



**Regione Emilia-Romagna**



Operazione Rif PA **2021-15946/RER**

approvata con DGR **1263/2021 del 02/08/2021**

Progetto n. **1** - Edizione n. **1**

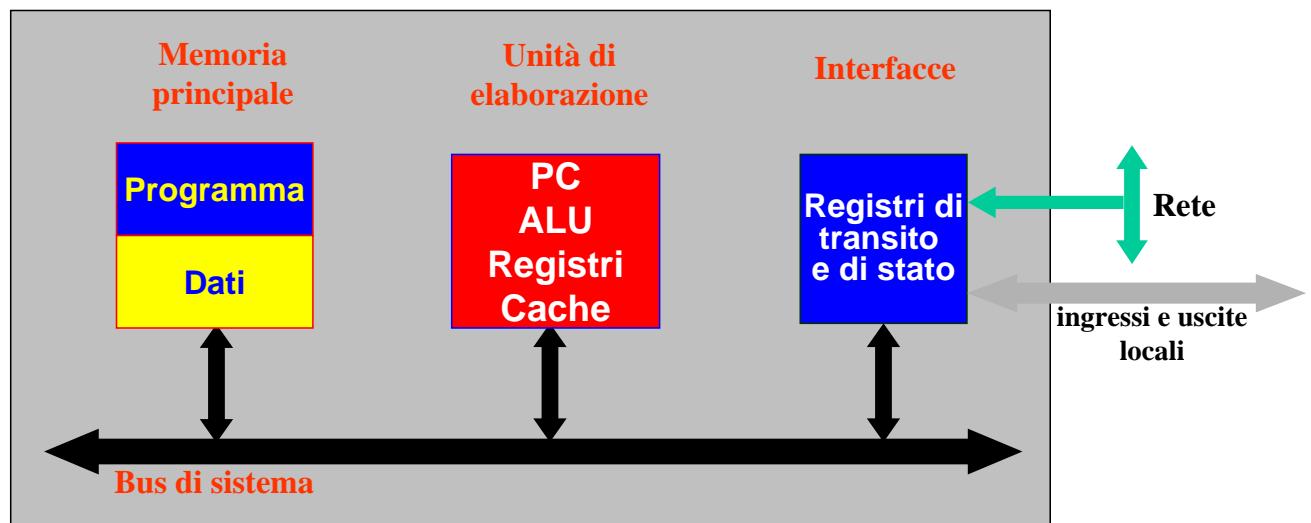
## **TECNICO PER LA PROGETTAZIONE E LO SVILUPPO DI APPLICAZIONI INFORMATICHE**

**MODULO: N. 6 Titolo: AMMINISTRAZIONE DI SISTEMI SERVER  
DURATA : 34 ORE DOCENTE: MARCO PRANDINI**

## **Sistema operativo Linux**

# Il calcolatore è una macchina digitale

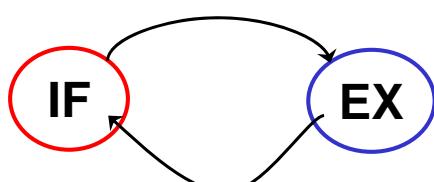
- All'interno di un calcolatore tutte le informazioni (es. dati e istruzioni) sono codificate in forma binaria, quindi:
  - nell'unità di elaborazione vengono elaborate variabili binarie
  - in memoria dati e istruzioni risiedono sotto forma di variabili binarie
- Il bus è il **supporto di interconnessione** tra i blocchi che costituiscono il calcolatore quindi:
  - sul bus transitano variabili binarie, pertanto i segnali del bus sono segnali digitali



15

## Modello di esecuzione del programma

- Il programma risiede in memoria ed è costituito da istruzioni codificate in forma binaria
- In memoria risiedono anche gli operandi delle istruzioni, cioè i dati elaborati e da elaborare
- Le istruzioni vengono eseguite in sequenza dalla CPU
- La CPU è una macchina sequenziale sincrona
- A livello di massima astrazione il suo automa ha due stati:
  - Stato in cui la CPU legge in memoria la prossima istruzione da eseguire (**INSTRUCTION FETCH o IF**)
  - Stato in cui la CPU esegue l'istruzione letta in IF (**EXECUTE o EX**)



- per funzionare la CPU ha bisogno almeno degli ingressi di **reset** e **clock** (stato iniziale)
- Se il reset non è attivo la CPU **perennemente** legge e esegue istruzioni, cambiando stato ad ogni impulso di clock
- la frequenza del clock è uno dei parametri che caratterizzano la CPU

16

# Accesso a dati e istruzioni da parte della CPU

## indirizzamento della memoria

- Durante l'esecuzione del programma la CPU legge le istruzioni e scambia dati e altre informazioni con la memoria principale attraverso il bus
- La memoria principale è vista dalla CPU come un vettore  $M[0..2^n-1]$  di  $2^n$  elementi detti celle o parole di memoria; questo vettore è detto “spazio di indirizzamento in memoria”
- L'indice  $i$  che identifica la cella  $M[i]$  si chiama **indirizzo della cella**
- L'accesso da parte della CPU all'informazione (istruzione o operando) residente in  $M[i]$  avviene *sempre mediante il rispettivo indirizzo*; pertanto sul bus di sistema devono transitare non solo i contenuti ma anche gli indirizzi delle celle di memoria (contemporaneamente o in successione)
- L'indirizzo di una cella in uno spazio di indirizzamento di  $2^n$  celle è una configurazione binaria di  $n$  bit; la dimensione dello spazio di indirizzamento di una CPU è uno dei parametri che ne caratterizza l'architettura
- Ogni cella è solitamente composta da 8 bit (un byte); in questo caso si dice che la memoria è organizzata in byte; il byte è quindi la più piccola quantità di memoria singolarmente indirizzabile

17

## Spazio di indirizzamento in memoria

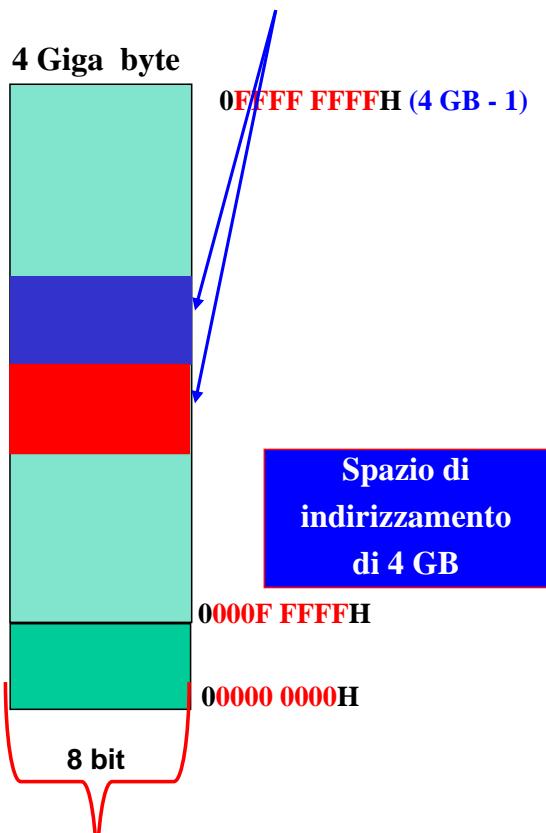


- Gli indirizzi si indicano solitamente in codice esadecimale
- la dimensione di uno spazio di indirizzamento si può esprimere in:
  - Kilobyte ( $1KB = 2^{10}$  Byte = 1024 B)
  - Megabyte ( $1MB = 1K KB = 2^{20}$  Byte = 1.048.576 B)
  - Gigabyte ( $1GB = 1K MB = 2^{30}$  Byte =  $1024 \cdot 2^{20} = \text{circa } 10^9$ )

