è molto piu' complesso**
  - Le differenze implementative però non nascondono il fatto che usiamo gli stessi principi

Automi FSM per protocollo rdt1.0

- Canale affidabile: non ci sono errori di bit, non c'è perdita di pacchetti
- Va definito un *automa* per il mittente, uno per il destinatario
  - Il mittente invia dati sul canale sottostante
  - Il destinatario legge dati dal canale sottostante



(a) rdt1.0: mittente



(b) rdt1.0: destinatario

Modulo 4:
Protocolli su canale con possibili errori nei bit

Trasferimento affidabile su canale con errori a livello di bit

2) Trasferimento su un canale con errore nei bit (protocollo rdt2.0)

- **Il canale trasmissivo può modificare il valore di un bit**
- **Possibile modalità per recuperare l'errore**
  - Acknowledgement (ACK): il destinatario comunica esplicitamente al mittente che il pacchetto ricevuto è OK
  - Acknowledgement negativo (NAK): il destinatario comunica esplicitamente al mittente che il pacchetto ricevuto ha un errore.

Il mittente ritrasmette il pacchetto se riceve un NAK

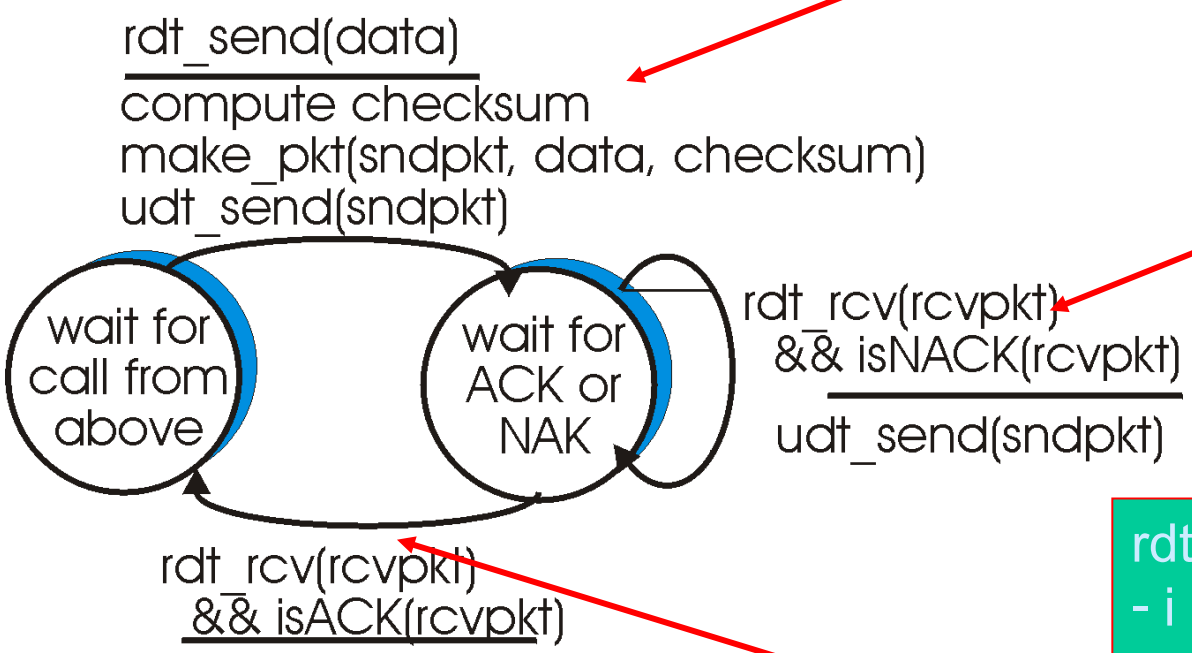
Differenze

- **Nuovi meccanismi in rdt2.0 rispetto a rdt1.0:**
  - Rilevamento dell'errore → checksum
  - Feedback da parte del destinatario → messaggio di controllo (ACK o NAK) inviato dal destinatario al mittente

Automa mittente per protocollo rdt2.0

Meccanismo di stop-and-wait

Il mittente invia un pacchetto e si mette in attesa della risposta (ACK o NAK)



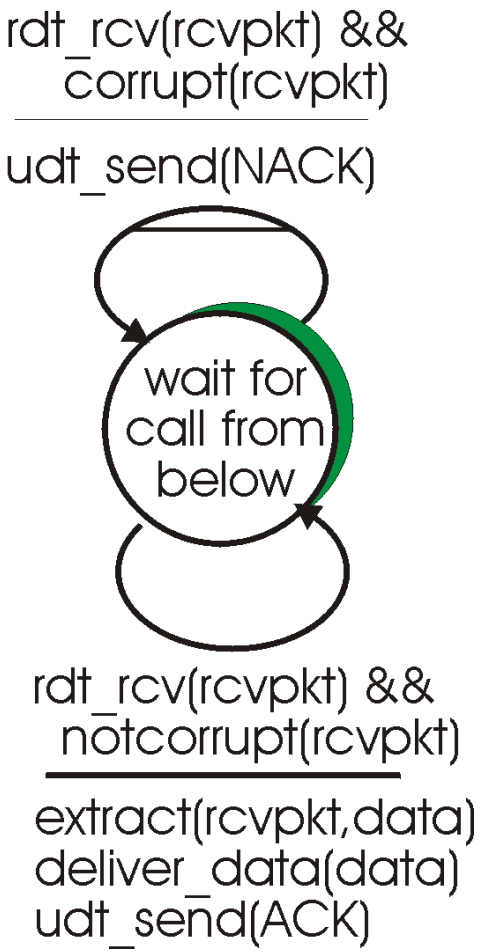
FSM per mittente

rdt_send(data)
- calcola checksum
- crea pacchetto
- invia pacchetto su canale inaffidabile

