Reti di Telecomunicazioni

Livello Data Link

Autori

Queste slides sono state scritte da

Michele Michelotto

michele.michelotto@pd.infn.it

che ne detiene i diritti a tutti gli effetti

Copyright Notice

Data Link Layer

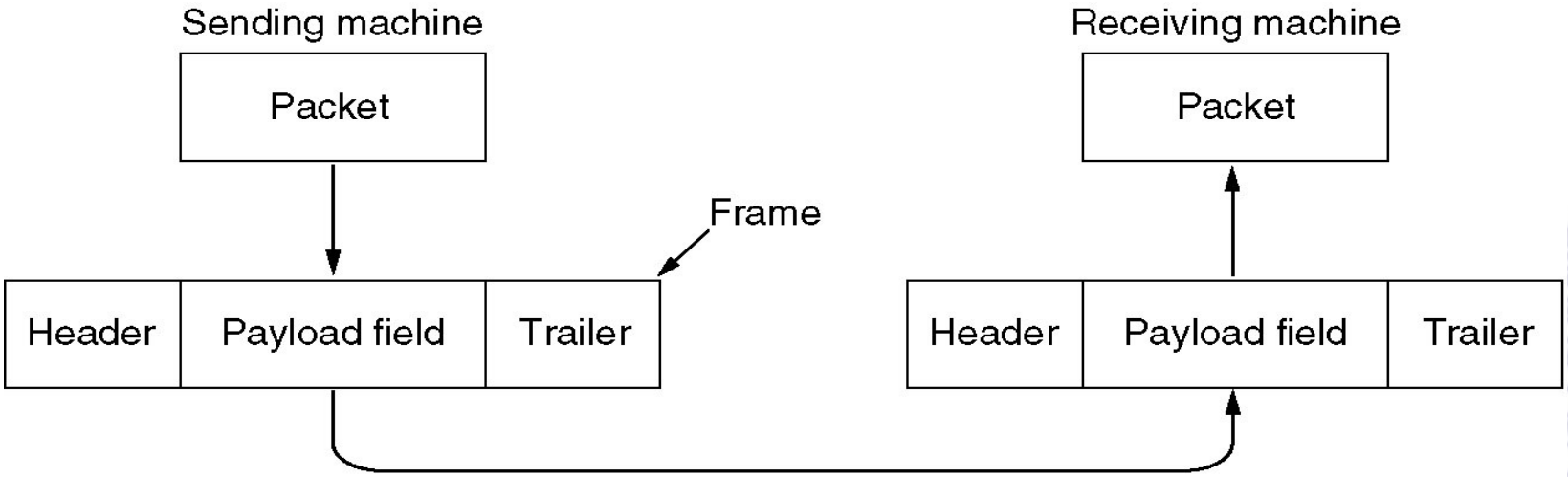
Introduzione
Layer: Modello OSI e TCP/IP
Physics Layer
Data Link Layer
MAC sublayer

Data Link Layer

- Come effettuare una comunicazione, affidabile, efficiente tra due macchine **adiacenti** a livello data link
- Adiacente significa che le macchine sono connesse da una canale **concettualmente** simile ad un filo (coax, doppino, linea telefonica, canale wireless)
- Un canale simile ad un filo implica che i bit sono consegnati nell'ordine di spedizione
- Non banale, bit soggetti ad errore, ritardi di trasmissione

Criteri di design

- Interfaccia ben definita verso il livello network
- Meccanismo per gestire gli errori
- Meccanismo di gestione del data-flow
- Prende i pacchetti dal network layer e li incapsula in frame (trame? buste?)

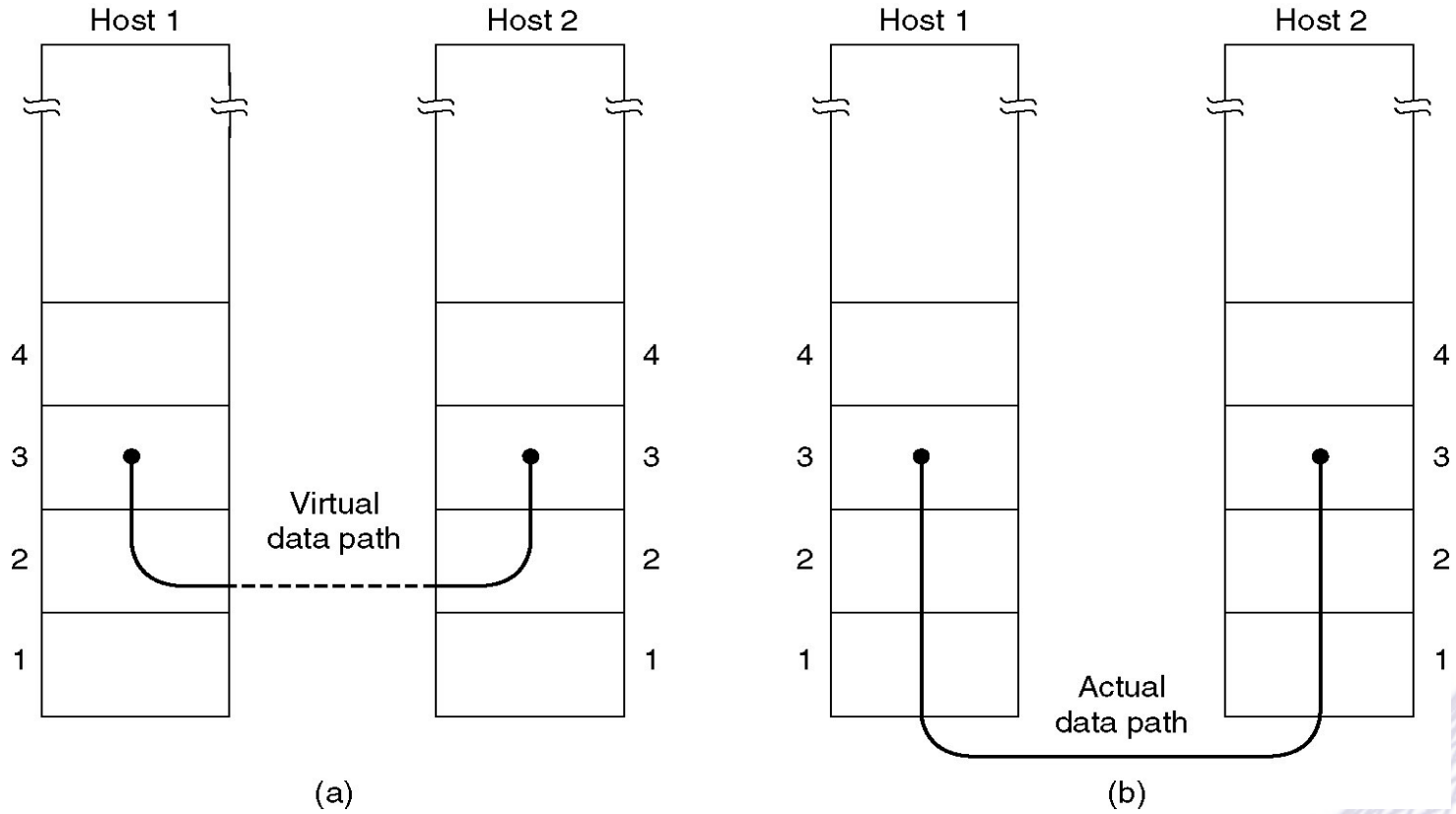


L3

L2

L1

Comunicazione virtuale



(a) Comunicazione virtuale (b) Comunicazione reale

Tre tipi di servizio

1 - Senza connessione e senza ricevuta:

- Mando frame indipendenti (cfr. cartolina postale) se un frame si perde non tento di recuperarlo. Da usare per error rate estremamente basso o per traffico real-time in cui dati in ritardo sono peggio di dati persi (voce, video)

2 - Senza connessione con ricevuta:

- Mando frame indipendenti ma voglio la ricevuta per ognuno (cfr. raccomandate A.R.). Il sender quindi sa se deve rimandare un frame. Utile per link non affidabili, tipo wireless
- (NB la mancanza di ricevuta a livello 2 può sempre essere compensata in un livello superiore per esempio a livello 3)

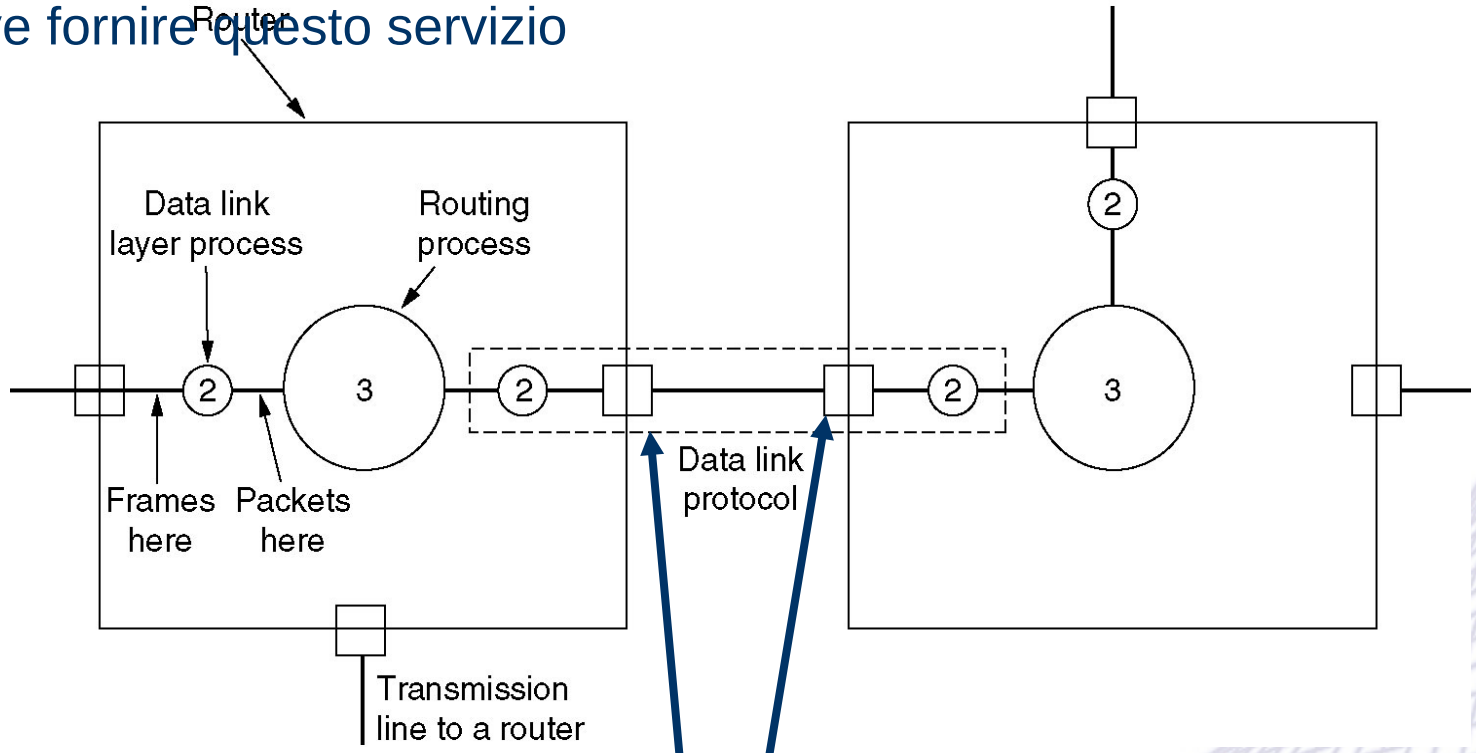
3 - Con connessione e ricevuta:

- Si stabilisce una connessione, poi ogni frame viene numerato
- il layer garantisce che ogni frame spedito sia ricevuto. Inoltre garantisce anche l'ordine di ricezione

Esempio

Il router gestische pacchetti, li accoda, li smista.

Però non vuole occuparsi troppo di pacchetti persi. Il protocollo di data link deve fornire questo servizio



Link affidabile fornito su link potenzialmente inaffidabile

Framing

- Fornisce servizi al Network layer e ha bisogno dei servizi forniti del Physical layer
- Physical layer prende uno stream di bit e cerca di consegnarli
- I bit ricevuti possono essere meno di quelli mandati (o anche di più) e avere valori errati. **Data Link deve accorgersi e se necessario correggere questi problemi**
- Questo viene fatto mettendo i bit in **frame** e calcolando un **checksum** per ogni frame

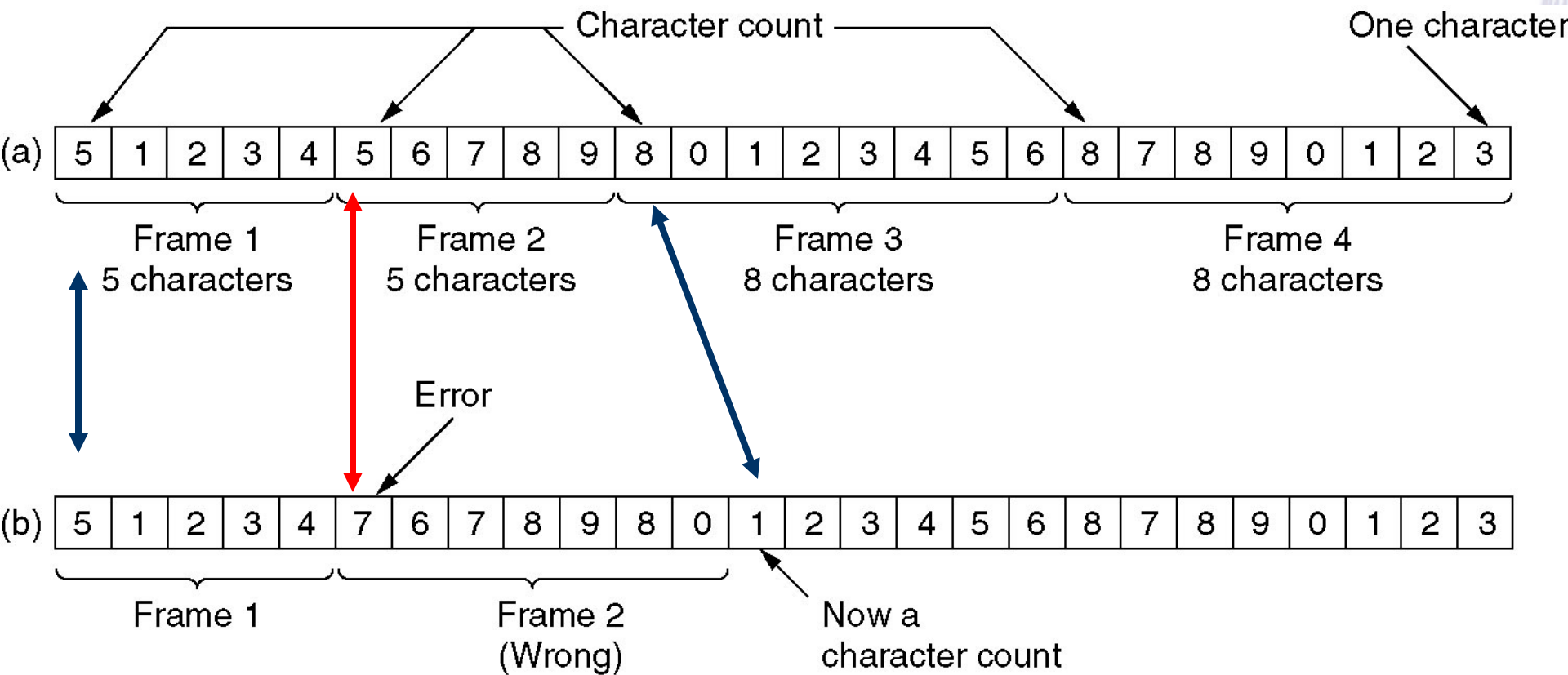
Metodi di framing

Non facile, non posso mettere spazi come con il testo scritto, difficile sincronizzarsi:

- Conteggio dei caratteri
- Flag byte senza bit stuffing
- Starting end Ending Flag byte con bit stuffing
- Violazioni del codice al layer fisico

Conteggio caratteri

- a) stream di dati
- b) In caso di errore me ne accorgo con il checksum ma non capisco dove avviene l'errore

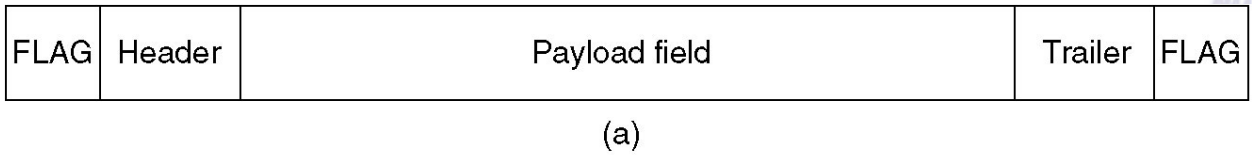


FLAG

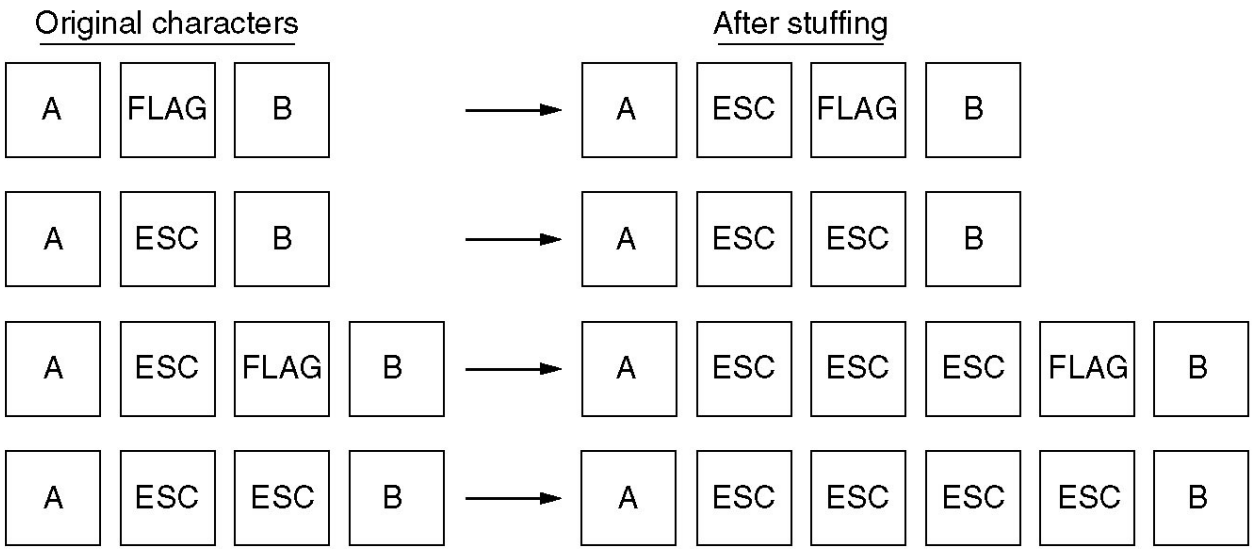
a) Il frame inizia e finisce con un pattern speciale di bytes detto **FLAG**, una volta erano diversi all'inizio dalla fine, ora si tende a metterli uguali. due FLAG indicano EOFrame, SOFrame

b) Se il FLAG dovesse essere una sequenza possibile di dati serve una sequenza di escape **ESC**

In passato i dati erano solo testuali per cui non c'era il problema che un dato fosse uguale al FLAG

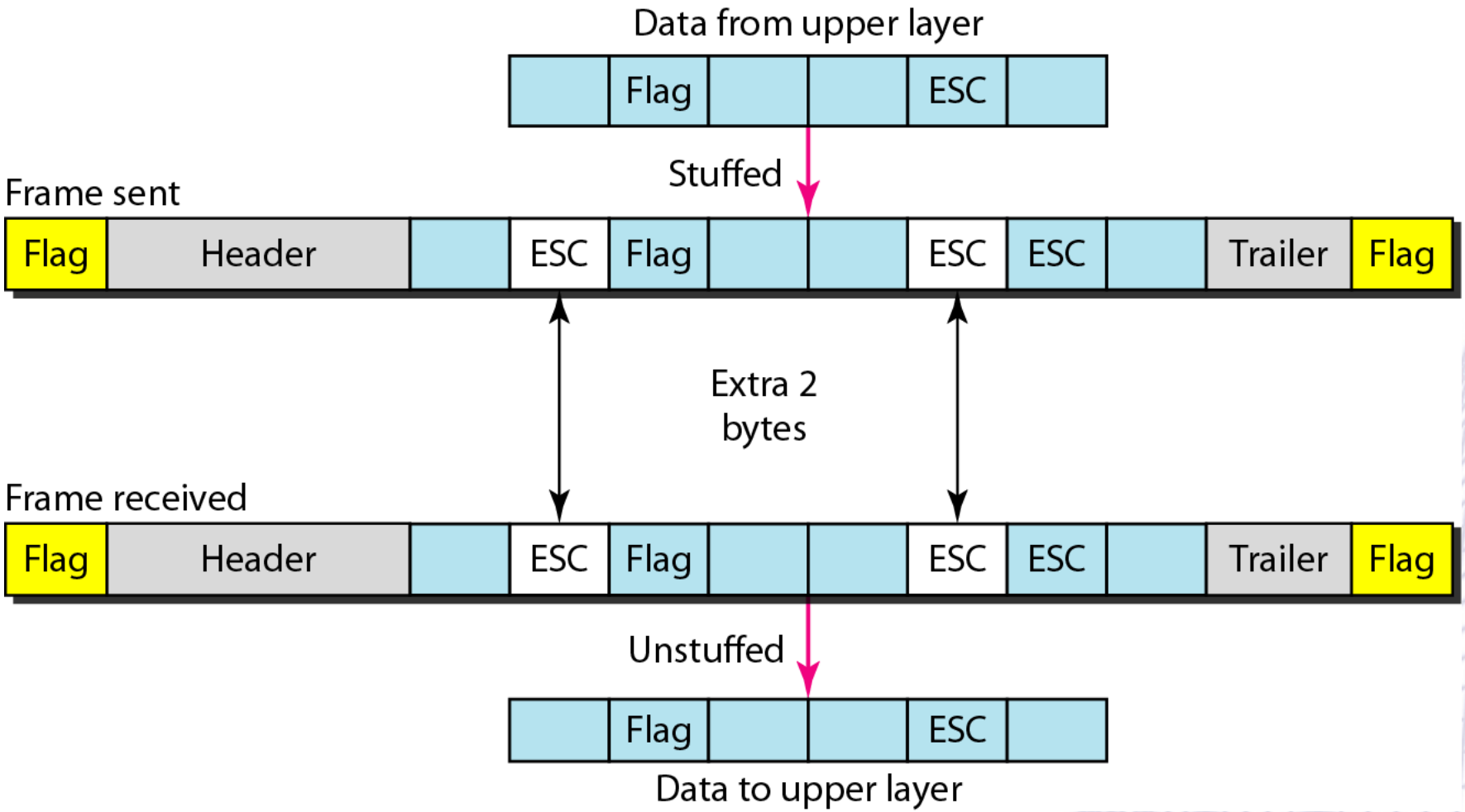


(a)



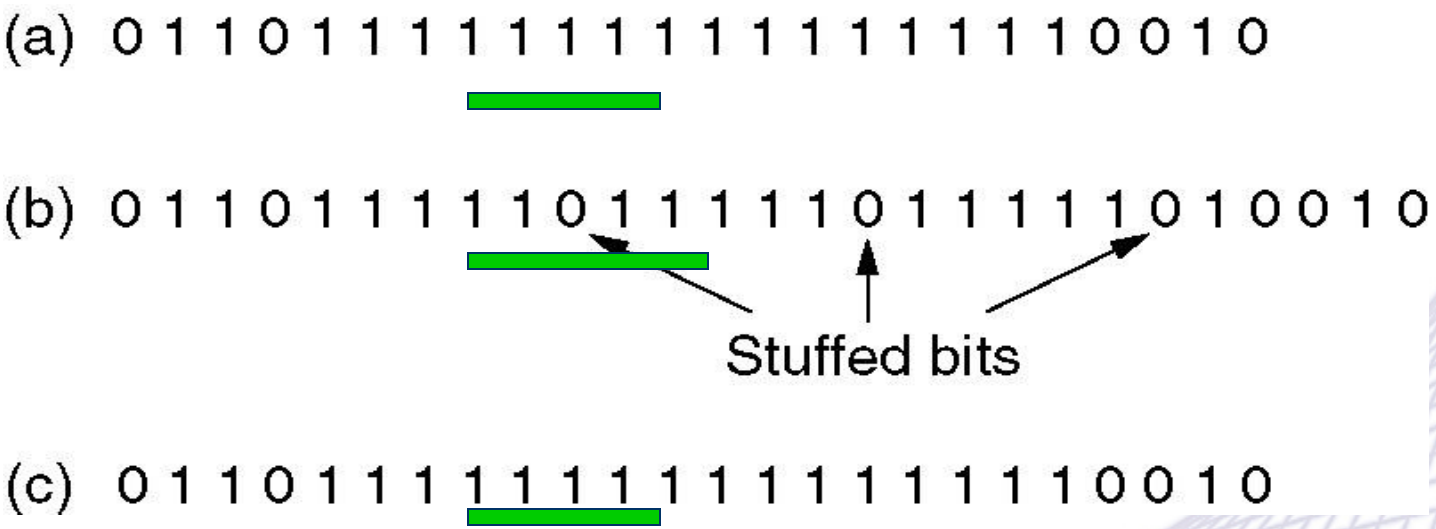
(b)

