Lezione 9: Comunicazione Seriale Asincrona

- **Comunicazione**: ad un certo istante, è possibile individuare
  - Un dispositivo che agisce da trasmettitore (Initiator o TX)
  - Un dispositivo che agisce da ricevitore (Target o RX)

Bus due dispositivi
c'è un RX e 1 TX

- **Seriale**: i bit che compongono il dato vengono trasmessi sul mezzo trasmissivo in sequenza
  - La trasmissione potrà essere anche wireless!
  - La trasmissione potrà usare 1 filo o più fili elettrici (modalità differenziale, cfr. Ethernet baseT)

Trasmetto seriamente un bit alla volta

- **Asincrona**: la trasmissione dei dati avviene seguendo un protocollo di comunicazione asincrono
  - Nessun segnale di clock è associato alla comunicazione

a distanza non posso permettermi un clock

no segnale di clock, serve un metodo per capire quando inizio e finisco la trasmissione

Caratteristiche principali di una ASC

Comunicaz. seriale asincrona

- La velocità della comunicazione è caratterizzata dal numero di bit trasmessi per secondo o bit-rate
  - Se T è il tempo per trasmettere un bit, 1/T è il bit-rate

frequenza con cui trasmetto bit, o baud rate

periodo

BIT RATE, frequenza di trasmiss.

FORMATO TRASMISSIONE:

- Il dato è costituito da una trama (frame) di bit
  - L'informazione è tipicamente racchiusa tra un header o preambolo e un trailer o coda
  - L'header serve a marcare l'inizio della trasmissione
  - Il trailer serve a marcare la fine della trasmissione e a trasmettere un codice di verifica (CRC o parità)

payload, bit che voglio effettivamente trasmettere

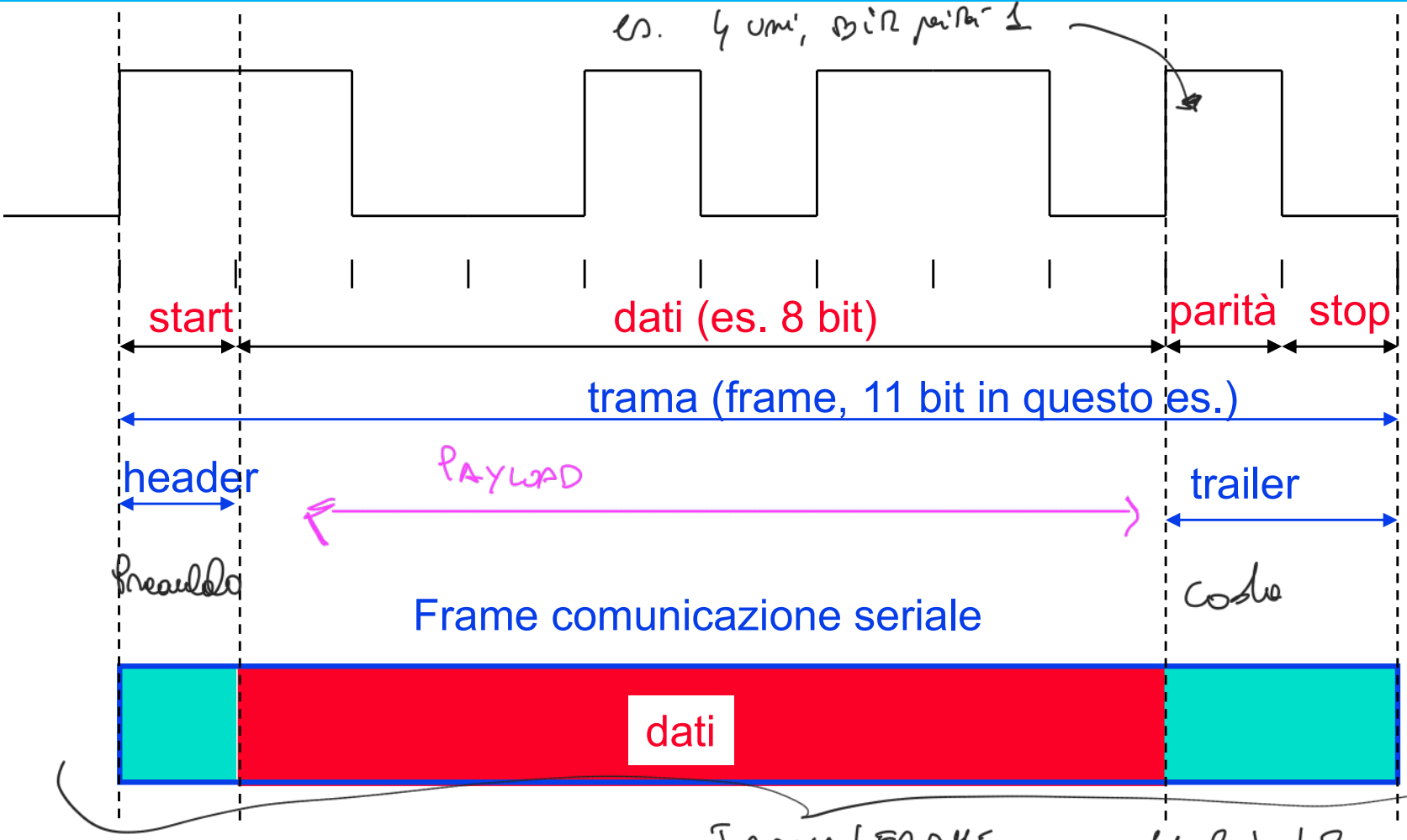
ulteriori info da trasmettere a priori

es. # di 1 nel payload è pari, => CRC = 1

- Ricevitore e trasmettitore si sincronizzano usando l'header
  - Il clock in ricezione cerca così di ottenere stessa fase del clock del trasmettitore: la frequenza deve però essere nota a priori!

Ricevitore deve campionare, appena riceve l'header, si sincronizza con quella freq. di clock e continua a campionare

