Università di Ferrara

# Architettura di Reti

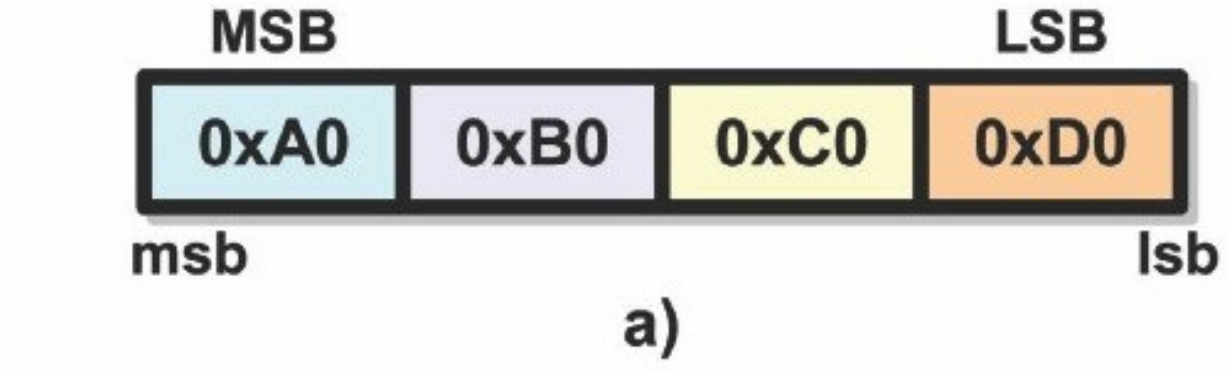
## Gestione dell'Eterogeneità

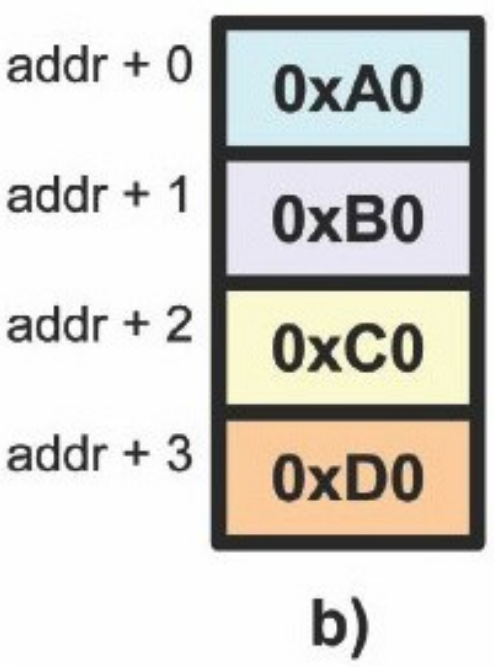
Carlo Giannelli
carlo.giannelli@unife.it
http://www.unife.it/scienze/informatica/insegnamenti/architettura-reti/
http://docente.unife.it/carlo.giannelli

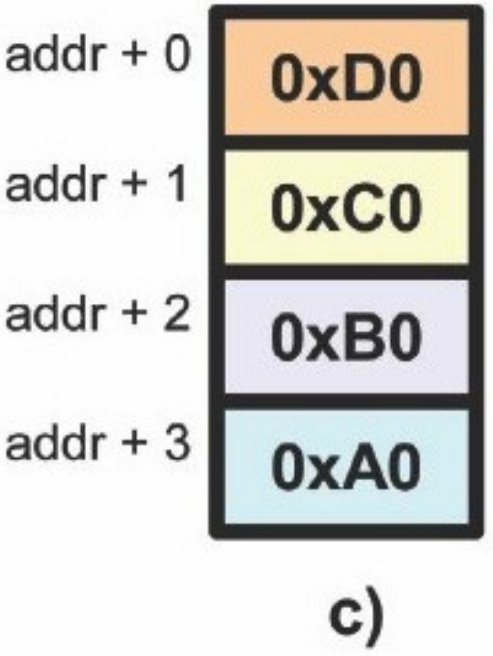
# Il Problema dell'eterogeneità

- Nelle reti di calcolatori vi è un'estrema eterogeneità di sistemi hw/sw
- Client e Server possono eseguire su architetture diverse che usano differenti rappresentazioni dei dati
  - **rappresentazione informazioni**
    - caratteri: US-ASCII, ISO 8859, Unicode…
    - interi: rappresentazione in complemento a 1 o a 2…
  - **dimensione dati**
    - interi: dimensione 4 o 8 byte
    - reali: lunghezza esponente e mantissa
    - caratteri: US-ASCII 1 byte, UTF-8 da 1 a 4 byte, UTF-16 da 2 a 4 byte
  - **ordine byte** all'interno di una parola di più byte
    - little endian vs. big endian
- Necessità di definire una rappresentazione comune dei dati e di implementare meccanismi per gestirla

# Es. little endian o big endian




Big endian

Little endian

# Caveat

- Attenzione! Il problema della rappresentazione eterogenea dei dati tra diverse piattaforme HW/SW presenta complessità che vanno ben oltre le comunicazioni di rete
- Ad esempio, anche la ricompilazione dello stesso codice sorgente su piattaforme diverse può presentare spiacevoli sorprese. Si veda l'articolo:

"Twice the Bits, Twice the Trouble:
Vulnerabilities Induced by Migrating to 64-Bit Platforms"
*https://www.tu-braunschweig.de/Medien-DB/sec/pubs/2016-ccs.pdf*

# Rappresentazione dati

- Per comunicare tra nodi eterogenei sono possibili due tipi di soluzioni:
  - ogni nodo converte i dati nel formato specifico del destinatario (prestazioni)
  - si concorda un formato comune di rappresentazione dei dati che i nodi useranno per comunicare tra loro (flessibilità)
- Supponendo di avere N diversi tipi di nodi, nel primo caso avrò bisogno di N*(N-1) procedure di conversione dati, nel secondo caso solo di 2*N procedure

