Costruzione di Messaggi

(alcune considerazioni)

Framing & Parsing

Client e Server devono accordarsi su come l' informazione debba essere codificata

Framing è il problema di formattare il messaggio in modo che il ricevente possa farne il parsing

(Es: individuare inizio e fine del messaggio, limiti tra i campi del messaggio, ecc. …..)

Caso1: il messaggio ha dimensione fissata, nota a priori

Si riceve il numero di byte atteso in un buffer

Caso2: il messaggio contiene delimitatori

Si implementa una procedura di parsing per individuare i delimitatori di campi, per es. "Nome#Cognome#"

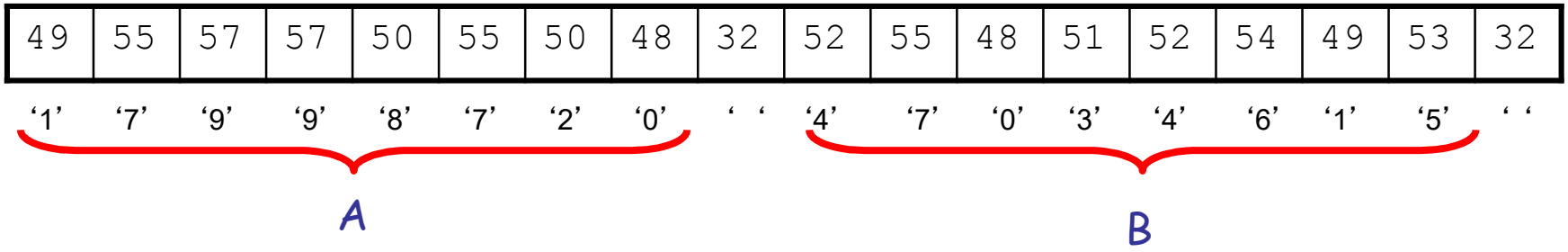
Codifica dell' informazione

Client e Server devono accordarsi su come l' informazione debba essere codificata

possibilità 1 :

Stringhe di cifre decimali: *sequenze di byte i cui valori sono determinati in accordo con una qualche codifica (es: ASCII)*

Per rappresentare *17.998.720* ***e*** *47.034.615*




```
sprintf(sendbuf,"%d%d ", A, B);
send( m_socket, sendbuf, strlen(sendbuf), 0 );
```

Buffer contenente i dati da trasmettere

Codifica dell' informazione

Possibilità 1: Stringhe di cifre decimali

```
sprintf(sendbuf,"%d%d ", A, B);
send( m_socket, sendbuf, strlen(sendbuf), 0 );
```

- *sendbuf deve essere abbastanza grande (numeri più grandi, segno .....)*
- *un comune errore:*

```
#define BUFSIZE 132
.
.
.
Char sendbuf[BUFSIZE]
.
.
sprintf(sendbuf,"%d%d ", A, B)
send( m_socket, sendbuf, BUFSIZE, 0 );
```

- *Il ricevente riceve byte extra "spazzatura"*

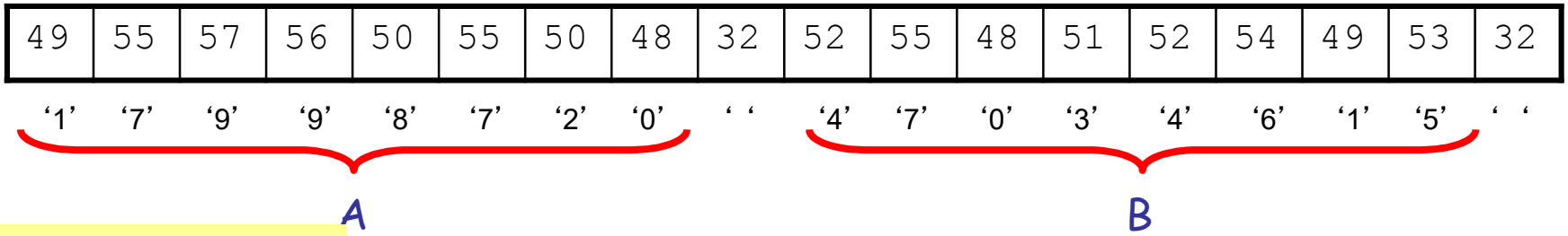
Codifica dell' informazione

Client e Server devono accordarsi su come l' informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare 17.998.720 e 47.034.615