18

# Spazio di indirizzamento in memoria

*Dispositivi fisici che realizzano la porzione di memoria*

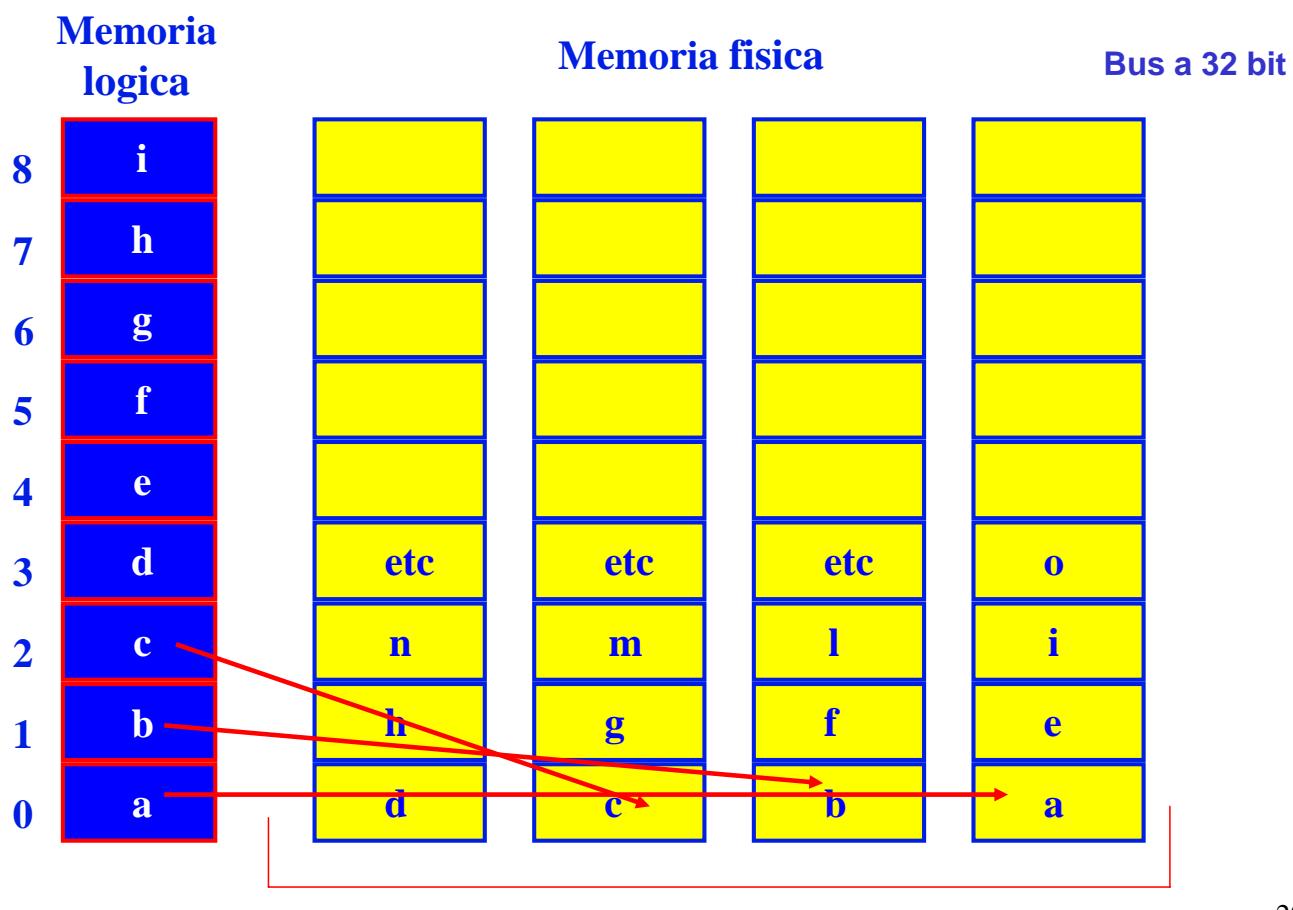


- **Esempi**

- l'8085 aveva uno spazio di indirizzamento di 64 KB (indirizzi di 16 bit)
- l'8086 aveva uno uno spazio di indirizzamento di 1 MB (indirizzi di 20 bit rappresentabili con 5 cifre esadecimale)
- il Pentium aveva uno spazio di indirizzamento di 4 GB (indirizzi di 32 bit rappresentabili con 8 cifre esadecimale)
- il Pentium IV indirizza 64 GB (indirizzi di 36 bit)
- I nuovi processori fino a 256GB
- Memorie a stato solido e memorie di massa

19

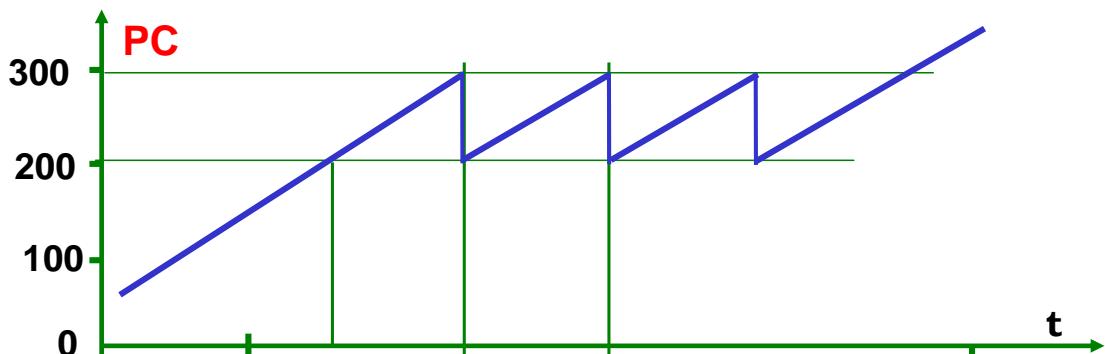
# Spazio di indirizzamento e parallelismo



20

# Il program counter e la sua dinamica durante l'esecuzione di un programma

- Per poter eseguire le istruzioni in sequenza la CPU dispone al suo interno di un contatore detto **Program Counter (PC)**
- Il **PC** viene incrementato ad ogni **FETCH**
- Il **PC** contiene *l'indirizzo in memoria* della prossima istruzione da leggere nella prossima fase di **FETCH**; in questo modo il **PC** individua la prossima istruzione da eseguire



Il grafico mostra qualitativamente la dinamica del PC quando il calcolatore esegue un loop di n iterazioni

La pendenza delle linee blu è un indicatore del numero di istruzioni eseguite nell'unità di tempo

21

## Accesso alle interfacce di ingresso/uscita Spazio di indirizzamento in I/O

- Durante l'esecuzione del programma anche i dati scambiati tra CPU e interfacce di ingresso/uscita transitano per il bus
- Così come i dispositivi di memoria, anche le interfacce di ingresso/uscita sono mappate in uno spazio di indirizzamento
- Le interfacce di I/O possono essere mappate in uno spazio distinto da quello della memoria oppure nello stesso; in quest'ultimo caso si dice che l'I/O è mappato in memoria (**memory mapped I/O**)
- Lo spazio di indirizzamento in I/O è solitamente più piccolo dello spazio di indirizzamento in memoria; es: nelle architetture Intel IA16 e IA32 lo spazio di indirizzamento in I/O è rimasto sempre di 64 KB
- Anche se i due spazi di indirizzamento sono distinti, i segnali del bus che portano l'indirizzo sono comuni (i segnali che portano gli indirizzi di I/O sono un sottoinsieme di quelli che portano gli indirizzi negli accessi alla memoria)
- La distinzione tra i due spazi di indirizzamento viene affidata a appositi segnali del bus



22

# Trasferimenti di informazioni sul bus comandati dalla CPU

- La CPU può comandare i seguenti trasferimenti di informazioni sul bus:
  - trasferimenti da e verso dispositivi mappati nello spazio di indirizzamento in memoria; esempi:
    - ◊ lettura di istruzioni e dati (operandi delle istruzioni) dalla memoria
    - ◊ scrittura di risultati in memoria
  - trasferimenti da e verso dispositivi mappati nello spazio di indirizzamento in I/O; esempi:
    - ◊ lettura di dati dalle interfacce
    - ◊ scrittura di risultati sulle interfacce
- Per gestire correttamente questi trasferimenti il bus di sistema deve disporre dei tre gruppi di segnali descritti nel lucido seguente