# Ma quale ISO livello 6?!?!

- Socket API è un'interfaccia di basso livello (tra ISO L4 e L5) che purtroppo non fornisce alcuno strumento di questo tipo
- Non esiste un unico standard per il livello di presentazione
  - molte soluzioni, con caratteristiche molto diverse, sono stati sviluppate per ambiti specifici
- La soluzione giusta da adottare andrà valutata caso per caso

# eXternal Data Representation (XDR)

- Sun XDR è una soluzione realizzata all'interno dello stack Sun/ONC RPC
- XDR fornisce un insieme di procedure di conversione per trasformare la rappresentazione nativa dei dati in una *rappresentazione esterna* (XDR) e viceversa
- XDR fa uso di uno stream (contenuto in un buffer) che permette di creare un messaggio con i dati in forma XDR
- I dati vengono inseriti/estratti nello/dallo stream XDR uno alla volta, tramite operazioni di serializzazione e/o deserializzazione (marshalling/unmarshalling)

# Esempio di serializzazione XDR

```
int i = 260;
char str[80] = “pippo”;
XDR *xdrs; /* stream XDR */
char buf[BUFSIZE]; /* buffer vuoto, da preparare */
...
/* Creazione stream XDR in memoria */
xdrmem_create(xdrs, buf, sizeof(buf), XDR_ENCODE);

/* Inserimento nello stream di un intero, convertito in formato XDR */
xdr_int(xdrs, &i);

/* Inserimento nello stream di una stringa, convertita in formato XDR */
xdr_string(xdrs, &str, strlen(str));

/* Scrittura su socket */
write(sd, buf, xdr_getpos(xdrs));
```

# Esempio di deserializzazione XDR

```
int i;
char str[80];
XDR *xdrs;
char buf[BUFSIZE]; /* buffer vuoto, da preparare */
…
/* Lettura da socket */
read(sd, buf, sizeof(buf));

/* Creazione stream XDR in memoria */
xdrmem_create(xdrs, buf, sizeof(buf), XDR_DECODE);

/* Lettura di un intero dallo stream, convertito dal formato XDR */
xdr_int(xdrs, &i);

/* Lettura di una stringa dallo stream, convertita dal formato XDR */
memset(str, 0, sizeof(str));
xdr_string(xdrs, &str, sizeof(str)-1);
```

# Interface Definition Language - IDL

- Oltre ai tipi di dati primitivi, per cui fornisce già routine di serializzazione e di deserializzazione, XDR permette di gestire tipi di dati complessi
- In XDR, il formato delle strutture dati è definito attraverso un apposito linguaggio *IDL (Interface Definition Language)* simile al C
- Uso di *IDL compiler* per generare automaticamente le procedure di codifica e decodifica dei dati complessi

Additional Material

# Esempio di IDL (Sun/ONC RPC)

```
/* definisci massima dimensione stringhe */
const MAXNAMELEN = 255;

/* parametro: nome directory */
typedef string nametype<MAXNAMELEN>;

/* valore di ritorno: lista di file */
typedef struct namenode *namelist;

struct namenode {
        nametype name; /* nome del file */
        namelist pNext; /* prossimo file */
};
```

# Altre soluzioni

- CORBA Common Data Representation (CDR)
- ASN.1/X.680 (ITU-T/OSI)
- Soluzioni Web-oriented:
  - Google Protocol Buffers (protobuf)
  - Avro https://avro.apache.org/
  - MessagePack https://msgpack.org/
  - Apache Thrift https://thrift.apache.org/

# Google Protocol Buffers - protobuf

- https://developers.google.com/protocol-buffers/
- "Protocol buffers are Google's **language-neutral**, **platform-neutral**, **extensible** mechanism for serializing **structured data**"

- Step principali:
  1. definire i tipi di messaggio "protocol buffer" in un file .proto
  2. compilare il file .proto col compilatore di protobuf per generare codice language-specific
  3. utilizzare il codice generato per serializzare e deserializzare
  4. compilare e linkare tutto assieme

# protobuf – pre-requisiti

- Installazione librerie
  - sudo apt install protobuf-c-compiler
  - sudo apt-get install libprotobuf-dev
  - sudo apt-get install libprotobuf-c-dev

# File .proto per definizione messaggi

- Ad esempio, file di testo “AMessage.proto”
- Ogni campo ha a) un nome, b) un tipo, c) un identificativo univoco numerico
- required → uno ed uno solo di questi campi
- optional → non più di uno di questi campi

```
message AMessage {
  required int32 a = 1;
  optional int32 b = 2;
  required string c = 3;
}
```

# Tipi di dato

| .proto | C++ | Java | Python | C# | PHP |
|---|---|---|---|---|---|
| **double** | double | double | float | double | float |
| **float** | float | float | float | float | float |
| **int32** | int32 | int | int | int | integer |
| **int64** | int64 | long | int/long | long | integer/string |
| **bool** | bool | boolean | bool | bool | boolean |
| **string** | string | String | str/unicode | string | string |
| **bytes** | string | ByteString | str | ByteString | string |

Dettagli e note in https://developers.google.com/protocol-buffers/docs/proto

"[…] A string must always contain UTF-8 encoded or 7-bit ASCII text. […]"

# Compilazione file .proto (1)

- "protoc-c --c_out=. amessage.proto" per generare "amessage.pb-c.c" e "amessage.pb-c.h"
- **AMessage**: struct che contiene i campi a, b, has_b, c

**amessage.pb-c.h**

```
typedef struct _AMessage AMessage;

struct  _AMessage
{
  ProtobufCMessage base;
  int32_t a;
  protobuf_c_boolean has_b; // b is optional
  int32_t b;
  char *c;
};
```

# Compilazione file .proto (2)

- "protoc-c --c_out=. amessage.proto" per generare "amessage.pb-c.c" e "amessage.pb-c.h"

**amessage.pb-c.h**

```
void   amessage__init (AMessage *message);
size_t amessage__get_packed_size (const AMessage   *message);
size_t amessage__pack (const AMessage
                                *message, uint8_t *out);
size_t amessage__pack_to_buffer (const AMessage *message,
                                      ProtobufCBuffer *buffer);

AMessage * amessage__unpack (ProtobufCAllocator  *allocator,
                                      size_t len,
                                      const uint8_t *data);
void   amessage__free_unpacked (AMessage *message,
                           ProtobufCAllocator *allocator);
```