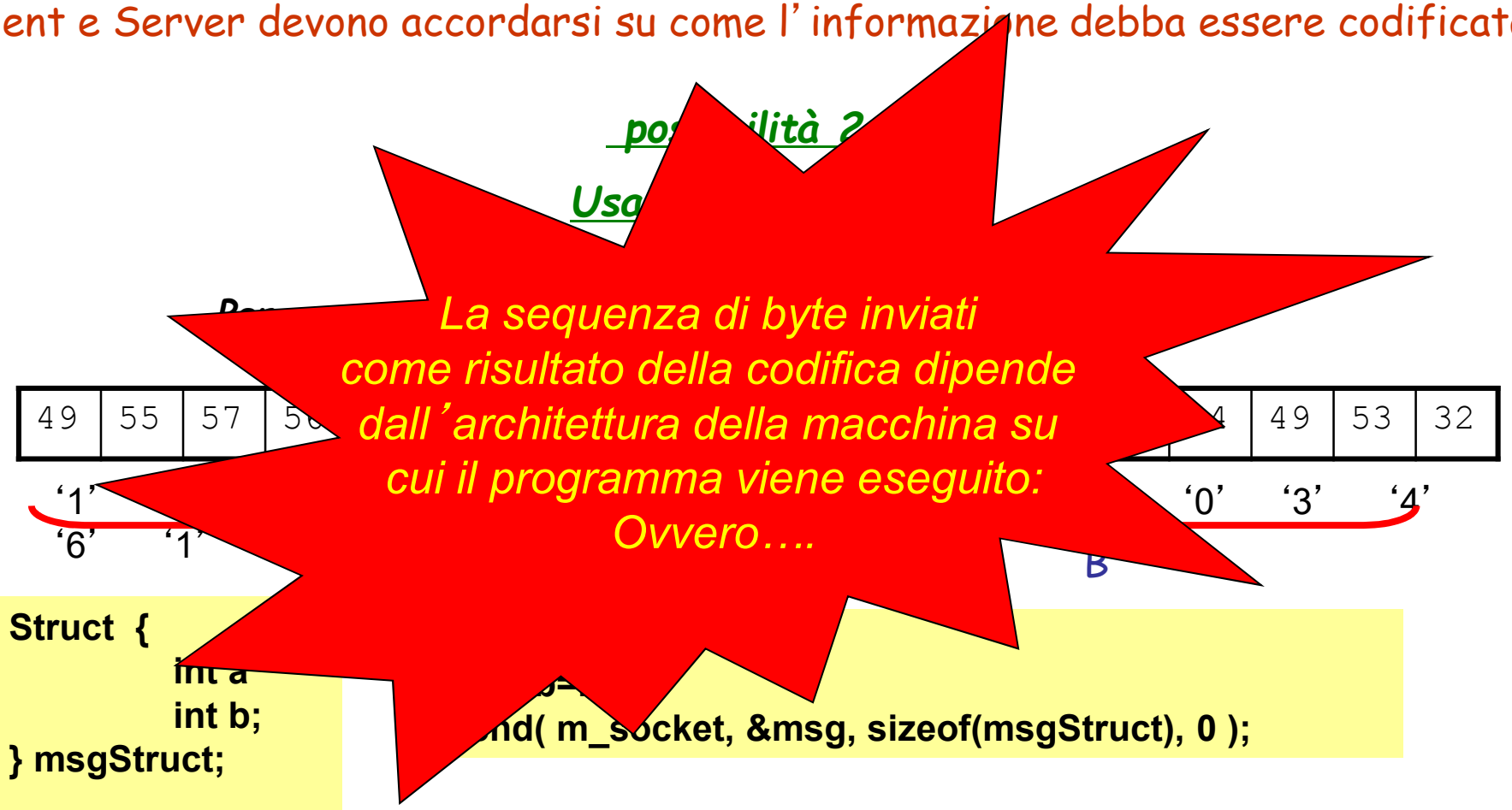



```
Struct  {
            int a;
            int b;
} msgStruct;
..
struct msgStruct msg;
```

```
msg.a=A;
msg.b=B;
send( m_socket, &msg, sizeof(msgStruct), 0 );
```