Trama seriale



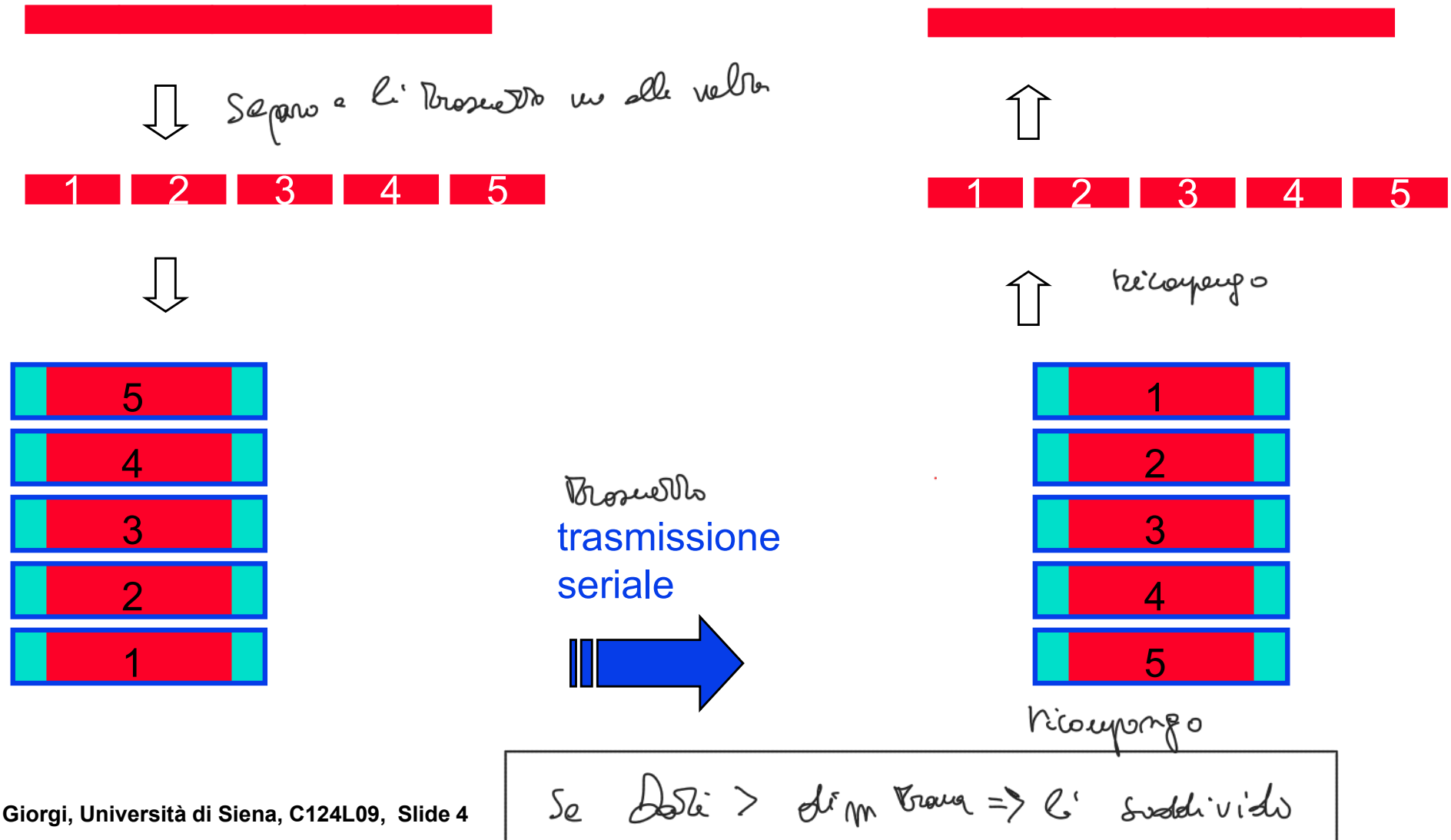
- **Informazioni che è necessario sapere a priori**
  - **Velocità di trasmissione (bit-rate, es. 1200 bit/s)**
  - **Formato della trama**
    (es. 1 bit di start, 8 bit dati, 1 bit di parità, parità positiva, 1 bit di stop)

Pacchettizzazione

iterando questo meccanismo posso trasmettere ogni numero di byte

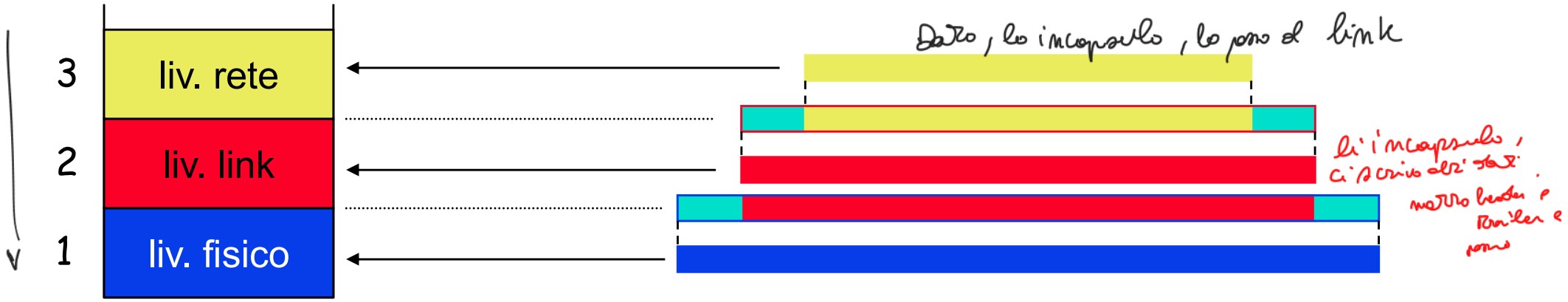
Es. Trasmissione di 5 byte

Se devo trasmettere 5 byte



Se dati > dim frame => li suddivido

Stratificazione (Stack di Comunicazione)