# Funzioni/macro amessage.pb-c.h

- Serializzazione
  - **AMESSAGE__INIT**: macro per inizializzare correttamente una variabile di tipo AMessage
  - **amessage__get_packed_size**: funzione per ottenere quale sarà la dimensione di una variabile di tipo AMessage a valle del processo di serializzazione
  - **amessage__pack**: funzione che effettua la serializzazione
- Deserializzazione
  - **unpack**: funzione che effettua la deserializzazione
  - **amessage__free_unpacked**: funzione che dealloca la memoria allocata da unpack

# Uso del codice generato per serializzare

```
AMessage msg = AMESSAGE__INIT; // AMessage
void *buf;          // Buffer to store serialized data
unsigned len;       // Length of serialized data

msg.a = atoi(argv[1]);
if(argc==3) { msg.has_b = 1; msg.b = atoi(argv[2]); }
msg.c = "ciao";

len = amessage__get_packed_size(&msg);
buf = malloc(len);
// buf will store the serialized version of msg
amessage__pack(&msg, buf);

// Use buf, e.g., sending it via socket */

free(buf); // Free the allocated serialized buffer
```

# Uso del codice generato per deserializzare

```
AMessage *msg;

// Read packed message and size
uint8_t buf[MAX_MSG_SIZE];
size_t msg_len = read_buffer (MAX_MSG_SIZE, buf);

// Unpack the message (implicit memory allocation)
msg = amessage__unpack(NULL, msg_len, buf);
if (msg==NULL){ fprintf(stderr, "error\n");exit(1); }

// Now it is possible to use msg fields
printf("Received: a=%d",msg->a);
if (msg->has_b) printf("  b=%d",msg->b);
printf("  c=%s",msg->c);

// Free the unpacked message
amessage__free_unpacked(msg, NULL);
```

# Funzione di supporto read_buffer

```
read_buffer (unsigned max_length, uint8_t *out) {

  size_t cur_len = 0;
  size_t nread;

  while ((nread=fread(out + cur_len, 1, max_length - cur_len,
                                                    stdin)) != 0){
    cur_len += nread;
    if (cur_len == max_length) {
       fprintf(stderr, "max message length exceeded\n");
       exit(1); }
  }

  return cur_len;
}
```

# Compilazione e linking

- Compilazione file generati
  - "gcc -c amessage.pb-c.c"

- Compilazione e linking 1) del codice che utilizza protobuf, 2) dei file generati e 3) delle librerie protobuf
  - gcc -o example.out example.c amessage.pb-c.c -L/usr/lib -lprotobuf-c

# Message boundary

- Protobuf risolve il problema dell'eterogeneità
- Protobuf non risolve il problema del message boundary
  - OK singolo messaggio protobuf in singolo datagram
  - KO più messaggi protobuf in singolo datagram
  - KO più mesaggi protobuf consecutivi sullo stesso stream
- Soluzioni per message boundary
  - ZeroMQ http://zeromq.org/ come soluzione production-ready: "**ZMTP delimits the TCP stream as 'frames'.** […] A frame consists of a flags field, followed by a 'length" field and a frame body of 'length' octets". https://rfc.zeromq.org/spec:15/ZMTP/
  - Ipotesi semplificative: ack applicativo, prima si invia dimensione del pacchetto e poi pacchetto, buffer di dimensione costante…
  - ...

# Protobuf & ZeroMQ – Client (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <zmq.h>
#include "amessage.pb-c.h"

#define MAX_MSG_SIZE 4096

int main (int argc, const char *argv[])
{
  AMessage msg = AMESSAGE__INIT; // AMessage
  uint8_t *send_buf;             // Buffer to store serialized data
  // uint8_t recv_buf[MAX_MSG_SIZE];
  int rc, cc;
  size_t len;
  void *context, *requester;

  // Allow one or two integers
  if (argc != 2 && argc != 3) {
    fprintf(stderr, "Usage: %s a [b]\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  // Create REQREP socket
  context = zmq_ctx_new();
  if (context == NULL) {
    perror("zmq_ctx_new");
    exit(EXIT_FAILURE);
  }
```

Additional Material

# Protobuf & ZeroMQ – Client (2)

```
requester = zmq_socket(context, ZMQ_REQ);
if (requester == NULL) {
  perror("zmq_socket");
  exit(EXIT_FAILURE);
}

rc = zmq_connect(requester, "tcp://localhost:5555");
if (rc != 0) {
  perror("zmq_connect");
  exit(EXIT_FAILURE);
}

msg.a = atoi(argv[1]);
if (argc == 3) { msg.has_b = 1; msg.b = atoi(argv[2]); }
msg.c = "ciao";
len = amessage__get_packed_size(&msg);

send_buf = malloc(len);
amessage__pack(&msg, send_buf);
```

# Protobuf & ZeroMQ – Client (3)

```
  // invio richiesta contenente messaggio
  fprintf(stdout, "Writing %lu serialized bytes\n", len);
  cc = zmq_send(requester, send_buf, len, 0);
  if (cc < 0) {
    perror("zmq_send");
    exit(EXIT_FAILURE);
  }

  // Free the allocated serialized buffer
  free(send_buf);

  zmq_close(requester);
  zmq_ctx_destroy(context);

  return 0;
}
```

Additional Material

# Protobuf & ZeroMQ – Server (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <zmq.h>
#include "amessage.pb-c.h"

#define MAX_MSG_SIZE 4096

int main(void)
{
  uint8_t recv_buf[MAX_MSG_SIZE];
  AMessage *msg;
  void *context, *responder;
  int cc, rc;
  // char *risposta = "Ok!";

  // Create REQREP socket
  context = zmq_ctx_new();
  if (context == NULL) {
    perror("zmq_ctx_new");
    exit(EXIT_FAILURE);
  }
```