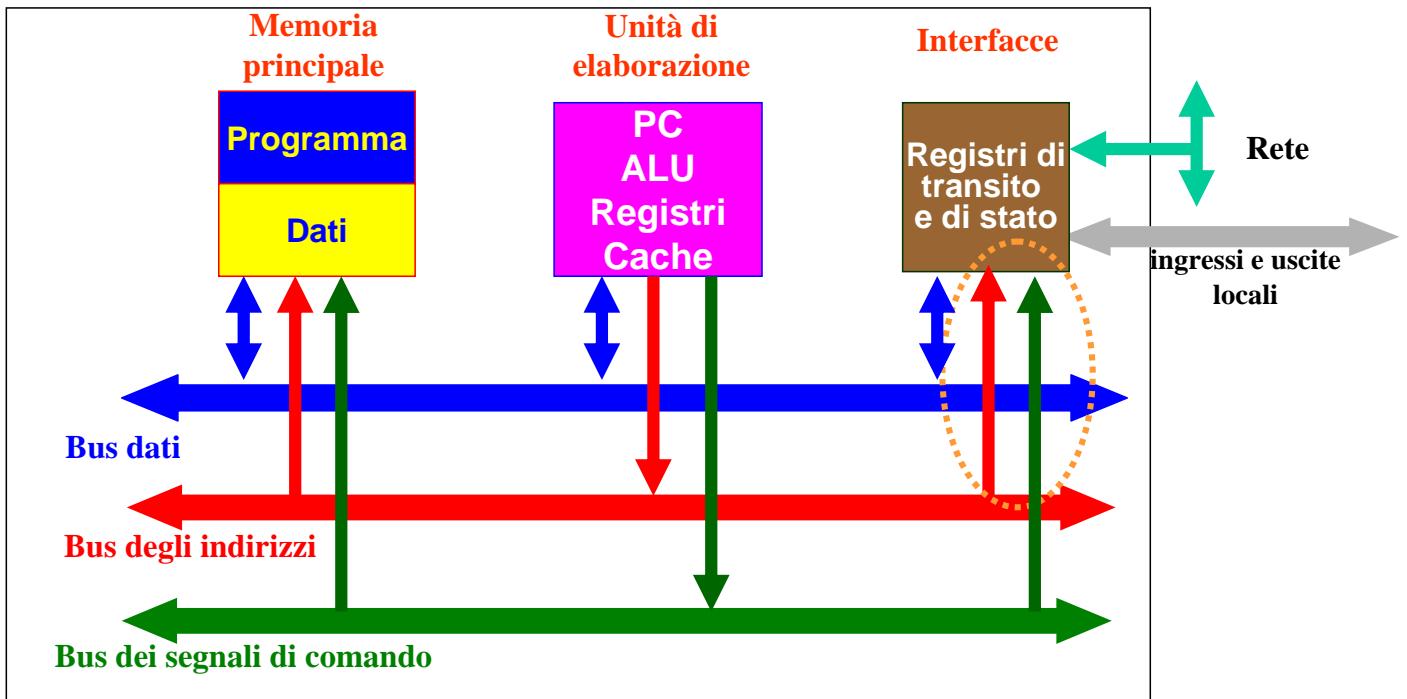
23

## Struttura del bus di sistema

- I segnali del bus di sistema sono suddivisi in tre gruppi:
  - bus dati, bus degli indirizzi e bus dei segnali di comando
- il bus dati è costituito dai segnali che portano istruzioni e operandi; il suo parallelismo è multiplo di un byte secondo una potenza di 2 (es. 1, 2, 4, 8, 16 byte); il parallelismo del bus dati è un altro parametro caratteristico dell'architettura della CPU. Il bus dati è identificato dal vettore di m bit **D[m-1..0]**
- il bus del Core è a 64 bit; quello dell'Itanium è a 128 Bit
- il bus degli indirizzi è costituito dai segnali che identificano la posizione delle informazioni trasferite nello spazio di indirizzamento a cui si intende accedere; il bus degli indirizzi è solitamente identificato dal vettore di n bit **A[n-1..0]**
- il bus dei segnali di comando è composto dai segnali che comandano i trasferimenti di dati sul bus; esempi (non esaustivo) di segnali di comando sono:
  - il comando con cui la CPU esegue una lettura nello spazio di indirizzamento in memoria (**MRDC#**)
  - il comando con cui la CPU esegue una scrittura in memoria (**MWRC#**)
  - il comando con cui la CPU esegue una lettura nello spazio di indirizzamento in I/O (**IORDC#**)
  - il comando con cui la CPU esegue una scrittura in I/O (**IOWRC#**)

24

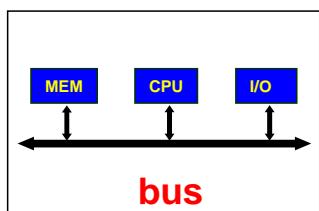
# Bus dati, bus degli indirizzi e bus dei segnali di comando



In questo schema a blocchi la CPU genera i segnali di **indirizzo** e **di comando** per la memoria e le interfacce

25

## Agenti master e slave sistemi a singolo master



- Gli agenti del bus si suddividono in:
  - agente master
  - agente slave
  - agenti master/slave

- Si chiama agente del bus (bus agent) un modulo che si affaccia al bus
- Si chiama agente master un agente che genera indirizzi e segnali di comando; per i master indirizzi e comandi sono segnali di uscita
- Si chiama agente slave un agente indirizzato dal master; per gli agenti slave indirizzi e comandi sono segnali di ingresso
- Un sistema in cui c'è un solo agente master è detto sistema a singolo master
- Nell'architettura del lucido precedente la CPU è l'unico agente master; memoria e interfacce sono agenti slave
- Un agente che alterna nel tempo il ruolo di master con quello di slave si chiama agente master/slave

26

# I cicli di bus

- Il trasferimento di un'informazione tra agenti del bus avviene con una sequenza di eventi detti nel loro insieme **ciclo di bus**
- In ogni ciclo di bus un agente master indirizza un agente slave
- Ad esempio, per leggere in memoria all'indirizzo i la CPU (master del bus) genera un ciclo di bus costituito dalla seguente sequenza di eventi:
  - la CPU mette sul bus degli indirizzi l'indirizzo nonché l'indicazione del numero di bytes che vuole trasferire a partire dall'indirizzo i
  - la CPU attiva un segnale di comando che identifica l'operazione desiderata (MRDC# per la lettura in memoria)
  - la CPU attende che la memoria indirizzata (slave) metta l'informazione desiderata sul **bus dati**
  - la CPU legge (cioè campiona) sul **bus dati** i segnali generati dalla memoria
- Questa sequenza di eventi si chiama **ciclo di lettura in memoria (memory read cycle)**

27

## Esempi di cicli di bus

- Altri esempi di cicli di bus sono i seguenti:
  - **ciclo di scrittura in memoria (memory write cycle)** caratterizzato dal segnale di comando **MWRC#**
  - **ciclo di lettura in I/O (I/O read cycle)** caratterizzato dal segnale di comando **IORDC#**
  - **ciclo di scrittura in I/O (I/O write cycle)** caratterizzato dal segnale di comando **IOWRC#**
- Durante il corso studieremo in dettaglio questi e altri cicli di bus

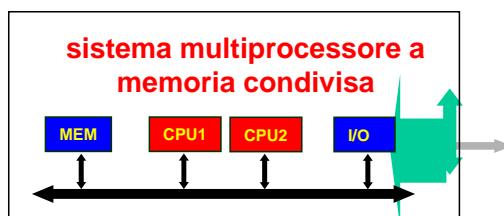
28

# Sistemi multimaster

- Un sistema in cui più bus master sono interconnessi a un bus si chiama “sistema multimaster”
- In questo caso il bus diventa una risorsa condivisa gestita a divisione di tempo: un solo bus master è attivo (**bus owner**) per ogni ciclo di bus
- Quando un master è attivo gli altri master possono:
  - osservare l'attività sul bus al fine di stabilire se ciò che avviene sul bus li riguarda (in questo caso l'agente si chiama anche **snooping agent**)
  - rimanere temporaneamente isolati dal bus
- E' necessario prevedere un meccanismo di arbitraggio che assegna in base a una regola di priorità il bus ai master che ne fanno richiesta
- I cicli di bus sono indivisibili; se un master inizia un ciclo, allora lo termina prima di cedere il bus a un altro master (questa è una regola generale con qualche eccezione non considerata in questo corso)