Escape

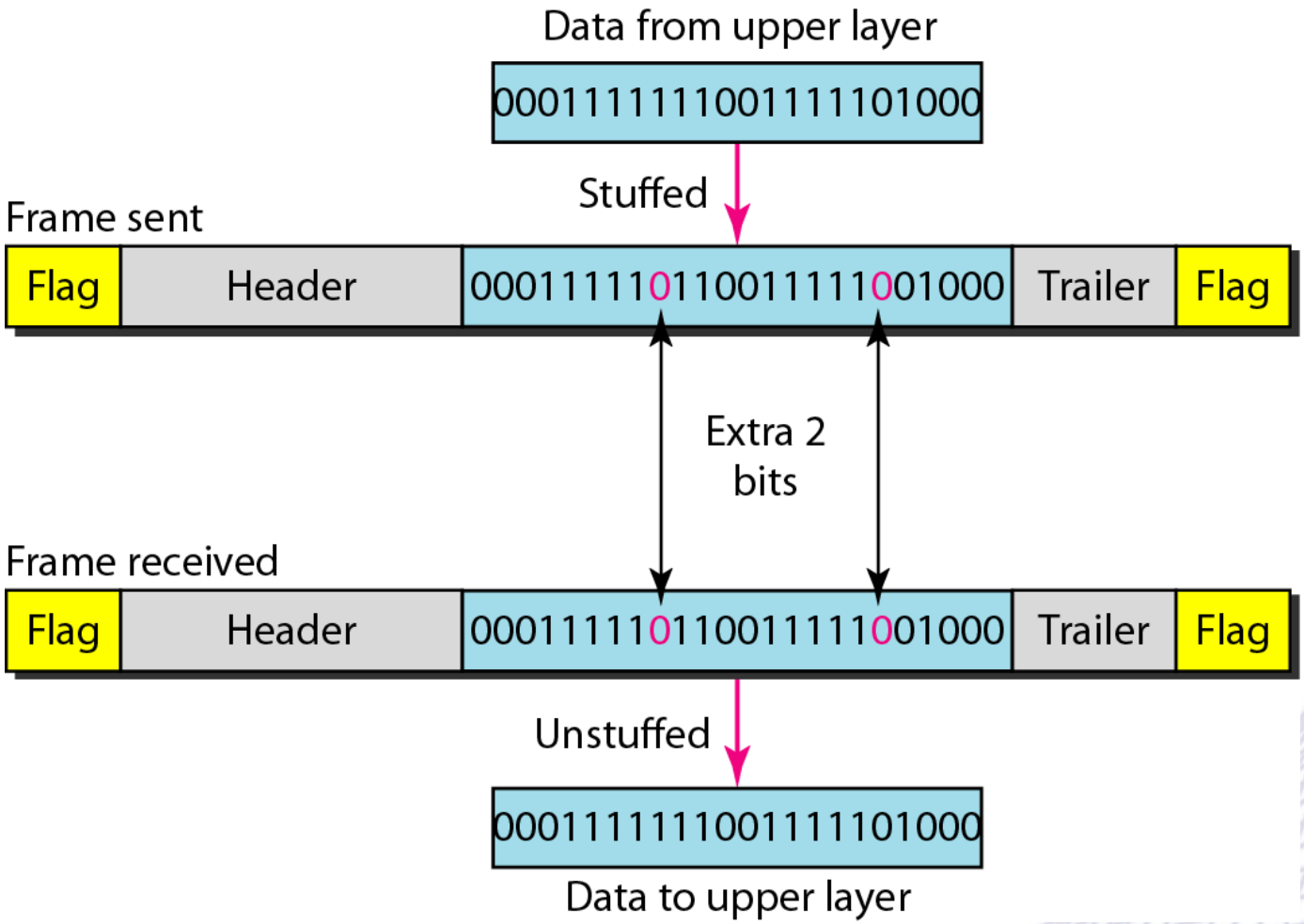

Data from upper layer
Flag
ESC
Stuffed
Frame sent
Flag
Header
ESC
Flag
ESC
ESC
Trailer
Flag
Extra 2
bytes
Frame received
Flag
Header
ESC
Flag
ESC
ESC
Trailer
Flag
Unstuffed
Flag
ESC
Data to upper layer

Bit stuffing

- Se non si vuole che il framing dipenda dai byte (esempio uso Unicode a 16bit) si usa **bit stuffing.**
- Es: comincio il frame con un FLAG **01111110**
- Quando **a)** sender vede 5 bit 1 consecutivi inserisce **b)** un bit 0, il receiver a sua volta quando **c)** vede 5 bit 1 seguiti da uno 0 lo rimuove
- Quando il receiver perder il sync cerca il FLAG che può essere solo a inizio frame e non nei dati



Bit stuffing



Physical Violation

- Applicabile a reti in cui l'encoding sia ridondante
- Es. 1 bit di dati su due bit fisici
  - Bit 1 definito come coppia alto-basso (10)
  - Bit 0 come coppia basso-alto (01)
  - Ogni bit ha quindi una transizione e permette di riconoscere il confine dei bit. Le combinazione basso-basso (00) o alto-alto (11) non sono legali e possono essere usate per delimitare i frame

Controllo degli errori

- Quando il ricevitore riceve un frame manda un feedback
  - positivo se arrivato bene
  - negativo se arriva male
- Se non arriva nulla cosa faccio? O se la ricevuta si perde?
  - Devo mettere un timeout
  - Se non arriva entro il timeout presumo che il frame sia andato perso
  - Se presumo il frame sia perso lo ritrasmetto ma devo numerarlo nel caso fosse arrivato e io abbia perso solo la ricevuta, devo assegnare un numero di sequenza in modo che il ricevente sappia distinguere la ritrasmissione dall'originale

Controllo di flusso

- Sender troppo veloce rispetto al receiver
  - macchine di diversa potenza, con diverso carico
  - Anche se la trasmissione è esente da errori ad un certo punto il ricevente non ce la fa a ricevere
- Feedback-based flow control
  - Il receiver manda indietro informazione dando il permesso di mandare altri dati o almeno dicendo al sender come se la sta cavando
- Rate-based flow control
  - Il protocollo ha un meccanismo intrinseco che limita il rate a cui il sender inserisce i dati senza avere bisogno di feedback
  - Solitamente non si usa a livello data-link

Gestione Errori

- Gli errori sono rari su linee digitali ma quando si viaggia su local loop o in wireless si possono avere errori
- A volte gli errori arrivano a bursts e non singoli
  - Pro: Se ho blocchi di 1000 bit con 0.001 error rate quasi ogni blocco ha un errore. Se vengono in burst di 100 solo uno o due blocchi su 100 ha errori
  - Contro: Difficile correggere burst di errori
- Due strategie:
  1. Inserire molta ridondanda da capire comunque cosa si voleva trasferire (**error-correcting codes**)
  2. Inserire poca ridondanza, solo la necessaria per dedurre la presenza di un errore, ma non quale e richiedere la trasmissione di quel blocco di dati (**error detection codes**)

Error detection

- Parity Checking: viene aggiunto un bit detto parity bit in modo tale che il numero di “1” sia pari (o dispari)
  - Es. **1011010** mandato con parità pari: viene aggiunto un bit **0** e diventa **10110100**
  - Riesce a scoprire un singolo errore
- Ci sono decine di algoritmi. Uno molto usato, basato su algebra dei polinomi e implementato in hardware si chiama CRC.

Error correction Code

- Supponiamo di avere un codice in cui le uniche codeword valide sono: {0000000000, 0000011111, 1111100000 1111111111}
- Se arriva il codice 00000**0**1111 il numero originale doveva essere 00000**1**1111 (quindi correggo un bit errato)
- Se arriva il codice 0000000111 il numero originale doveva essere 0000011111 (quindi correggo fino a due bit errati)
- In caso di triplo errore che trasforma 0000011111 in 0000000011 il codice non sa se veniva da 0000000000 o da 0000011111

Parity bidimensionale

Parity →

0	1	0	1	1	1	0	**0**
0	1	1	1	1	1	0	**1**
1	0	1	1	1	1	0	**1**
1	1	1	1	1	1	1	**1**
0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	1	**1**
1	1	0	0	1	1	0	**0**
0	1	1	0	0	0	1	**1**
1	**1**	**0**	**0**	**1**	**1**	**1**	1

Permette Correzione

0	1	0	1	1	1	0	**0**
0	1	0	1	1	1	0	**1**
1	0	1	1	1	1	0	**1**
1	1	1	1	1	1	1	**1**
0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	1	**1**
1	1	0	0	1	1	0	**0**
0	1	1	0	0	0	1	**1**
1	**1**	**0**	**0**	**1**	**1**	**1**	1

Segnala Errore

Al terzo bit!

Doppio Errore

Parity →

0	1	0	1	1	1	0	**0**
0	1	1	1	1	1	0	**1**
1	0	1	1	1	1	0	**1**
1	1	1	1	1	1	1	**1**
0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	1	**1**
1	1	0	0	1	1	0	**0**
0	1	1	0	0	0	1	**1**
1	**1**	**0**	**0**	**1**	**1**	**1**	1

0	1	0	1	1	1	0	**0**
0	1	**0**	1	**0**	1	0	**1**
1	0	1	1	1	1	0	**1**
1	1	1	1	1	1	1	**1**
0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	1	**1**
1	1	0	0	1	1	0	**0**
0	1	1	0	0	0	1	**1**
1	**1**	**0**	**0**	**1**	**1**	**1**	1

Non Segnala Errore

Due segnalazioni

Error detection

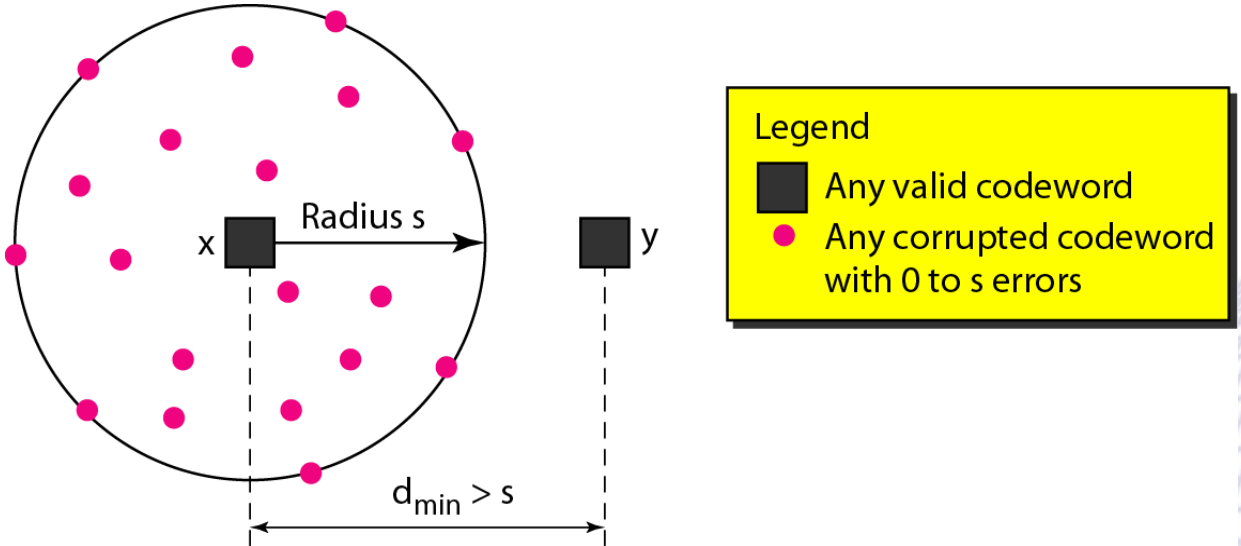
- Si possono trasmettere informazioni ridondanti che permettono di rivelare la presenza di un errore o di diversi errori, si possono anche correggere alcuni errori.
- Messaggio di **m** bit + **r** bit di ridondanza = **n** bit di codeword. Esempi:
  - Parity Bit: un bit viene aggiunto ad ogni frame in modo da rendere la codeword pari (o dispari). Quindi **10110101** diventa **101101011** (pari). Si possono rivelare solo errori singoli

Hamming distance

- Date due codeword è possibile calcolare di quanti bit differiscono
  - Es **10001001** e **10110001** differiscono di 3 bit
  - Questa si chiama Hamming distance
  - Se due codeword hanno una distanza ***d*** significa che servono ***d*** errori a singolo bit per andare ad una all'altra
  - In un codice con **m** bit ho $\mathbf{2^m}$ data word e $\mathbf{2^n}$ codeword ma non tutte sono legali. Con la lista delle codeword legali trovo le due codeword con distanza minima, questa è la distanza del codice completo

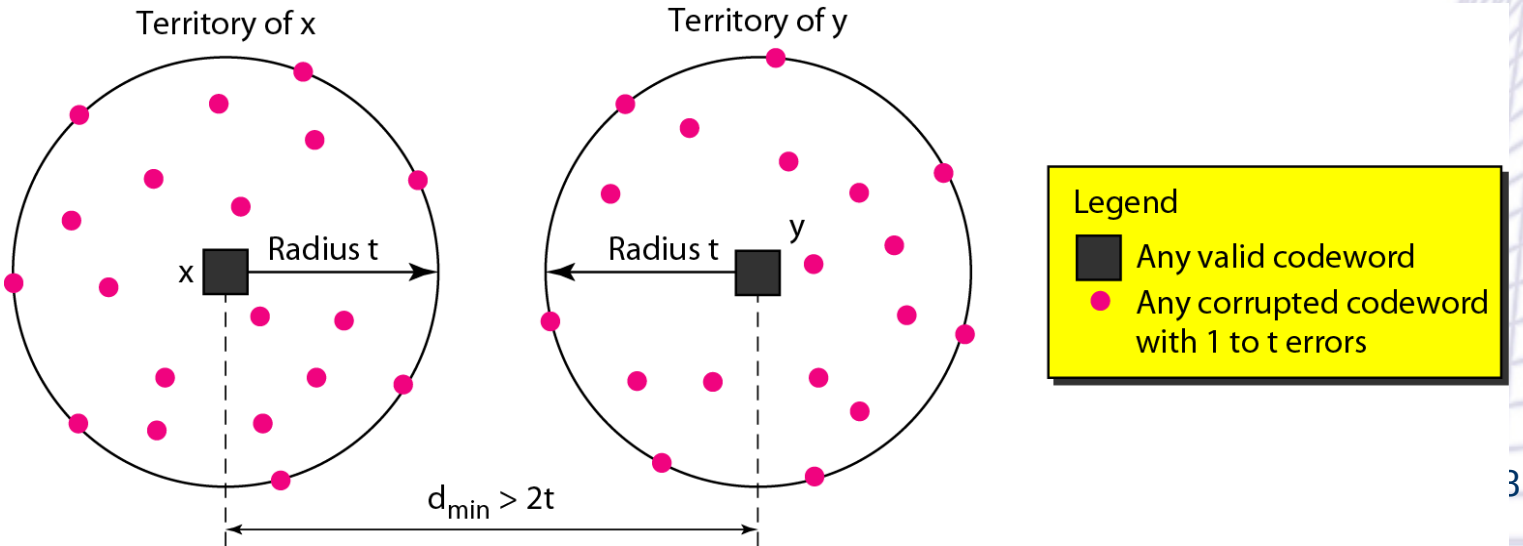
Hamming distance

- Le proprietà di *error correction* e di *error detection* di un codice dipendono dalla sua Hamming distance
  - Per rilevare **d** errori devo avere un codice con distanza ***d+1*** perché non c'è modo che **d errori single bit** cambino un codeword valida in un'altra



Hamming distance

- Per correggere **d** errori devo avere un codice con distanza ***2d+1*** perché due codeword legali sono così distanti che anche con ***d*** modifiche la codeword originale è ancora più vicina di ogni altra codeword



Parity

- Un codice con un singolo bit di parity aggiunto ha distanza 2
  - Ogni errore a singolo bit produce una codeword sbagliata. Utile solo per rivelare errori singoli
- Il codice con solo 4 codeword su 10 bit che abbiamo visto alla slide Error Correction ha una Hamming distance di 5 quindi corregge fino a 2 errori
  - Se mi arriva 00000.00111 il ricevente risale a 00000.11111
  - Se ho un triplo errore che cambia 00000.00000 in 00000.00111 non riesco a correggerlo.

Codici parity

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

CRC

- Codici CRC
  - Sono codici ciclici, una famiglia di codici lineari, in cui ogno codice si ottiene dallo shift di un altro codice (es 1011000 → 0101100
  - Interessanti perché facilmente implementabili in hardware, sono molto usati in LAN e in WAN
  - Si basano sulle proprietà delle divisioni di numeri binari, in pratica il resto deve essere sempre un certo numero, per es. 0, se un bit camba, il risultato della divisione cambia, soprattuto il resto.
  - Es CRC(7,4)

CRC(7,4)

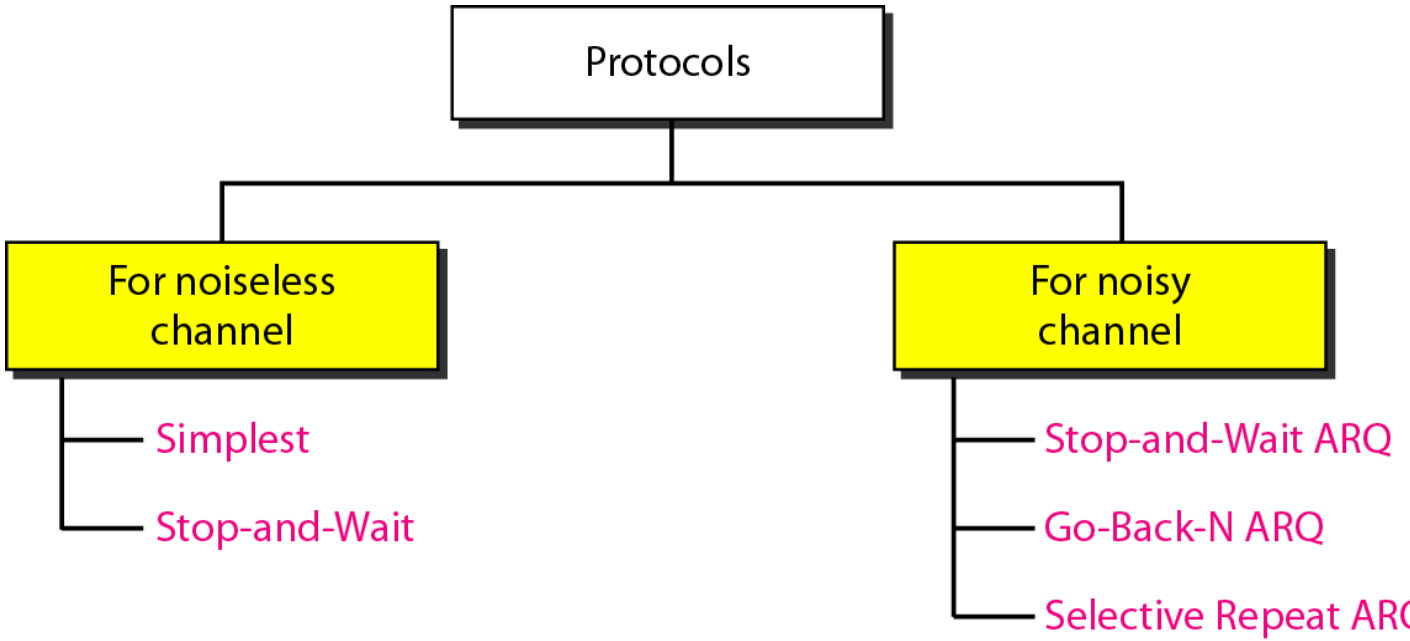
Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Esempi

- CRC-8 usato in ATM
- CRC-10 usato in ATM-AAL
- CRC-16 usato in HDLC
  - Ad esempio il **CRC16** rivela errori singoli e errori doppi, tutti gli errori con un numero dispari di bit, tutte le sequenze (error burst) di 16 bit consecutivi errati, il 99.997% di 17 bit e 99.998% di 18 bit consecutivi errati.
- CRC-32 usato in LAN

Flow Control

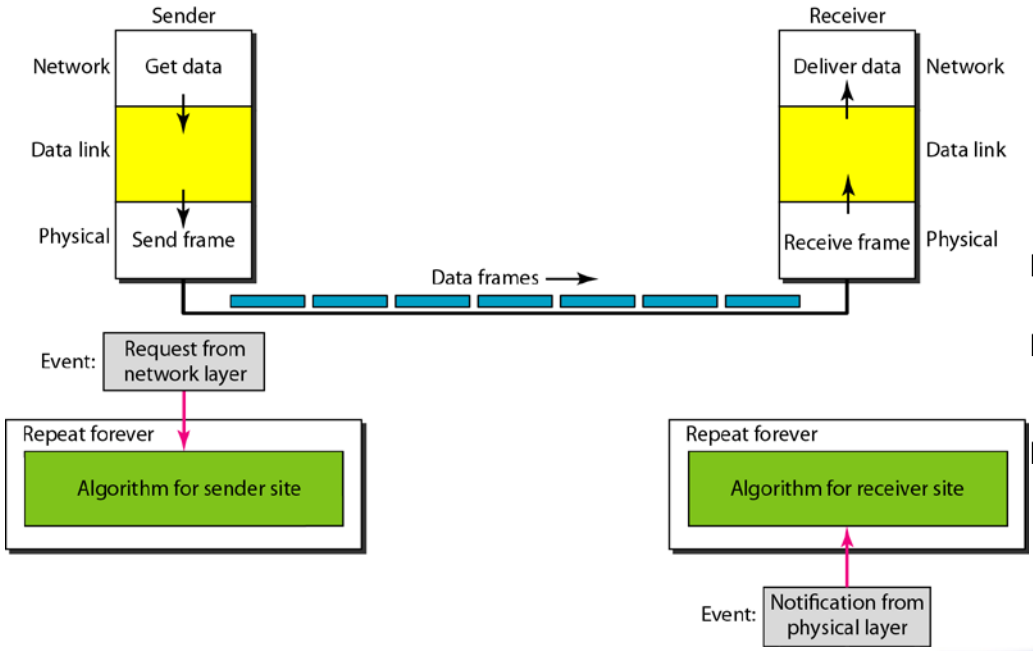
- Vediamo come si implementa il controllo di flusso
- Pseudo codice di esempio
- Definiamo prima alcune strutture e funzioni primitive
- ARQ = **A**utomatic **R**epeat re**Q**uest

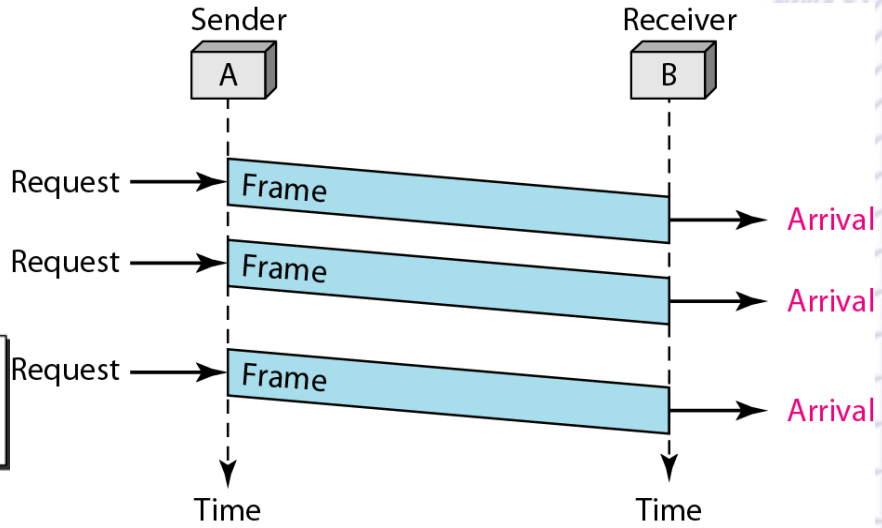


```
#define MAX_PKT 1024                                /* determines packet size in bytes */

typedef enum {false, true} boolean;                 /* boolean type */
typedef unsigned int seq_nr;                        /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;           /* frame_kind definition */

typedef struct {                                    /* frames are transported in this layer */
  frame_kind kind;                                  /* what kind of a frame is it? */
  seq_nr seq;                                       /* sequence number */
  seq_nr ack;                                       /* acknowledgement number */
  packet info;                                      /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Simplest Unrestricted

- Il più semplice: Dati in un'unica direzione
- Network layer ad entrambi i lati sempre pronti, buffer infiniti, processing time trascurabile, il canale di comunicazione non perde o danneggia i frame
- Nessun numero di sequenza: Uno spedisce sempre, l'altro aspetta e processa tutto in un attimo





```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                                /* buffer for an outbound frame */
  packet buffer;                          /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer); /* go get something to send */
      s.info = buffer;                    /* copy it into s for transmission */
      to_physical_layer(&s);              /* send it on its way */
  }                                       /* Tomorrow, and tomorrow, and tomorrow,
                                             Creeps in this petty pace from day to day
                                             To the last syllable of recorded time.
                                                 - Macbeth, V, v */
}


void receiver1(void)
{
  frame r;
  event_type event;                       /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);             /* only possibility is frame_arrival */
      from_physical_layer(&r);            /* go get the inbound frame */
      to_network_layer(&r.info);          /* pass the data to the network layer */
  }
}
```

Codice

- In alto il codice semplificato per il sender

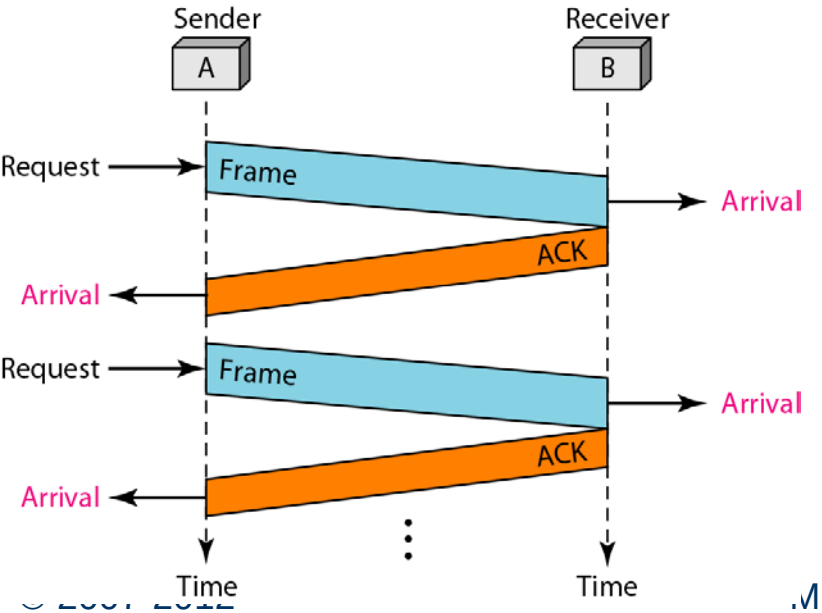
```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6       GetData();
7       MakeFrame();
8       SendFrame();                    //Send the frame
9    }
10 }
```

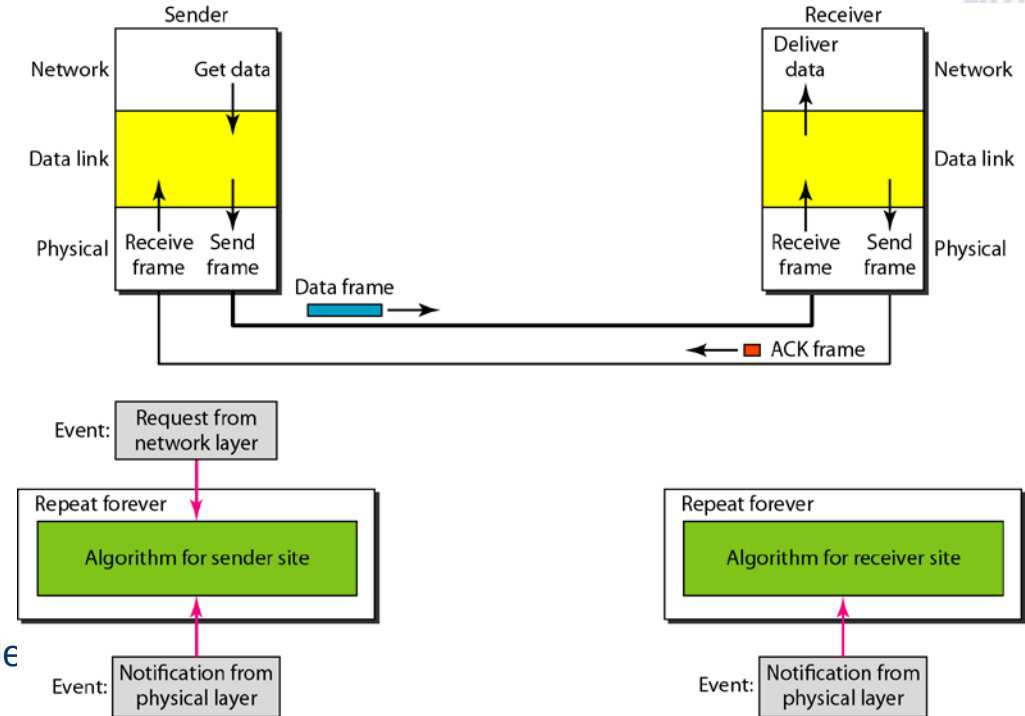
- In basso per il receiver

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();                  //Deliver data to network layer
9    }
10 }
```