Codifica dell' informazione

Client e Server devono accordarsi su come l' informazione debba essere codificata

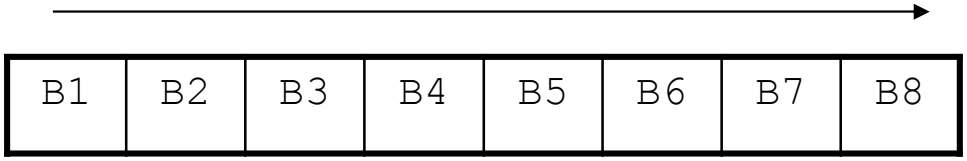



```
Struct  {
            int a
            int b;
} msgStruct;
..
struct msgStruct msg;
```

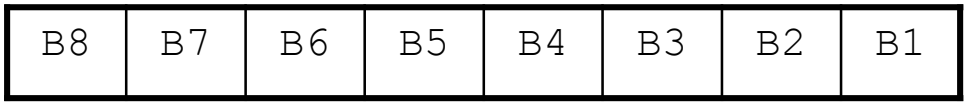
```
nd( m_socket, &msg, sizeof(msgStruct), 0 );
```

Big-Endian vs. Little-Endian

in base a come assumono che siano accodati i byte delle rappresentazioni multi-byte.



Ordine crescente indirizzi di memoria



Big-Endian: byte memorizzati da SX a DX ⇔ Il byte PIU' significativo ha lo stesso indirizzo dell' intera parola

Little-Endian: byte memorizzati da DX a SX ⇔ Il byte MENO significativo ha lo stesso indirizzo dell' intera parola (ordine inverso rispetto al Big-Endian)

Attenzione

Se due programmi che comunicano in rete hanno un formato di ordinamento di byte opposto possono insorgere problemi

Soluzione:

Per convenzione si è scelto il Big-Endian come standard per il formato di ordinamento dei byte nel networking

Sono inoltre necessarie delle funzioni di conversione dal formato dell'host a quello standard di rete Big-Endian

-Se l'host è Big-Endian le funzioni di conversione non cambiano l'ordine dei byte

-Se invece l'host è Little-Endian l'ordine è invertito

Alcune funzioni utili

- **`inet_addr()`**
  - converte una stringa espressa nella cosiddetta notazione dotted-decimal ("127.0.0.1") in un numero a 32 bit espresso nella rappresentazione della rete
- **`inet_ntoa()`**
  - esegue la conversione inversa
- **`htons()`**
  - converte uno short integer (2 byte) dalla rappresentazione della piattaforma a quella della rete
- **`ntohs()`**
  - esegue la conversione inversa
- **`htonl()`**
  - converte long integer (4 byte) dalla piattaforma alla rete
- **`ntohl()`**
  - esegue la conversione inversa