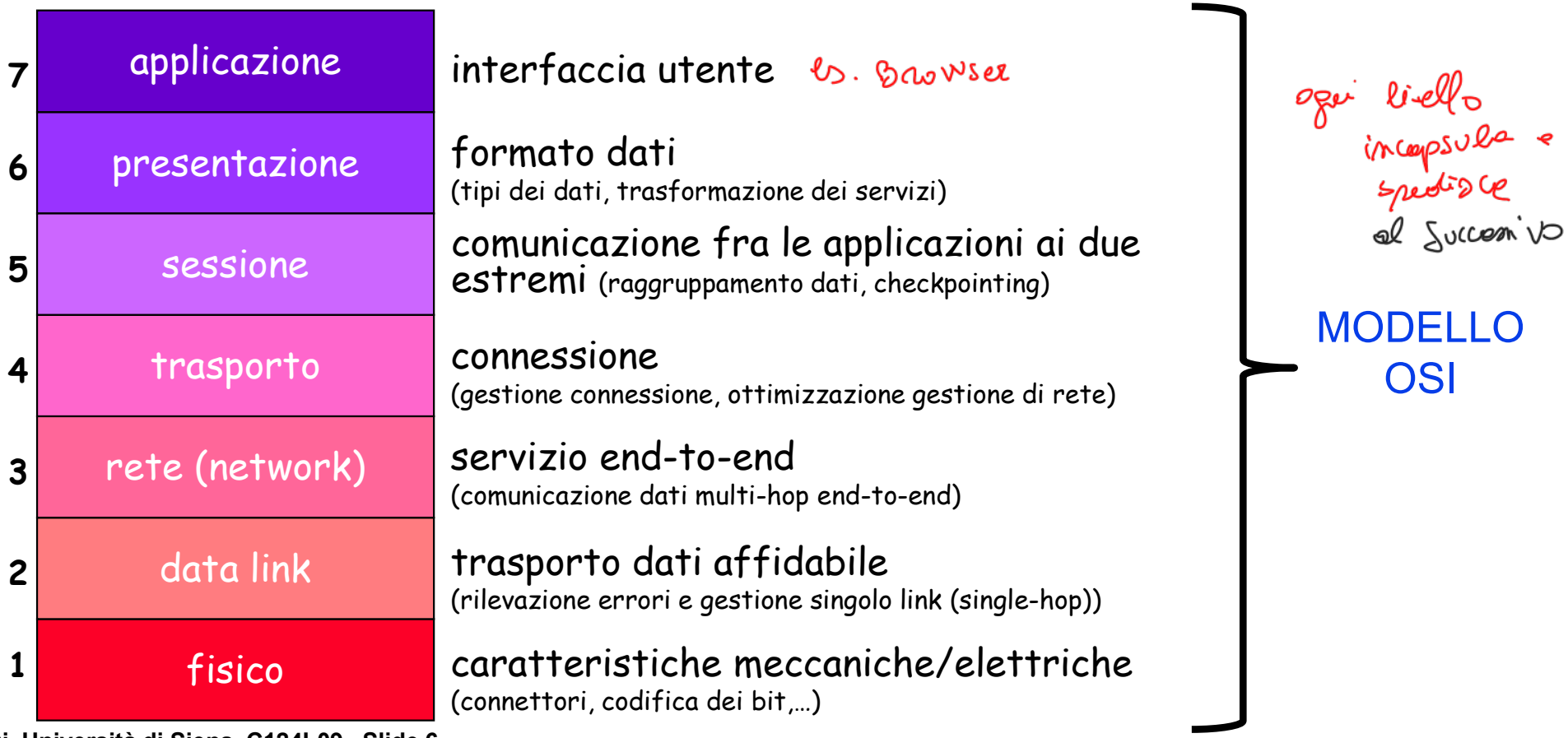


Stack di comunicazione:

- verso l'alto, ogni livello fornisce una comunicazione con più proprietà
- verso il basso i dati vengono 'arricchiti' con ulteriori informazioni di controllo

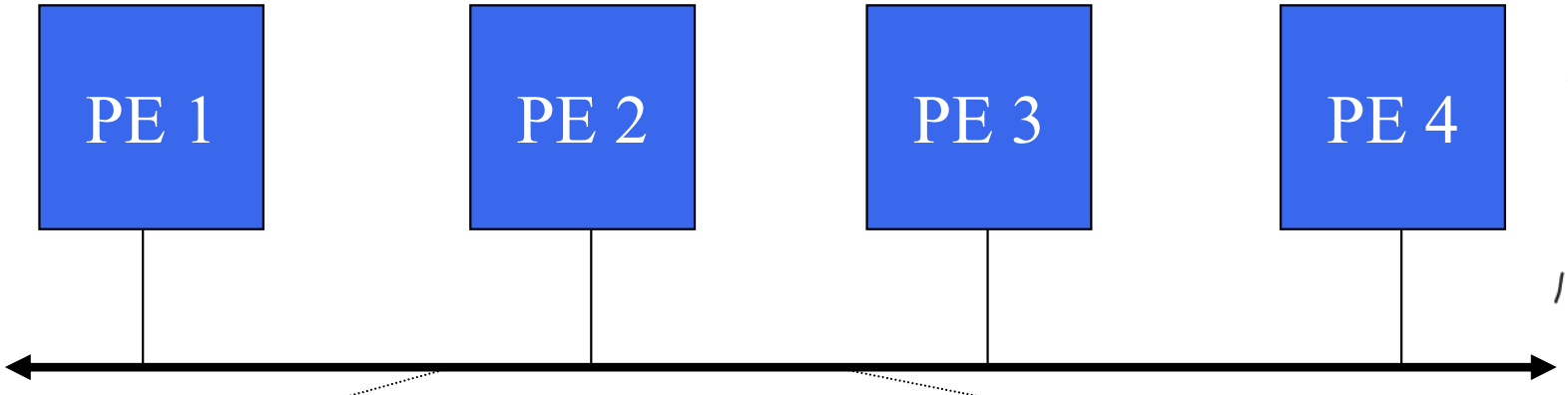
Gestione delle reti per astrazioni successive

- International Standards Organization (ISO) ha sviluppato un modello per descrivere le reti chiamato Open Systems Interconnection (OSI)
  - Modello a 7 livelli divenuto standard noto come ISO/OSI
- Fornisce uno standard per classificare i componenti delle reti e le operazioni coinvolte



Livelli 1 & 2: Reti Seriali a Bus

- **Una unica connessione fisica comune**



packet format

- **I2C, USB, FireWire, Ethernet, Wi-Fi…**

→ Standard Ethernet – Livelli 1 & 2

- Risale ai primi anni '80
- Standardizzato come IEEE 802.3
- Dominante nelle reti locali (LAN)
- Versioni:
  - 10 Mb/s (IEEE 802.3, o Ethernet)
  - 100 Mb/s (IEEE 802.3u, o Fast Ethernet)
  - 1 Gb/s (IEEE 802.3ab, o Gigabit Ethernet)
  - 10 Gb/s (IEEE 802.3ae, o 10-Gigabit Ethernet)
  - 100 Gb/s (IEEE 802.3ba, o 100-Gigabit Ethernet)

  - Wireless LAN(IEEE 802.11, o Wi-Fi)
  - Low-Power WLAN (IEEE 802.15, o Bluetooth)

Reti seriali a bus

questi oggetti trasmettono come

- Topologia: "a bus"

quando trasmetto, trasmetto a tutti; come faccio a dopo se posso stabilire una comunicazione tra due e solo lì qualcuno?

- Obiettivo: ottenere una comunicazione affidabile usando un mezzo trasmissivo eventualmente non affidabile

TECNICA PER STABILIRE COMUNICAZ PIÙ AFFIDABILE

CSMA/CD: Carrier Sense Multiple Access with Collision Detection

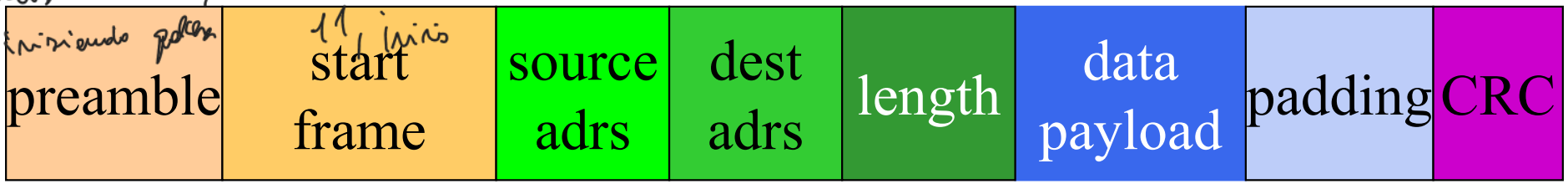
- Un nodo "ascolta prima di parlare" (carrier sense)
- Se il mezzo è libero, si attende un tempo "Defer Time"
- Se ci sono collisioni (multiple access) l'energia sul mezzo aumenta
- Un nodo cerca di rilevare collisioni durante la trasmissione (collision detect)
- Se un nodo rileva una collisione, continua a trasmettere 4-6 bytes per essere sicuro che la collisione venga rilevata dagli altri nodi e poi "lascia" il mezzo
- Se un nodo ha rilevato collisione, la trasmissione viene ripetuta dopo R*Δt per altri 15 tentativi
- Dopo 16 tentativi l'errore viene segnalato ai livelli superiori
- Δt è fisso mentre R viene calcolato secondo un "algoritmo di back-off"

Algoritmo di back-off per il calcolo di R

- Sia n l'n-esimo tentativo di trasmissione (n=1,2,…, 15)
- K= min(n,10)
- R è un numero casuale t.c. 0 <= R <= 2K