Stop-and-Wait

- Togliamo la supposizione più improbabile: **Il network layer lato ricevente non ha buffer infinito** (o ha un tempo di processing finito)
- Il ricevente manda indietro un feedback, un frame dummy, dopo aver processato il frame ricevuto, dando il permesso di mandarne un altro al mittente
- Il traffico dati è simplex in un'unica direzione ma le frame vanno in entrambe le direzioni.





```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;                     /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* bye-bye little frame */
      wait_for_event(&event);           /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
      to_physical_layer(&s);            /* send a dummy frame to awaken sender */
  }
}
```

Simplex su Noisy channel

- Ora il canale ha rumore che potrebbe creare errori, frame persi o danneggiati
- Aggiungiamo un timer, il ricevente manda l'ack solo se il frame ricevuto è ok, se è danneggiato lo scarta
- Dopo un po' il sender rimanda il frame e lo fa fino a quando il frame arriva intatto
- Problema! E se si perde il frame di ack?
  - Il sender pensa che non sia arrivato e lo rimanda.
  - Il ricevente si trova con due copie dello stesso frame
  - Serve un numero di sequenza

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                       /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;            /* seq number of next outgoing frame */
  frame s;                              /* scratch variable */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;               /* initialize outbound sequence numbers */
  from_network_layer(&buffer);          /* fetch first packet */
  while (true) {
       s.info = buffer;                 /* construct a frame for transmission */
       s.seq = next_frame_to_send;      /* insert sequence number in frame */
       to_physical_layer(&s);           /* send it on its way */
       start_timer(s.seq);              /* if answer takes too long, time out */
       wait_for_event(&event);          /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
            from_physical_layer(&s);    /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                 stop_timer(s.ack);     /* turn the timer off */
                 from_network_layer(&buffer);     /* get the next one to send */
                 inc(next_frame_to_send);     /* invert next_frame_to_send */
            }
       }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);          /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {    /* a valid frame has arrived. */
            from_physical_layer(&r);    /* go get the newly arrived frame */
            if (r.seq == frame_expected) {/* this is what we have been waiting for. */
                 to_network_layer(&r.info);     /* pass the data to the network layer */
                 inc(frame_expected); /* next time expect the other sequence nr */
            }
            s.ack = 1 − frame_expected;   /* tell which frame is being acked */
            to_physical_layer(&s);      /* send acknowledgement */
       }
  }
}
```

Sliding windows

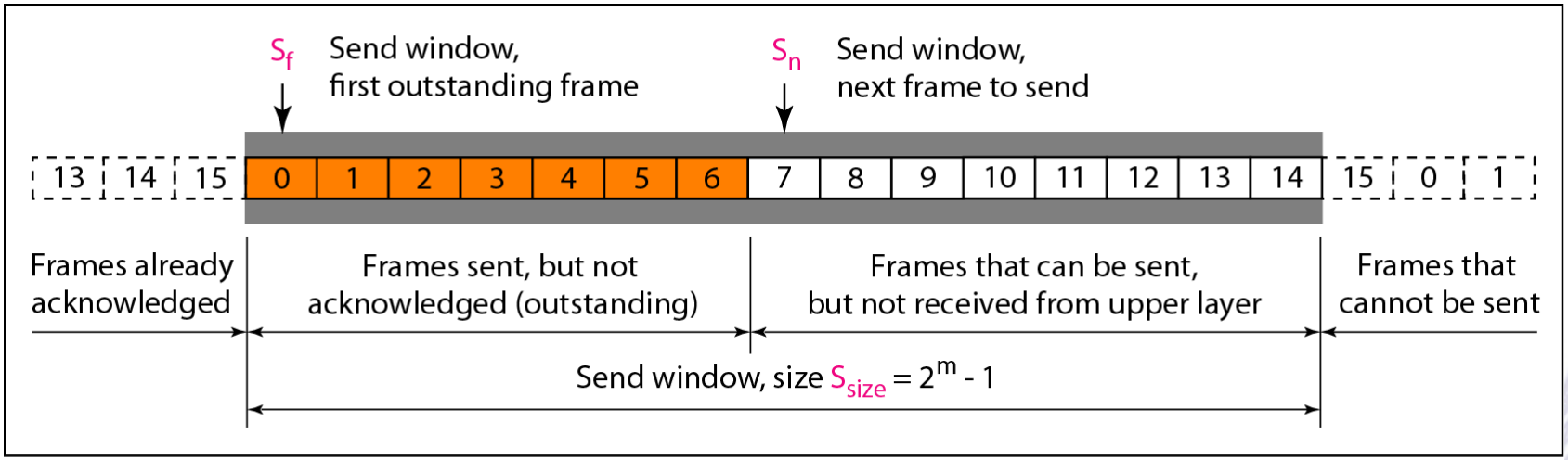
- Posso usare numeri di sequenza a **n** bit (da **0** a **2^n-1**)
- Il sender sa ad ogni istante quali sono i numeri di sequenza che può mandare
  - Questi sono nella cosiddetta “Sending window” e rappresentano i frame mandati ma non ancora acknowledged.
  - Quando un frame arriva dal network layer gli viene dato il numero più alto di sequenza
  - Quando arriva un **ack** il limite basso della finestra viene incrementato di uno
  - Il sender deve tenere in memoria tutti i buffer non acknowledged fino ad un massimo di “***n”.***
  - Se la finestra arriva al massimo n il sender deve chiedere al network layer di fermarsi

Sliding window

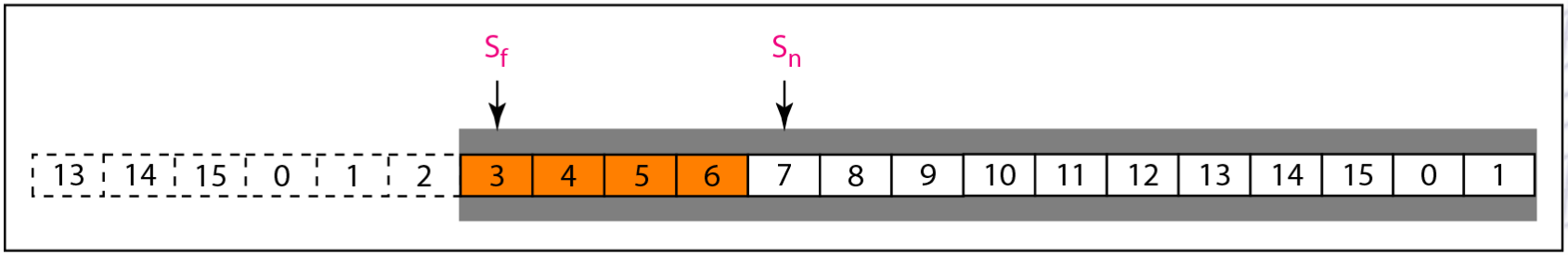
- Il ricevente ha un'equivalente finestra
  - Contiene i frame che può accettare
  - Se arriva un frame fuori finestra lo scarta
  - Quando arriva un frame con numero pari all'estremo inferiore della finestra, genera un ack e incrementa la finestra
  - Le dimensioni della finestra rimangono sempre uguali
  - Finestra di dimensioni 1 significa che vengono ricevuti i frame in ordine

Sender window m=4

- Sliding windows (go-back-N ARQ con m=4, finestra 0-15, lato mittente

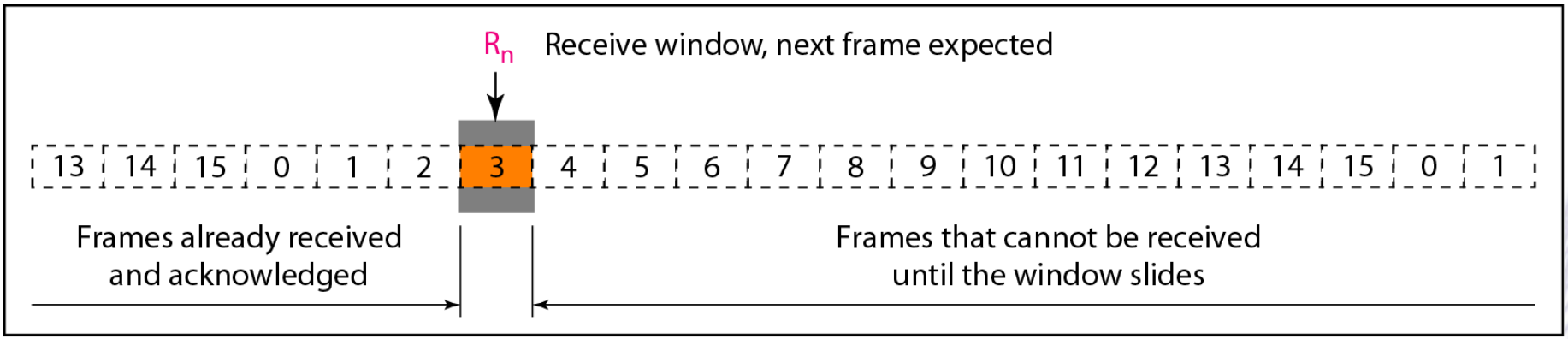


a. Send window before sliding

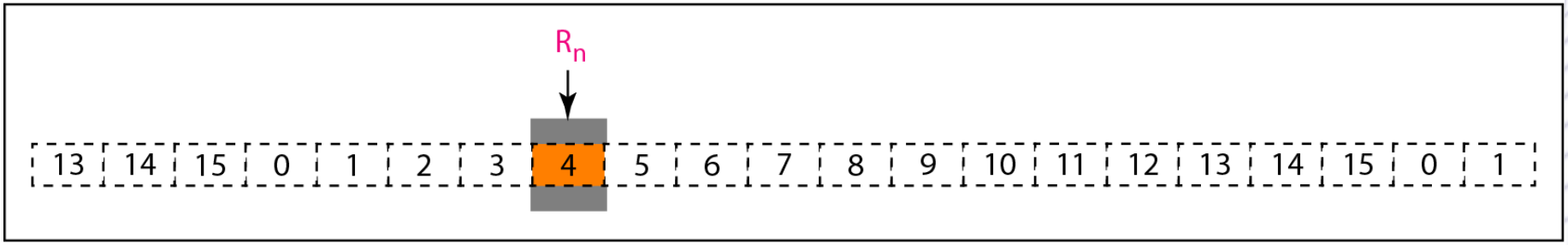


b. Send window after sliding

Lato receiver

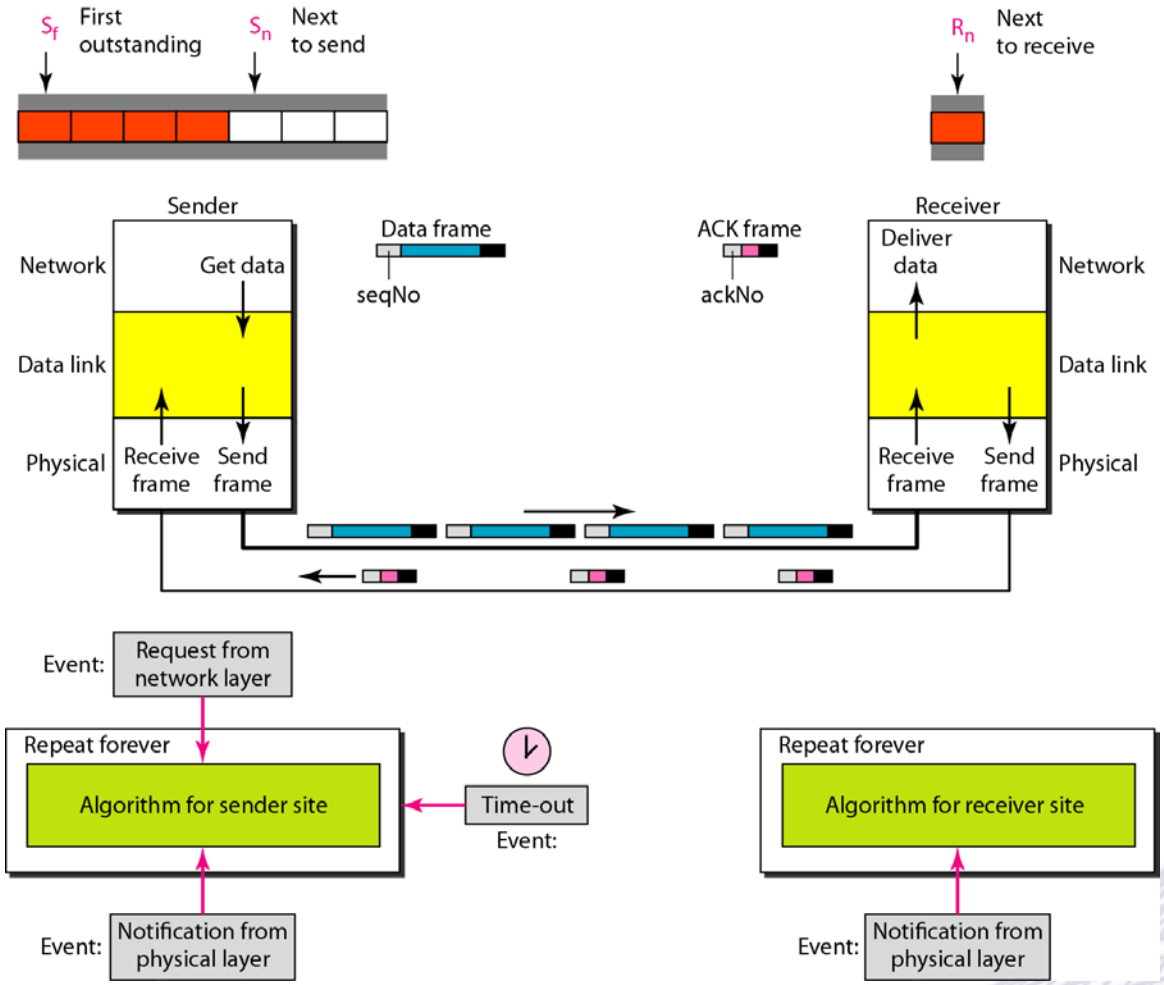


a. Receive window



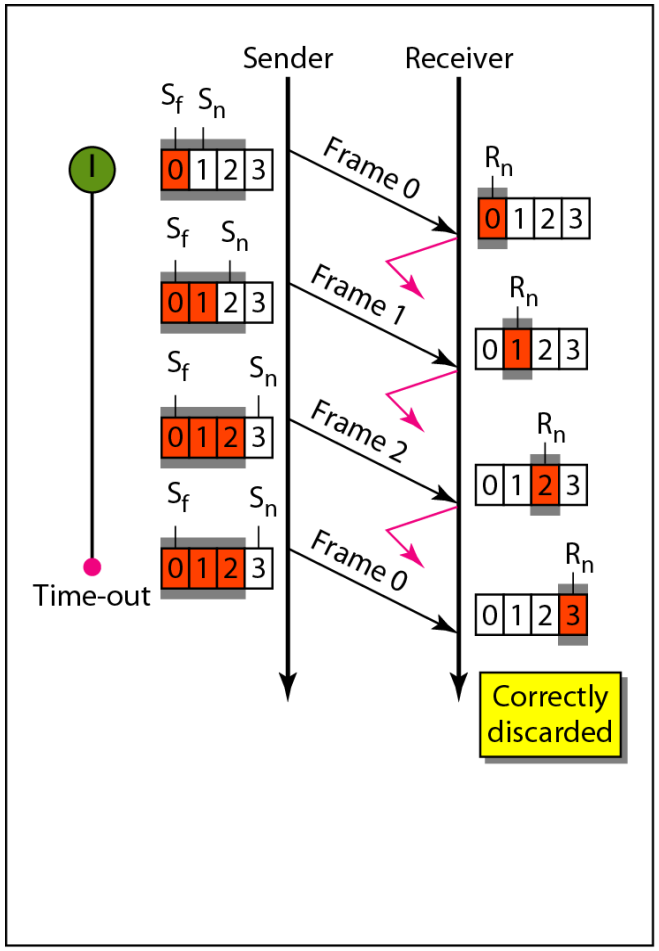
b. Window after sliding

Protocollo go-back-n ARQ

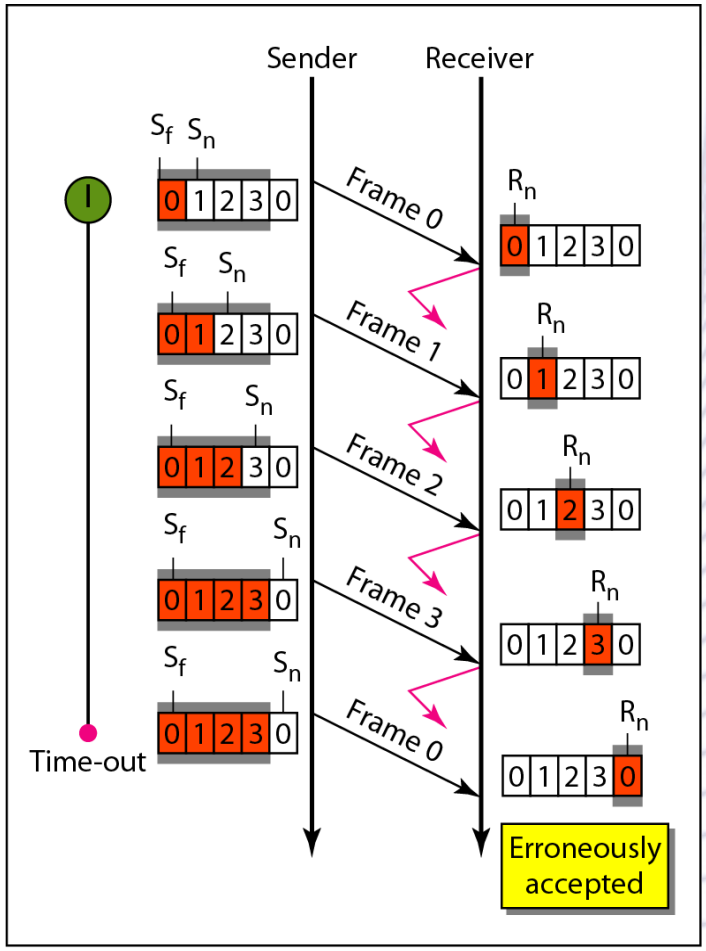


Flusso

- La grandezza della finestra mittente deve essere sempre minore di 2^m altrimenti si verifica il caso della seconda finestra



a. Window size < 2^m



b. Window size = 2^m

Codice sender