# Protobuf & ZeroMQ – Server (2)

```
responder = zmq_socket(context, ZMQ_REP);
if (responder == NULL) {
  perror("zmq_socket");
  exit(EXIT_FAILURE);
}

rc = zmq_bind(responder, "tcp://*:5555");
if (rc != 0) {
  perror("zmq_bind");
  exit(EXIT_FAILURE);
}

// Receive request
cc = zmq_recv(responder, recv_buf, sizeof(recv_buf), 0);
if (cc < 0) {
  perror("zmq_recv");
  exit(EXIT_FAILURE);
}
```

# Protobuf & ZeroMQ – Server (3)

```
  // Unpack the message using protobuf-c.
  msg = amessage__unpack(NULL, cc, recv_buf);
  if (msg == NULL)
  {
    fputs("Error unpacking incoming message", stderr);
    exit(EXIT_FAILURE);
  }

  // Display the message's fields.
  printf("Received: a=%d", msg->a); // required field
  if (msg->has_b) printf("  b=%d", msg->b);         // handle optional field

  printf("  c=%s\n", msg->c);

  // Free the unpacked message
  amessage__free_unpacked(msg, NULL);

  zmq_close(responder);
  zmq_ctx_destroy(context);

  return 0;
}
```

# Protocolli Testuali

- Nella realizzazione di applicazioni distribuite, l'adozione di protocolli testuali si è spesso rivelata vincente
  - facilità di testing e debugging
  - estendibilità

- *"When you feel the urge to design a complex binary file format, or a complex binary application protocol, it is generally wise to lie down until the feeling passes."* – Eric Raymond, "The Art of Unix Programming" http://www.catb.org/~esr/writings/taoup/html/

# US-ASCII

- Per decenni, lo standard per la rappresentazione del testo è stato US-ASCII.
- US-ASCII definisce un **set di caratteri e una loro rappresentazione in formato binario a 8-bit**.
- Solo i 7 bit meno significativi sono effettivamente utilizzati nello standard US-ASCII (127 caratteri). Il bit più significativo di ciascun byte è sempre settato a 0.

# Tabella US-ASCII

| Bits $b_4$ $b_3$ $b_2$ $b_1$ | Row ↓ / Column → | $b_7$ $b_6$ $b_5$ = 0 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1 0 0 | 1 0 1 | 1 1 0 | 1 1 1 |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 0 0 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p |
| 0 0 0 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q |
| 0 0 1 0 | 2 | STX | DC2 | " | 2 | B | R | b | r |
| 0 0 1 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s |
| 0 1 0 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t |
| 0 1 0 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u |
| 0 1 1 0 | 6 | ACK | SYN | & | 6 | F | V | f | v |
| 0 1 1 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w |
| 1 0 0 0 | 8 | BS | CAN | ( | 8 | H | X | h | x |
| 1 0 0 1 | 9 | HT | EM | ) | 9 | I | Y | i | y |
| 1 0 1 0 | 10 | LF | SUB | * | : | J | Z | j | z |
| 1 0 1 1 | 11 | VT | ESC | + | ; | K | [ | k | { |
| 1 1 0 0 | 12 | FF | FC | , | < | L | \ | l | \| |
| 1 1 0 1 | 13 | CR | GS | - | = | M | ] | m | } |
| 1 1 1 0 | 14 | SO | RS | . | > | N | ^ | n | ~ |
| 1 1 1 1 | 15 | SI | US | / | ? | O | _ | o | DEL |

# ISO 8859

- I caratteri della tabella US-ASCII sono sufficienti per la rappresentazione dell'alfabeto inglese, ma non per quella di molte altre lingue europee. Ad esempio, US-ASCII non supporta accenti (à, è, è…) e umlaut (ä, ö, ü).
- **Lo standard ISO 8859 estende US-ASCII utilizzando anche l'ottavo bit**, portando così il set di caratteri supportati a 255.
- Diverse mappe di caratteri *8859-n* per coprire le varie lingue (es. per l'italiano si hanno 8859-1 "Latin 1" e 8859-15 "Latin 9").

# Tabella ISO 8859-15

| | -0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -A | -B | -C | -D | -E | -F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0- | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 000A | 000B | 000C | 000D | 000E | 000F |
| 1- | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 | 0016 | 0017 | 0018 | 0019 | 001A | 001B | 001C | 001D | 001E | 001F |
| 2- | 0020 | ! 0021 | " 0022 | # 0023 | $ 0024 | % 0025 | & 0026 | ' 0027 | ( 0028 | ) 0029 | * 002A | + 002B | , 002C | - 002D | . 002E | / 002F |
| 3- | 0 0030 | 1 0031 | 2 0032 | 3 0033 | 4 0034 | 5 0035 | 6 0036 | 7 0037 | 8 0038 | 9 0039 | : 003A | ; 003B | < 003C | = 003D | > 003E | ? 003F |
| 4- | @ 0040 | A 0041 | B 0042 | C 0043 | D 0044 | E 0045 | F 0046 | G 0047 | H 0048 | I 0049 | J 004A | K 004B | L 004C | M 004D | N 004E | O 004F |
| 5- | P 0050 | Q 0051 | R 0052 | S 0053 | T 0054 | U 0055 | V 0056 | W 0057 | X 0058 | Y 0059 | Z 005A | [ 005B | \ 005C | ] 005D | ^ 005E | _ 005F |
| 6- | ` 0060 | a 0061 | b 0062 | c 0063 | d 0064 | e 0065 | f 0066 | g 0067 | h 0068 | i 0069 | j 006A | k 006B | l 006C | m 006D | n 006E | o 006F |
| 7- | p 0070 | q 0071 | r 0072 | s 0073 | t 0074 | u 0075 | v 0076 | w 0077 | x 0078 | y 0079 | z 007A | { 007B | \| 007C | } 007D | ~ 007E | 007F |
| 8- | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 008A | 008B | 008C | 008D | 008E | 008F |
| 9- | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 | 0096 | 0097 | 0098 | 0099 | 009A | 009B | 009C | 009D | 009E | 009F |
| A- | 00A0 | ¡ 00A1 | ¢ 00A2 | £ 00A3 | € 20AC | ¥ 00A5 | Š 0160 | § 00A7 | š 0161 | © 00A9 | ª 00AA | « 00AB | ¬ 00AC | - 00AD | ® 00AE | ¯ 02C9 |
| B- | ° 00B0 | ± 00B1 | 2 00B2 | 3 00B3 | Ž 017D | µ 00B5 | ¶ 00B6 | · 00B7 | ž 017E | 1 00B9 | º 00BA | » 00BB | Œ 0152 | œ 0153 | Ÿ 0178 | ¿ 00BF |
| C- | À 00C0 | Á 00C1 | Â 00C2 | Ã 00C3 | Ä 00C4 | Å 00C5 | Æ 00C6 | Ç 00C7 | È 00C8 | É 00C9 | Ê 00CA | Ë 00CB | Ì 00CC | Í 00CD | Î 00CE | Ï 00CF |
| D- | ð 00F0 | Ñ 00D1 | Ò 00D2 | Ó 00D3 | Ô 00D4 | Õ 00D5 | Ö 00D6 | × 00D7 | Ø 00D8 | Ù 00D9 | Ú 00DA | Û 00DB | Ü 00DC | Ý 00DD | Þ 00DE | ß 00DF |
| E- | à 00E0 | á 00E1 | â 00E2 | ã 00E3 | ä 00E4 | å 00E5 | æ 00E6 | ç 00E7 | è 00E8 | é 00E9 | ê 00EA | ë 00EB | ì 00EC | í 00ED | î 00EE | ï 00EF |
| F- | ð 00F0 | ñ 00F1 | ò 00F2 | ó 00F3 | ô 00F4 | õ 00F5 | ö 00F6 | ÷ 00F7 | ø 00F8 | ù 00F9 | ú 00FA | û 00FB | ü 00FC | ý 00FD | þ 00FE | ÿ 00FF |