29

## Esempi di Sistemi Multimaster Architetture multiprocessor

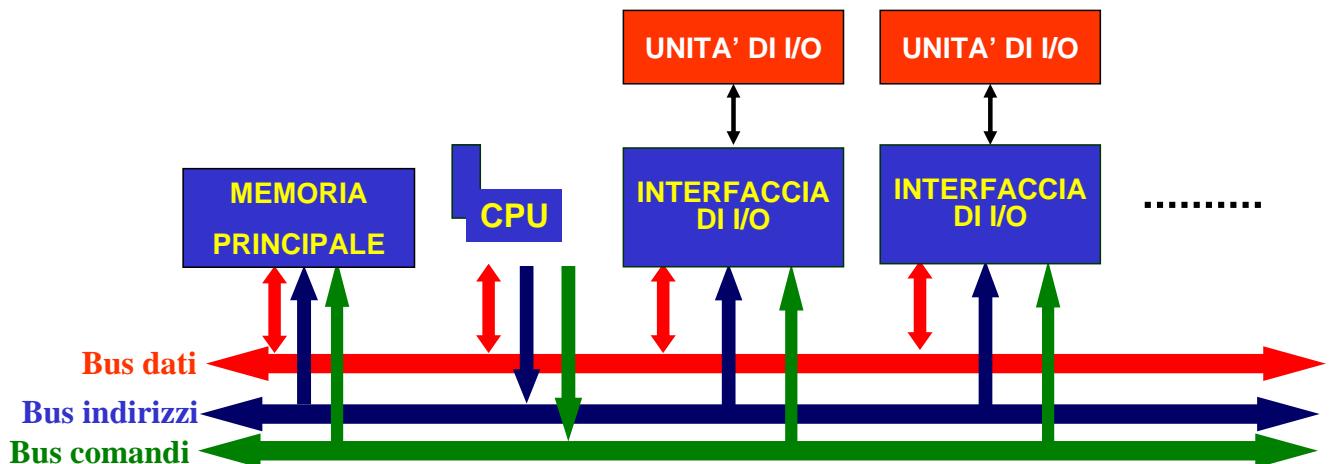


- Un sistema con più CPU affacciate al bus di sistema è un sistema multimaster
- Un sistema di questo tipo si chiama **sistema multiprocessore a memoria condivisa**
- L'architettura viene anche chiamata architettura **UMA (Uniform Memory Access)** a indicare che tutte le CPU accedono nello stesso modo alla memoria
- Non solo la memoria ma anche le interfacce vengono condivise da tutte le CPU

30

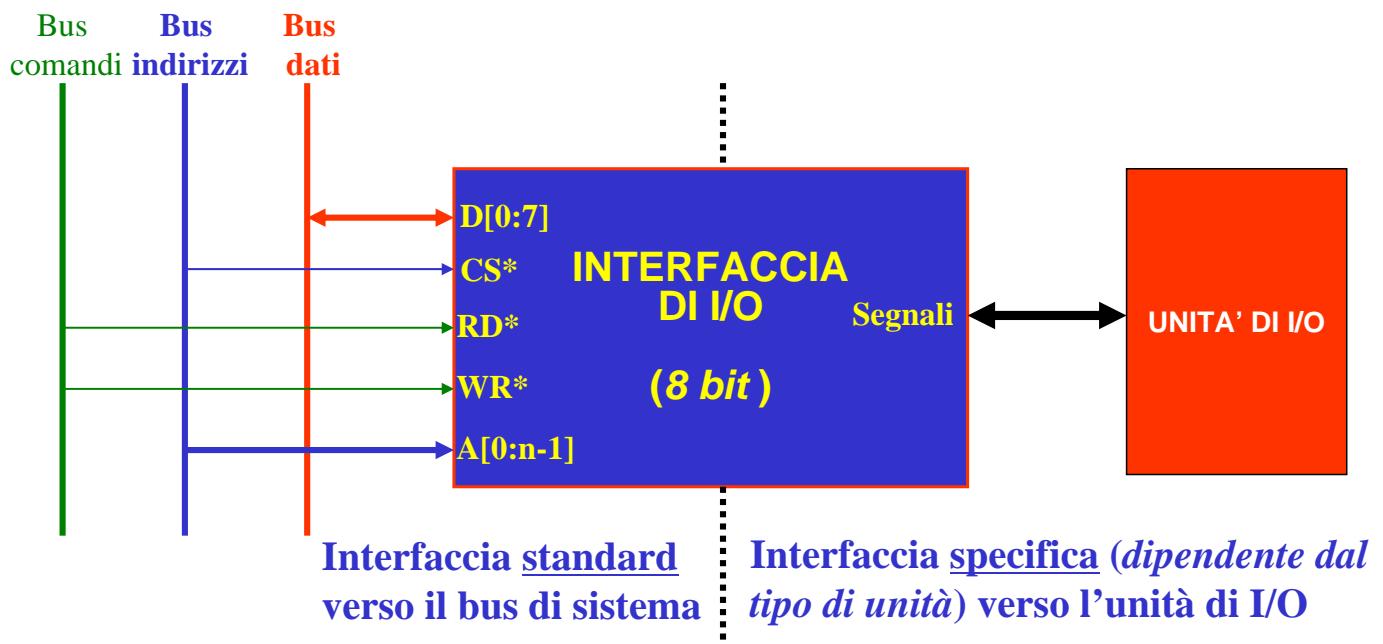
# Il sottosistema di I/O

- Il sottosistema di I/O consente la comunicazione fra il calcolatore ed il mondo esterno. Fanno parte del sottosistema i dispositivi (*Unità di I/O*) per la comunicazione uomo/macchina (video, terminali, stampanti), quelli utilizzati per la memorizzazione permanente delle informazioni (unità a disco, nastri magnetici), la rete.
- Tutte le Unità di I/O sono collegate al bus di sistema mediante dispositivi detti **Interfacce di I/O** (o anche *Periferiche, Controllori di I/O, Porte di I/O*)



2

## Interfacce di I/O



- L'interfaccia svolge una funzione di *adattamento* fra la modalità di trasferimento dei dati utilizzata all'interno del sistema (cicli di bus) e quella utilizzata dall'Unità di I/O.

5

# Interfacce di I/O

## • Buffer Dati

Strettamente associata alla funzionalità di *adattamento* fra calcolatore ed unità di I/O è la presenza all'interno dell'interfaccia di *registri di appoggio* (“*buffer*”) utilizzati nei trasferimenti dei dati da CPU ad Unità di I/O e viceversa.

Per inviare un dato all'Unità di I/O la CPU effettua una scrittura del dato su un buffer dell'interfaccia, da cui poi quest'ultima si occupa di trasferire il dato all'Unità di I/O.

Per prelevare un dato dall'Unità di I/O la CPU effettua una lettura da un buffer dell'interfaccia, su cui quest'ultima ha precedentemente appoggiato il dato proveniente dall'Unità di I/O (può essere anche connessa via tri-state).

## • Programmazione e Registri di Controllo

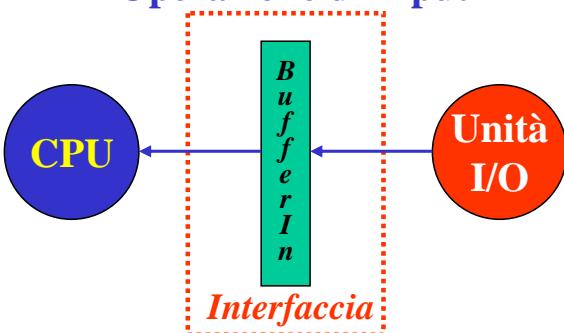
Le interfacce di I/O devono poter funzionare secondo un certo insieme di modalità differenti (es.: un'interfaccia per comunicazioni deve poter scambiare dati impiegando trame e frequenze differenti). Conseguentemente le interfacce di I/O sono tipicamente *programmabili*: sono presenti cioè al loro interno un certo numero di *registri di controllo* che vengono scritti dalla CPU all'atto dell'inizializzazione del dispositivo per impostare la modalità di funzionamento desiderata.