Risolve in maniera probabilisticamente accettabile i conflitti fino a 1024 nodi…

Formato del Pacchetto Ethernet



- **In dettaglio**
  - **Preamble** — 10101010101010… — 64 bit
  - **Start frame** — 11
  - **Source address** — 6 Bytes
  - **Destination address** — 6 Bytes
  - **Length** — 2 Bytes
  - **Data Payload** — i dati veri e propri!
  - **Padding** — Eventuali byte in coda al payload
  - **CRC** — 4 Bytes

8 Bytes

46-1500 Bytes

Ethernet FRAME

64-1518 Bytes

- **Segue un IPG (Inter Packet Gap) di almeno 96 bit (12 bytes)…**

Prestazioni della rete Ethernet = non ho la Ethernet

- La Qualità del servizio (Quality-of-Service, o QoS) tende a decrescere in maniera non-lineare ad alti livelli di carico dopo 15 tentativi, abbandono
  Ethernet non ha una qualità del servizio, non è garantita affidabilità
- Una conseguenza dell'algoritmo di back-off è che non è possibile stabilire il tempo massimo entro cui verrà trasmessa con successo una trama
  - Non si possono garantire deadline real-time!
  - Si possono ottenere livelli molto buoni di servizi purché si riesca a mantenere il carico ad un livello appropriato

Se voglio garanzie, non uso ethernet

Rete non deterministica
Se voglio più affidabilità
Cavi standard

NON HO GARANZIA SULLA QUALITÀ del servizio

ABBIAMO VISTO IL TIPO di comunicaz. seriale.... vediamo i dispositivi che la supportano

Porte Seriali

- **Metodo efficiente di comunicazione tra dispositivi**
  - **Tipicamente denominata UART (Universal Asynchronous Receiver and Transmitter)**
  - **Es. 16550° (es. Personal Computer), 16450, 8250, 8251A, MC68681**
- **Semplice porta seriale costituita da:**
  - **2 registri a scorrimento (shift register)**
    - **Uno usato dal lato trasmettitore (Tx)**
    - **L'altro usato dal lato ricevente (Rx)**
  - **L'uscita del Tx di un dispositivo è collegata all'ingresso Rx dell'altro dispositivo**
  - **Trasmissione e ricezione guidate dallo stesso clock (generato localmente, non trasmesso!)**
  - **Quando Tx è vuoto o Rx è pieno (ricevuto 1 byte) viene generato un interrupt**
- **Nel caso di trasmissione seriale asincrona full-duplex**
  - **trasmissione contemporanea in entrambi i versi: i due dispositivi fungono sia da Tx che da Rx**
  - **Sono sufficienti 3 fili di collegamento (Tx, Rx, GND)**
- **Trasmissione seriale asincrona half-duplex**
  - **La trasmissione, in un dato istante, avviene in una sola direzione: uno dei due dispositivi fa da Tx l'altro da Rx (successivamente i ruoli si possono scambiare)**
  - **Sono sufficienti 2 fili (Tx, GND)**

Uso di un buffer FIFO

First in First out

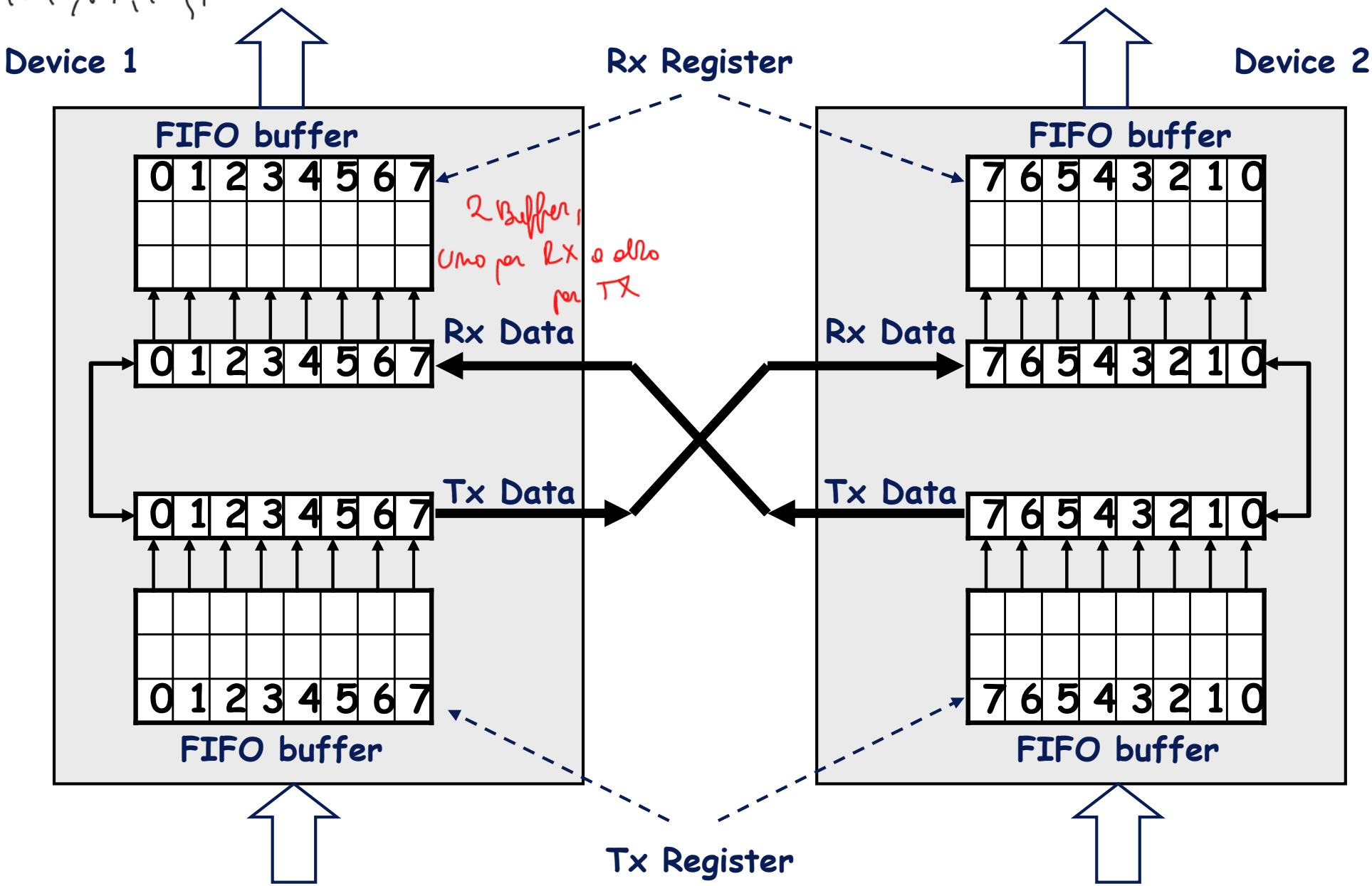
- **Utilizzo di un buffer FIFO per evitare la perdita di dati**
  - **in Rx:**
    dato ricevuto prima che sia stato letto quello precedente
  - **in Tx:**
    dato inserito prima che sia stato trasmesso il precedente

- **Dimensione del buffer FIFO:**
  - **Aumenta il throughput della porta**
  - **Riduce l'overhead della CPU (attesa della Tx o Rx)**