rdt_rcv(pacchetto NAK)
- ri-invia pacchetto su canale inaffidabile

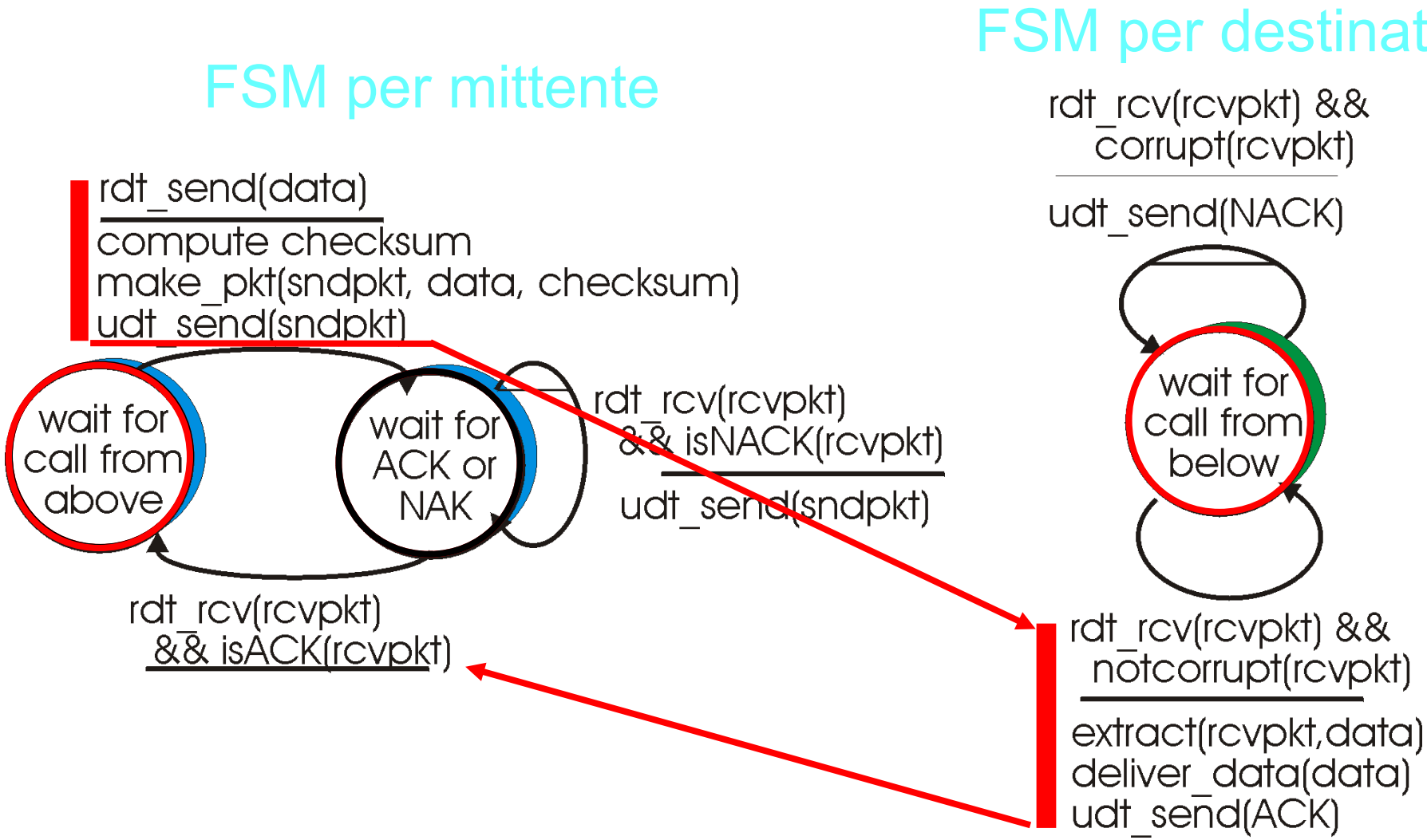
rdt_rcv(pacchetto ACK)
- i dati sono arrivati correttamente quindi è possibile mettersi in attesa di altri dati da inviare

Automa destinatario per rdt2.0



FSM per destinatario

Protocollo rdt2.0 nel caso di trasferimento corretto



Protocollo rdt2.0 nel caso di trasferimento con errore di bit

Il mittente che ha inviato un pacchetto ed è in attesa della risposta ACK o NAK del destinatario, cicla in questo stato rinviando il pacchetto finché non riceve un ACK

Problema del protocollo rdt2.0

Cosa succede se ad essere danneggiati sono i messaggi di ACK/NAK?

→ A causa del danneggiamento, il mittente non sa se il destinatario ha ricevuto o meno il pacchetto (ricordarsi che non c'è perdita di pacchetti)

Problema del protocollo rdt2.0

- **Prima soluzione:**
  - ritrasmettere il pacchetto quando si ha un messaggio ACK o NAK difettoso → non funziona perché vi è la possibilità di duplicazione dei pacchetti. E il protocollo rdt2.0 non gestisce la duplicazione di pacchetti (ovvero il destinatario non sa se un pacchetto è nuovo o uno di quelli ritrasmessi)
- **Seconda soluzione:**
  - inviare ACK/NAK in risposta a ACK/NAK del destinatario → non funziona perché se un ACK/NAK di risposta si danneggia, c'è rischio di ritrasmissioni continue

Protocollo rdt2.1: possibile miglioramento del protocollo rdt2.0

SOLUZIONE → Protocollo rdt2.1: Trasferimento su un canale con errore sui bit e gestione dei numeri di sequenza