6

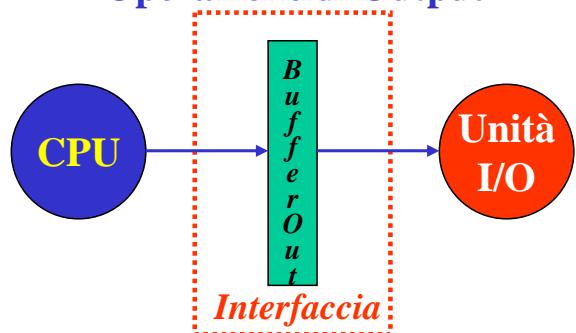
# Le modalità di gestione dell'I/O

Affrontiamo ora in modo più approfondito la problematica della sincronizzazione fra CPU (*programma in esecuzione sulla CPU*) ed Unità di I/O. Come già osservato, la funzionalità di sincronizzazione è a carico dell'interfaccia di I/O.

## Operazione di Input



## Operazione di Output

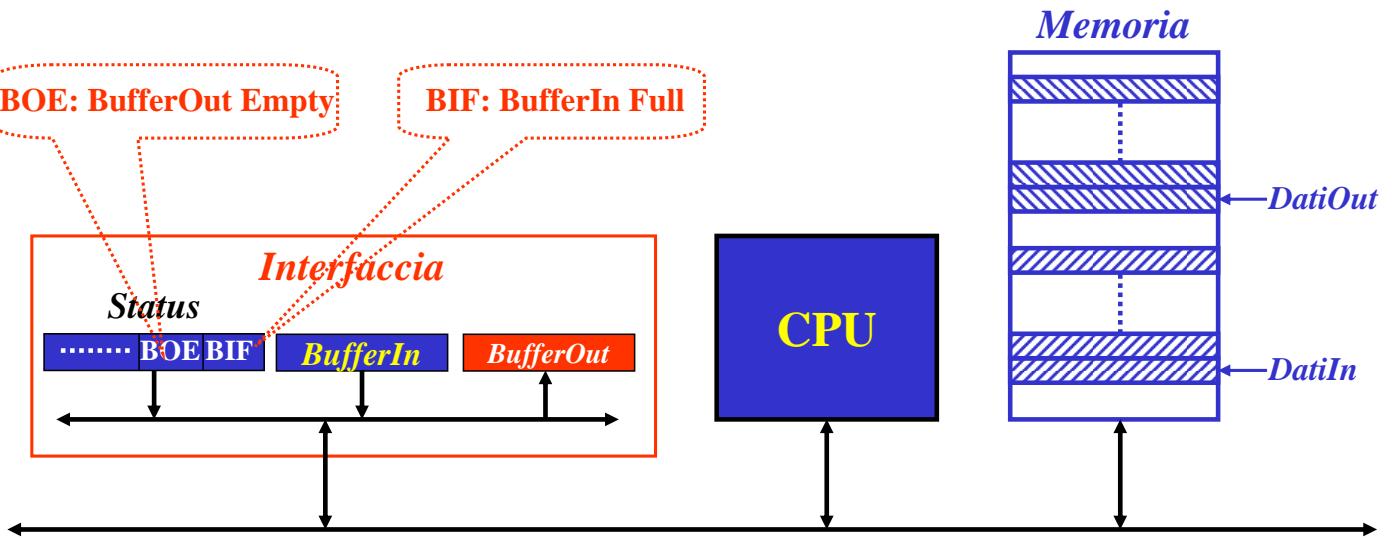


L'interfaccia deve comunicare alla CPU che l'Unità di I/O ha reso disponibile un nuovo dato su *BufferIn* (condizione di “**BufferIn Full**”)

L'interfaccia deve comunicare alla CPU che l'Unità di I/O ha prelevato da *BufferOut* il dato precedentemente inviato dalla CPU e quindi la CPU può inviare un nuovo dato (condizione di “**BufferOut Empty**”).

Le interfacce possono comunicare alla CPU le informazioni di sincronizzazione in due modi diversi, cui corrispondono due differenti modalità di gestione dell'I/O dette **Gestione a Polling** (Controllo di Programma) e **Gestione ad Interrupt** (Interruzione).

# Gestione a Polling



L'interfaccia rende disponibili le informazioni di sincronizzazione su 2 bit del registro di stato. La CPU, ogni volta deve eseguire un trasferimento, va prima a leggere il registro di stato (**BIF** oppure **BOE**) per verificare se l'interfaccia è pronta per il trasferimento. Se l'interfaccia è pronta la CPU esegue il trasferimento, viceversa legge in continuazione il registro di stato fino a che questo non segnala che l'interfaccia è pronta, quindi esegue il trasferimento.

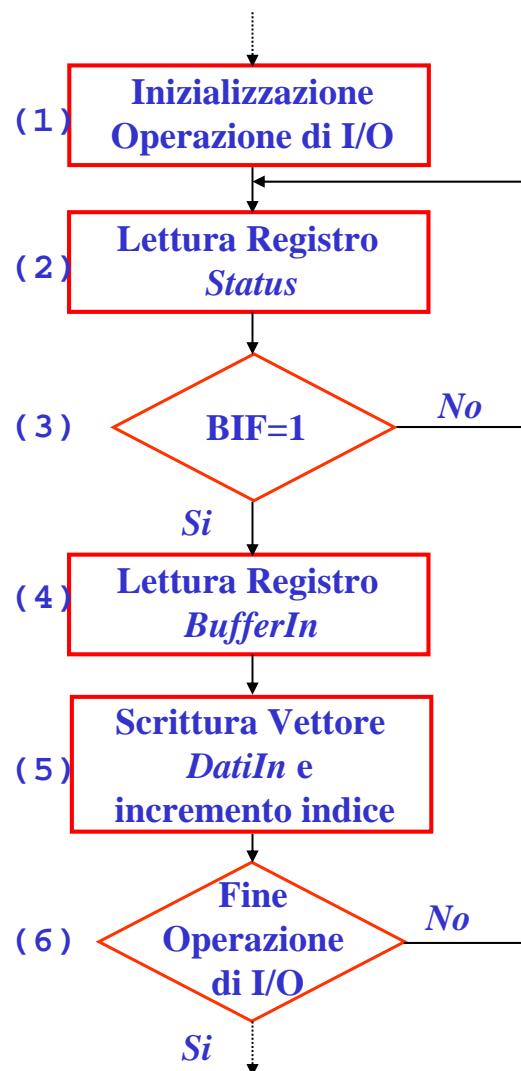
16

**Flusso di un  
programma  
che esegue una  
*Operazione di Input  
a Polling.***

### Codifica in Assembler

```

(1)          MOV SI, 0
(2) WaitData:    IN AL,
                Status
                TEST AL, 1
                JZ WaitData
                IN AL, BufferIn
                MOV DatiIn[SI], AL
                INC SI
                CMP SI, N
                JNE WaitData
  