```
 1  Sw = 2^m - 1;
 2  Sf = 0;
 3  Sn = 0;
 4
 5  while (true)                          //Repeat forever
 6  {
 7   WaitForEvent();
 8    if(Event(RequestToSend))            //A packet to send
 9    {
10       if(Sn-Sf >= Sw)                 //If window is full
11             Sleep();
12       GetData();
13       MakeFrame(Sn);
14       StoreFrame(Sn);
15       SendFrame(Sn);
16       Sn = Sn + 1;
17       if(timer not running)
18            StartTimer();
19    }
20
```

Codice sender

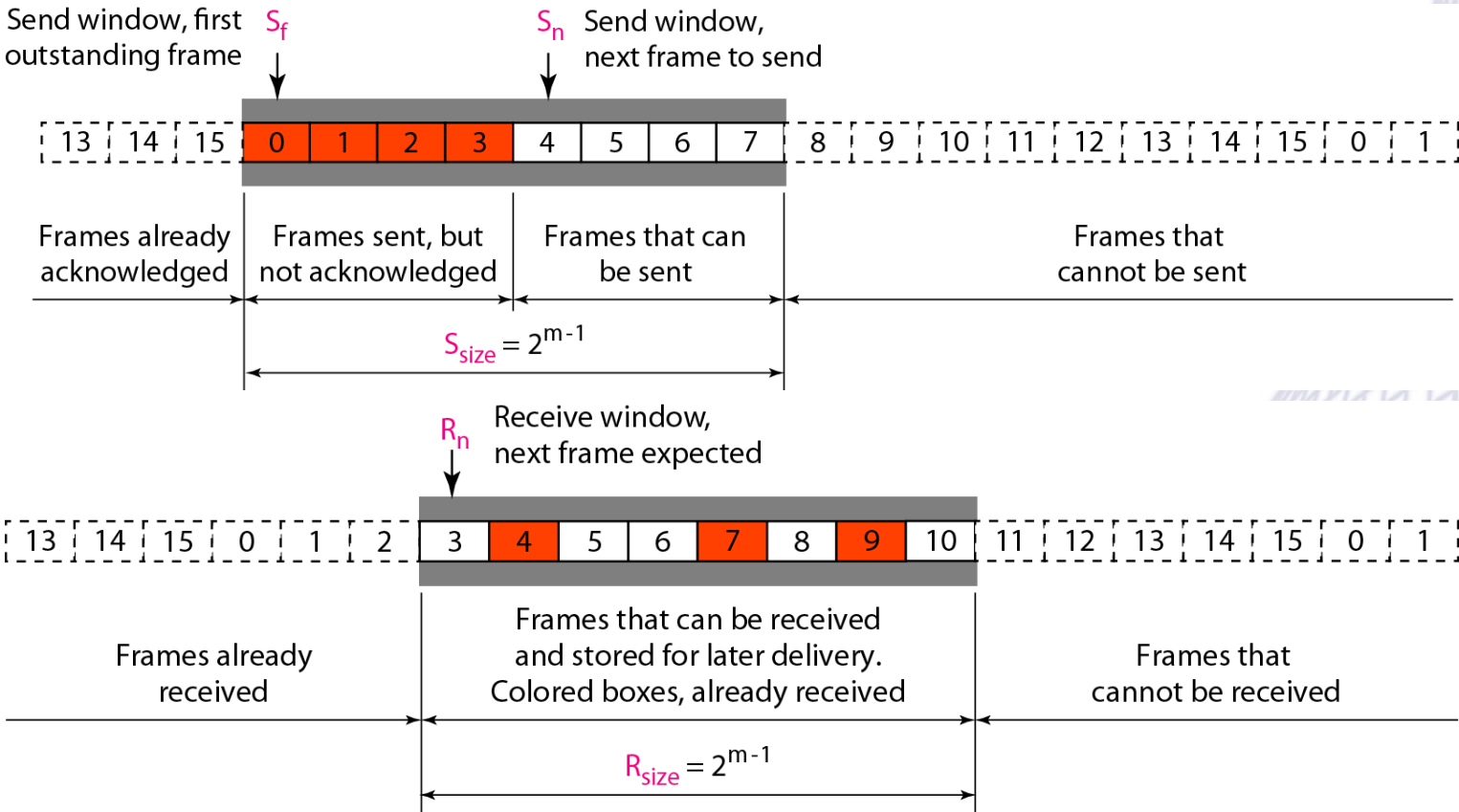
```
21  if(Event(ArrivalNotification)) //ACK arrives
22  {
23     Receive(ACK);
24     if(corrupted(ACK))
25          Sleep();
26     if((ackNo>Sf)&&(ackNo<=Sn))  //If a valid ACK
27     While(Sf <= ackNo)
28      {
29       PurgeFrame(Sf);
30       Sf = Sf + 1;
31      }
32      StopTimer();
33  }
34
35  if(Event(TimeOut))              //The timer expires
36  {
37   StartTimer();
38   Temp = Sf;
39   while(Temp < Sn);
40    {
41     SendFrame(Sf);
42     Sf = Sf + 1;
43    }
44  }
45 }
```

Codice receiver

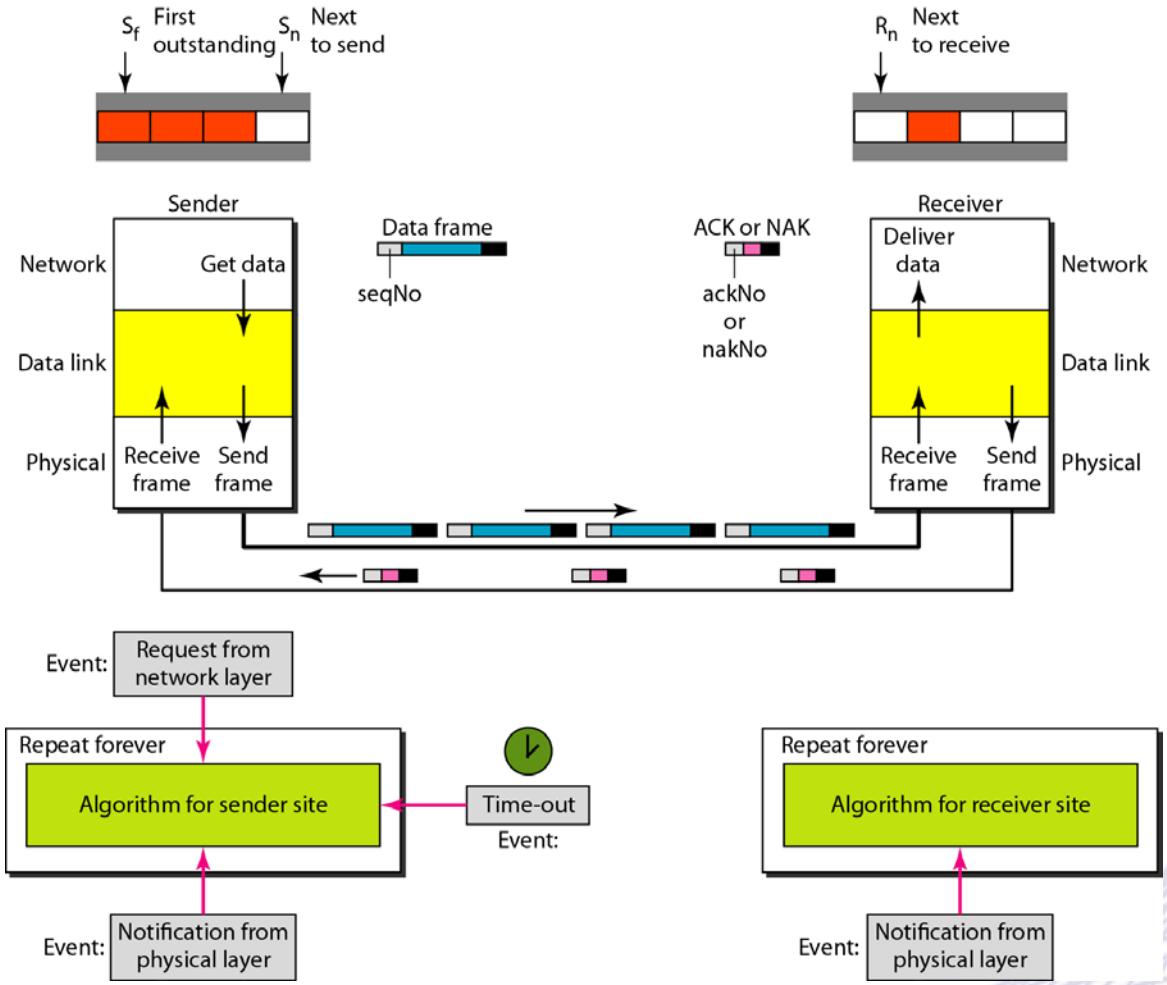
```
 1  Rn = 0;
 2
 3  while (true)                          //Repeat forever
 4  {
 5    WaitForEvent();
 6
 7    if(Event(ArrivalNotification)) /Data frame arrives
 8    {
 9       Receive(Frame);
10       if(corrupted(Frame))
11            Sleep();
12       if(seqNo == Rn)                  //If expected frame
13       {
14         DeliverData();                 //Deliver data
15         Rn = Rn + 1;                   //Slide window
16         SendACK(Rn);
17       }
18    }
19  }
```

Selective repeat

- Con sliding window, se non arriva l'ACK del frame 3 ma sono arrivati quelli di 4,5 e 6, devo rispedire tutti da 3 a 6
- Invece potrei tenermi una finestra anche nel ricevente e richiedere la spedizione solo di quelli

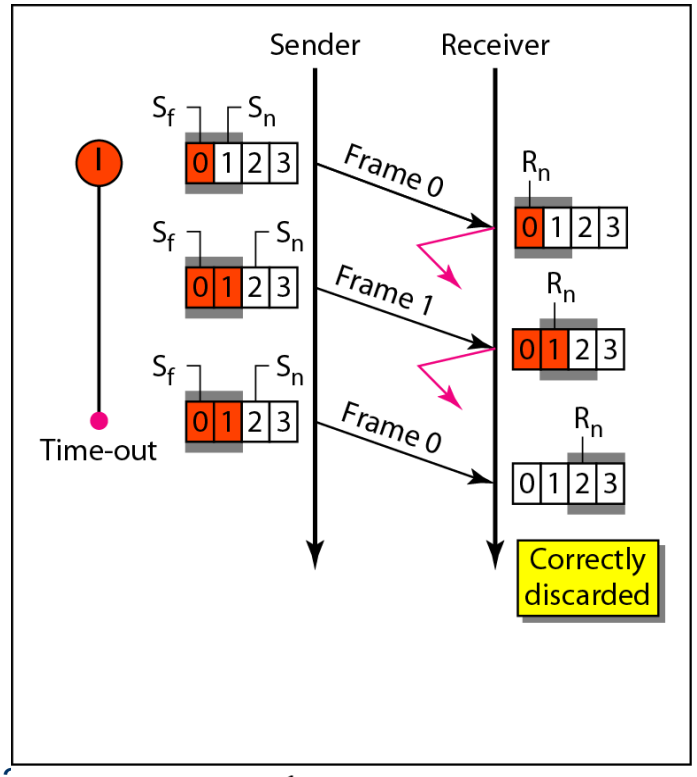


Selective repeat

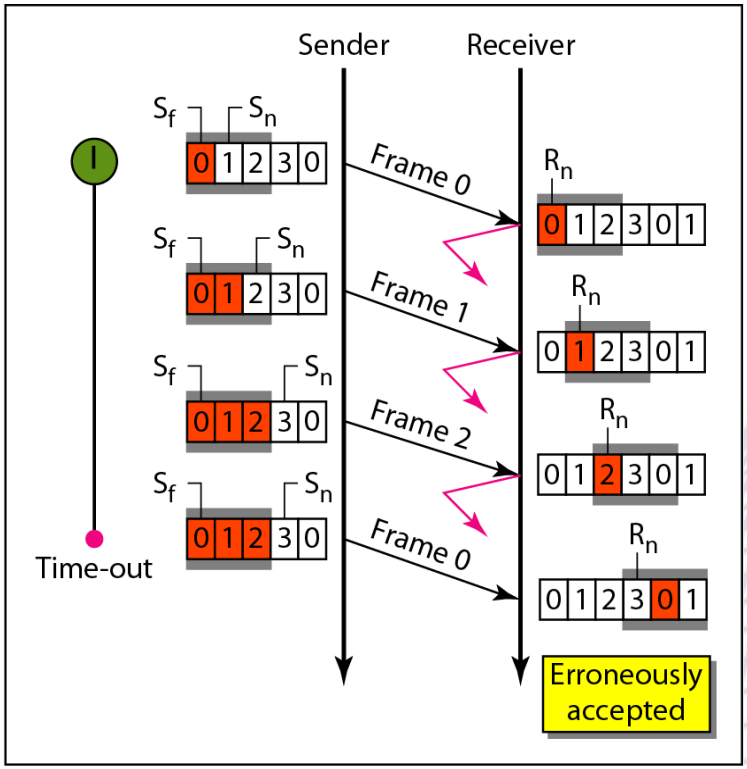


finestra

- La dimensione della finestra del sender e del receiver deve essere al massimo $2^{(m-1)}$, quindi la metà di 2^m
  - esempio con m=2

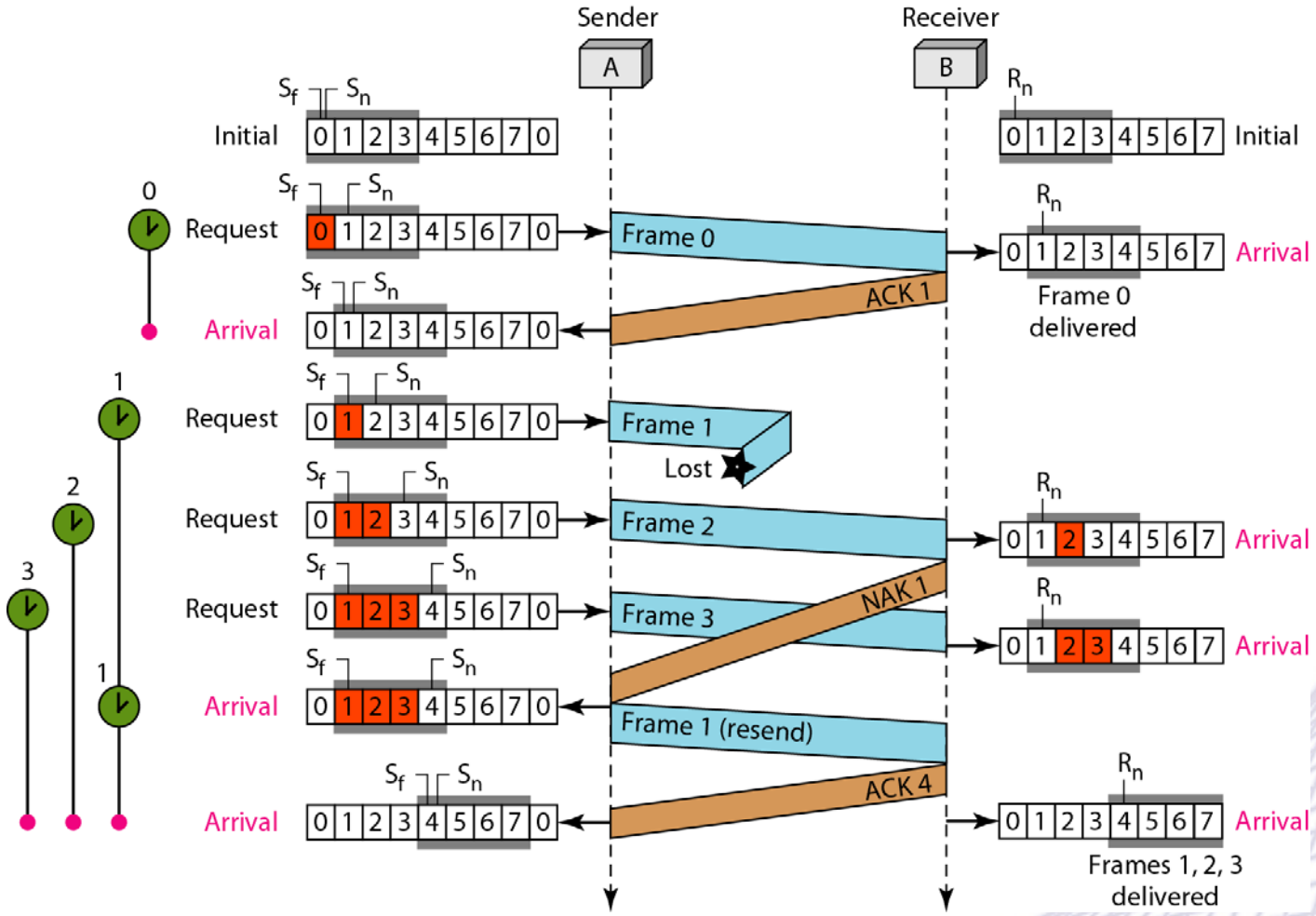


a. Window size = 2^{m-1}



b. Window size > 2^{m-1}

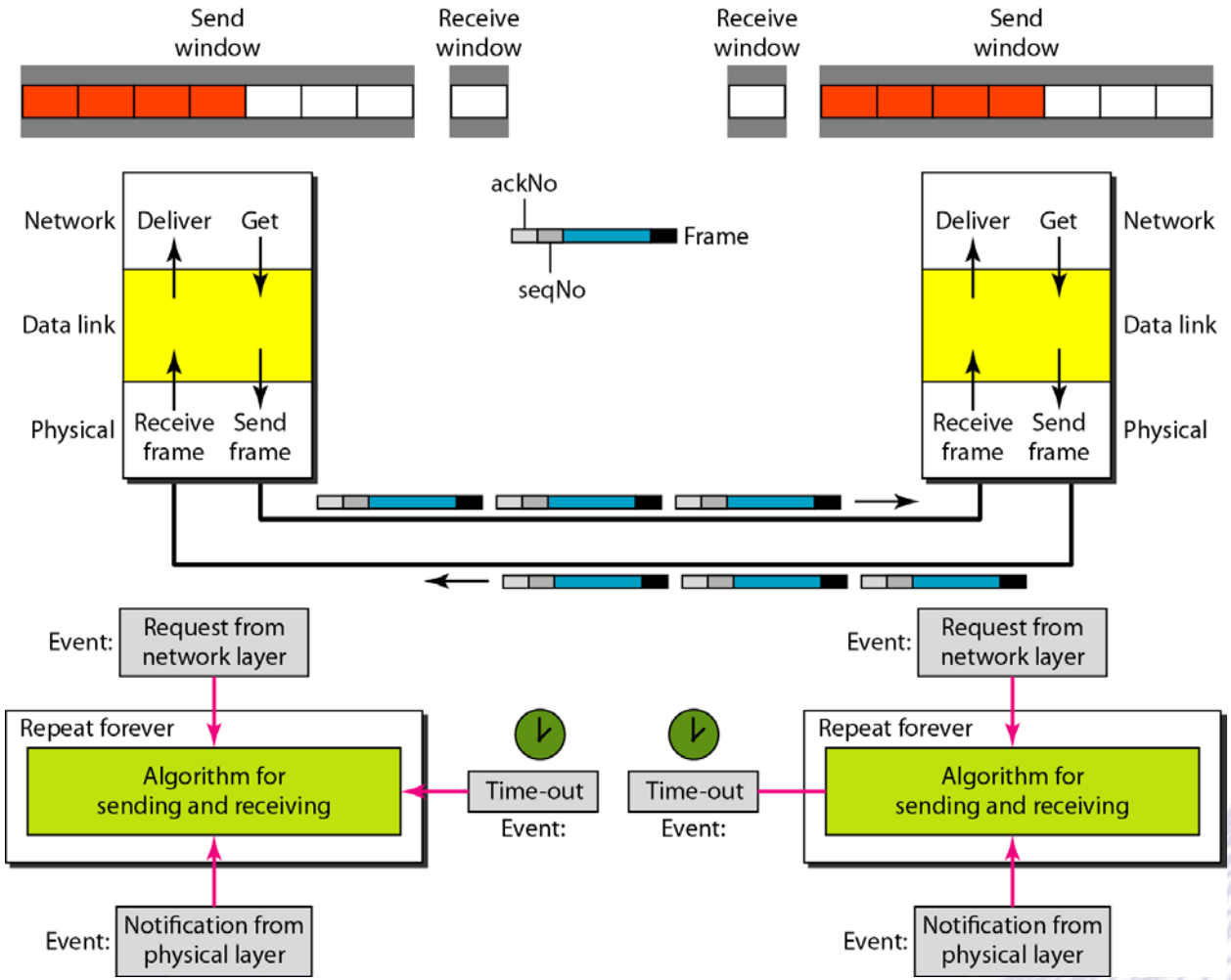
Flusso


Sender
Receiver
A
B
Initial
Request
Arrival
Frame 0
ACK 1
Frame 0 delivered
Frame 1
Lost
Frame 2
Frame 3
NAK 1
Frame 1 (resend)
ACK 4
Frames 1, 2, 3 delivered

Miglioramenti

- Uno spreco usare il canale di ritorno solo per gli ack
  - Posso usare un campo “type” che mi dice se il frame è di dati o di ack
  - Posso anche aspettare di mandare il frame di ack per mandarlo insieme al prossimo frame di dati in quella direzione (**piggybacking**)
  - In questo modo spreco solo pochi bit invece di mandare un intero frame, completo di header, inoltre ho meno interrupt da gestire
  - Non posso aspettare troppo altrimenti l’altro lato va in timeout e mi rimanda il frame anche se lo avevo gia’ ricevuto ok.

Go-back-N + piggyback



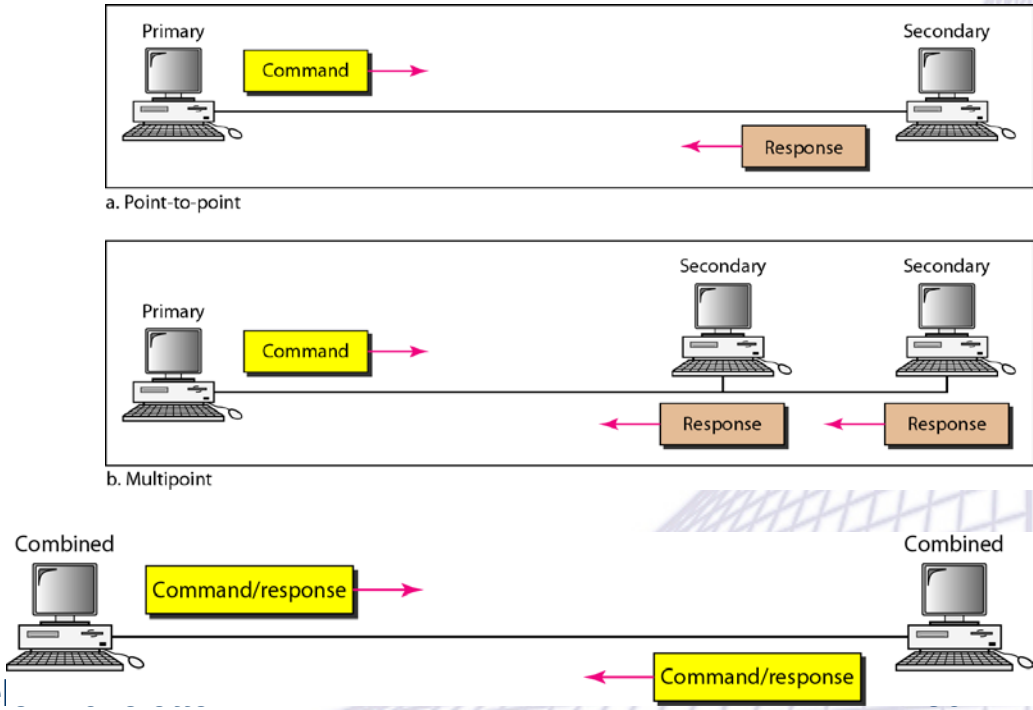
Esempi pratici

- HDLC classico protocollo bit oriented usato da diverse applicazioni con diverse varianti
- PPP il protocollo data link usato per connettersi a internet da casa

HDLC

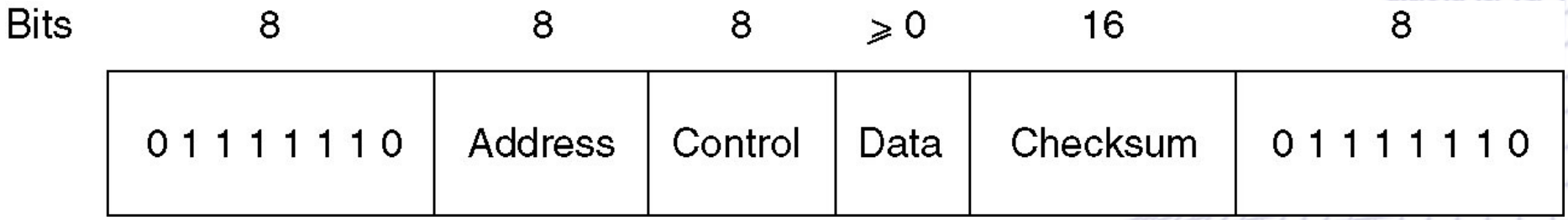
- Famiglia di protocolli bit-oriented
  - vecchia ma ancora molto usata: SDLC (mainframe IBM) ADCCP (ANSI)
  - HDLC (ISO) di cui esistono diverse versioni. Tutte molto simili a meno di piccoli ma irritanti differenze

- NRM – Normal Response Mode: Una stazione primaria e una o più stazioni secondarie che possono solo rispondere
- ABM – Asyncronous Balanced Mode: ognuna delle due stazioni è sia primaria che secondaria. Usata per collegamenti punto a punto, la più utilizzata attualmente

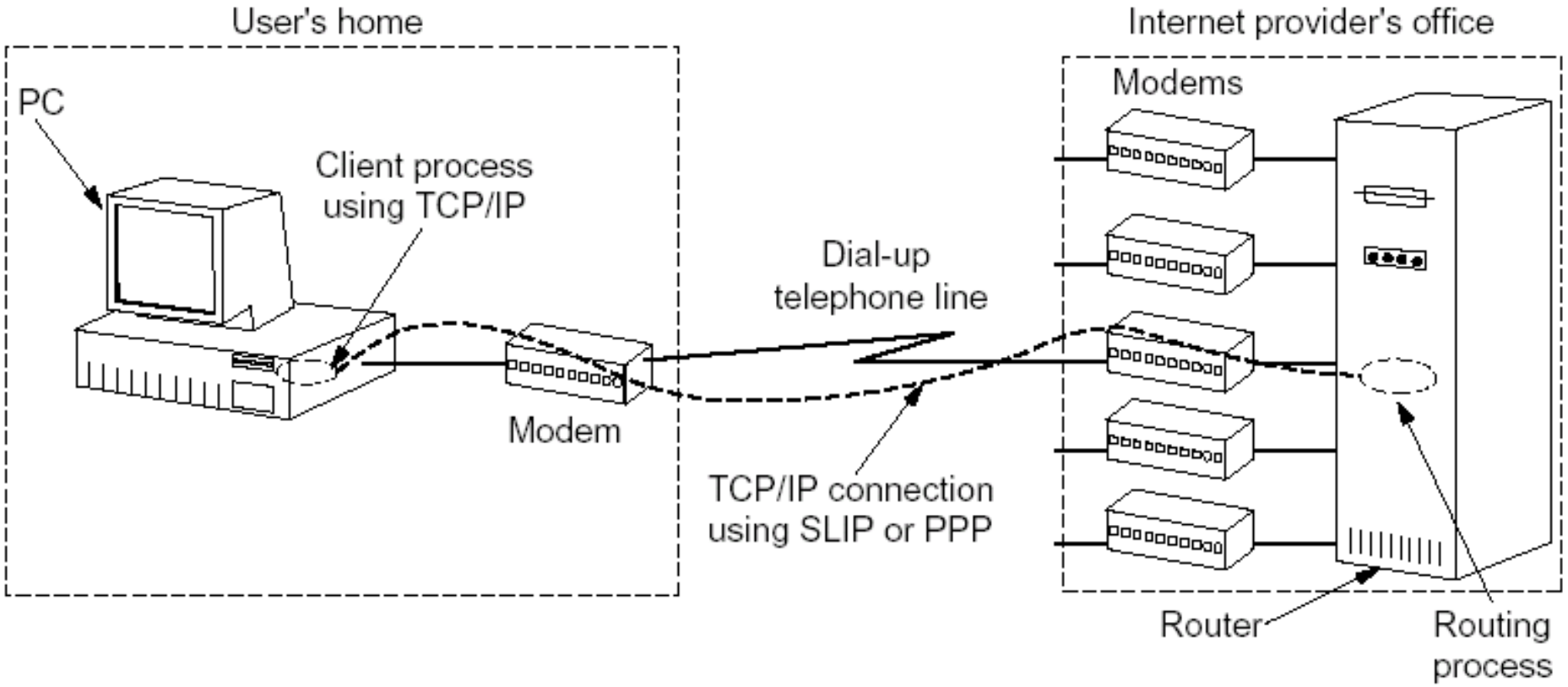


HDLC

- Frame HDLC
  - Address: per linee con terminali multipli per identificarli
  - Control: Numeri di sequenza, acknowledgements
  - Data: Qualsiasi informazione di lunghezza arbitraria, l'efficenza del checksum (fissa a 16) scende con l'aumentare della lunghezza