Nuovi meccanismi in rdt2.1: numeri di sequenza

- il mittente aggiunge un numero di sequenza a ciascun pacchetto
- il mittente ritrasmette il pacchetto se il messaggio di ACK/NAK è danneggiato (e quindi scartato)
- il destinatario scarta (nel senso che non consegna al livello superiore) i pacchetti duplicati

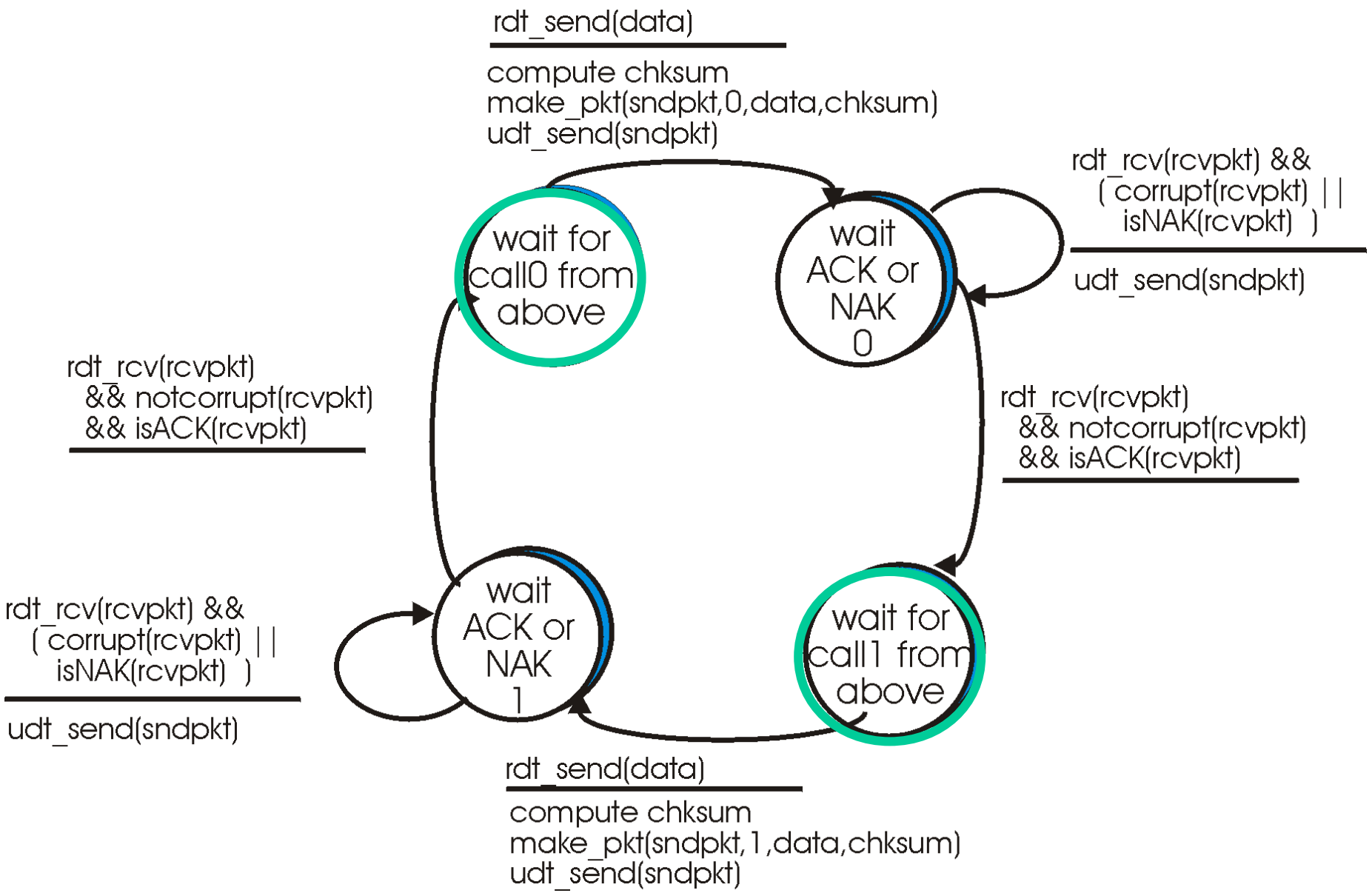
NOTA

Nell’ipotesi di un sistema di comunicazione STOP-AND-WAIT

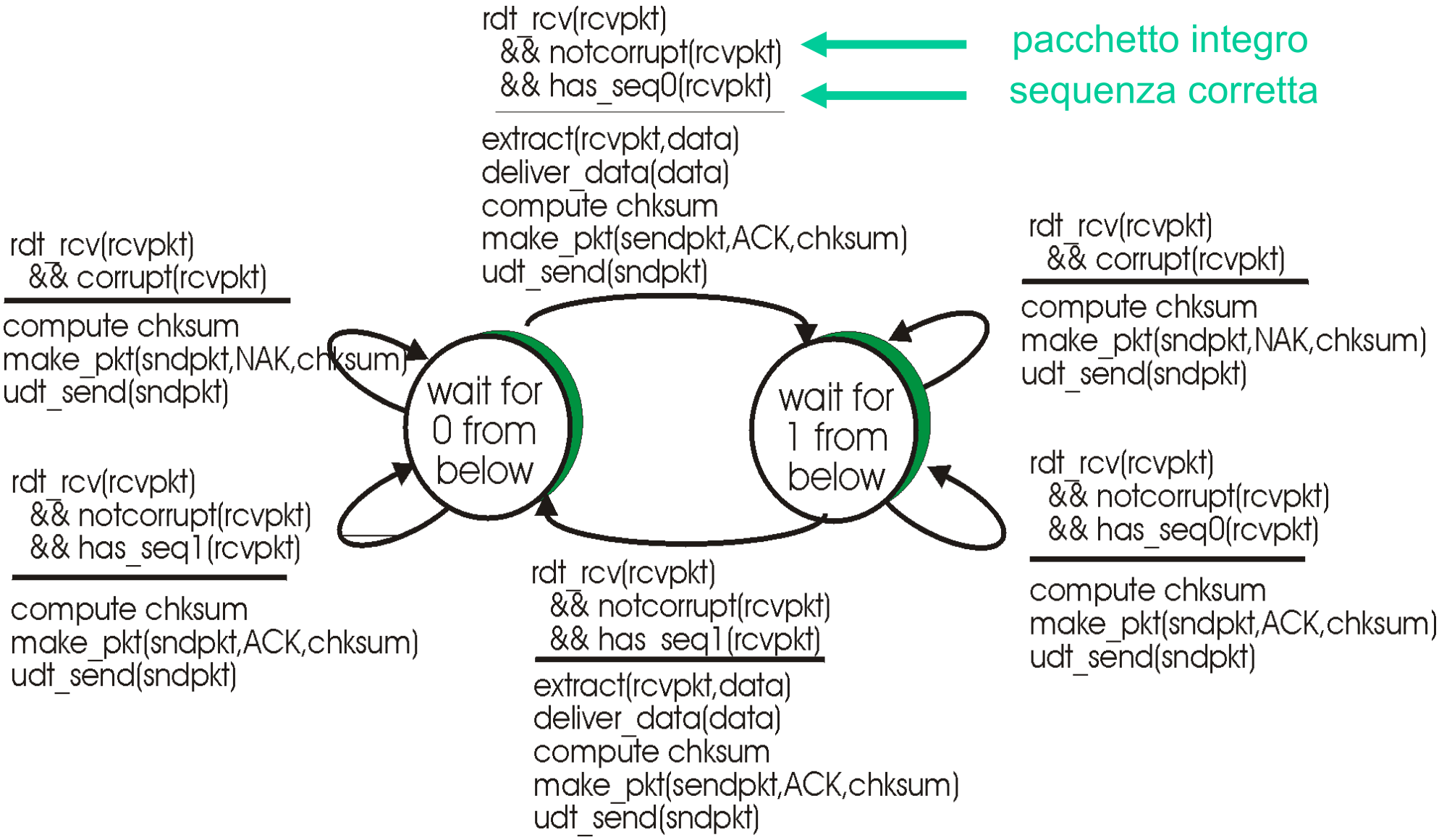
→ è sufficiente un bit come numero di sequenza

- **Se il numero di sequenza è lo stesso → il mittente ha rispedito un pacchetto**
- **Se il numero di sequenza è incrementato di 1 (modulo 2) → il mittente ha spedito un nuovo pacchetto**

Automa FSM per protocollo rdt2.1-mittente



Automa FSM per protocollo rdt2.1-destinatario



Analisi del protocollo rdt2.1

- **Mittente**
  - Numero di sequenza aggiunto a ciascun pacchetto
  - Due sequenze di numeri (0,1) sono sufficienti nel caso di ACK/NAK per singolo pacchetto in caso di trasmissione STOP-AND-WAIT
  - Deve verificare se anche ACK/NAK è danneggiato
  - Serve il doppio di stati rispetto a rdt2.0 in quanto lo stato corrente deve “ricordarsi” se il pacchetto corrente ha un numero di sequenza pari a 0 o 1

Analisi del protocollo rdt2.1

- Destinatario
  - Deve controllare se il pacchetto ricevuto è duplicato (lo stato consente di verificare se il numero di sequenza del pacchetto è 0 o 1)
  - Il destinatario può non sapere se il suo ultimo ACK/NAK è stato ricevuto correttamente dal mittente

Protocollo rdt2.2: possibile miglioramento del protocollo rdt2.1

Nel protocollo rdt2.1, per ogni pacchetto ricevuto, il destinatario deve inviare sempre un pacchetto di ACK o NAK al mittente