```



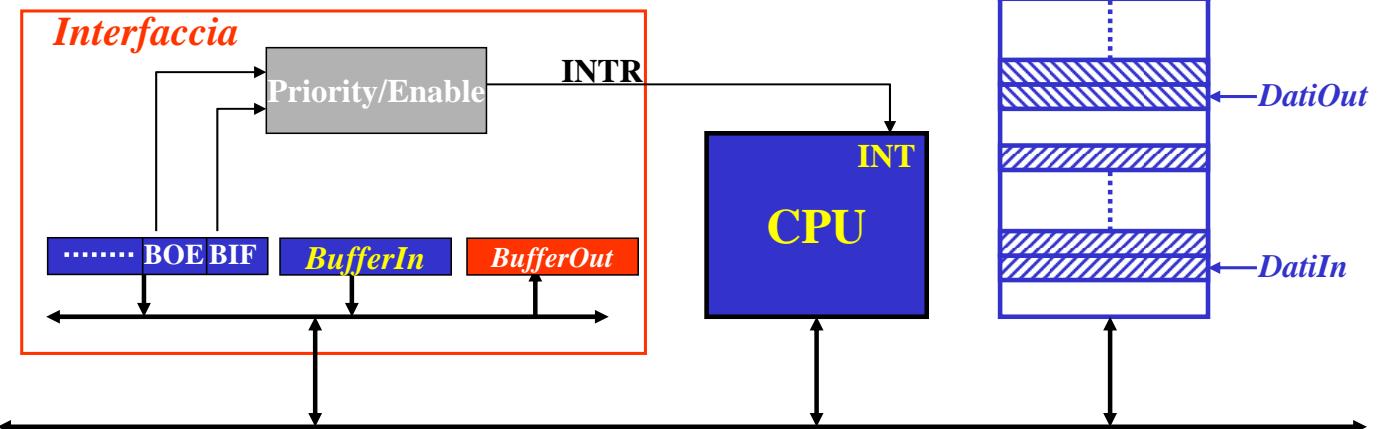
17

# Considerazioni sulla Gestione a Polling

- Nella Gestione a Polling la CPU è costantemente impegnata ad osservare lo stato della periferica al fine di eseguire un trasferimento non appena quest'ultima risulta pronta.
- Conseguentemente, la CPU non può svolgere nessuna altra attività durante gli intervalli di tempo che intercorrono fra due situazioni successive di “interfaccia pronta” (tempo necessario al trasferimento del dato fra interfaccia e Unità di I/O). Poiché le Unità di I/O sono tipicamente molto più lente della CPU, ciò implica una scarsa efficienza nell’impiego delle CPU.
- La Gestione ad Interrupt, che vedremo successivamente, ha proprio l’obiettivo di impiegare la CPU in modo più efficiente, consentendogli di eseguire delle istruzioni “utili” negli intervalli di tempo che intercorrono fra due situazioni successive di “interfaccia pronta”.
- Tuttavia, la Gestione a Polling è molto più semplice di quella ad Interrupt. Inoltre, quest’ultima presenta dei vantaggi reali solo quando è possibile strutturare il software in maniera tale che esistano delle attività utili che possono essere svolte simultaneamente all’Operazione di I/O.

18

## Gestione ad Interrupt



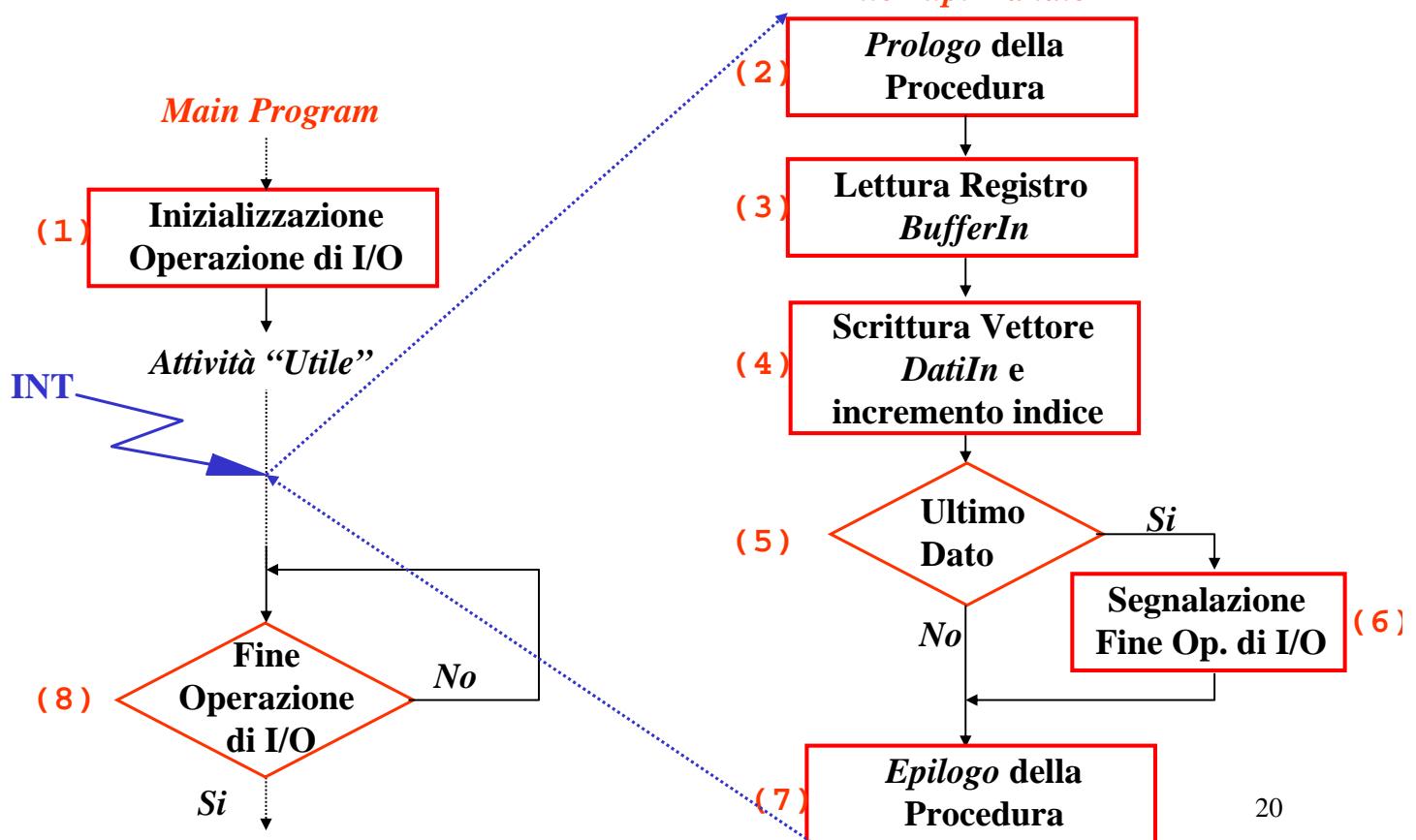
Quando l’interfaccia è pronta ad eseguire un nuovo trasferimento invia alla CPU una *richiesta di servizio* (detta *Interrupt*) mediante un apposito segnale (*INTR*, che per ora possiamo considerare connesso all’ingresso *INT* della CPU). All’atto della ricezione della *richiesta di servizio* la CPU interrompe il programma in esecuzione e trasferisce il controllo (secondo un meccanismo che studieremo successivamente) ad una apposita *procedura di servizio* (detta *Interrupt Handler*) che si occupa di effettuare il trasferimento del dato. Una volta effettuato il trasferimento la procedura di servizio restituisce il controllo al programma che era in esecuzione all’atto della ricezione dell’Interrupt.

Nella Gestione ad Interrupt la CPU può eseguire qualsiasi attività durante gli intervalli di tempo che intercorrono fra due situazioni successive di “interfaccia pronta” (cioè fra due interrupt).