# Un mondo post US-ASCII

- Nel ventunesimo secolo, non è più possibile limitare le nostre applicazioni ai set di caratteri US-ASCII o ISO 8859.
- Necessità di nuovi standard, che supportino anche:
  - nuovi alfabeti (cinese, arabo, ebraico, cirillico, ecc.)
  - caratteri composti
  - lingue right-to-left

# ISO 10646

- Lo *standard normativo ISO 10646* definisce lo ***Universal Character Set (UCS)*, un set di caratteri che contiene tutti i caratteri universalmente noti**
- Ciascun carattere ha un codice a esso associato e viene rappresentato con una notazione esadecimale come U+12345678
- Separazione tra Basic Multilingual Plane (BMP) (caratteri da U+0000 a U+FFFF) e supplementary plane (astral plane)
- Supporto a caratteri composti (ad esempio il carattere Ä è rappresentabile tramite la coppia di caratteri U+0041 U+0308)

# Unicode

- *Unicode* è uno *standard implementativo* sviluppato a partire dagli anni '80 da un consorzio di industrie che realizzavano software multilingua
- Inizialmente sviluppato parallelamente e indipendentemente da ISO 10646, si è allineato con quest'ultimo nel 1991
- **Unicode *definisce degli standard (UTF-*/UCS-*) per l'encoding* dei caratteri dello UCS**
  - assunzione di lavoro (2003): i caratteri **UCS hanno codici rappresentabili con al massimo 21 bit** (UTF-16 non è in grado di rappresentare caratteri oltre U+10FFFF)

# UTF-32 / UCS-4

- UTF-32 (anche noto come UCS-4) è l'encoding più semplice: **ogni carattere viene rappresentato con 4 byte**
- Molto spesso questo tipo di codifica è **troppo onerosa**
  - i caratteri al di fuori del Basic Multilingual Plane sono talmente rari da poter essere ignorati per molte applicazioni
  - i caratteri all'interno del Basic Multilingual Plane sarebbero rappresentabili con 16 bit
- Codifica *non compatibile con US-ASCII*
- Encoding raramente adottato per le comunicazioni ed essenzialmente utilizzato solo all'interno delle librerie di gestione del testo

# UTF-16 / UCS-2

- UTF-16 (che estende il precedente UCS-2) è una **codifica a lunghezza variabile (2 o 4 byte)**:
  - 16 bit per i caratteri del Basic Multilingual Plane (U+0000-U+FFFF)
  - 32 bit per gli altri caratteri (U+10000-U+10FFFF)
- Rappresentazione piuttosto compatta
- **Encoding standard di Java e Windows**
- Codifica *non compatibile con US-ASCII*
- Due diverse versioni: UTF-16LE (little endian) e UTF-16BE (big endian)
  - ove non specificato, uso di U+FEFF come Byte Order Mark

# UTF-8

- UTF-8 è una **codifica a lunghezza variabile (da 1 a 4 byte) che supporta tutto il set di caratteri UCS**
  - ricordiamo che al momento si assume che i caratteri UCS abbiano codici rappresentabili con al massimo 21 bit
- La codifica UTF-8 associa a ciascun carattere una sequenza di byte di lunghezza variabile (da 1 a 4)
- Encoding standard di XML, di JSON e della maggior parte dei sistemi Unix moderni

# UTF-8 e retrocompatibilità

- UTF-8 è lo strumento messo a disposizione da UCS e Unicode per fornire un certo livello di **compatibilità con il passato**
  - i caratteri da U+0000 a U+007F sono identici ai caratteri della tabella US-ASCII
  - **UTF-8 usa un solo byte per rappresentare i caratteri da U+0000 a U+007F**, quelli della tabella US-ASCII
  - UTF-8 permette stringhe NULL-terminated
- Quindi, *l'encoding UTF-8 è del tutto compatibile con l'encoding US-ASCII* per il subset di caratteri da quest'ultimo supportati

# Encoding UTF-8

| Caratteri UCS | Rappresentazione binaria in UTF-8 |
|---|---|
| Da U+000000 a U+00007F | 0XXXXXXX |
| Da U+000080 a U+0007FF | 110XXXXX 10XXXXXX |
| Da U+000800 a U+00FFFF | 1110XXXX 10XXXXXX 10XXXXXX |
| Da U+010000 a U+10FFFF | 11110XXX 10XXXXXX 10XXXXXX 10XXXXXX |

- **Velocità di decodifica**: il primo byte dice quanti altri byte bisogna leggere per ottenere un carattere completo
- **Auto-sincronizzazione**: si riesce a capire facilmente e velocemente dove inizia un carattere guardando in un intorno di 3 byte dalla posizione corrente

Additional Material

# UTF-8 - Validazione

- **Attenzione!** Poiché in UTF-8 i caratteri hanno dimensione variabile, si deve fare particolare attenzione a **verificare che un buffer di memoria non contenga dei caratteri incompleti** prima di utilizzarlo!
  - con una `write` da un file o da una IPC potrei leggere in un buffer solo una parte di una stringa UTF-8, che non contiene tutti i byte che codificano l'ultimo carattere ricevuto!