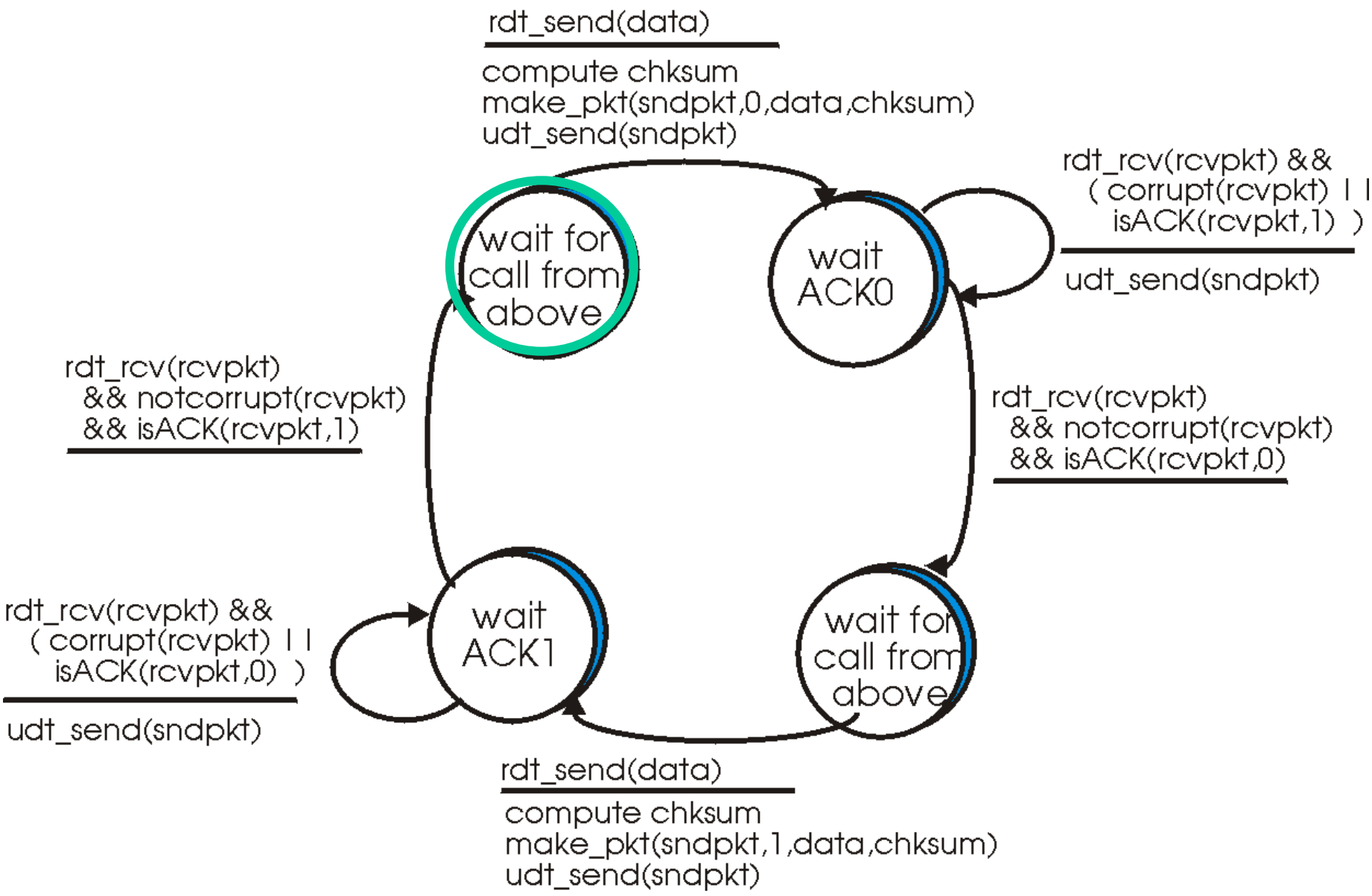
Possibile miglioramento:

Invece di inviare NAK → inviare solo messaggi di ACK per l'ultimo pacchetto ricevuto correttamente (il destinatario deve includere il numero di sequenza del pacchetto a cui si riferisce l'ACK)

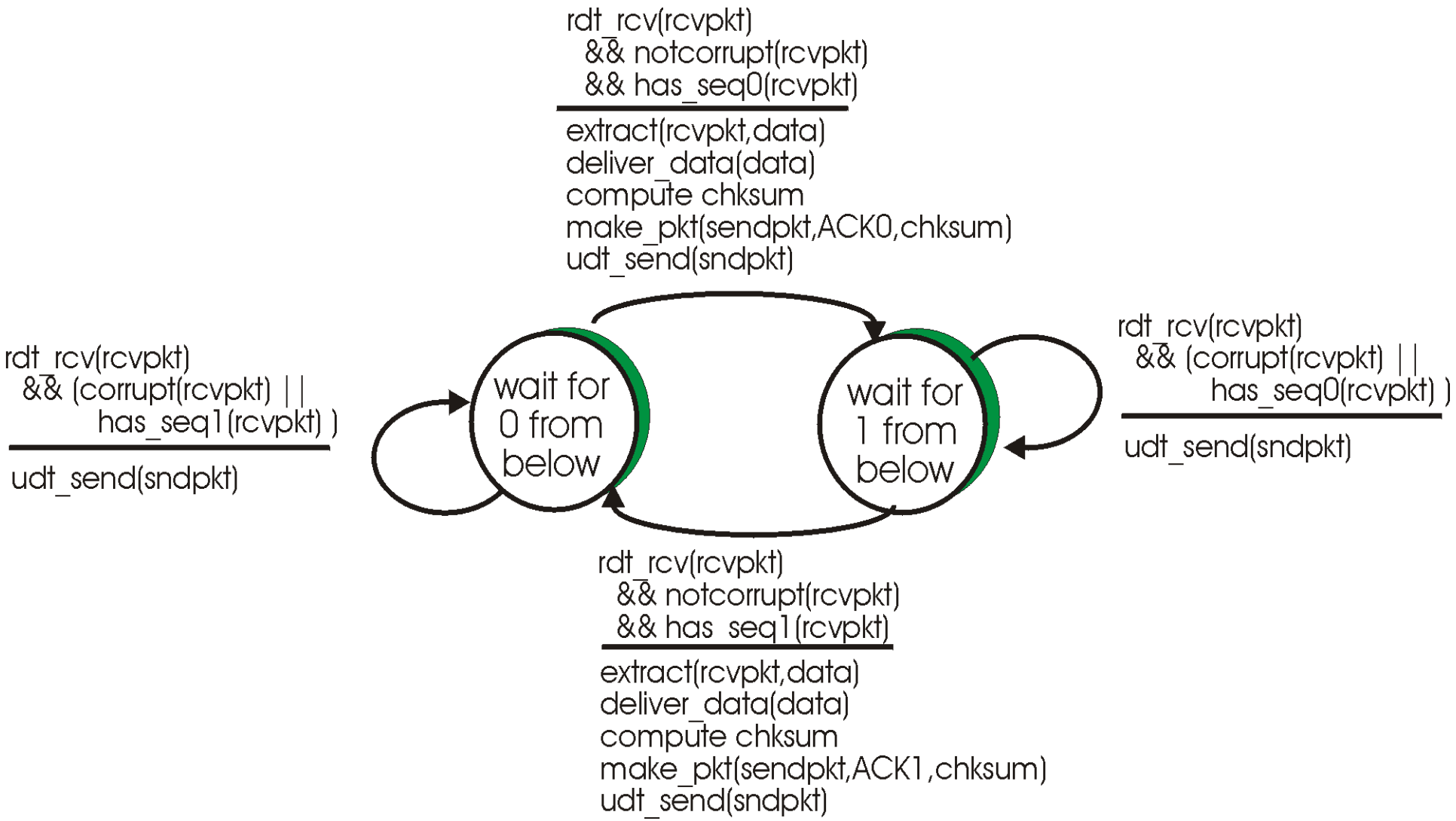
Protocollo rdt2.2: possibile miglioramento del protocollo rdt2.1