19

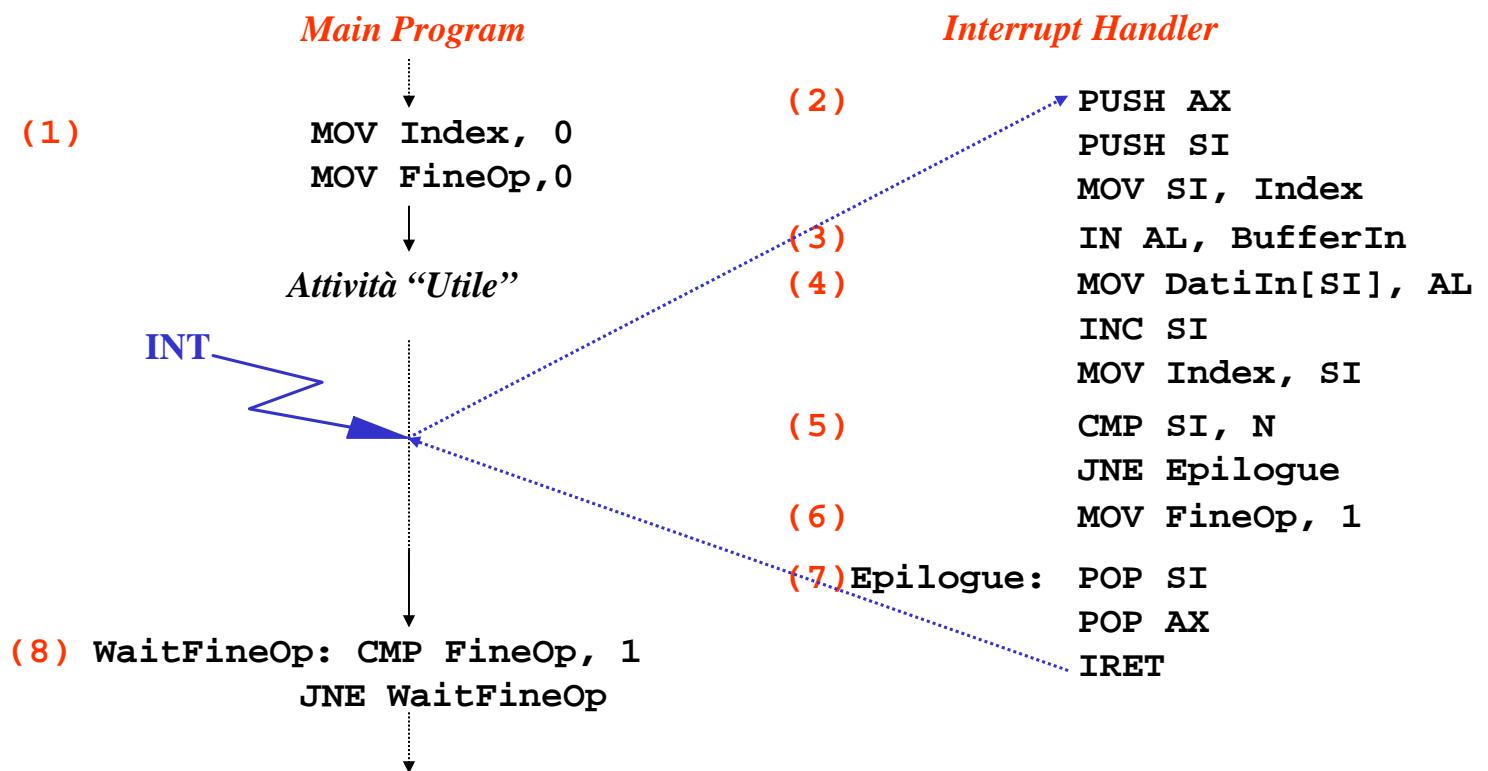
# Flusso di un programma che esegue una Operazione di Input ad Interrupt

*Interrupt Handler*



20

## Codifica in Assembler



21

# Considerazioni sulla Gestione a Interrupt

- La Gestione ad Interrupt consente un utilizzo efficiente della CPU: la CPU può essere impegnata in altre attività durante i “tempi morti” che intercorrono fra gli istanti in cui l’interfaccia è pronta ad eseguire un nuovo trasferimento.
- E’ opportuno sottolineare che se l’Operazione di I/O richiede il trasferimento di  $N$  gruppi di dati l’interfaccia genererà  $N$  richieste di servizio (cioè  $N$  Interrupt) e conseguentemente l’*Interrupt Handler* verrà eseguito  $N$  volte. Ad ogni esecuzione l’*Interrupt Handler* trasferisce un solo gruppo (che può corrispondere, ovviamente, a un solo dato)
- Poiché i trasferimenti sono eseguiti dall’*Interrupt Handler*, è necessario prevedere un meccanismo di sincronizzazione fra quest’ultimo ed il *Main Program*. Nell’esempio mostrato la sincronizzazione è effettuata tramite la variabile *FineOp*, azzerata dal *Main Program* nella fase di inizializzazione e messa a 1 dall’*Interrupt Handler* quando ha completato l’Operazione di I/O.
- La Gestione ad Interrupt è particolarmente adatta ai *Sistemi Multitasking*: un *task* (programma), dopo aver “lanciato” l’Operazione di I/O viene “sospeso” dal *Sistema Operativo* e la CPU viene assegnata ad un altro *task*. Il *task* sospeso viene poi rimesso in esecuzione quando l’Operazione di I/O è stata completata dall’*Interrupt Handler* (che fa parte del *kernel* del *Sistema Operativo*). <sup>22</sup>

## Che cos’è un Sistema Operativo (SO)?

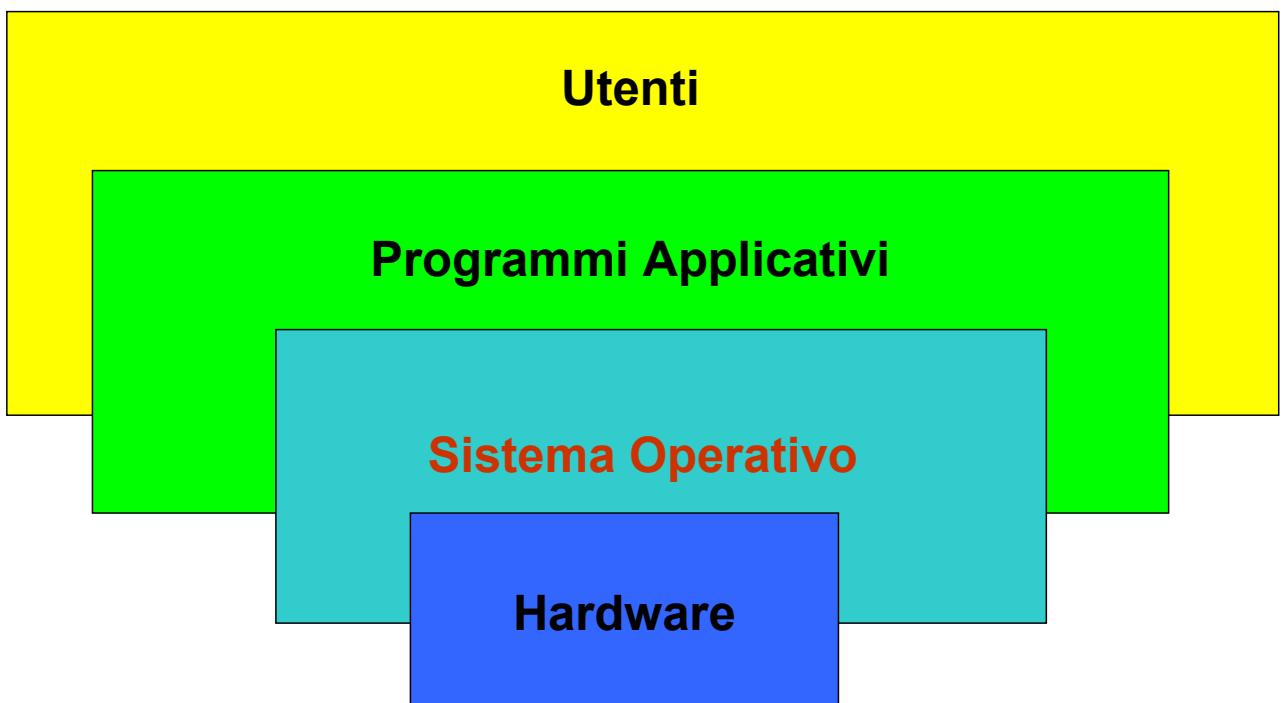
È un **programma** (o un insieme di programmi) che agisce come **intermediario tra l’utente e l’hardware** del computer:

- fornisce un **ambiente di sviluppo e di esecuzione** per i programmi applicativi
- fornisce una **visione astratta** dell’HW
- **gestisce** efficientemente le risorse del sistema di calcolo

# SO e Hardware

- SO interfaccia programmi applicativi o di sistema con le risorse HW:
  - CPU      - memoria volatile e persistente
  - dispositivi di I/O    - connessione di rete
  - dispositivi di comunicazione      - ...
- SO *mappa* le risorse HW in **risorse logiche**, accessibili attraverso interfacce ben definite:
  - *processi* (CPU)
  - *file system* (dischi)
  - *memoria virtuale* (memoria), ...