- **Attenzione!** Oltre a verificare che un buffer non contenga caratteri incompleti bisogna **verificare che i dati rappresentati siano effettivamente validi**!
  - i caratteri tra U+D800 e U+DFFF sono riservati per UTF-16 e non possono essere usati in UTF-8
  - rappresentazioni UTF-8 di caratteri con un numero di bit significativi maggiore di 21 vanno scartate

Additional Material

# UTF-8 - Validazione

UTF-8

| | _0 | _1 | _2 | _3 | _4 | _5 | _6 | _7 | _8 | _9 | _A | _B | _C | _D | _E | _F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0_ | NUL 0000 0 | SOH 0001 1 | STX 0002 2 | ETX 0003 3 | EOT 0004 4 | ENQ 0005 5 | ACK 0006 6 | BEL 0007 7 | BS 0008 8 | HT 0009 9 | LF 000A 10 | VT 000B 11 | FF 000C 12 | CR 000D 13 | SO 000E 14 | SI 000F 15 |
| 1_ | DLE 0010 16 | DC1 0011 17 | DC2 0012 18 | DC3 0013 19 | DC4 0014 20 | NAK 0015 21 | SYN 0016 22 | ETB 0017 23 | CAN 0018 24 | EM 0019 25 | SUB 001A 26 | ESC 001B 27 | FS 001C 28 | GS 001D 29 | RS 001E 30 | US 001F 31 |
| 2_ | SP 0020 32 | ! 0021 33 | " 0022 34 | # 0023 35 | $ 0024 36 | % 0025 37 | & 0026 38 | ' 0027 39 | ( 0028 40 | ) 0029 41 | * 002A 42 | + 002B 43 | , 002C 44 | - 002D 45 | . 002E 46 | / 002F 47 |
| 3_ | 0 0030 48 | 1 0031 49 | 2 0032 50 | 3 0033 51 | 4 0034 52 | 5 0035 53 | 6 0036 54 | 7 0037 55 | 8 0038 56 | 9 0039 57 | : 003A 58 | ; 003B 59 | < 003C 60 | = 003D 61 | > 003E 62 | ? 003F 63 |
| 4_ | @ 0040 64 | A 0041 65 | B 0042 66 | C 0043 67 | D 0044 68 | E 0045 69 | F 0046 70 | G 0047 71 | H 0048 72 | I 0049 73 | J 004A 74 | K 004B 75 | L 004C 76 | M 004D 77 | N 004E 78 | O 004F 79 |
| 5_ | P 0050 80 | Q 0051 81 | R 0052 82 | S 0053 83 | T 0054 84 | U 0055 85 | V 0056 86 | W 0057 87 | X 0058 88 | Y 0059 89 | Z 005A 90 | [ 005B 91 | \ 005C 92 | ] 005D 93 | ^ 005E 94 | _ 005F 95 |
| 6_ | ` 0060 96 | a 0061 97 | b 0062 98 | c 0063 99 | d 0064 100 | e 0065 101 | f 0066 102 | g 0067 103 | h 0068 104 | i 0069 105 | j 006A 106 | k 006B 107 | l 006C 108 | m 006D 109 | n 006E 110 | o 006F 111 |
| 7_ | p 0070 112 | q 0071 113 | r 0072 114 | s 0073 115 | t 0074 116 | u 0075 117 | v 0076 118 | w 0077 119 | x 0078 120 | y 0079 121 | z 007A 122 | { 007B 123 | \| 007C 124 | } 007D 125 | ~ 007E 126 | DEL 007F 127 |
| 8_ | • +00 128 | • +01 129 | • +02 130 | • +03 131 | • +04 132 | • +05 133 | • +06 134 | • +07 135 | • +08 136 | • +09 137 | • +0A 138 | • +0B 139 | • +0C 140 | • +0D 141 | • +0E 142 | • +0F 143 |
| 9_ | • +10 144 | • +11 145 | • +12 146 | • +13 147 | • +14 148 | • +15 149 | • +16 150 | • +17 151 | • +18 152 | • +19 153 | • +1A 154 | • +1B 155 | • +1C 156 | • +1D 157 | • +1E 158 | • +1F 159 |
| A_ | • +20 160 | • +21 161 | • +22 162 | • +23 163 | • +24 164 | • +25 165 | • +26 166 | • +27 167 | • +28 168 | • +29 169 | • +2A 170 | • +2B 171 | • +2C 172 | • +2D 173 | • +2E 174 | • +2F 175 |
| B_ | • +30 176 | • +31 177 | • +32 178 | • +33 179 | • +34 180 | • +35 181 | • +36 182 | • +37 183 | • +38 184 | • +39 185 | • +3A 186 | • +3B 187 | • +3C 188 | • +3D 189 | • +3E 190 | • +3F 191 |
| 2-byte C_ | 192 | 193 | Latin 0080 194 | Latin 00C0 195 | Latin 0100 196 | Latin 0140 197 | Latin 0180 198 | Latin 01C0 199 | Latin 0200 200 | IPA 0240 201 | IPA 0280 202 | IPA 02C0 203 | accents 0300 204 | accents 0340 205 | Greek 0380 206 | Greek 03C0 207 |
| 2-byte D_ | Cyril 0400 208 | Cyril 0440 209 | Cyril 0480 210 | Cyril 04C0 211 | Cyril 0500 212 | Armeni 0540 213 | Hebrew 0580 214 | Hebrew 05C0 215 | Arabic 0600 216 | Arabic 0640 217 | Arabic 0680 218 | Arabic 06C0 219 | Syriac 0700 220 | Arabic 0740 221 | Thaana 0780 222 | N'Ko 07C0 223 |
| 3-byte E_ | Indic 0800 224 | Misc. 1000 225 | Symbol 2000 226 | Kana, CJK 3000 227 | CJK 4000 228 | CJK 5000 229 | CJK 6000 230 | CJK 7000 231 | CJK 8000 232 | CJK 9000 233 | Asian A000 234 | Hangul B000 235 | Hangul C000 236 | Hangul D000 237 | PUA E000 238 | Forms F000 239 |
| 4-byte F_ | SMP, SIP 10000 240 | 40000 241 | 80000 242 | SSP, SPUA C0000 243 | SPUA-B 100000 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |