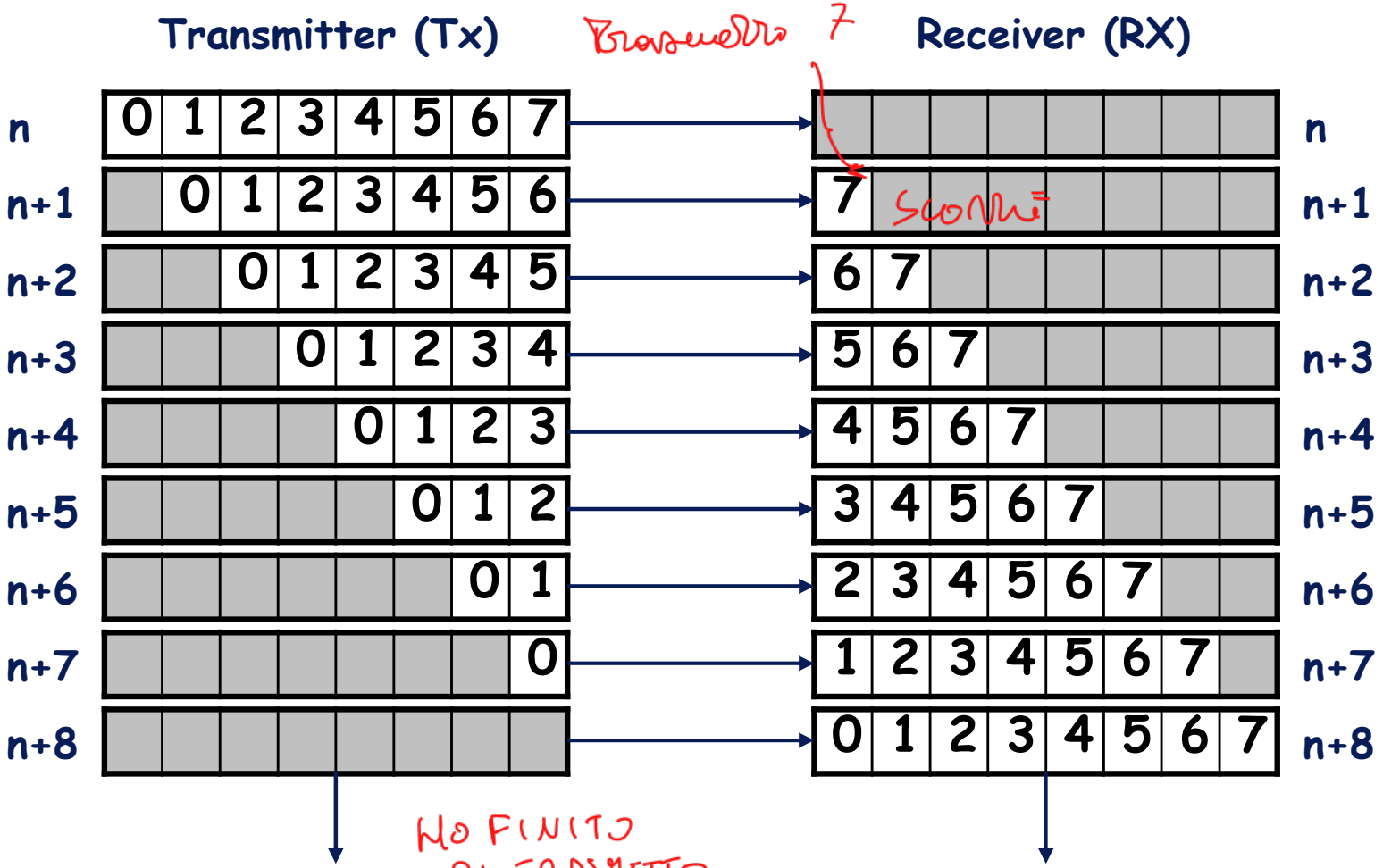
se buffer si riempie devo comunicarlo

coda, anche solo 16 byte, basta per bufferizzare ciò che sta arrivando, così la CPU può leggere byte, scriverla in memoria ecc senza bloccare la comunicazione. Comunque se il BUFFER dovesse riempirsi, devo comunicare che non posso più ricevere e dunque interrompere la ricezione, altrimenti perdo

Interfaccia Seriale Asincrona con Buffer FIFO



Es: trasmissione di un byte



Interrupt: Tx empty

Interrupt: Rx full

Esempio di trasmissione su porta seriale

- **Lato Tx:**
  - Appena il dato è nel Tx-Register, viene trasmesso
  - Viene generato un interrupt per segnalare che il byte è stato trasmesso
- **Lato Rx:**
  - Il buffer riceve bit dopo bit il dato
  - Appena viene ricevuto un intero byte, viene trasferito nell'Rx-register
  - Viene generato un interrupt ed il byte viene letto
- **In entrambi i casi**
  - Viene messo a 1 un bit nel registro di stato (a fine trasmissione o ricezione)
  - Per poter proseguire, tale bit deve essere azzerato
    - Es. dalla routine di servizio

Controllo del flusso

Se BUFFER FIFO pieno, devo segnalarlo in 2 modi

- **Necessario per impedire che Tx invii più dati di quelli che Rx può leggere**

Tecnica HW e SW

- **Due metodi** Se Buffer pieno, devo avvertire Trasmettitore
  - **Hardware:**
    - il chip UART del Rx individua un potenziale intasamento e attiva una linea apposita per avvertire il Tx
    - Tx interrompe la trasmissione
    - Appena Rx può ricevere dati, la linea viene disattivata

linea aggiuntiva, filo apposito

  - **Software:**
    - Il Rx invia caratteri speciali di controllo (XON e XOFF) al Tx
    - XOFF interrompe una trasmissione
    - XON riprende la trasmissione

INVIO BIT Speciali NO FILI AGGIUNTIVI

Ricevitore richiede che non venga più trasmesso

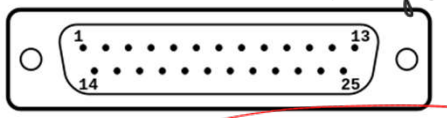
} byte con valore particolare XON, XOFF

no fili aggiuntivi, costringo a trasmettere qualcosa nella linea dati

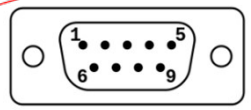
carattere speciale per codificare XON e XOFF

Standard EIA-RS232C – segnali e connettori

- **Segnali:**
  - In trasmissione, 1 logico=[-15V,-5V], 0 logico=[+5V,+15V]
  - In ricezione, 1 logico=[<-3V], 0 logico=[>+3V]
  - Il Tx deve vedere una capacità <2.5nF, una resistenza <7KOhm
  - Il Tx deve poter supportare un corto circuito con corrente <0.5A
  - Massima lunghezza cavo:
    per 20Kb/s →16m, per 9.6Kb/s →75m, per 1.2Kb/s →1000m
- **I connettori più comuni sono due:**
  - DB-25: connettore D-type a 25 pin
  - DB-9 : connettore D-type a 9 pin