  - Checksum: un codice di tipo CRC (2 o 4 Byte)
  - FLAG: 01111110 trasmesso di continuo su linee idlee punto-punto
  - Il frame minimo sono 32bit (0 dati) esclusi i FLAG



Internet via modem L2

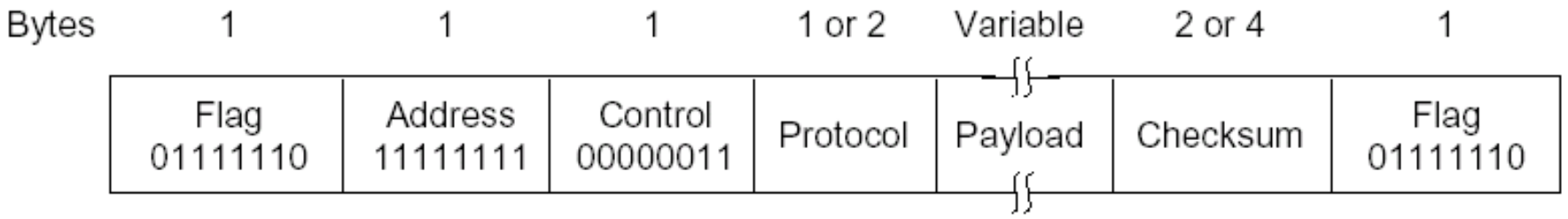


SLIP

- Un protocollo di DataLink per traffico Internet sopra una linea telefonica
- Attacca ad ogni pacchetto IP un carattere 0x**C0**
- Se presente nei dati viene usata la sequenza 0x**DB**, 0x**DC**
- Alcune implementazioni di SLIP mettono il pacchetto all'inizio e alla fine
- Problemi:
  - Non ha error detection o error correction
  - Supporta solo IP e non per esempio Novell
  - Entrambi i lati devono conoscere gli indirizzi IP reciproci in anticipo
  - Non esiste alcuna forma di autenticazione
  - Non esiste uno standard Internet approvato

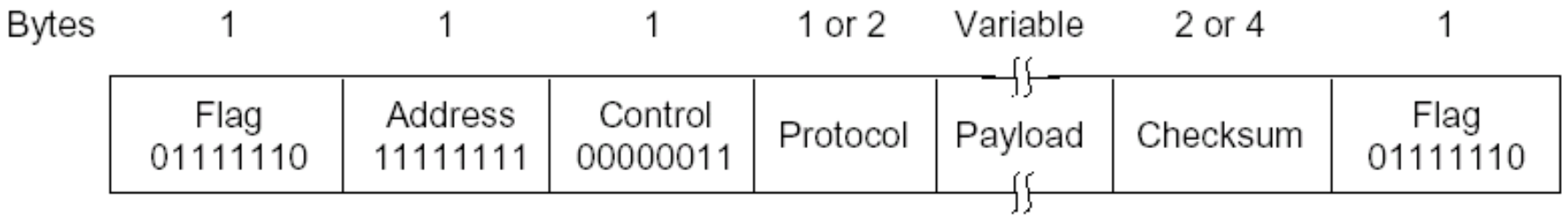
PPP – RFC 1661

- Protocollo Byte oriented
- supporta diversi protocolli di rete
- negozia i numeri IP al momento della connessione (utile per connessioni da casa in cui c'è bisogno di un indirizzo IP temporaneo)
- Definisce procedure di autenticazione.
- Non supporta controllo di flusso
- ha un controllo di errori molto semplice, se ci sono errori il frame viene scartato senza notifica, quindi possono anche arrivare frame fuori ordine



PPP – RFC 1661

- Non ha meccanismi di indirizzamento per configurazioni multipunto
- Il FLAG ha lo stesso valore di quello HDLC ma viene trattato come un byte
- I campi Control e Address hanno valori fissi e possono essere omessi in seguito a contrattazione
- Campo Protocol: serve per decidere quale protocollo si trova nel payload (IP, IPX, OSI CLNP, XNS, AppleTalk)
- Payload di lunghezza variabile (fino al default 1500) se i dati non bastano a riempire il frame viene fatto byte stuffing

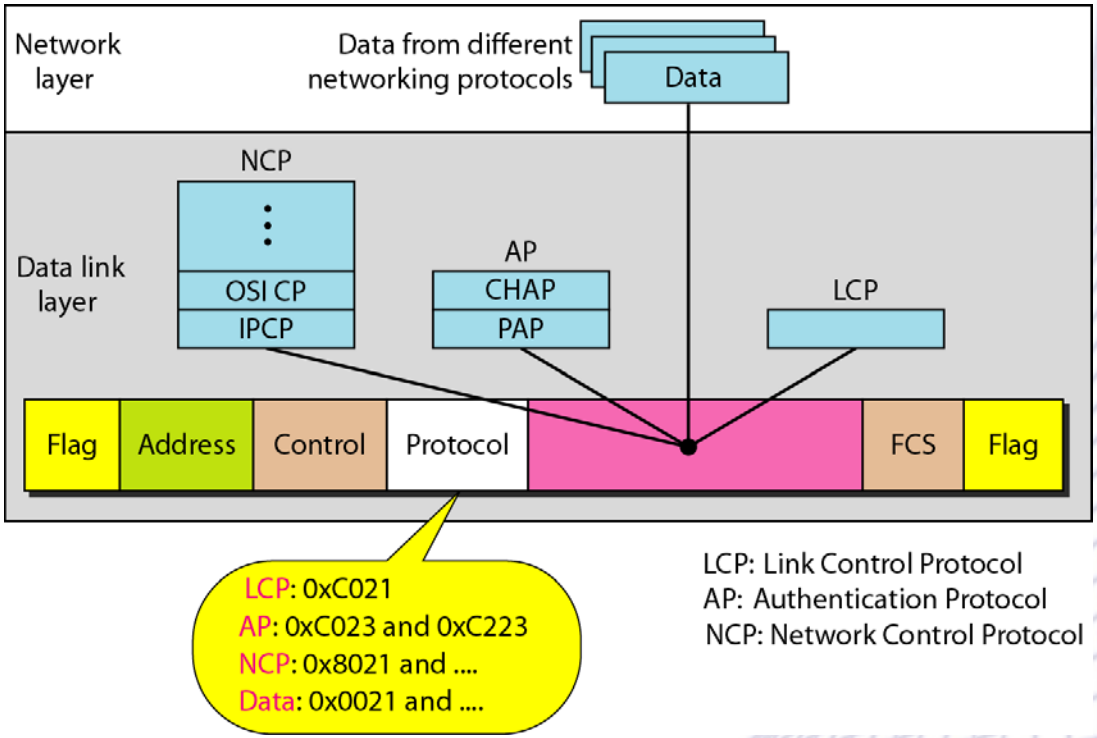


PPP

- L’utente da casa chiama l’ISP per rendere il PC un host della rete internet, usando un modem.
- Il modem dell’ISP risponde e stabilisce una connessione fisica.
- Il PC manda alcuni pacchetti LCP (Link Control Protocol) nel payload di pacchetti PPP per selezione i vari parametri di PPP
- Quindi vengono scambiati i pacchetti NCP (Network Control Protocol) per configurare il L3 (indirizzo IP etc…)
- Da questo momento il PC manda e riceve pacchetti come un qualsiasi host collegato permanentemente

Differenze

- PPP scelto per somigliare a HDLC
- PPP è byte oriented mentre HDLC è bit oriented
- Con PPP tutti i frame hanno un numero intero di byte
- Supporta protocolli multipli



Campi di PPP

- Address vale sempre 11111111 per indicare che tutte le stazioni devono accettare il frame
- Control vale 00000011 per indicare un frame non numerato.
  - Quindi PPP non fornisce trasmissioni affidabili usando numeri di sequenza e acknoledgement di default
  - Lo si può fare in ambienti noisy ma in pratica lo si fa raramente.

Campi PPP

- Il payload è variabile fino ad un massimo contrattato (nella fase LCP), altrimenti vale 1500 Bytes
- Checksum ha 2 bytes negoziabili fino a 4

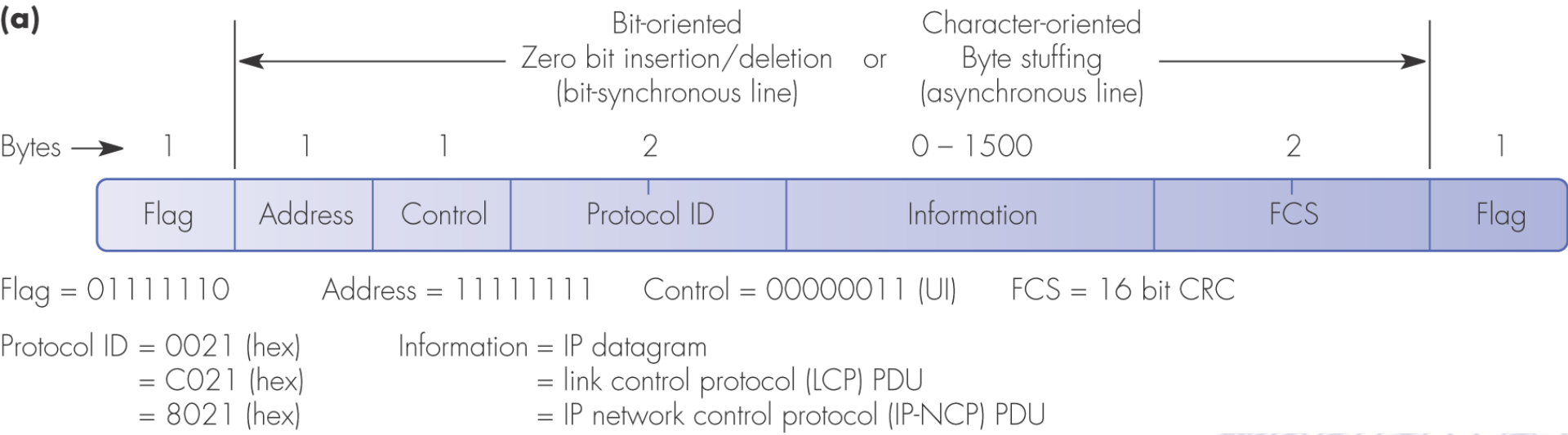
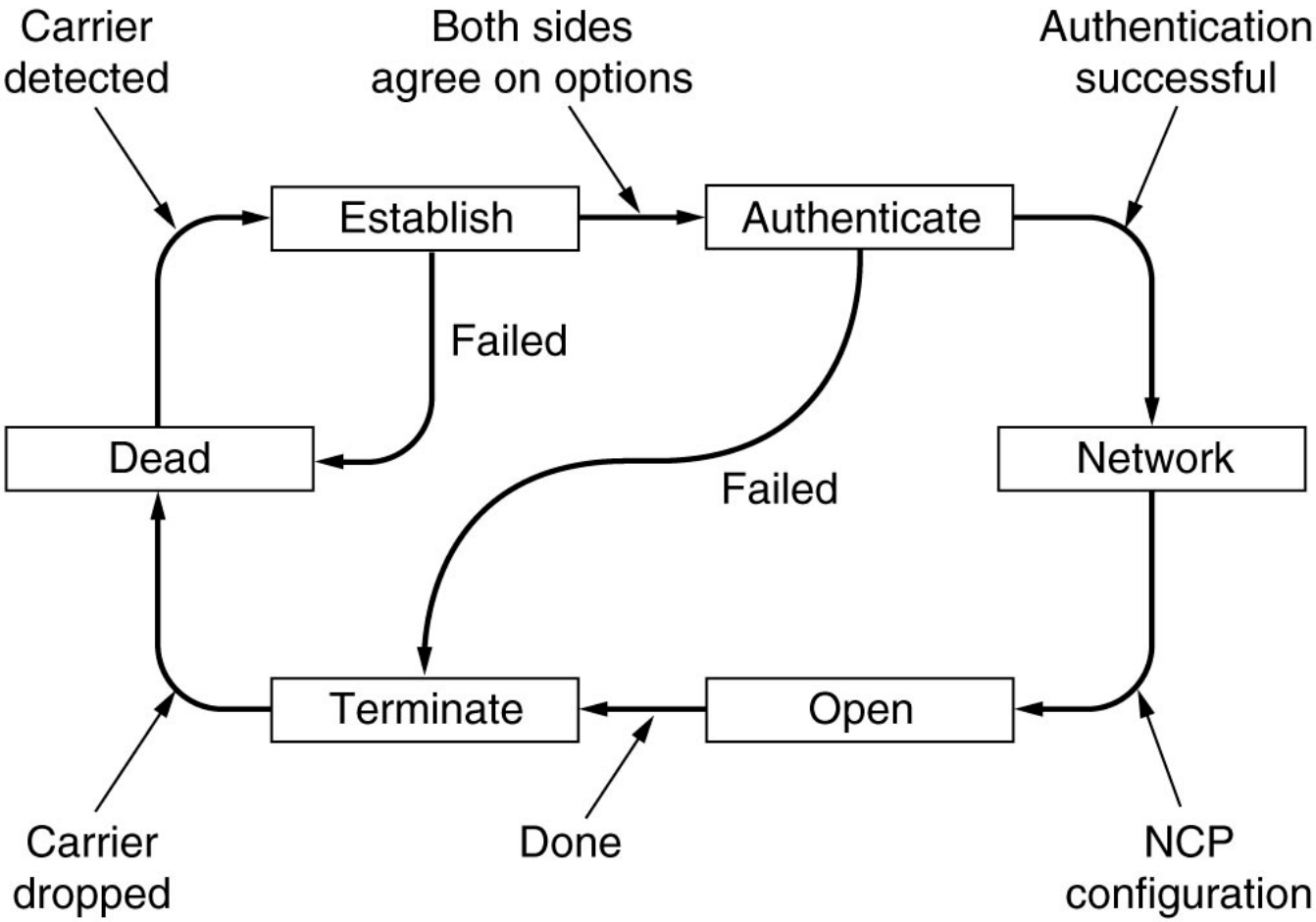
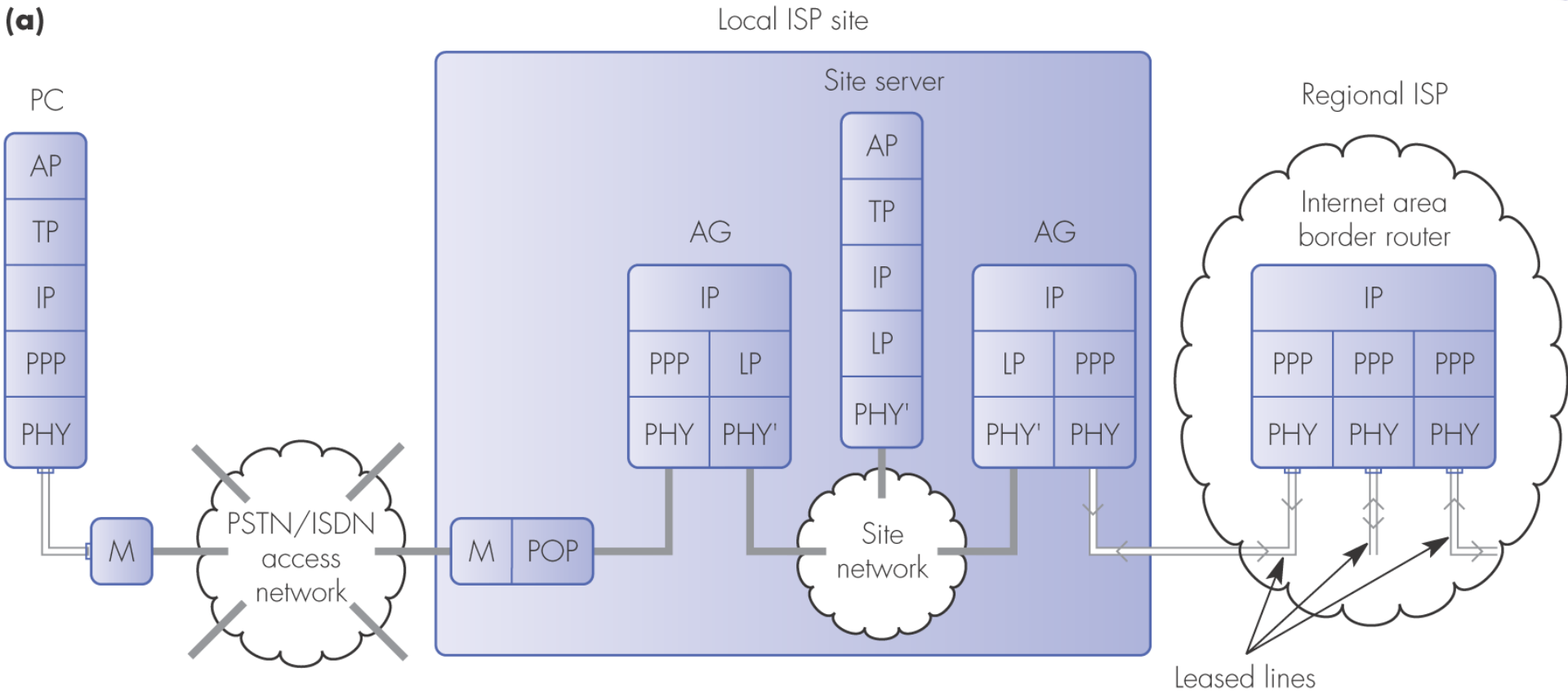


Diagramma di fase



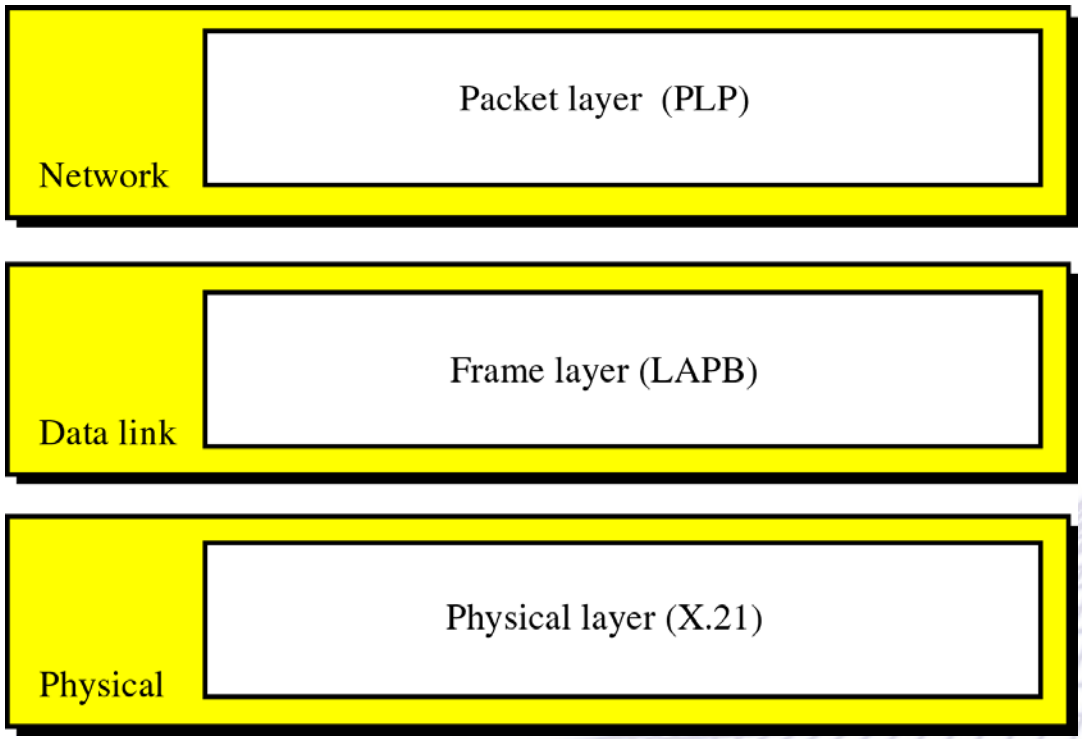
PPP nello stack



POP = point of presence
AG = access gateway
LP = site link protocol
AP = application protocol
TP = transport protocol
PPP = point-to-point protocol

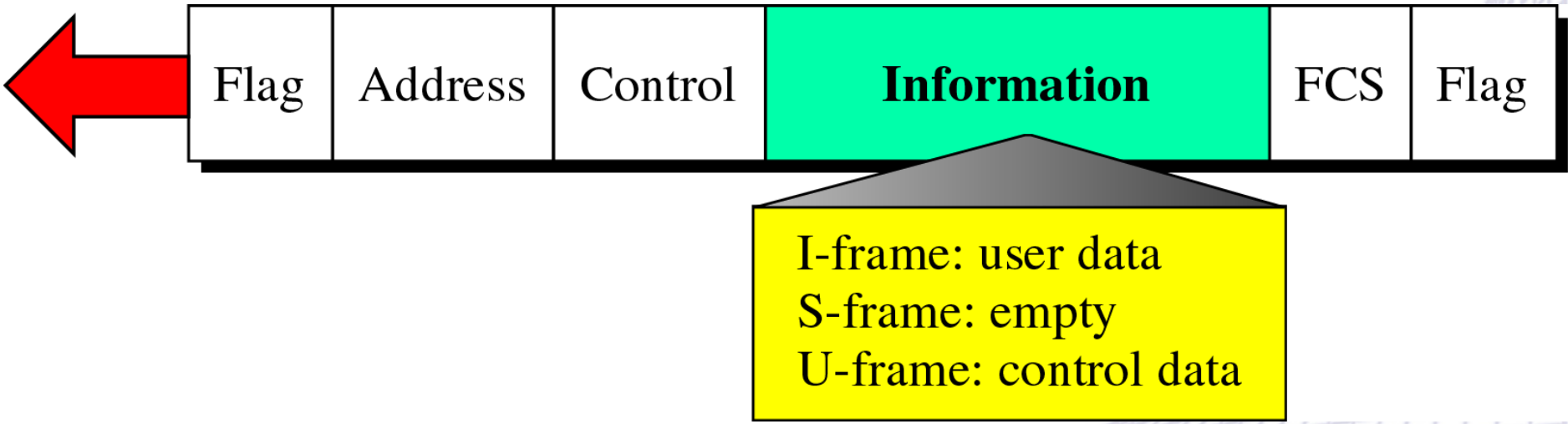
X.25 la prima rete pubblica

- In realtà progettato per reti private, quindi i pacchetti dati degli utenti devono essere incapsulati nei pacchetti di rete di X.25
- Nasce negli anni ’70 nel periodo dei monopoli telefonici
- Molto lento, 64 kbps

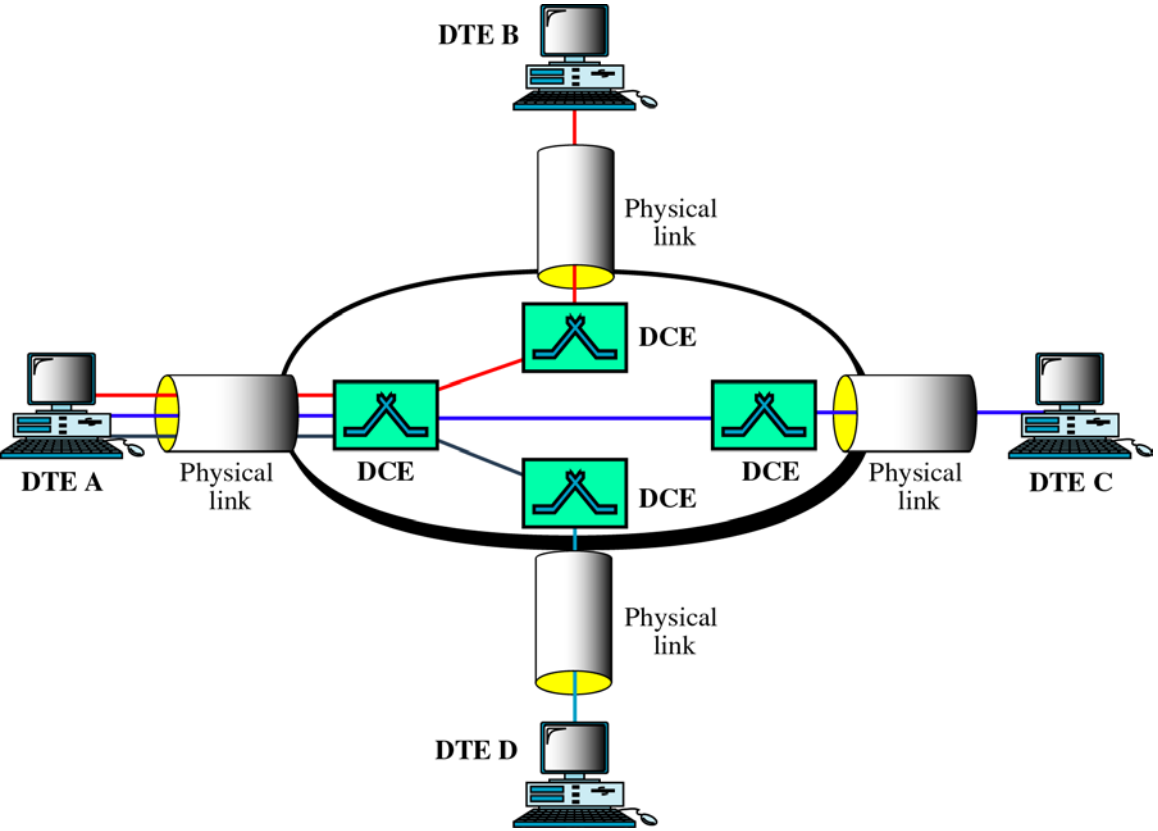


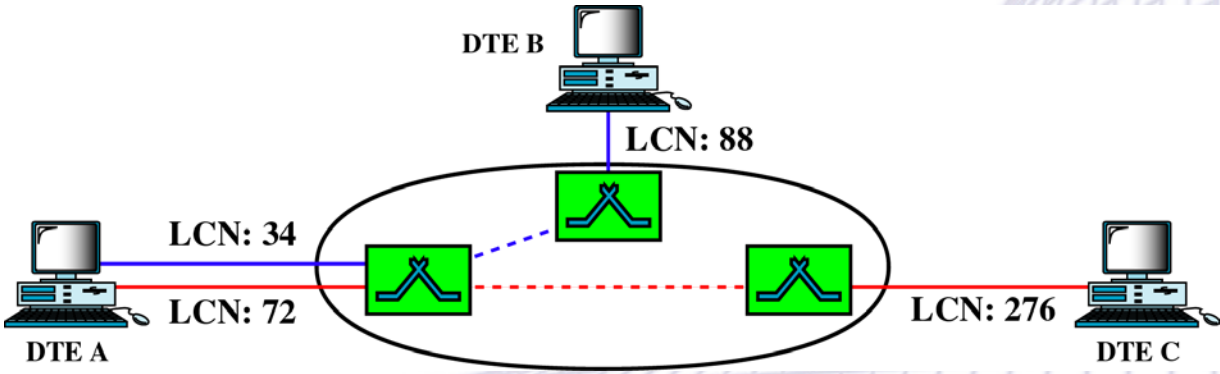
X.25

- Vari meccanismi di controllo e di flusso, sia a livello di rete che a livello data-link. Molto sovraccarico necessario negli anni ’70 quando i mezzi trasmissivi erano più soggetti ad errori