La tabella (presa da Wikipedia) mostra che alcuni byte in una sequenza di dati codificata in UTF-8 sono sicuramente (rosso) o possibilmente (rosa) rappresentazioni di caratteri non validi.

Additional Material

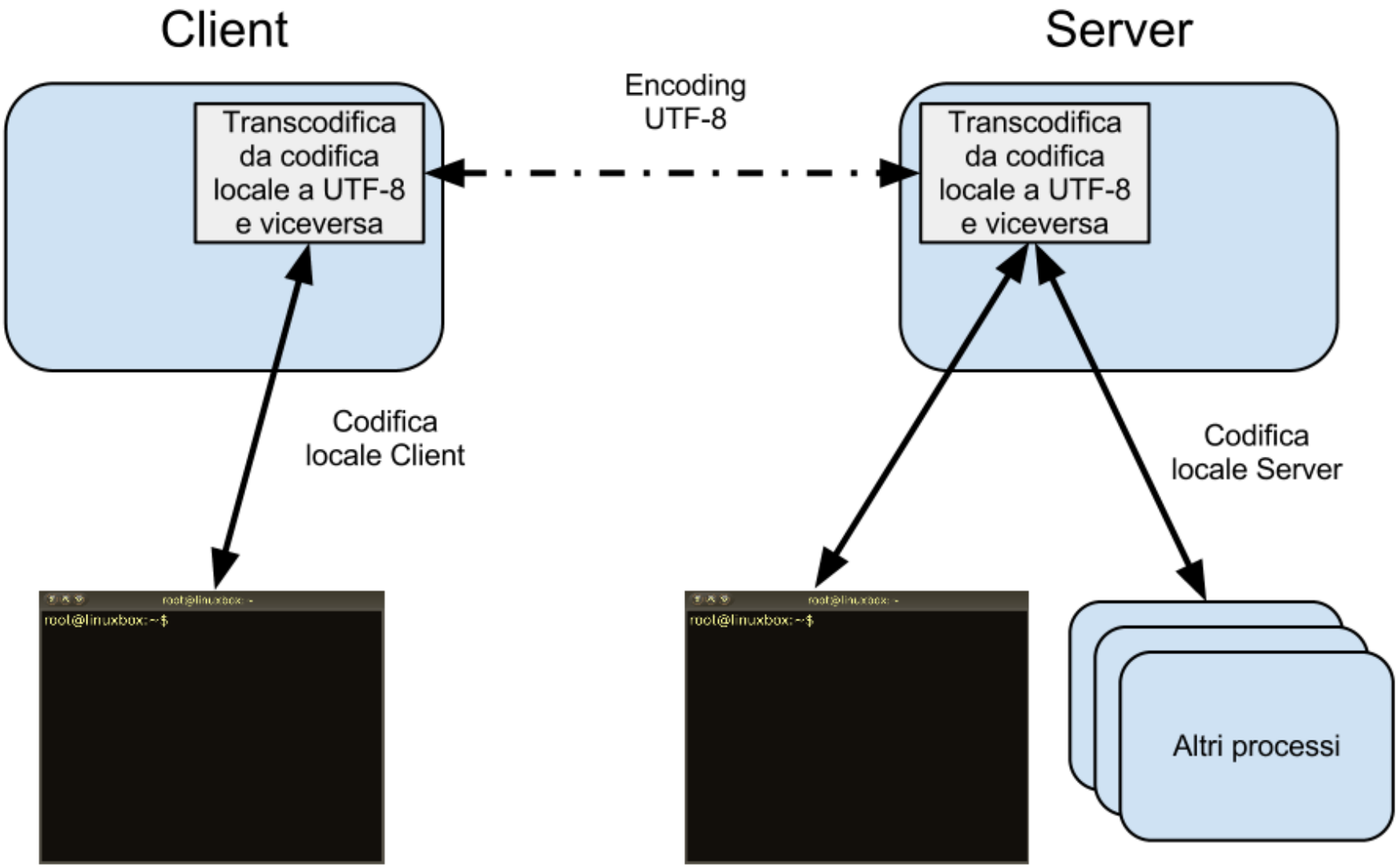
# UTF-8 – Lunghezza stringhe

- Chiaramente, **non possiamo più usare `strlen` per contare il numero di caratteri in una stringa UTF-8 NULL-terminated**
  - le funzioni di libreria `str*` sono state progettate assumendo una codifica US-ASCII che usa un byte per carattere
  - è necessario convertire la stringa UTF-8 a un array di caratteri UTF-32 (`wchar_t` su Unix) e usare funzioni `wcs*`
  - per il caso specifico di `strlen`, al posto di `strlen(str)` si può usare `mbstowcs(NULL,str,0)`
- Inoltre, poiché UCS ammette caratteri composti, non è detto che la lunghezza di una stringa equivalga al numero effettivo di caratteri che essa stamperà a video
  - uso di `wcswidth` per determinare il numero di colonne richieste per la stampa di un array di caratteri UTF-32