DB25 - Connettore maschio*



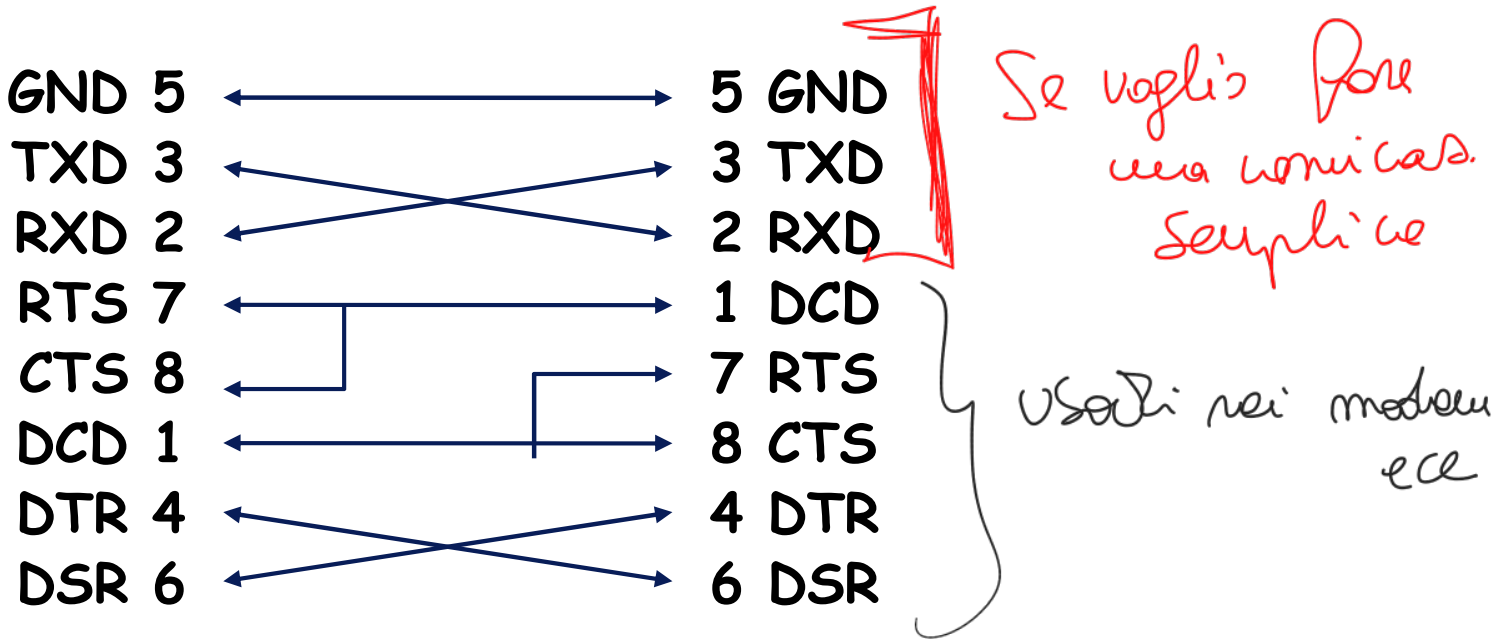
DB9 - Connettore maschio*

DB-25	Sigla	Segnale	DB-9
1	GND	Chassis Ground	Not Used
2	TXD	Trasmit Data	3
3	RXD	Receive Data	2
4	RTS	Request To Send	7
5	CTS	Clear To Send	8
6	DSR	Data Set Ready	6
7	GND	Signal Ground	5
8	DCD	Data Carrier Detect	1
20	DTR	Data Terminal Ready	4
22	RI	Ring Indicator	9

- **TXD (Trasmit Data):**
  linea di trasmissione dei dati connessa alla linea RXD del Rx
- **RXD (Receive Data):**
  linea di ricezione del dato, connessa alla linea TXD del Tx

Modem Cables

- Molte funzionalità di questi segnali derivano dal fatto che inizialmente furono pensati per connettere modem
- Per connessioni con altre periferiche, come stampanti, i segnali relativi ai modem devono essere "annullati" con l'uso di
  - cavi diretti oppure
  - cavi null-modem, es. per connettere due PC



Connessione NULL-MODEM

COME SUPPORTARE LA COMUNICAZ seriale IN UN PC?

Esempi di dispositivi di I/O: la Porta Seriale nei Personal Computer

- **Tipicamente implementata con un chip tipo 16550A** Serve un dispositivo I/O
  - **Bit-rate: compreso fra 50 e 115000** baud — fra 50 e 115K baud
  - **Formato trama:**
    - 1 bit di start
    - da 5 a 8 bit di dati
    - con o senza parità
    - 1, 1.5, 2 bit di stop
  - **Riconoscimento dell'errore**
    - Di parità (numero di 0 o 1 non corrispondente)
    - Di framing (bit di stop non ricevuti correttamente)
    - Di overrun (arrivo di un nuovo dato prima di aver prelevato il precedente)
  - **Emissione/ricezione di break** per interrompere comunicaz.
    - Persistenza in stato attivo per un tempo più lungo della trama
  - **Buffer dati da 16 byte** FIFO — buffering nella FIFO — Buffer FIFO di 16 byte

Allocato nello spazio di memoria 03F8, internamente ha 8 registri

- **PORTE DI I/O ASSEGNATE e INTERRUPT (Risorse)** | nei PC, registri nei PC, nei processori
  - 0x03F8-0x3FF porta seriale #1 (COM1 o ttyS0) con IRQ4 (interrupt)
  - 0x02F8-0x2FF porta seriale #2 (COM2 o ttyS1) con IRQ3
  - 0x03E8-0x3EF porta seriale #3 (COM3 o ttyS2) con IRQ4
  - 0x02E8-0x2EF porta seriale #4 (COM4 o ttyS3) con IRQ3

registri si dicono

3F8, 3F8 ... 3FF

- **Interfacciamento RS-232 compatibile** gestione
  - poiché i pin SIN, SOUT, /RTS, /CTS, /DSR, /DCD, /RI sono a livelli TTL (0-5V), si rendono necessari dei chip 75188, 75189 per adattare i livelli di tensione a quelli richiesti dallo standard RS-232C