- **DESTINATARIO**
  - Se il pacchetto i ricevuto è corretto (giusta sequenza, pacchetto integro), inviare un ACK con etichetta i
  - Se il pacchetto i ricevuto non è corretto e il pacchetto i-1 era corretto, inviare un ACK con etichetta i-1
- **MITTENTE**
  - Se il mittente riceve due ACK per il pacchetto i-1, può dedurre che il destinatario non ha ricevuto correttamente il pacchetto i e quindi deve agire come se fosse un (NAK, i) → ritrasmettere il pacchetto i

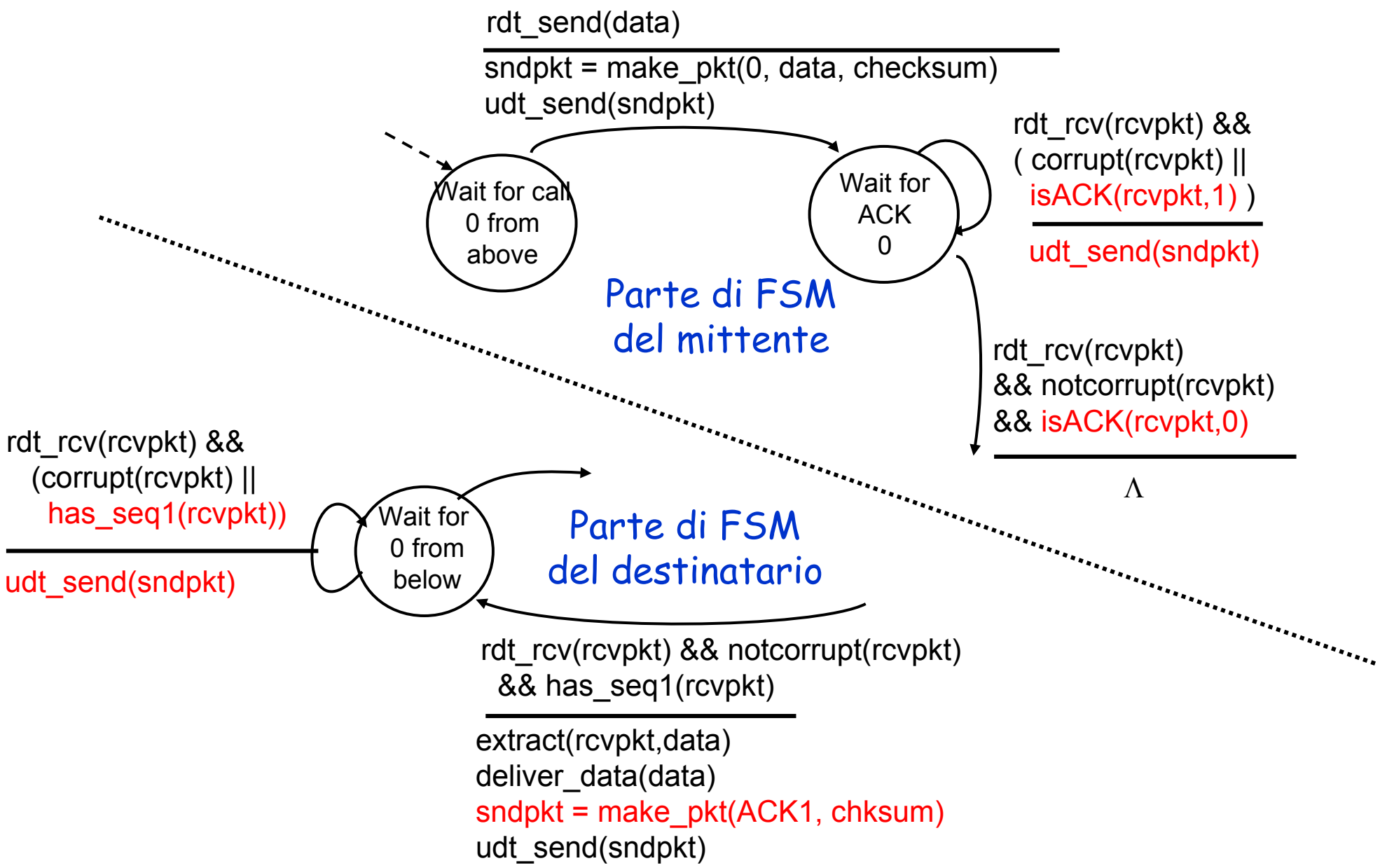
Automa FSM per protocollo rdt2.2-mittente



Automa FSM per protocollo rdt2.2-destinatario



Parti di automi FSM per rdt2.2



Modulo 5: Protocolli su canale con errori a livello di bit e perdita pacchetti

Nuova ipotesi

Il canale di trasmissione può causare errori sui bit e perdita di pacchetti (sia dati, sia ACK)