# Unicode e Applicazioni Distribuite

- È bene quindi realizzare applicazioni distribuite che comunichino con l'esterno utilizzando UTF-8
- Nel caso il formato di rappresentazione delle stringhe nella nostra piattaforma di sviluppo non sia UTF-8 (ad esempio su Windows), dobbiamo prevedere una transcodifica da UTF-8 alla codifica locale e viceversa

# Esempio


Client
Server
Encoding
UTF-8
Transcodifica
da codifica
locale a UTF-8
e viceversa
Transcodifica
da codifica
locale a UTF-8
e viceversa
Codifica
locale Client
Codifica
locale Server
root@linuxbox:~$
root@linuxbox:~$
Altri processi

# Transcodifica

Per la transcodifica tra UTF-8 e codifica locale:

- in **Java** si usano le funzioni di transcodifica fornite da InputStreamReader e OutputStreamWriter (il secondo parametro del costruttore specifica la codifica "esterna")
- in **Unix/C** lo standard è rappresentato dalla funzione *iconv* fornita dalle librerie di sistema, che però è piuttosto difficile da utilizzare. È pertanto consigliabile valutare l'uso di librerie di più alto livello, come utf8proc https://julialang.org/utf8proc, *glib* http://www.gtk.org, o addirittura ICU http://site.icu-project.org/.

# Esempi

- Validazione di una stringa UTF-8 a partire da buffer NULL-terminated con glib:

```
#include <glib.h>

/* mi assicuro che il buffer sia NULL-terminated */
char buffer[4096];
memset(buffer, 0, sizeof(buffer));
read(fd, buffer, sizeof(buffer)-1);

if (g_utf8_validate(buffer, NULL, NULL)) {
  /* stringa valida */
} else {
  /* stringa non (interamente) valida, posso provare
     a sanitizzarla o rigettarla */
}
```

Additional Material

# Esempi

- Estrazione sub-stringa UTF-8 valida da un buffer (non necessariamente NULL-terminated) con glib:

```
#include <glib.h>
#include <string.h>

char buff[4096]; int used_buf; char *valid_upto;

used_buf = read(fd, buffer, sizeof(buffer));
if (g_utf8_validate(buff, used_buf, &valid_upto)) {
  /* tutta la stringa è valida */
}
else if (valid_upto > buff) {
  /* estraggo porzione di stringa valida e lascio quella non
     valida nel buffer */
  size_t valid_bytes = valid_upto - buff;
  /* devo ricordarmi di deallocare la stringa con free */
  char *valid_str = strndup(buff, valid_bytes);
  memmove(buff, valid_upto, used_buf - valid_bytes);
  used_buf -= valid_bytes;
}
```

Additional Material

# Esempi

- Sanitizzazione di testo UTF-8 in un buffer (non necessariamente NULL-terminated) con glib:

```
#include <glib.h>

char buff[4096];
int used_buf;
char *sanitized_str;

used_buf = read(fd, buffer, sizeof(buffer));

/* ricordarsi di deallocare la stringa con free */
sanitized_str = g_utf8_make_valid(buff, used_buf);
```

**Attenzione!** La sanitizzazione dell'input è un'operazione molto delicata dal punto di vista della sicurezza! Lasciarla a una funzione di libreria di cui non si conosce perfettamente l'implementazione potrebbe non essere una buona idea.

Additional Material

# Per approfondire

- Per ulteriori informazioni su Unicode e UTF-8 si vedano:
  - http://www.joelonsoftware.com/articles/Unicode.html
  - http://utf8everywhere.org/
  - http://www.cl.cam.ac.uk/~mgk25/unicode.html
  - http://hackaday.com/2013/09/27/utf-8-the-most-elegant-hack/
  - https://eev.ee/blog/2015/09/12/dark-corners-of-unicode/
  - http://nullprogram.com/blog/2017/10/06/
  - http://bjoern.hoehrmann.de/utf-8/decoder/dfa/

# Dati strutturati

- È possibile scambiare **dati di tipo strutturato anche al di sopra di protocolli di tipo testuale**, utilizzando standard come XML e JSON
- Di solito, la perdita di performance legata alla trasformazione da rappresentazione binaria a rappresentazione testuale (e viceversa) è più che ripagata dalla flessibilità, dalla robustezza e dalla facilità di debug dei protocolli testuali

# XML

- XML è un linguaggio di descrizione specializzabile per settori specifici (ovverosia un metalinguaggio)
- Standardizzato da W3C, è molto utilizzato anche al di fuori del Web
- Tecnologia sviluppata in ottica machine-oriented, per facilitare la generazione automatica di codice che valida e/o manipola tipi di dati strutturati rigorosamente definiti
- XML è usato sia per la rappresentazione di dati e messaggi scambiati che per la definizione del loro formato

# Esempio XML

```
<? xml version=“1.0” encoding=“utf-8” ?>
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```

# XML Schema

- XML Schema è un linguaggio derivato da XML che consente di definire tipi di documento XML contenenti tipi di dati con strutture complesse o non regolari
- XML Schema permette di
  - **definire tipi di dati complessi**, basandosi su tipi di dati predefiniti
  - **specificare l'ordine** in cui gli elementi di un dato devono comparire
  - **definire delle regole** che specificano il numero di volte che ciascun elemento può o deve comparire
- Uso di XML Namespace per evitare ambiguità

# Esempio XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Person" type="PersonType"/>
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="firstName" type="xsd:string"
           minOccurs="0" MaxOccurs="1"/>
      <xsd:element name="lastName" type="xsd:string"
           minOccurs="1" MaxOccurs="1"/>
      <xsd:element name="age" type="AgeType" />
           minOccurs="0" MaxOccurs="1"/>
    ...
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="AgeType">
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

# JSON

- JSON è un formato di rappresentazione dati particolarmente leggero e molto utilizzato nel Web
  - molto più compatto e significativamente più performante e facile da processare rispetto a XML
- Pensato per applicazioni Web 2.0 e per la manipolazione dei dati in JavaScript

# Esempio JSON

```
{
   "firstName": "John",
   "lastName": "Smith",
   "age": 25,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      {
         "type": "home",
         "number": "212 555-1234"
      },
      {
         "type": "fax",
         "number": "646 555-4567"
      }
   ]
}
```