oltre TX e RX, spesso 5 segnali per Modem

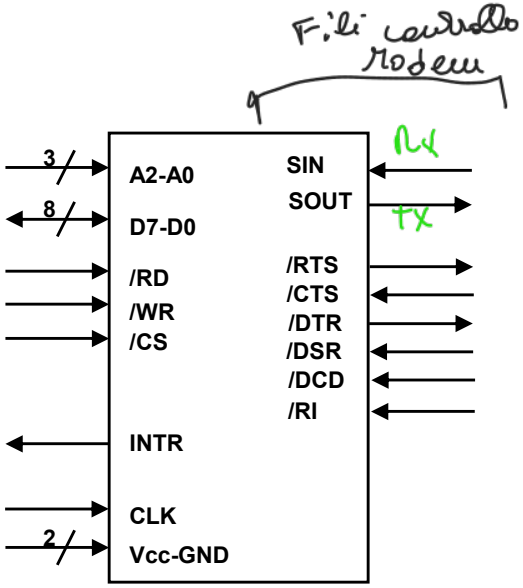
-15 +15 ✓

per gestire volt

Alcuni segnali compatibili

UART Intel-16550A

- **Visione Elettrica del chip:**



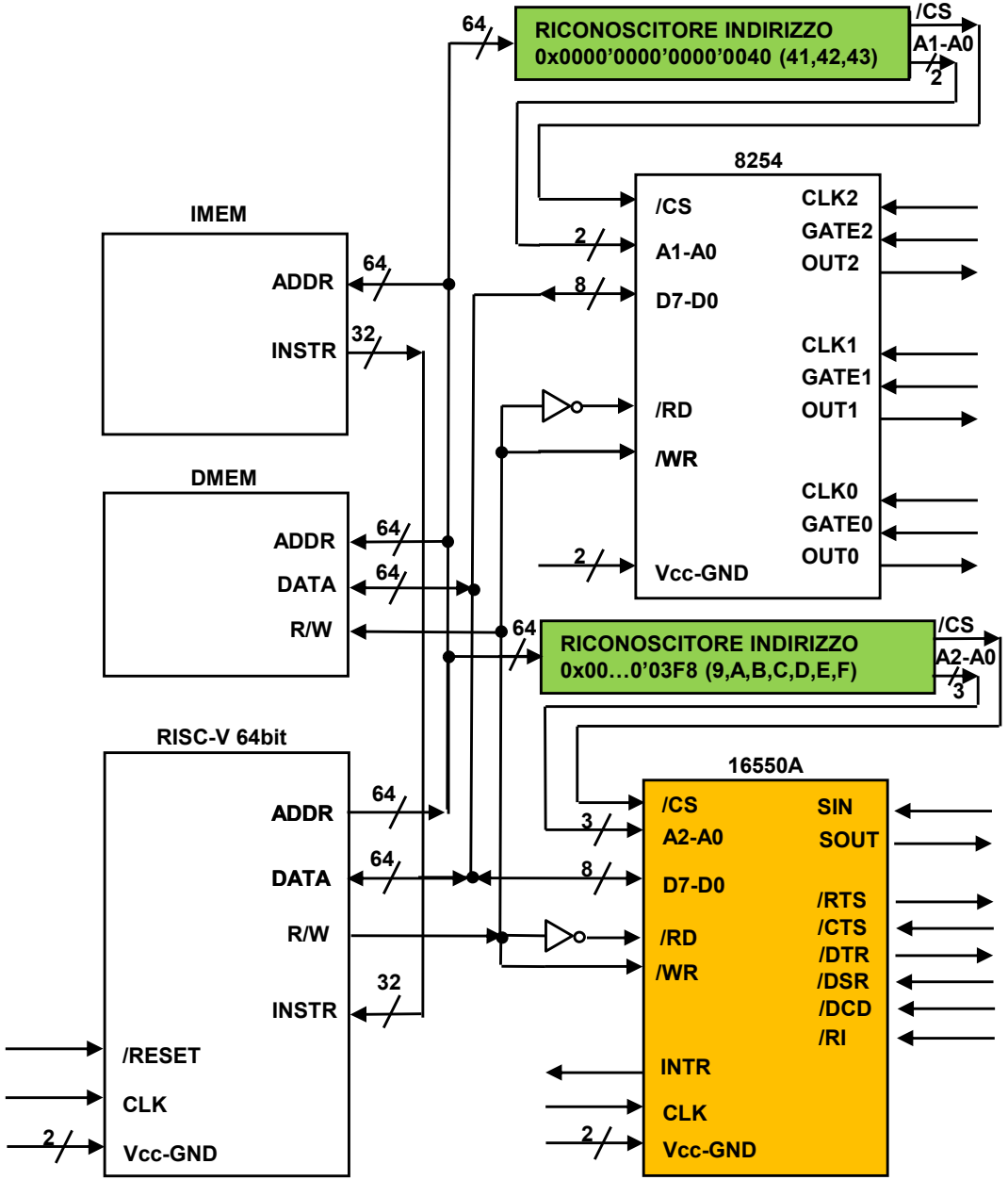
- **A2-A0: indirizzamento**
- **D7-D0: scambio dati su 8 bit**
- **/RD, /WR, /CS: Read, Write, Chip-Select**
- **Vcc-GND: alimentazione**
- **CLK: clock locale per derivare le temporizzazioni**

- **SIN, SOUT: segnali Tx e Rx**

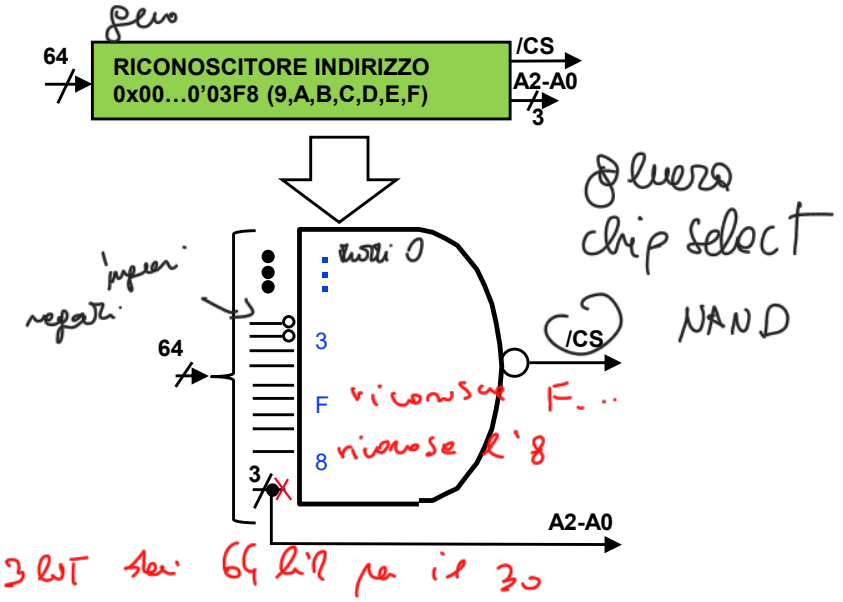
- **/RTS, /CTS, /DTR, /DSR, /DCD, /RI: segnali per l'interfacciamento secondo EIA-RS232C**

- **Altri pin possono essere presenti per usi generali e in dipendenza delle versioni specifiche del chip**

UART Intel-16550A: visione di insieme



- /CS: Chip-Select
- A2-A0: indirizzamento
- D7-D0: scambio dati su 8 bit
- /RD, /WR: Read, Write
- Vcc-GND: alimentazione
- CLK: clock locale per derivare le temporizzazioni
- SIN, SOUT: segnali Tx e Rx
- /RTS, /CTS, /DTR, /DSR, /DCD, /RI: segnali per interfacciamento EIA-RS232C
- Altri pin possono essere presenti per usi generali e in dipendenza delle versioni specifiche del chip



UART Intel-16550A

- **Visione logica del chip (tutti registri a 8 bit):**

A2-A0	access	DLAB bit	mnemonic		
000	R	0	RBR		Receive Buffer Register Dove si trova il dato ricevuto
000	W	0	THR		Transmitter Holding Register Dove si mette il dato da trasmettere
001	RW	0	IER		Interrupt Enable Register
010	R	0	IIR		Interrupt Identification Register
010	W	0	FCR		FIFO Control Register
011	RW	-	LCR		**Line Control Register**
100	RW	-	MCR		Modem Control Register I bit 0 e 1 sono dei latch su /RTS e /DTS (in scrittura gli altri bit possono essere messi a 0)
101	RW	-	LSR		Line Status Register
110	RW	-	MSR		Modem Status Register
111	RW	-	SCR		Scratchpad Register Registro di appoggio da usare liberamente
001	RW	1	DLM		**Divisor Latch MSB**
000	RW	1	DLL		**Divisor Latch LSB**