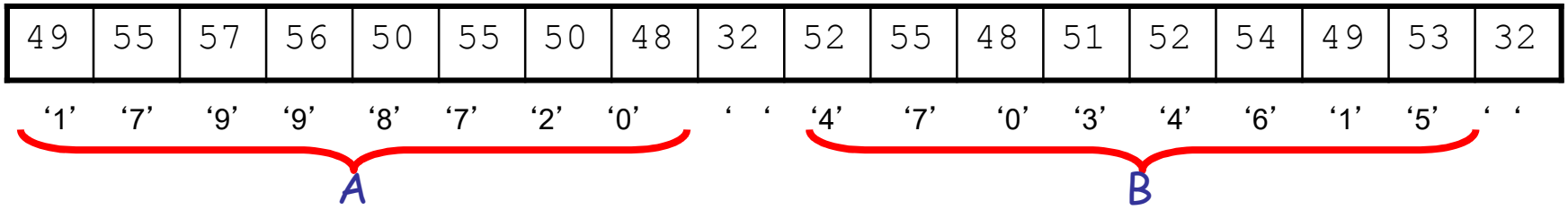
Codifica dell' informazione

Client e Server devono accordarsi su come l' informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare 17.998.720 e 47.034.615




```
Struct  {
            int a;
            int b;
} msgStruct;
..
```

```
struct msgStruct msg;          CLIENT SIDE
....
msg.a=htonl(A);
msg.b=htonl(B);
send( m_socket, &msg, sizeof(msgStruct), 0 );
```

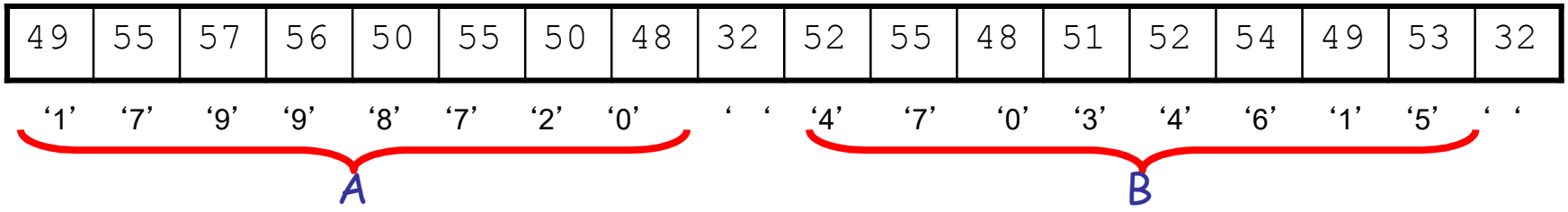
Codifica dell' informazione

Client e Server devono accordarsi su come l' informazione debba essere codificata

possibilità 2:

Usare una Struct

Per rappresentare `17.998.720` ***e*** `47.034.615`




```
Struct  {
            int a;
            int b;
} msgStruct;
..
```

```
struct msgStruct msg;          SERVER SIDE
....
recv( m_socket, &msg, sizeof(msgStruct), 0 );
msg.a=ntohl(msg.a)
msg.b=ntohl(msg.b)
```

Esercizio 2

Progettare ed implementare (scrivendo **codice portabile**) un'applicazione **TCP client/server** conforme al seguente protocollo:

1) Il server è avviato su una porta a scelta (NB: Il server resta sempre in ascolto).

2) Il client legge da tastiera l'IP e il numero di porta da utilizzare per contattare il Server

3) Il client contatta il server inviando il messaggio iniziale "Hello".

4) Ricevuto il messaggio iniziale, il server visualizza sullo std output un messaggio contenente il nome dell'host del client con cui ha stabilito la connessione. (Esempio: **"messaggio ricevuto dal client"**)

5) Il server invia al client la stringa "ack".

6) Il client riceve la stringa e la stampa.

7) Il client legge una dallo std input due interi n1 e n2 e li invia al server

8) Il server riceve i due numeri e visualizza sullo std output "la somma di n1+n2=res", dove res è il risultato;

9) Il server invia al client il risultato **res**

10) Il client riceve la stringa dal server. La visualizza sullo STD output e termina;

11) Il server rimane in ascolto.