- **Come realizzare una trasmissione affidabile?**
- **L'uso di checksum, numeri di sequenza dei pacchetti, ACK, ritrasmissioni sono utili, ma non sufficienti**

Nuova ipotesi

- **SOLUZIONE → COINVOLGERE IL “TEMPO”**
  - Dopo aver spedito il pacchetto i, il mittente attende
  - (ACK, i) per un intervallo di tempo “ragionevole”;
  - poi, se non ha ricevuto l’ACK, ritrasmette il pacchetto

Trasferimento affidabile su canale con errore sui bit e perdita pacchetti

- **3.0 → Trasferimento su un canale con errore sui bit e perdita di pacchetti (protocollo rdt3.0)**
- **Soluzione → il mittente attende l'ACK per un intervallo di tempo "ragionevole", poi ritrasmette**

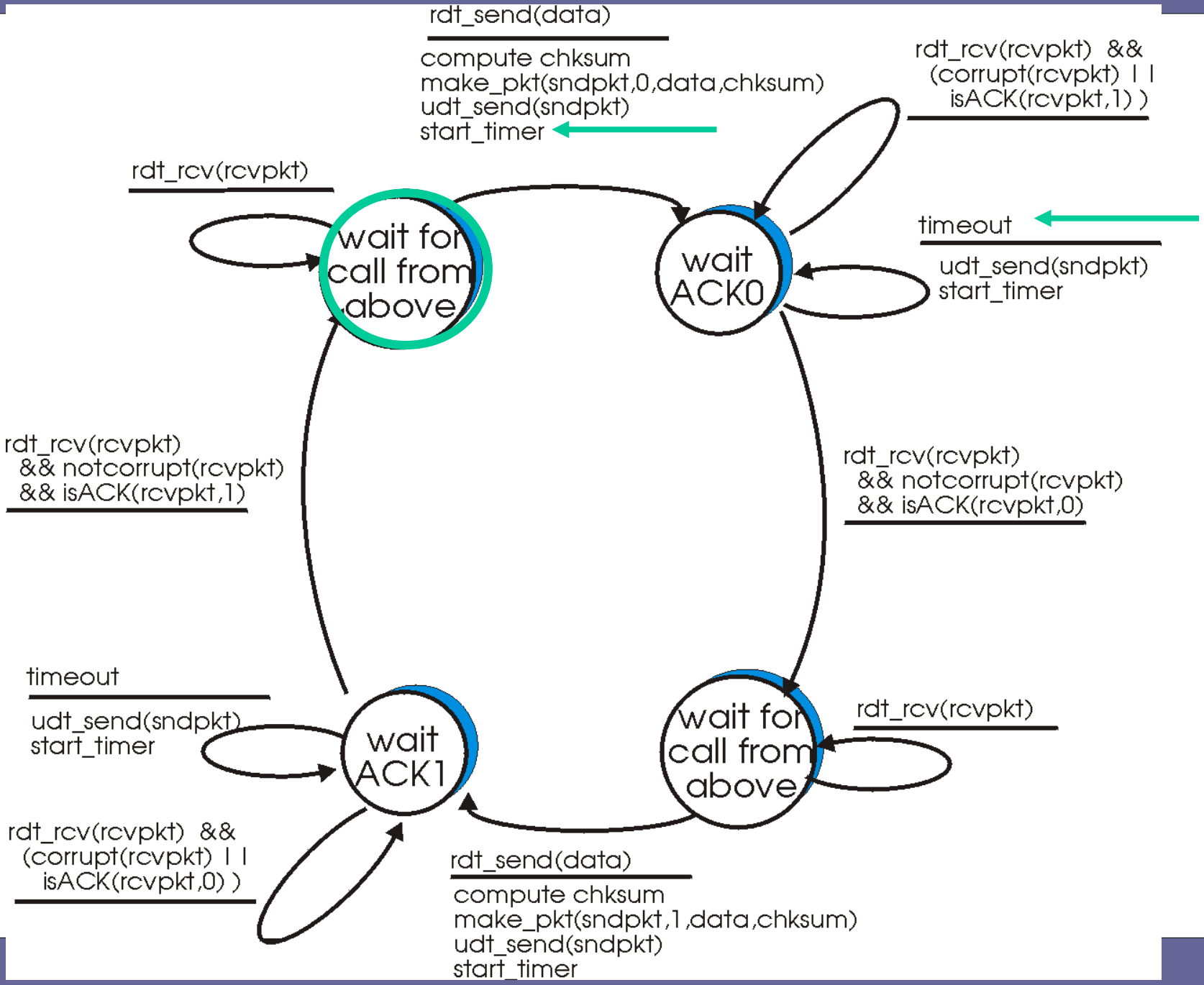
Trasferimento affidabile su canale con errore sui bit e perdita pacchetti

NECESSITA'