Velocità di trasmissione: DLM e DLL

- **Avendo posto a 1 il bit 7 di LCR (accesso ai Divisor Latches → DLAB=1) posso scrivere/leggere in DLM,DLL il valore della costante C che determina il bit-rate**
  - **DLM,DLL vengono interpretati come un intero senza segno a 16 bit**
  - **Detta FCK la frequenza del clock esterno e BR il bit-rate desiderato, si ha**

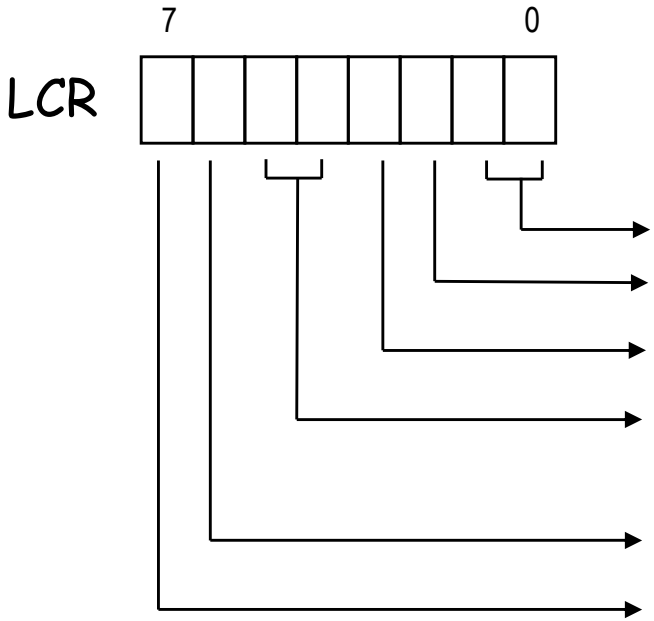
$$C_{ostante} = \frac{Fc}{16 \cdot BR}$$

Nota: nei PC Fc=1.8432MHz
Es. per ottenere un BR=9600
→C=12

BR	C
300	384
1200	96
4800	24
9600	12
115200	1

- **Note su DLAB**
  - **DLAB non è altro che il bit 7 di LCR**
  - **I registri RBR, THR, IER, IIR, FCR si accedono ponendo DLAB=0**
  - **Per i restanti registri il valore di DLAB è ininfluente**

Programmazione porta seriale: LCR, Line Control Register



n. di bit del dato 00→5, 01→6, 10→7, 11→8

n. di bit di stop: 0→1, 1→1.5 se il dato ha 5 bit, 2 altrimenti

1→bit di parità presente, 0→bit di parità assente

00→parità=#dispari di 1, 01→parità #pari di 1,
10→parità=#dispari a 0, 11→parità=#pari di 0

1→trasmetto un break (uscita a 0 logico)

Bit DLAB, se 1→posso leggere/scrivere la costante che determina il bit-rate

Esempio di programma che utilizza la porta seriale (1)

```
#include <sys/types.h>
#include <sys/stat.h>   /* open */
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>      /* stdin, stdout, stderr */
#include <sys/ioctl.h>
#include <termios.h>
#include <stdlib.h>
#include <sys/time.h>   /* timeval */
#include <stdlib.h>
#include <string.h>     /* memset used by FD_ZERO */

#define DEFAULT_DEV "/dev/ttyS0"
#define DEFAULT_SPEED 115200

int translate_speed(int spd);
void fd_setup(int fd, int speed, int flow);

int main(int argc, char **argv)
{
  int serfd, i, c=0, error=0, len, do_echo = 0;
  char txdata[127], rxdata[256];

  if (argc > 2) {
    printf("wrong syntax ! ./sertest [-e]"); exit(1);
  }

  if (argc == 2) {
    if ((strcmp("-e", argv[1]))==0) do_echo=1;
    else printf("wrong syntax ! ./sertest [-e]\n");
  }

  printf("Opening %s\n", DEFAULT_DEV);
  serfd = open(DEFAULT_DEV, O_RDWR);

  if (serfd < 0) exit(1);

  fd_setup(serfd, DEFAULT_SPEED, 1);

  for (i=0;i<127;i++) txdata[i]=i+64;
```

```
if (!do_echo)
  {
    while (!error){
      len = 0;
      i=0;
      printf("-%d- Sending data...", c++);
      write(serfd, txdata, 127);
      printf("done.\n");

      /* wait for 127 chars */
      while (len < 127)
      {
        len += read(serfd, rxdata+len, 127);
        printf("Got [%d/127]...\r", len);
        fflush(stdout);
      }

      /* now compare the received data */
      printf("Got response, comparing... ");

      if (memcmp(txdata, rxdata, 127) == 0)
        printf("OK.\n");
      else
        error = 1;
    }
  }
```

Se collego 2 pc Broadcast col un pc

Esempio di programma che utilizza la porta seriale (2)

```
else
  {
    printf("Running as echo server.\n");
    while (!error) {
      len = 0;

      /* wait for 127 chars */
      while (len < 127)
      {
        len += read(serfd, rxdata+len, 127);
        printf("Got [%d/127]...\r", len);
        fflush(stdout);
      }

       /* now compare the received data */
      printf("-%d- Checking RX data... ", c++);

      if (memcmp(txdata, rxdata, 127) == 0)
      {
        printf("OK...echoing.\n");
        write(serfd, rxdata, 127);
      }
      else
      {
        printf("ERROR\n");
        error = 1;
      }
    }
  }

  printf("Not OK !\n");
  for (i=0;i<127;i++)
    printf("%c", txdata[i]);

  printf("\n");

  exit(1);
}
```

```
int translate_speed(int spd)
{
  int speed_c = 0;

  switch(spd) {
   case 9600:   speed_c = B9600; break;
   case 19200:  speed_c = B19200; break;
   case 38400:  speed_c = B38400; break;
   case 57600:  speed_c = B57600; break;
   case 115200: speed_c = B115200; break;
   case 230400: speed_c = B230400; break;
   case 460800: speed_c = B460800; break;
   default:     printf(
     "Bad baudrate %d is not valid.\n", spd); break;
  }
  return speed_c;
}

void fd_setup(int fd, int speed, int flow)
{
  struct termios t;

  if (fd < 0) { perror("fd_setup"); exit(1); }

  if (tcgetattr(fd, &t) < 0)
     { perror("tcgetattr"); exit(1); }

  cfmakeraw(&t);

  t.c_cflag &= ~CBAUD;
  t.c_cflag |= translate_speed(speed) | CS8 | CLOCAL;
  t.c_oflag = 0; /* turn off output processing */
  t.c_lflag = 0; /* no local modes */

  if (flow) t.c_cflag |= CRTSCTS;
  else t.c_cflag &= ~CRTSCTS;

  if (tcsetattr(fd, TCSANOW, &t) < 0)
     { perror("fd_setup : tcsetattr"); exit(1); }
  return;
}
```