- Si telefona ad un computer remoto per stabilire una connessione
- Pacchetti di dati con 3 Byte di header e fino a 128 Byte di dati
- Header: 12 bit per il numero di connessione, un numero sequenziale e un ack



Circuiti virtuali X.25



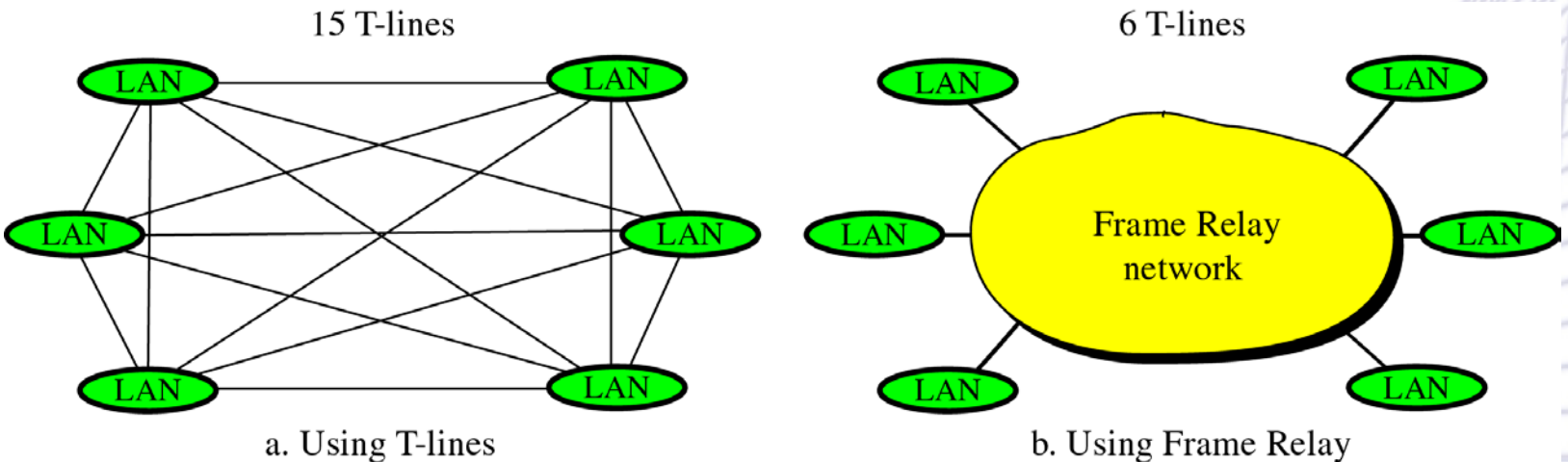


Frame Relay

- Successore di X.25 negli anni ’80
- Connection Oriented ma senza Error Correction o Flow Control (grazie a link migliori)
  - Implementa Error Detection ma solo a livello di DataLink. Eventuali frame danneggiati vengono gestiti da livelli superiori

Frame Relay

- Implementa solo strato fisico e di data-link
  - quindi può essere usato per esempio come trasporto per IP su internet
  - Viene usato per connettere diverse LAN su WAN dal momento che permette pacchetti fino a 9000 byte può mandare pacchetti senza problemi di frammentazione. Simula quindi una LAN estesa su WAN
  - Meno costosa di altre tecnologie LAN come ATM o T-lines (link dedicati o Sonet)

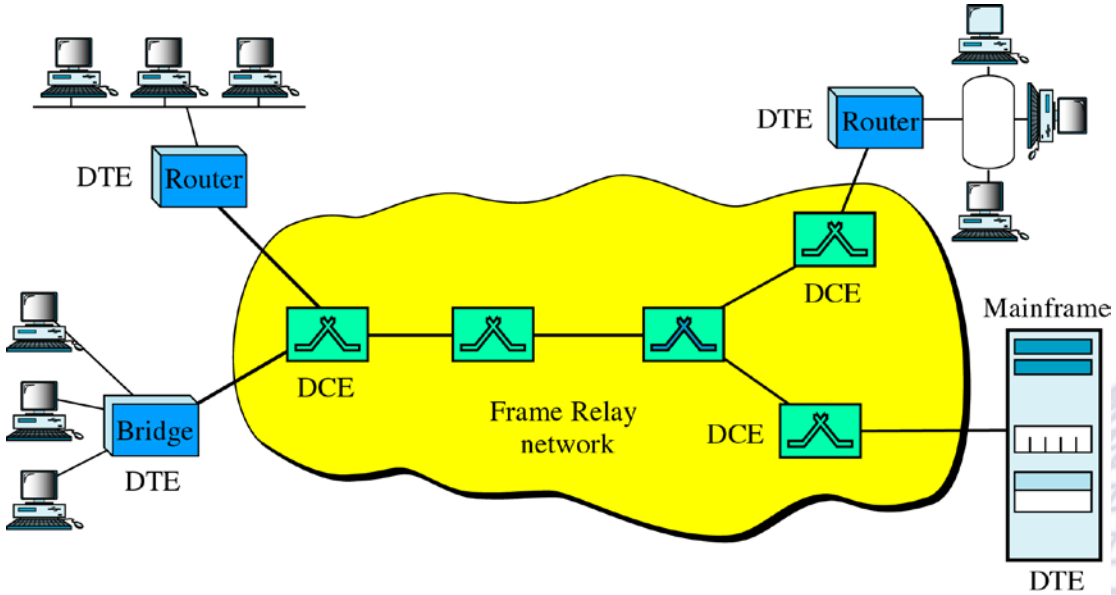


a. Using T-lines

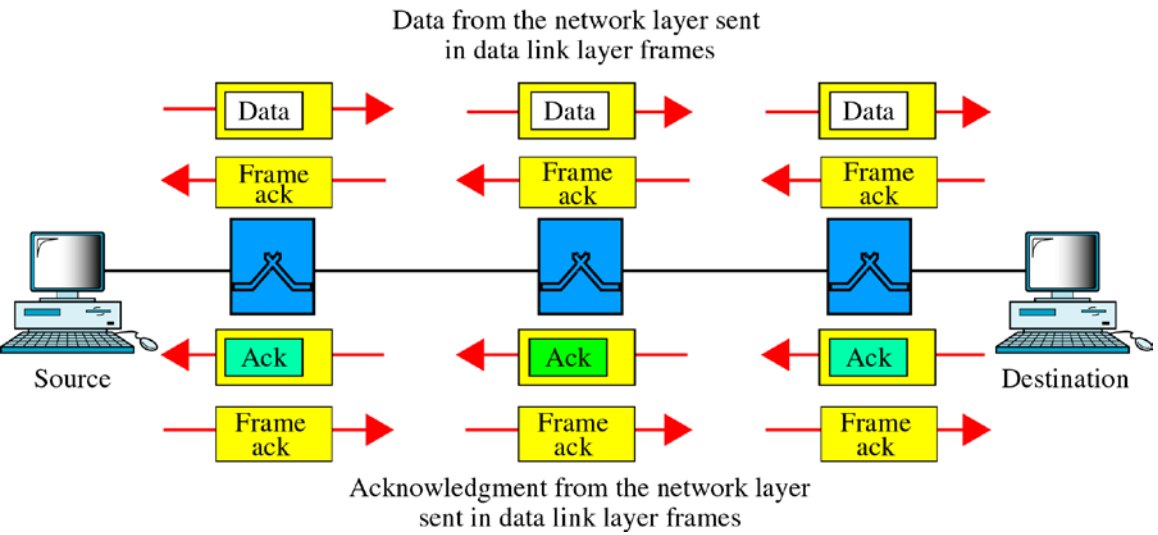
b. Using Frame Relay

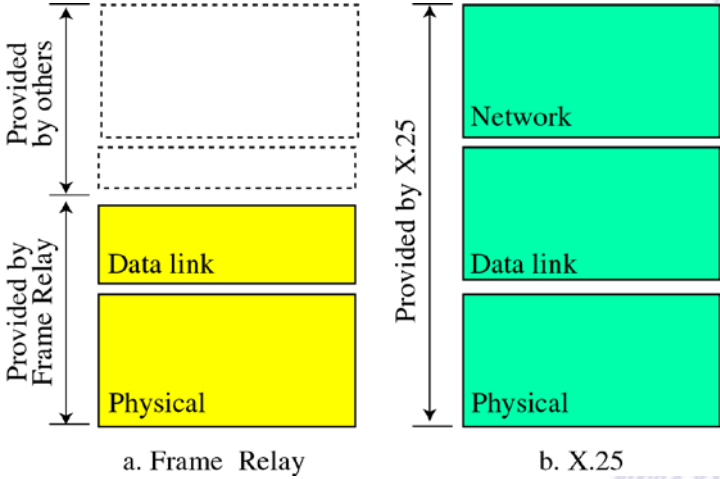
Frame Relay

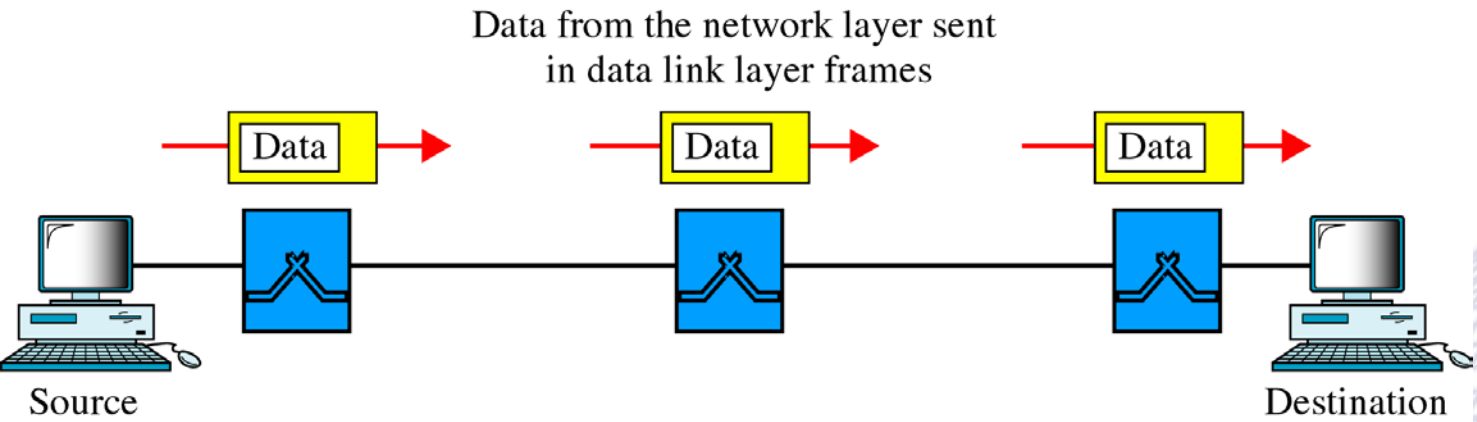
- Velocità da 1.544 Mbps fino a 44.376 Mbps in realtà possono anche essere minori
  - Link GARR da 256 kbps a 2 Mbps spesso sono su Frame Relay
- Permette di inviare dati con “banda a richiesta”
  - L’utente può richiedere banda più alta secondo il bisogno
  - Comunque si possono specificare banda minima garantita oltre che banda massima



Più snello di X.25

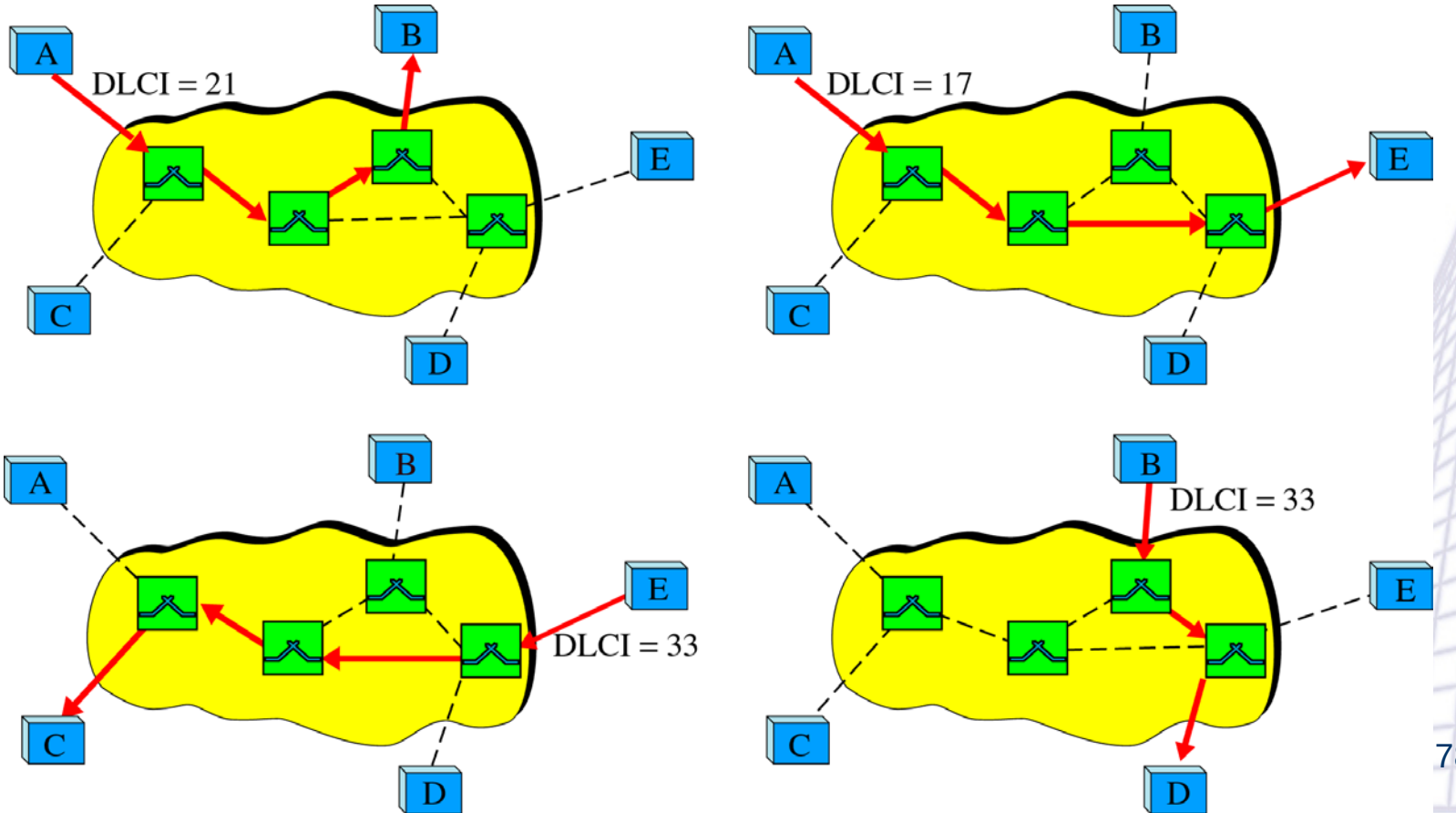






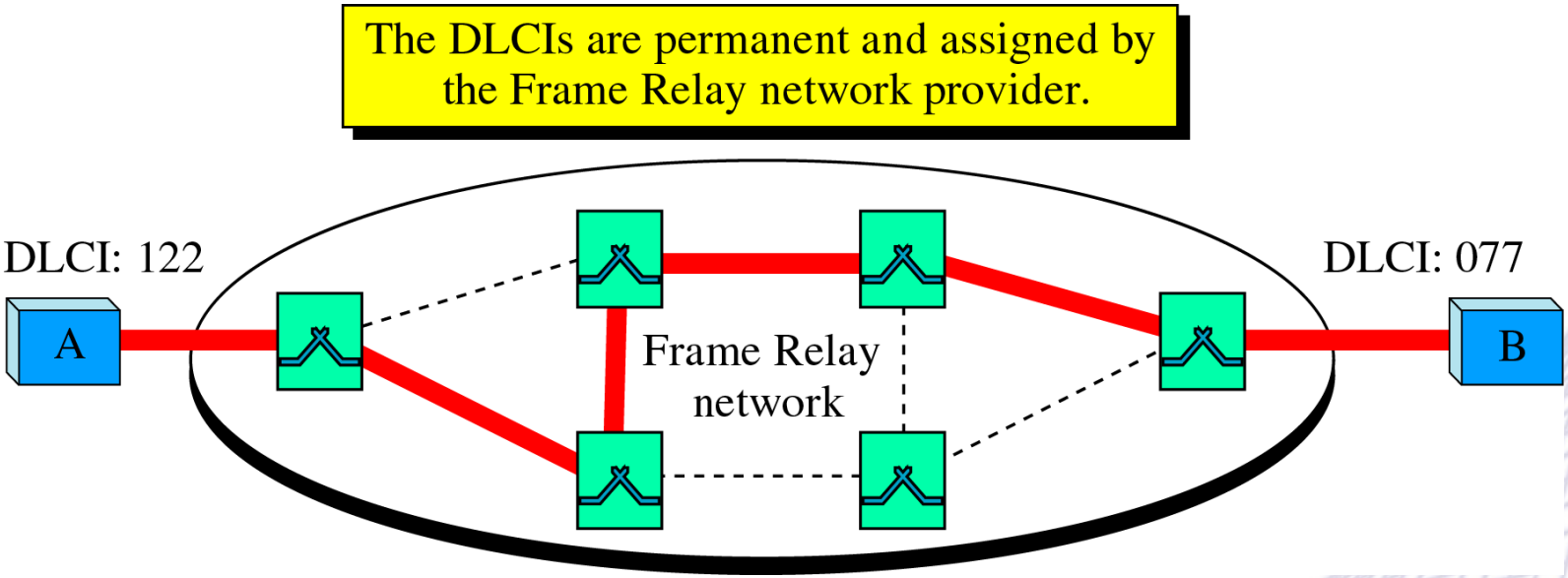
VCI Frame Relay: DLCI

- Frame Relay prevede l'uso di circuiti virtuali individuati da un numero DLCI (Data Link Connection Identifier)

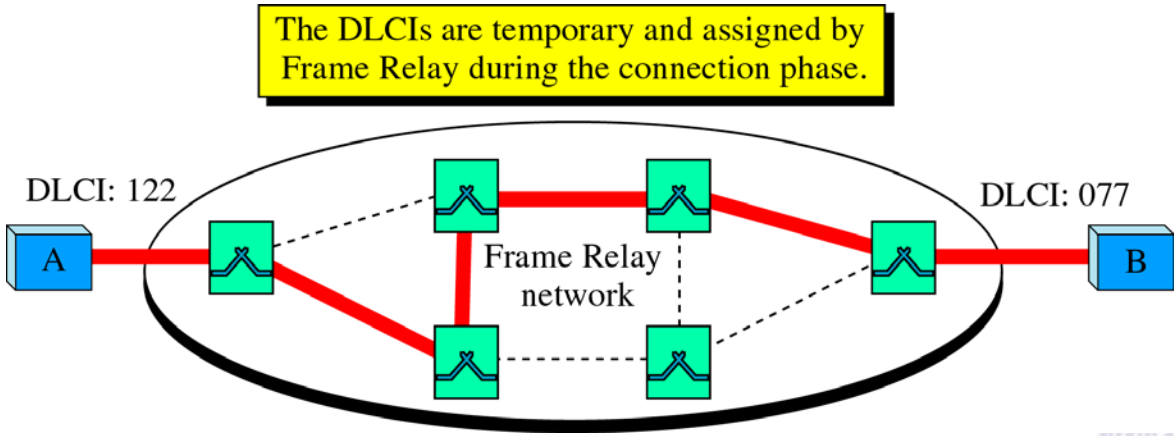


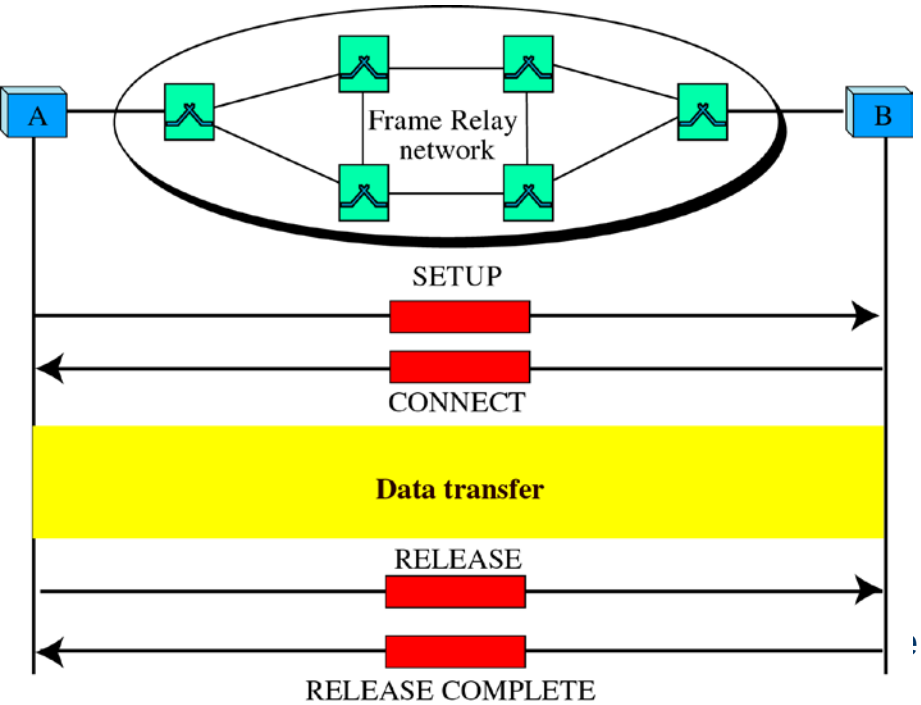
Permanent VC

- PVC: Circuiti Permanenti: L'amministratore definisce in tutti i nodi le informazioni per instradare i pacchetti. Molto comune ma spreca il link quando non c'è traffico



Switched VC



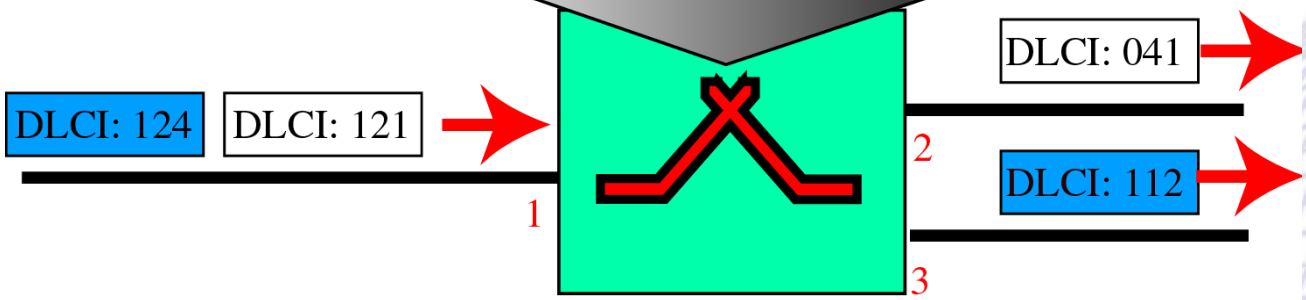


- SVC: Il circuito virtuale viene creato solo al momento del bisogno come in ISDN

Switch

- Ogni switch ha una tabella molto semplice
  - Mette in corrispondenza una coppia <porta di ingresso, DLCI> con una coppia <porta di uscita, DLCI>
  - Il DLCI è lungo 10 bit, ma è possibile avere indirizzi estesi di 16 o 23 bit

Incoming		Outgoing	
Interface	DLCI	Interface	DLCI
1	121	2	041
1	124	3	112
2	167	3	367
3	167	1	192

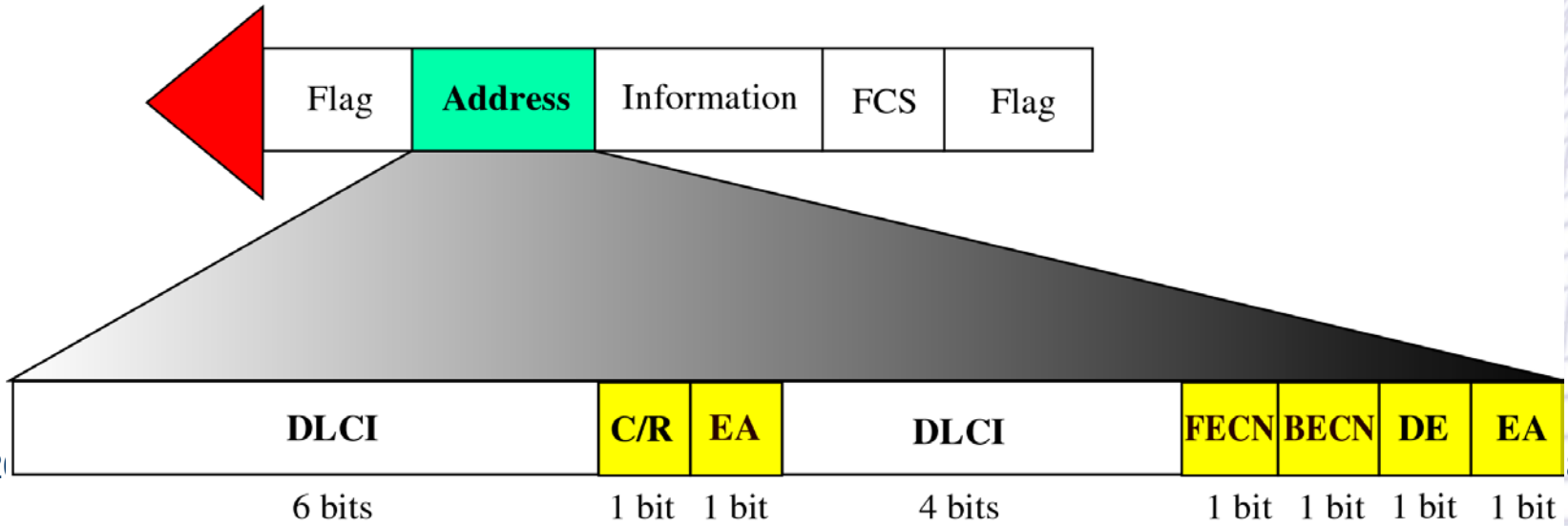


Formato di frame

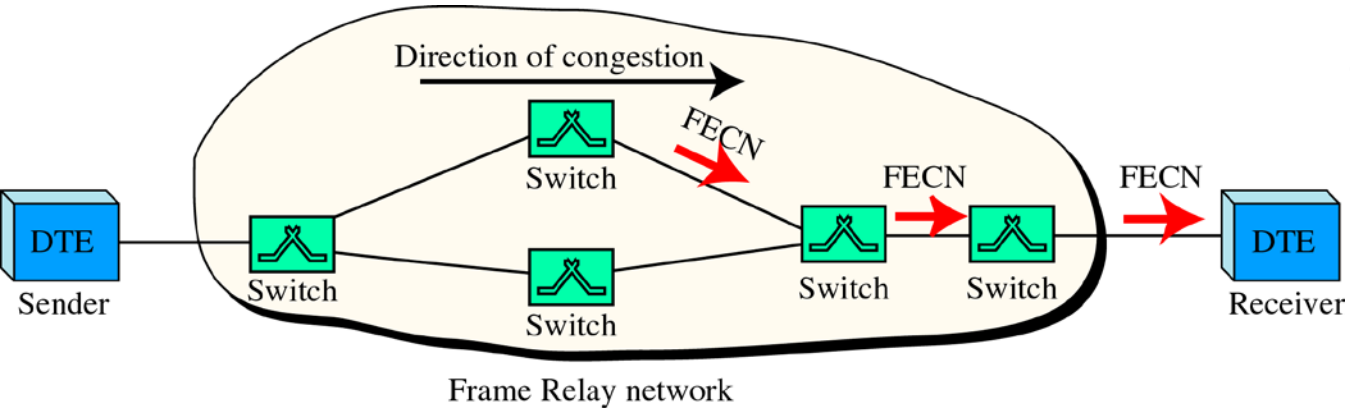
- EA per specificare extended address viene messo a 1
- FECN, BECN: Forward e Backward Explicit Congestion Notification. Per segnalare congestione
- DE: Discard Eligibility. I frame con questo bit a 1 sono eliminati prima di quelli con bit a 0

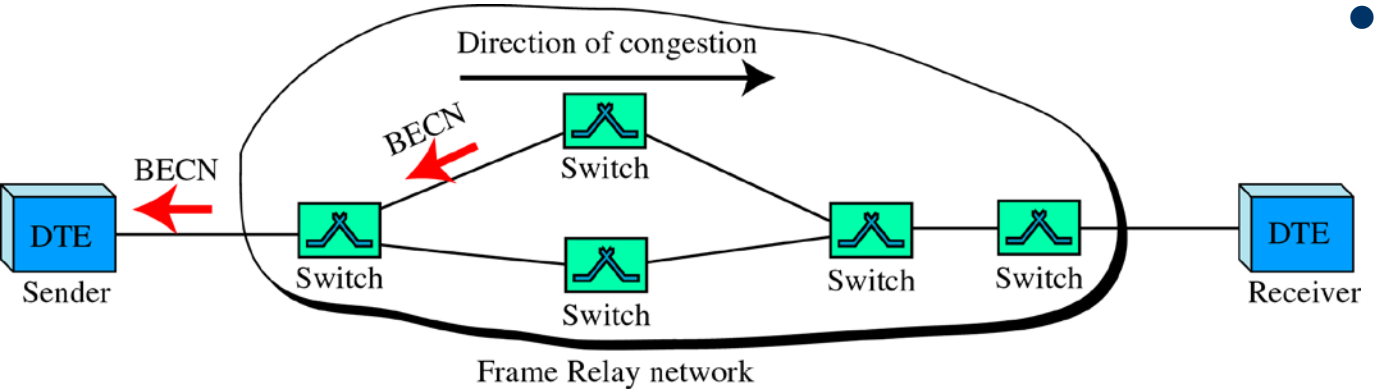
C/R: Command/response
EA: Extended address
FECN: Forward explicit congestion notification

BECN: Backward explicit congestion notification
DE: Discard eligibility
DLCI: Data link connection identifier



Gestione Congestione

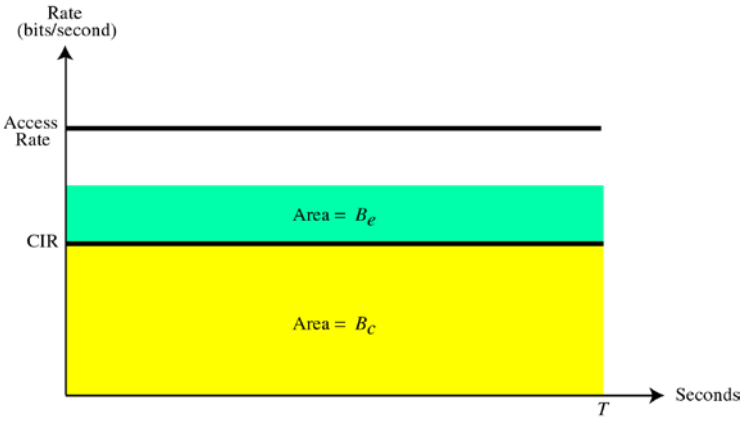




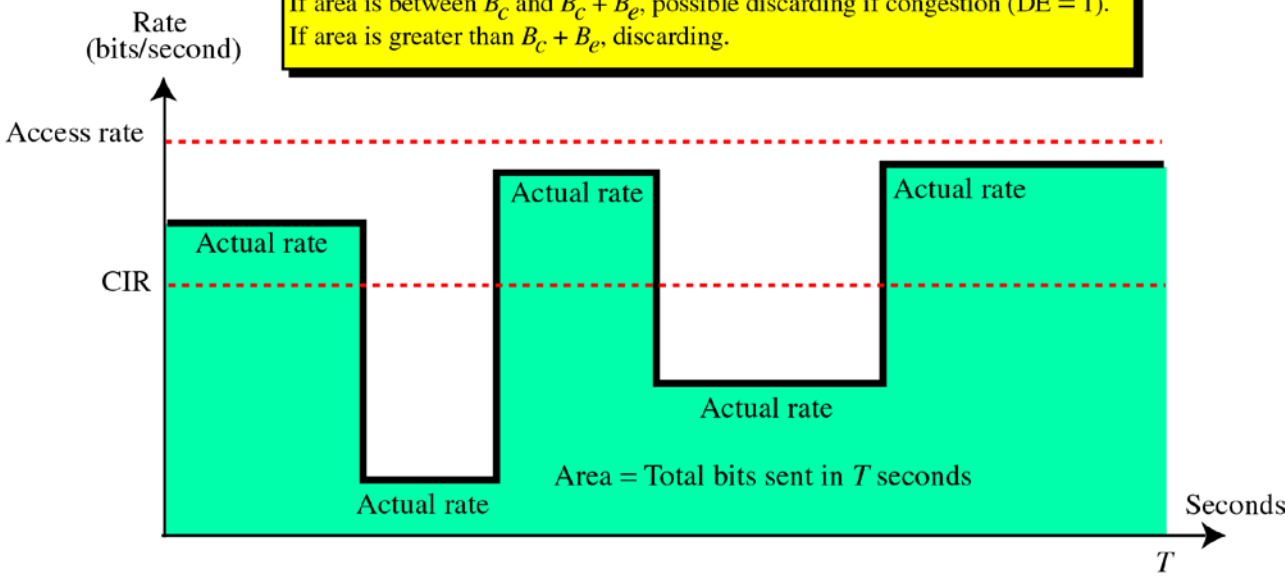
- FECN: Segnalo alla destinazione possibilità di congestione che si deve aspettare ritardi o perdita di frame
- BECN: Segnalo alla sorgente la possibilità di congestione affinchè spedisca meno velocemente

Uso di DE per CIR

- L'uso del bit BE permette di implementare il **Committed Rate**, il max data rate garantito
- L'utente ha la possibilità di usare tutta la banda (access rate) ma i frame sopra il CIR vengono taggati con DE=1



If area is less than B_c, no discarding (DE = 0).
If area is between B_c and $B_c + B_e$, possible discarding if congestion (DE = 1).
If area is greater than $B_c + B_e$, discarding.



CIR

- Commited Information Rate
  - Se CIR e Access Rate sono uquali allora Frame Relay equivale ad usare una linea dedicata