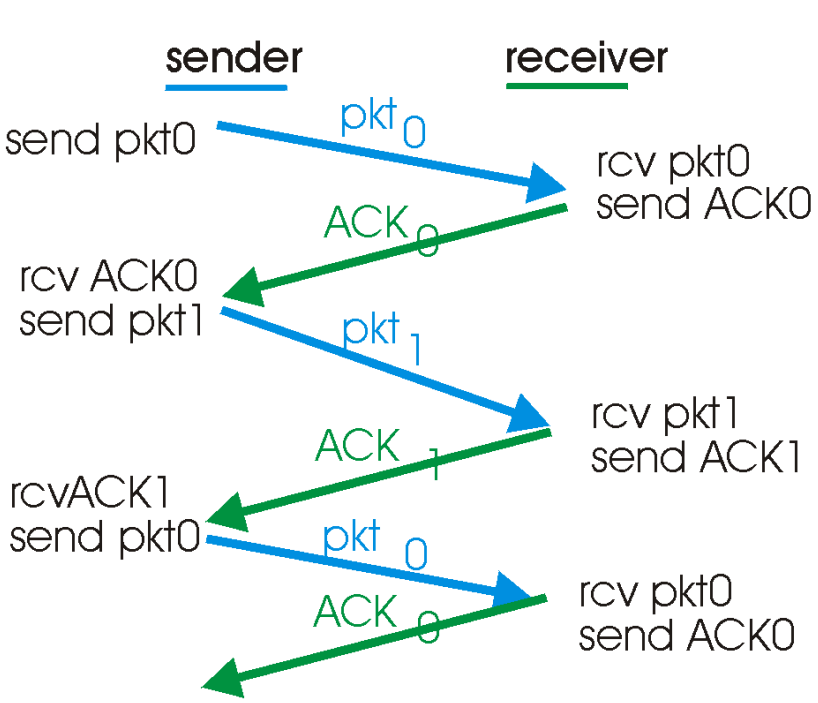
- **1) Se il pacchetto (messaggio o ACK) è in ritardo (ma non perso), il pacchetto ritrasmesso sarà duplicato**
  - Questo caso è già risolto dall'uso dei numeri di sequenza (per poter funzionare, anche il destinatario deve specificare il numero di sequenza del pacchetti per cui ha inviato ACK)

- **2) Intervallo di tempo ragionevole**
  - Richiede l'uso di un countdown timer

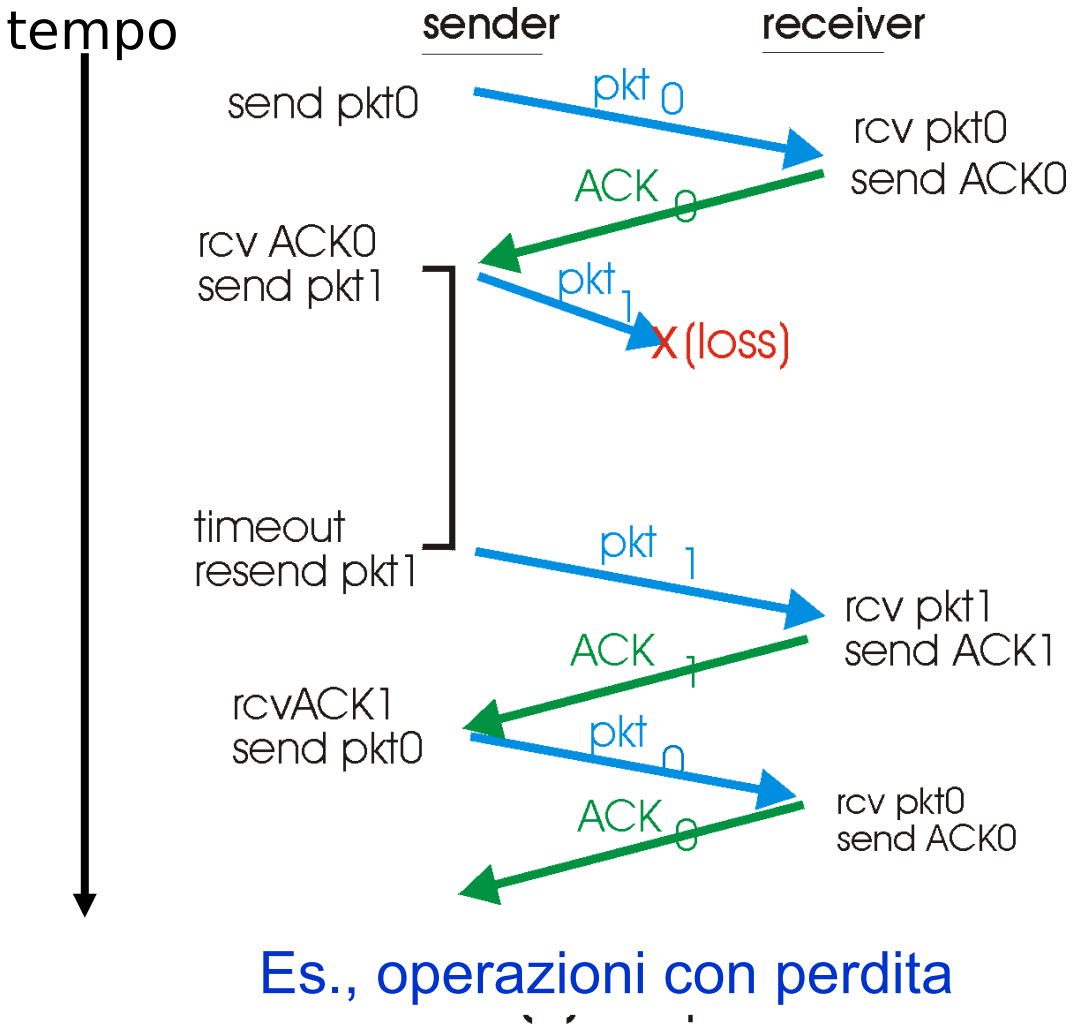
Automa FSM per protocollo rdt3.0 - mittente



Sequenza operazioni in rdt3.0



Es. operazioni senza perdite



Es., operazioni con perdita

Il meccanismo send-ack per singolo pacchetto è molto semplice ed affidabile, ma INEFFICIENTE

Prossimo obiettivo

- **Utilizzare questi risultati teorici per analizzare quali meccanismi utilizza il protocollo TCP per realizzare una “comunicazione affidabile” e “orientata alla connessione” su di un “canale inaffidabile”**