  - Se invece per esempio Access Rate vale 1 Mbps possiamo avere CIR a 256 kbps. In questo caso possiamo avere una subscription 4:1
  - Se il canale è poco usato si riesce ad arrivare a 1 Mbps. Se il canale è molto trafficato tutti riescono ad avere almeno 256 kbps.

ATM

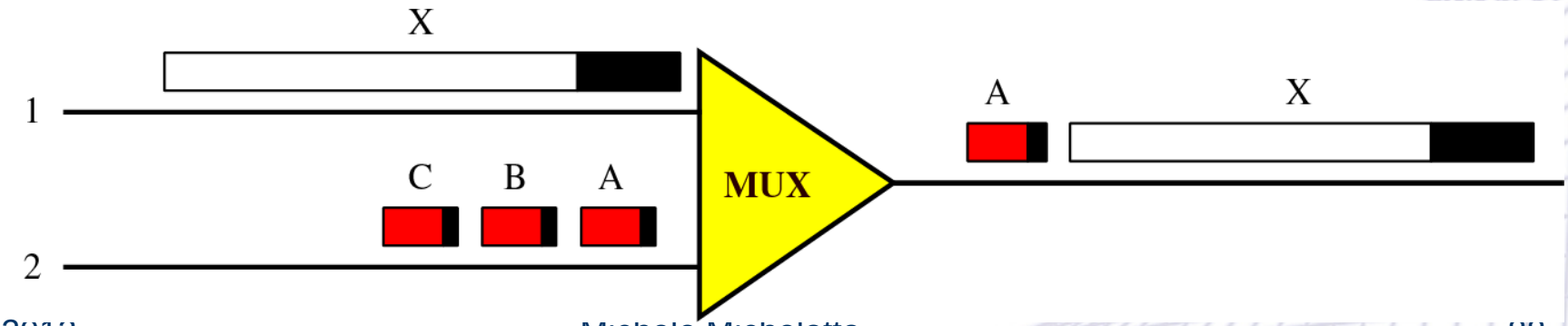
- Asyncronous Transfer Mode (le reti telefoniche sono solitamente sincrone)
- Progettato per portare tutto (dati, voce, tv, etc..) e spinto con grande dispendio di mezzi
  - Sopratutto su fibra ottica, molto costosa ma anche poco suscettibile a rumore ed errori
  - Spostare tutte le funzioni dal software verso l'hardware
- Usato molto meno, per gli stessi motivi di OSI (bad timing, technology, implementations, politics).
  - Molto usato a basso livello nelle reti telefoniche, anche per portare pacchetti IP. Essendo usato per trasporti interni gli utenti non lo conoscono.
- Link GARR da 34 o 155 Mbps spesso sono su ATM

Circuiti Virtuali ATM

- Essendo ATM connection oriented c'è una fase di call setup
- Il primo pacchetto percorre la rete e i diversi router (switch ATM) prendono nota e riservano le risorse necessarie.
- Vengono chiamate "Circuiti Virtuali" visto che simulano i circuiti fisici dei telefoni
- Solitamente si fanno "Circuiti Virtuali Permanenti" PVC simili alle "leased lines" delle reti dati

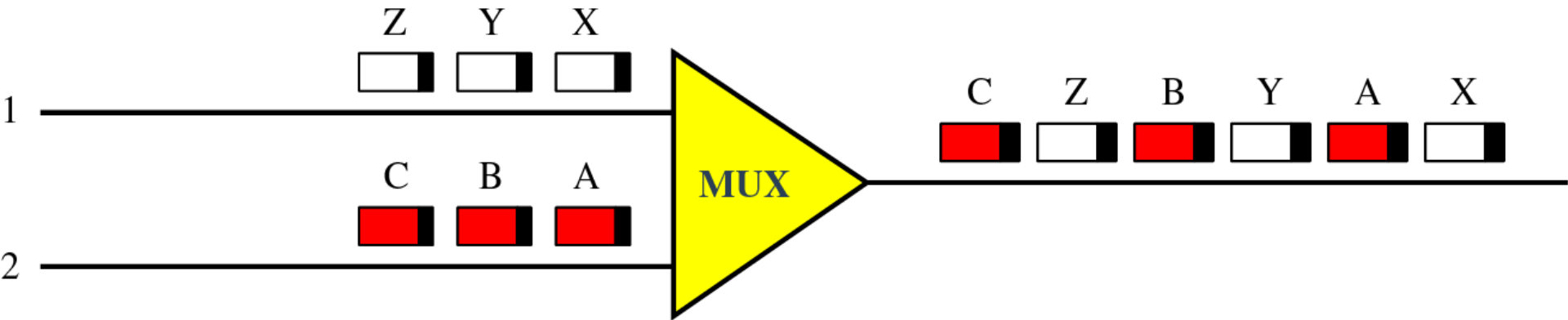
Reti Packet Switched

- Nelle reti Packet Switched ci sono diversi protocolli con frame di diversa dimensione e formato
- Quindi spesso ci sono header molto grandi e frame ancora più grandi di dimensione variabile
- Difficile consegnare a rate costante anche frame piccoli perchè se un frame piccolo (audio) arriva appena dopo un frame grande (bulk data) deve rimanere deve aspettare per essere inoltrato



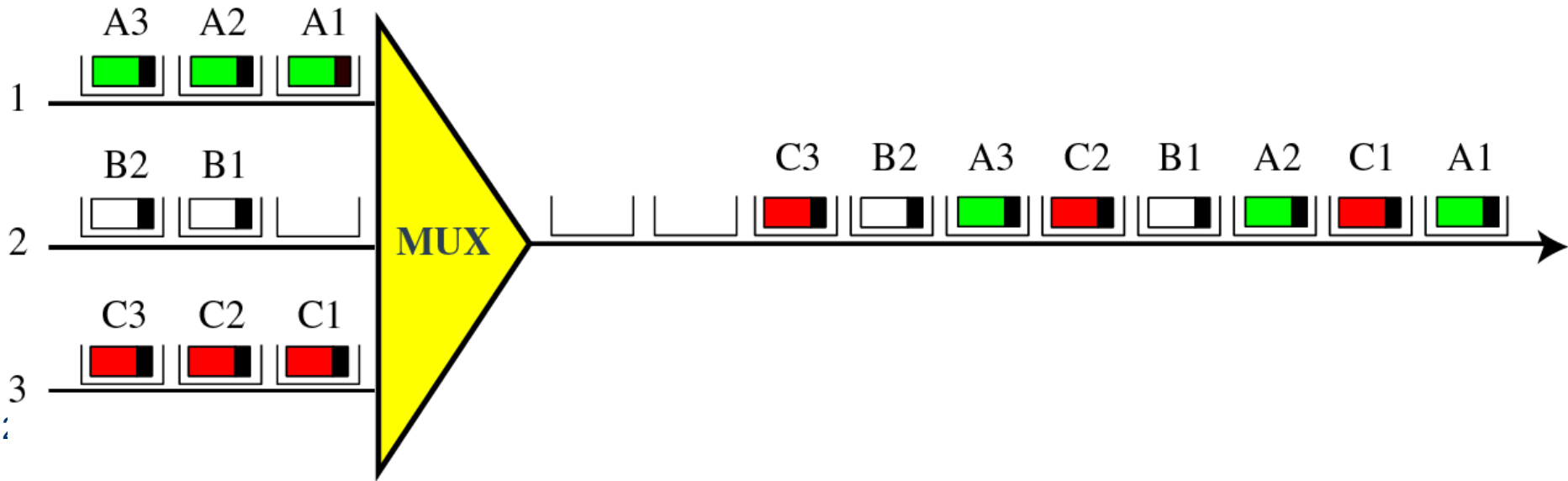
Reti a celle

- Frame molto piccoli di dimensione fissa, sono chiamati celle
- L’attesa massima viene minimizzata, quindi tutte le celle arrivano a destinazione a velocità costante



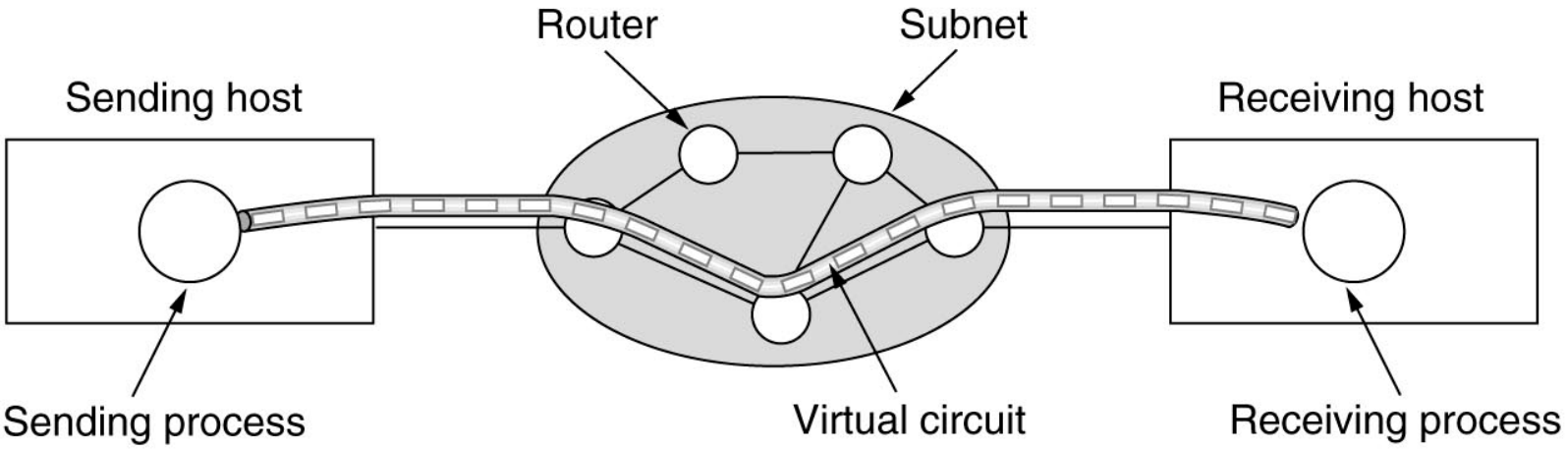
TDM Asincrono

- ATM usa un multiplexing a celle
- Un canale di output rimane inutilizzato solo se non nessuno dei canali di input ha qualcosa da spedire
- Dopo C1 arriva A2 perché non c'è niente su secondo canale in ingresso



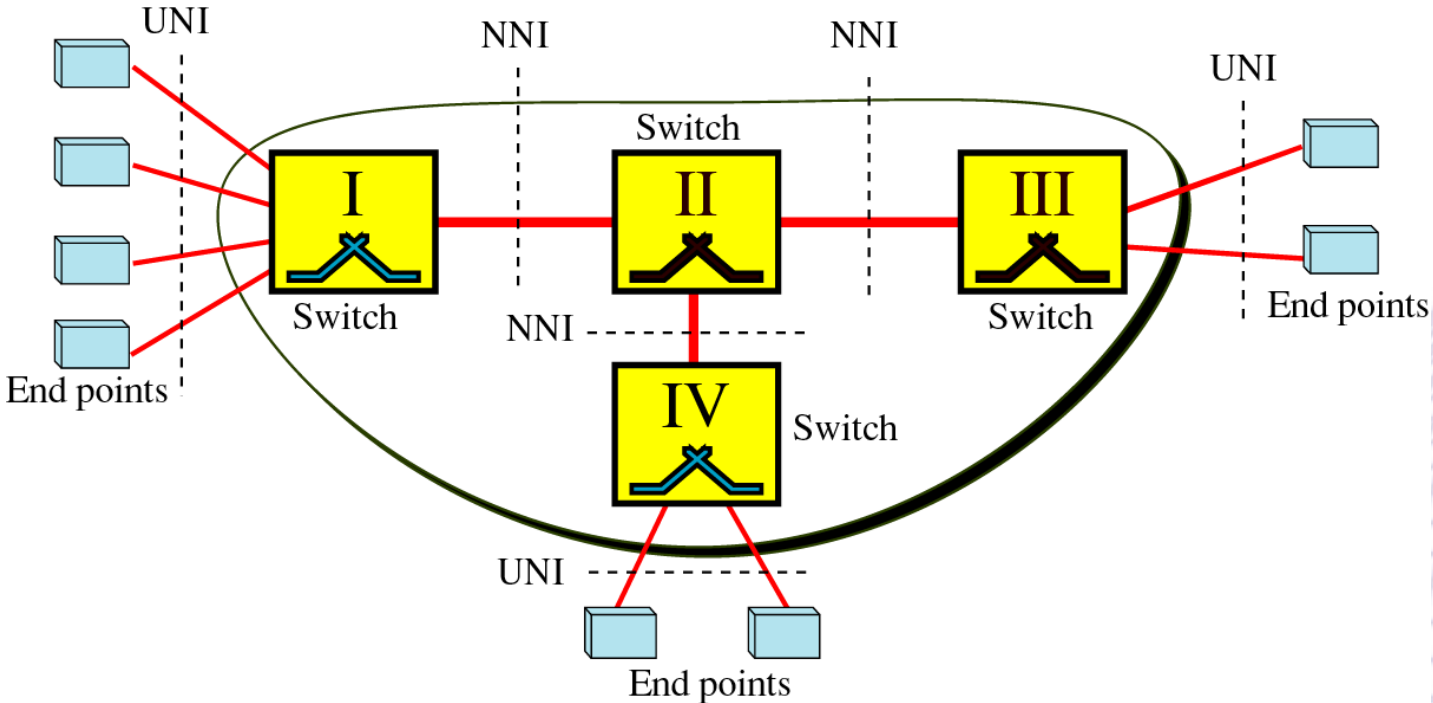
Circuiti Virtuali

- Ogni connessione ha un unico identificatore di connessione
- Stabilita la connessione entrambe le parti possono trasmettere dati



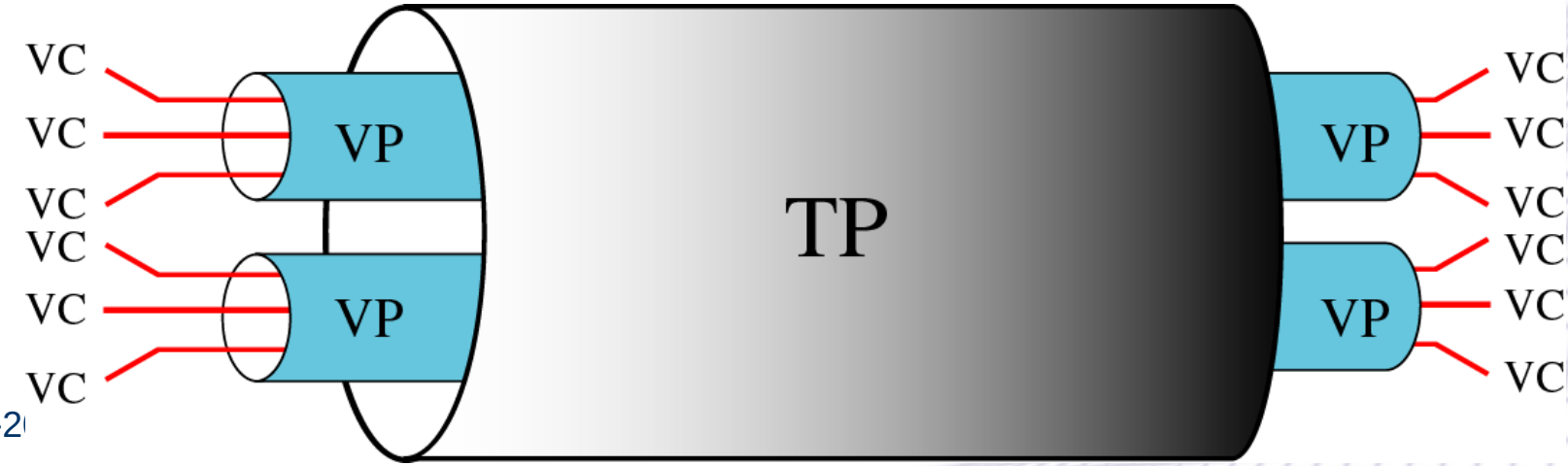
Architettura ATM

- Gli utenti accedono dagli End Points che sono connessi alla rete ATM con una UNI (User Network Interface)
- Gli switch sono collegati tra di loro da una NNI (Network Network Interface)

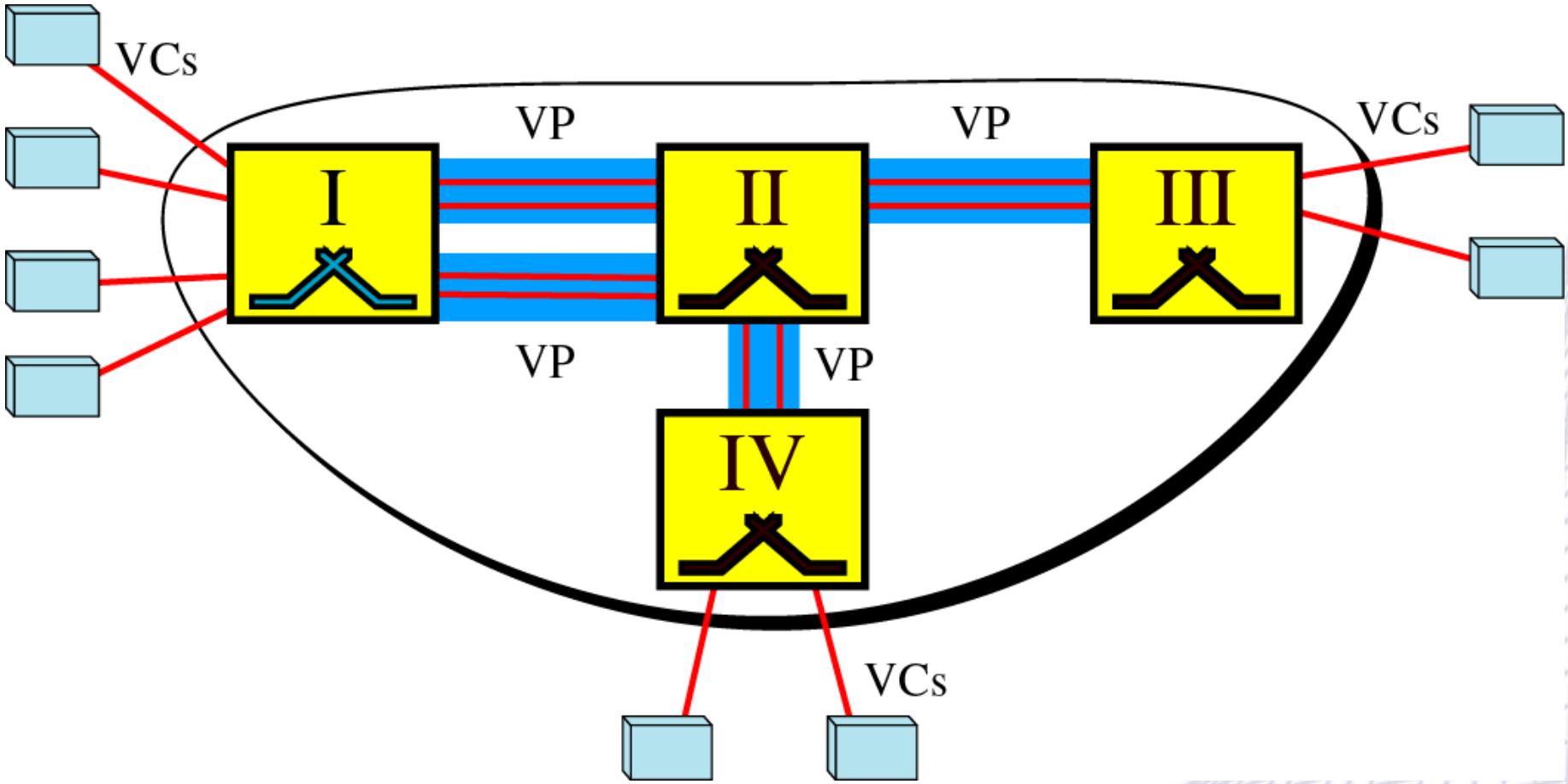


Connessioni virtuali

- La connessione avviene con
  - Canali di Trasmissione (TP: Transmission Path), in pratica la connessione fisica: cavo, satellite, fibra etc…
  - Canali Virtuali: (VP: Virtual Path) un TP viene diviso in diversi canali virtuali (come le strade che uniscono due città)
  - Circuiti Virtuali: (VC: Virtual Circuit) Una cella di una connessione segue lo stesso percorso all'interno di un VP (come una corsia in una strada)

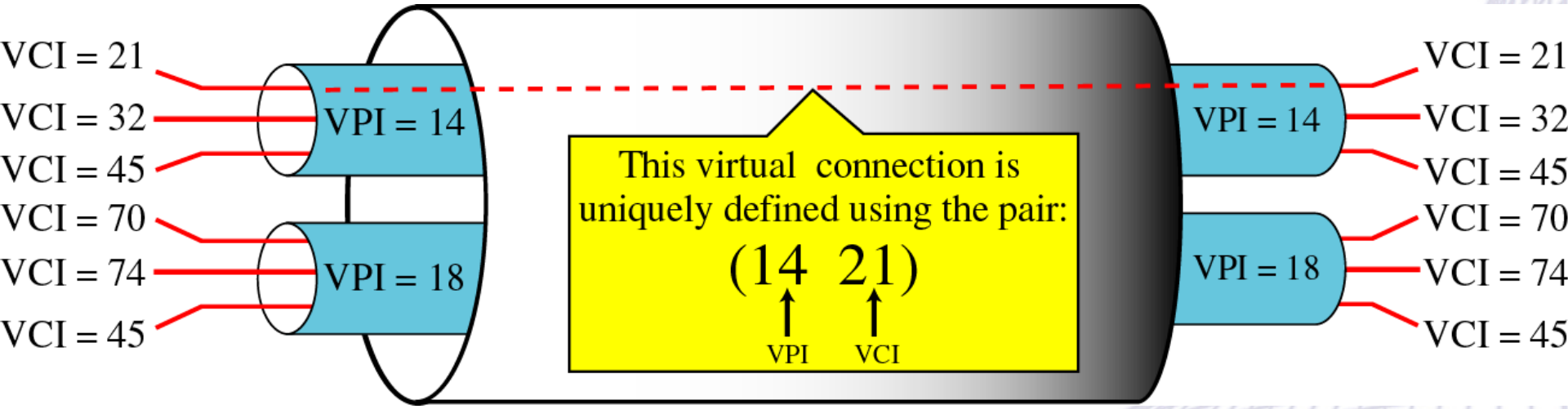


Relazione TP, VP, VC



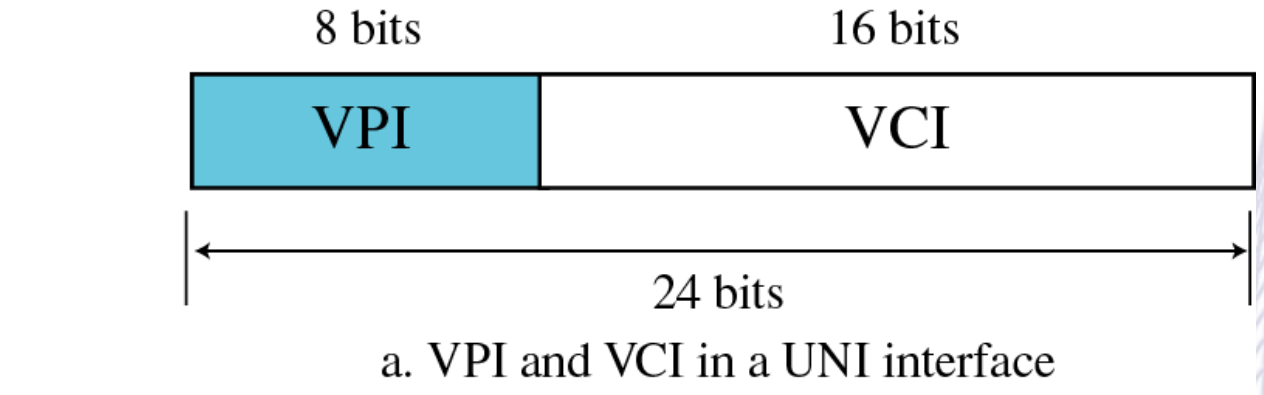
VPI e VCI

- Ogni connessione virtuale è identificata da un VP Identifier (VPI) e un VC Identifier (VCI)
- Serve per favorire un routing gerarchico, la maggior parte degli switch deciderà il routing guardando solo il VPI mentre solo gli switch di edge dovranno guardare anche il VCI

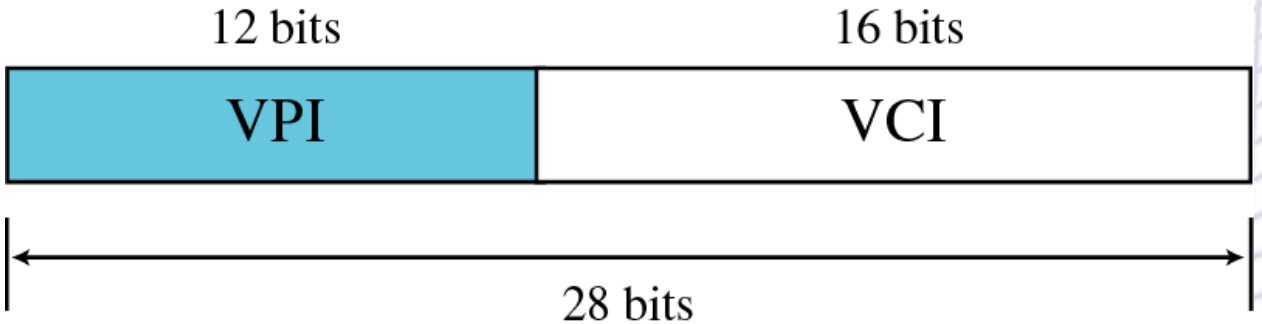


UNI vs NNI

- In una connessione UNI gli identificatori VPI e VCI hanno 8 e 16 bit
- In una connessione NNI invece hanno 12 e 16 bit



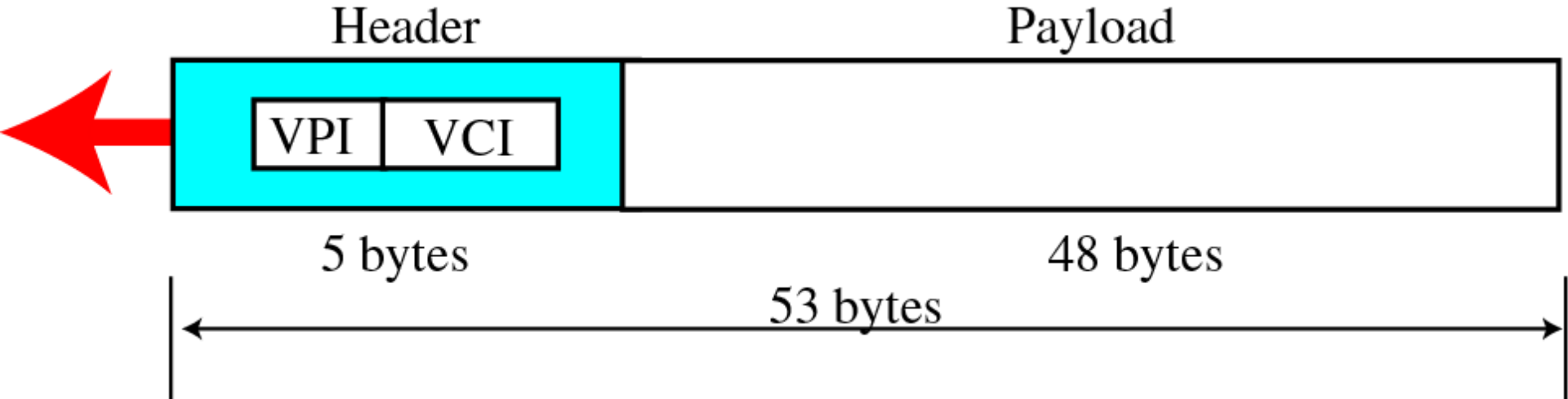
a. VPI and VCI in a UNI interface



b. VPI and VCI in an NNI interface

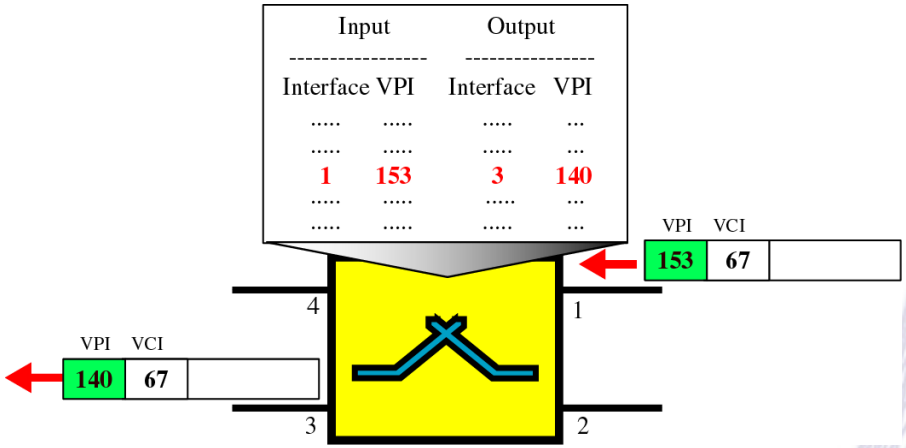
Celle

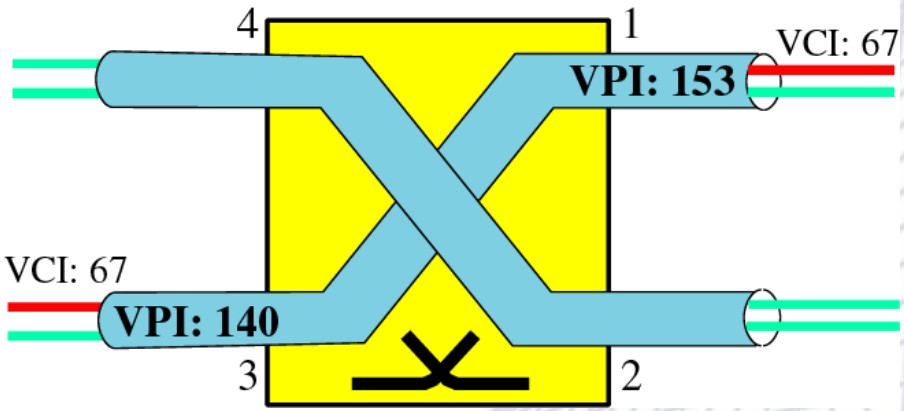
- Le celle hanno dimensione fissa di 53 bytes
  - Header di 5 byte che contiene il connection identifier
  - Payload di 48 byte (un compromesso tra 32 desiderato da alcune TCL e 64 desiderato da altre)



Routing

- Il routing è come quello visto per Frame Relay
- Tabella con le due coppie <porta, VPI>
- Concettualmente come la figura in basso
- Se deve usare anche la VCI saranno terne invece di coppie <porta, VPI, VCI>

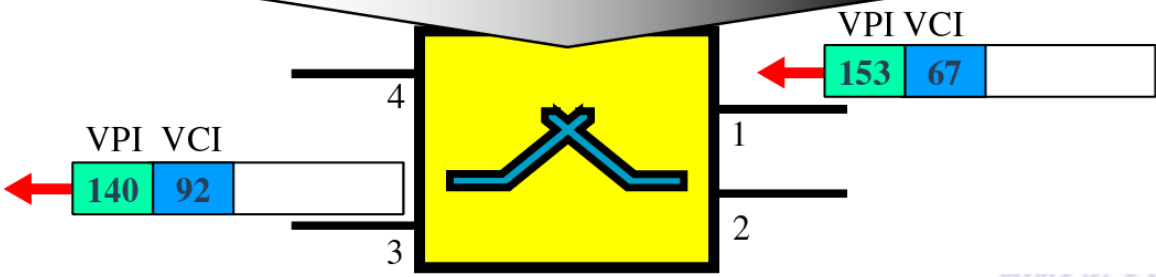


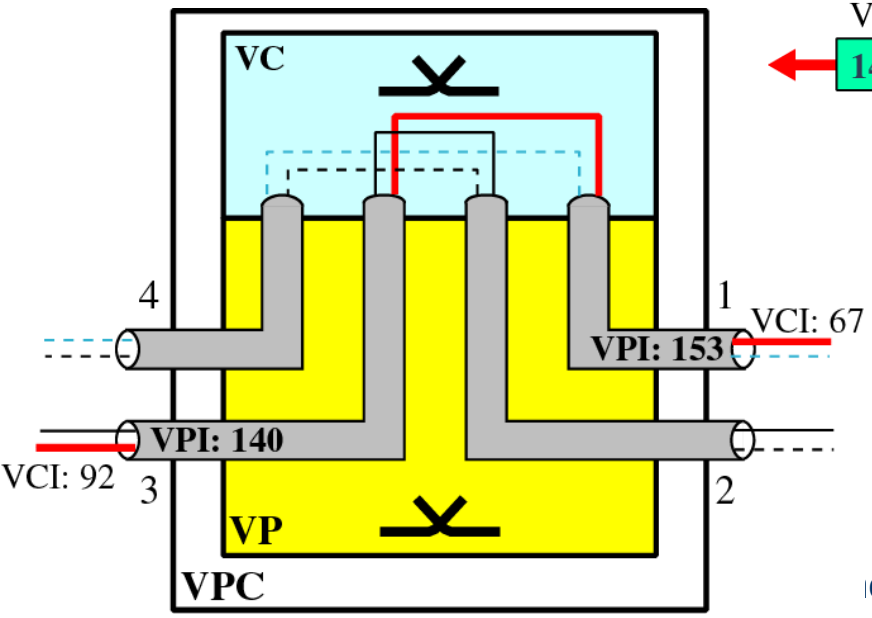


Routing

- Se deve usare anche la VCI saranno terne invece di coppie <porta, VPI, VCI>

Input			Output		
Interface	VPI	VCI	Interface	VPI	VCI
.....	.....	.....	.....	...	.....
.....	.....	.....	.....	...	.....
1	153	67	3	140	92
.....	.....	.....	.....	...	.....
.....	.....	.....	.....	...	.....





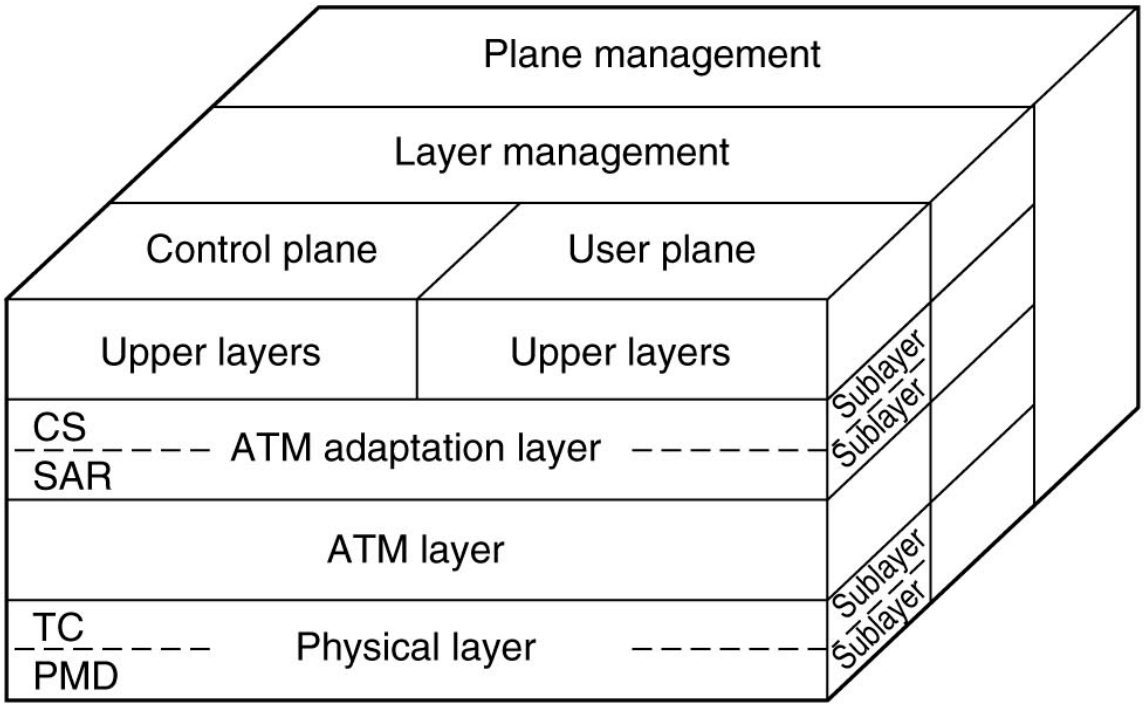
IP vs ATM

- Tutte le celle seguono lo stesso cammino
- Non viene garantita la delivery, ma se arrivano la cella 2 non supera la cella 1
- Una o entrambe possono non arrivare
- Questo viene gestito dai protocolli superiori
- Non è il massimo ma meglio di Internet che non garantisce l'ordine

ATM Reference Model

- Ha il suo modello, distinto da quello OSI o TCP/IP
- Il layer fisico si occupa di tensioni, bit timing etc..
  - Non ci sono regole ma solo si dice che le celle ATM possono essere mandate sul filo da sole o inserendole in un carrier → è indipendente dal sistema trasmissivo

ATM Reference Model



CS: Convergence sublayer
SAR: Segmentation and reassembly sublayer
TC: Transmission convergence sublayer
PMD: Physical medium dependent sublayer

Modello 3D

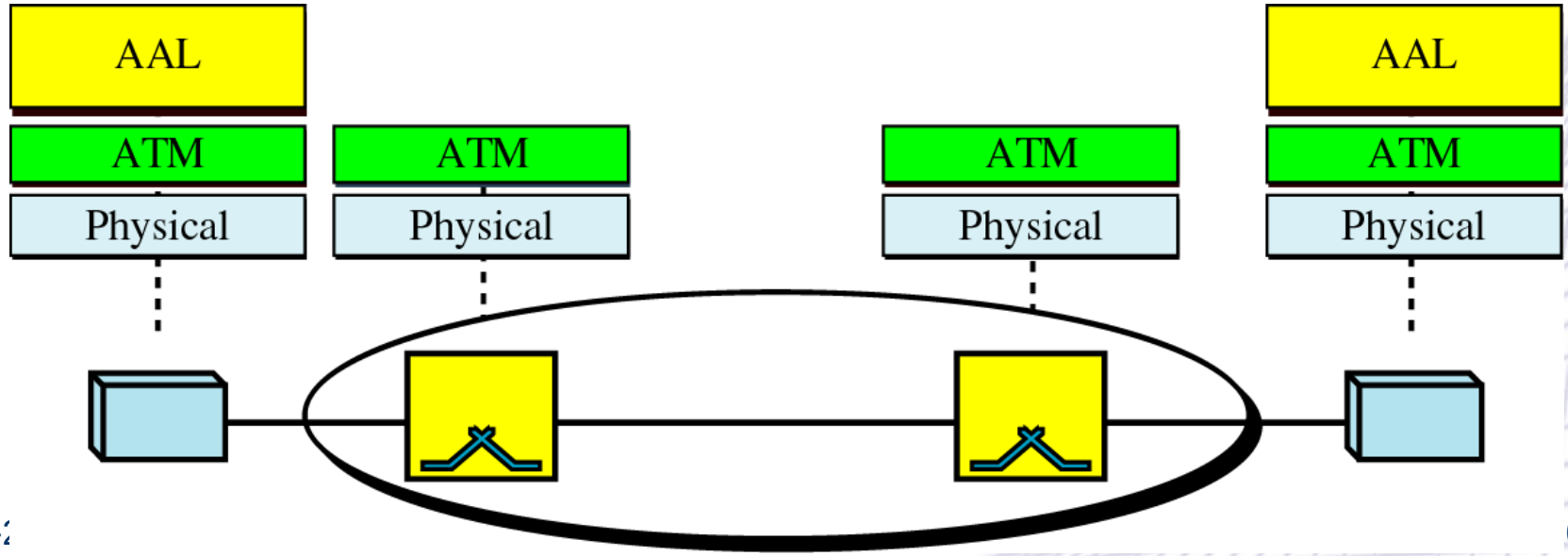
- È un modello 3D (non 2D come TCP/IP e OSI)
  - **User plane** gestisce trasporto dati, flow control, error correction
  - **Control plane** gestisce le connessioni
  - **Layer management** e **Plane management** sono legati alla gestione delle risorse e coordinamento tra i layer

Layer ATM e AAL

- Il layer ATM si occupa di celle e del loro trasporto
  - Definisce il layout della cella e il significato dei bit nell'header
  - Definisce anche call setup e release e gestisce la congestione
- Layer AAL per le applicazioni che non si vogliono occupare di celle
  - Gestisce inpacchettamento in celle e spachettamento

ATM Adaptation Layer

- Prende i dati che arrivano dal livello superiore (applicazione, IP o altro) e li segmenta in blocchi di 48 byte

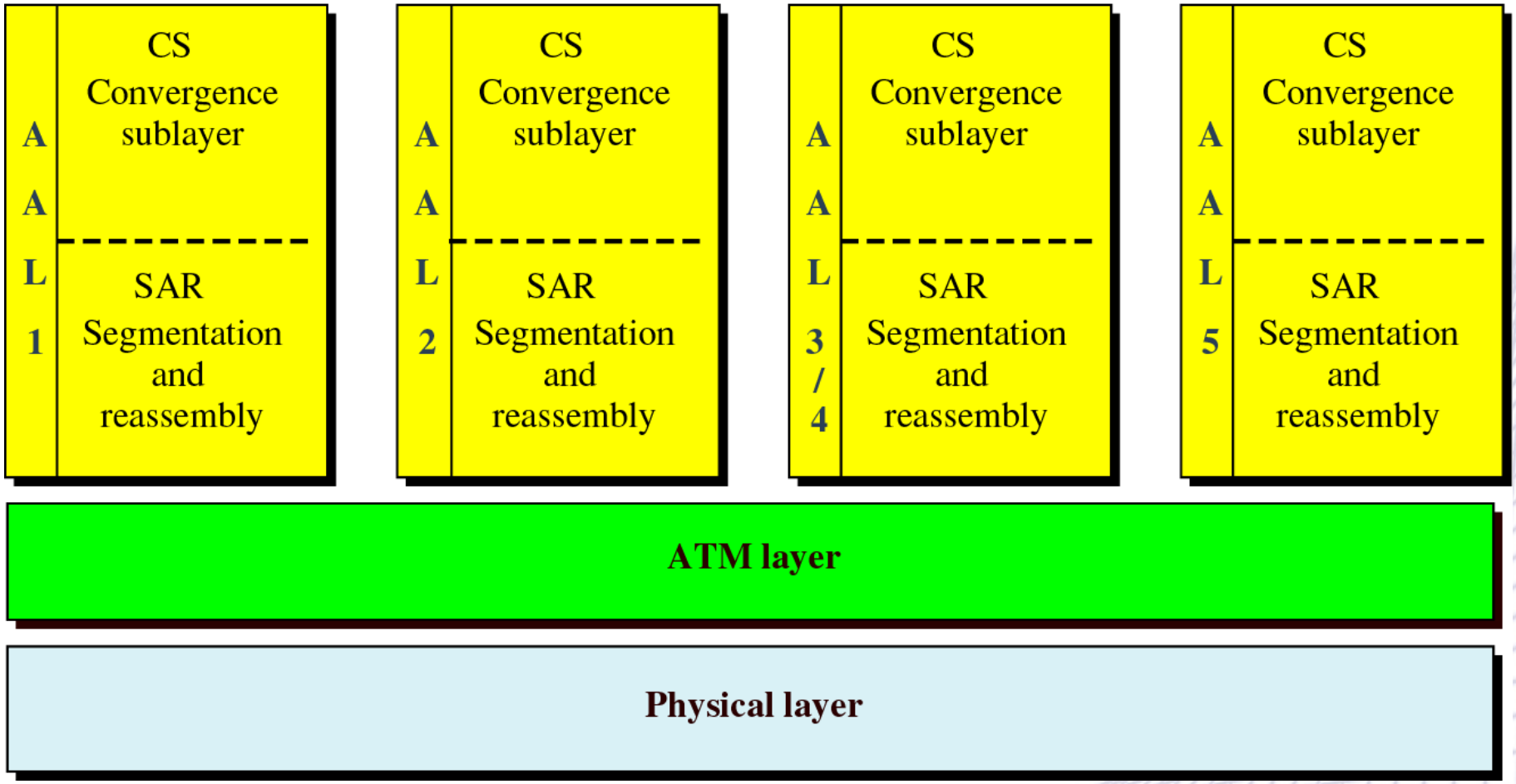


Layers e sottolayers ATM

OSI layer	ATM layer	ATM sublayer	Functionality
3/4	AAL	CS	Providing the standard interface (convergence)
		SAR	Segmentation and reassembly
2/3	ATM		Flow control Cell header generation/extraction Virtual circuit/path management Cell multiplexing/demultiplexing
2	Physical	TC	Cell rate decoupling Header checksum generation and verification Cell generation Packing/unpacking cells from the enclosing envelope Frame generation
1		PMD	Bit timing Physical network access

Ci sono diversi tipi di AAL che offrono diversi tipi di servizi a diverse applicazioni (file transfer, video on demand) con diversi requirements

Diversi tipi di AAL



Tipi di AAL

- AAL1 permette spedizioni di dati a velocità costante, ottimo per audio e video e per connettere ATM a reti telefoniche
- AAL2 progettato per flussi di dati a velocità variabile, è stato riprogettato per traffico a bassa velocità e frame piccoli, es telefonia mobile (permette mux di piccoli frame in una cella!)
- AAL3/4 progettati per supportare servizi orientati alle connessione e senza connessione, poi sono stati fusi
- AAL5 come AAL3/4 ma senza controllo di sequenza e controllo dell'errore, assumendo che questi controlli siano già svolti a livello superiore

Classi di servizio

- ATM può offrire diverse classi di servizio, permettendo un uso ottimale di un link fisico o logico
- CBR (Constant Bit Rate)
- VBR (Variable Bit Rate)
  - VBR Real Time
  - VBR Non Real Time
- ABR (Available Bit Rate)
- UBR (Unspecified Bit Rate)