## Che cos'è un Sistema Operativo?



# Che cos'è un Sistema Operativo?

- Un programma che **gestisce risorse** del sistema di calcolo in modo **corretto ed efficiente** e le **alloca** ai programmi/utenti
- Un programma che innalza il **livello di astrazione** con cui utilizzare le **risorse logiche** a disposizione

## Aspetti importanti di un SO

- **Struttura**: come è organizzato un SO?
- **Condivisione**: quali risorse vengono condivise tra utenti e/o programmi? In che modo?
- **Efficienza**: come massimizzare l'utilizzo delle risorse disponibili?
- **Affidabilità**: come reagisce SO a malfunzionamenti (HW/SW)?
- **Estendibilità**: è possibile aggiungere funzionalità al sistema?
- **Protezione e Sicurezza**: SO deve impedire **interferenze** tra programmi/utenti. In che modo?
- **Conformità a standard**: portabilità, estendibilità, apertura

# Evoluzione SO

## Prima generazione (anni '50)

- linguaggio macchina
- dati e programmi su schede perforate

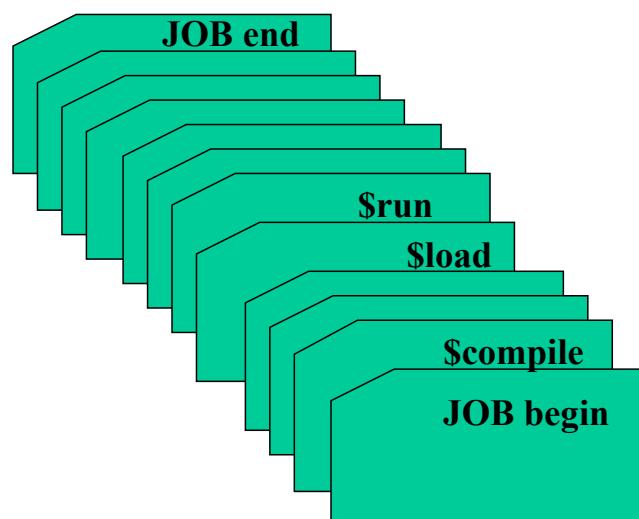
## Seconda generazione ('55-'65):

### sistemi batch semplici

- linguaggio di alto livello (fortran)
- input mediante schede perforate
- aggregazione di programmi in **lotti** (batch) con esigenze simili

## Sistemi batch semplici

**Batch:** insieme di programmi (*job*) da eseguire in modo sequenziale

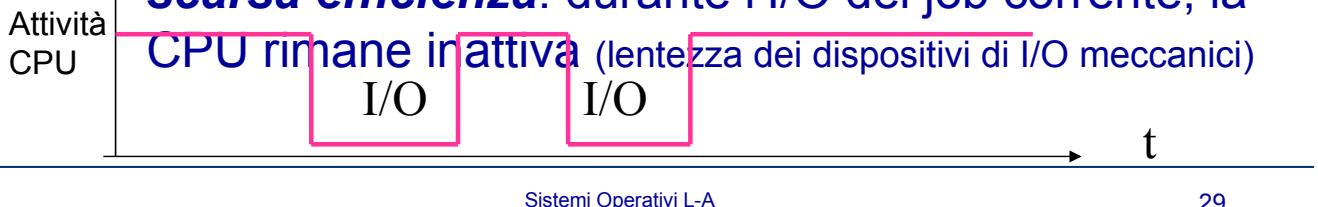


# Sistemi batch semplici

**Compito di SO (*monitor*):**  
***trasferimento di controllo*** da un job (appena terminato) al prossimo da eseguire

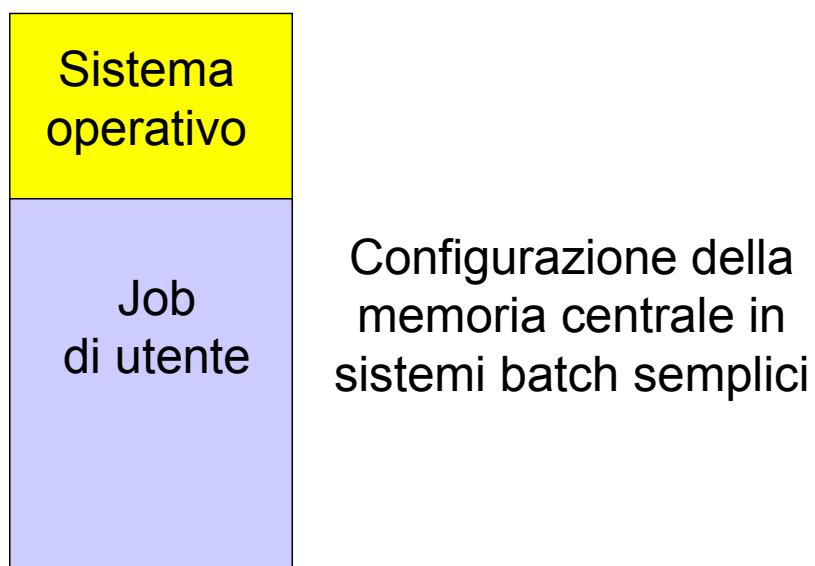
## Caratteristiche dei sistemi batch semplici:

- SO **residente in memoria** (monitor)
- **assenza di interazione** tra utente e job
- **scarsa efficienza**: durante l'I/O del job corrente, la CPU rimane inattiva (lentezza dei dispositivi di I/O meccanici)



# Sistemi batch semplici

In memoria centrale, ad ogni istante, è **caricato (al più) un solo job**:



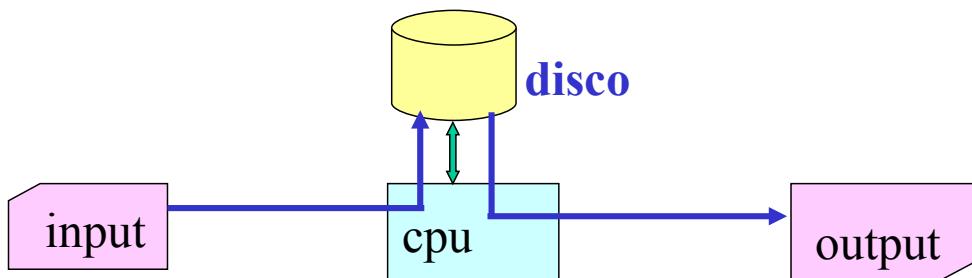
# Sistemi batch semplici

**Spooling (Simultaneous Peripheral Operation On Line):**

simultanetà di I/O e attività di CPU

disco viene impiegato come **buffer** molto ampio, dove

- leggere in anticipo i dati
- memorizzare temporaneamente i risultati (in attesa che il dispositivo di output sia pronto)
- caricare **codice e dati del job successivo**: -> possibilità di **sovraporre I/O** di un job **con elaborazione** di un altro job



# Sistemi batch semplici

## Problemi:

- finché il job corrente non è terminato, il **successivo non può iniziare l'esecuzione**
- se un job si **sospende** in attesa di un evento, la CPU rimane **inattiva**
- **non c'è interazione** con l'utente

# Sistemi batch multiprogrammati

**Sistemi batch semplici:** *I'attesa di un evento* causa inattività della CPU. Per evitare il problema

⇒ **Multiprogrammazione**

**Pool di job** contemporaneamente presenti su disco:

- SO seleziona un **sottoinsieme dei job** appartenenti al pool da **caricare in memoria centrale**
- mentre un job è in **attesa di un evento**, il sistema operativo **assegna CPU a un altro job**

# Sistemi batch multiprogrammati

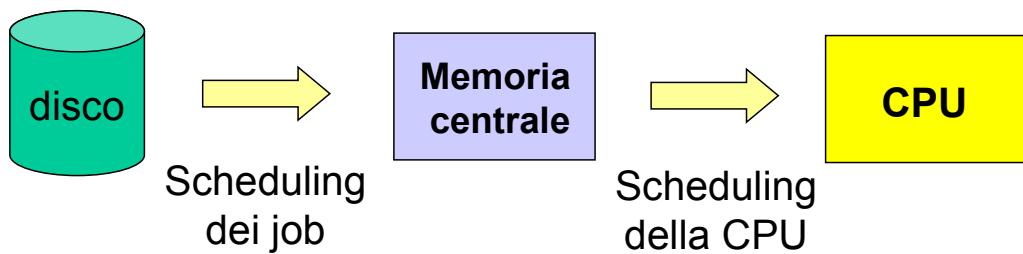
SO è in grado di **portare avanti** l'esecuzione di più job **contemporaneamente**

- Ad ogni istante:
  - **un solo job** utilizza la CPU
  - **più job**, appartenenti al pool selezionato e caricati in memoria centrale, attendono di acquisire la CPU
- Quando il job che sta utilizzando la CPU si **sospende in attesa di un evento**:
  - SO **decide** a quale job assegnare la CPU ed effettua lo scambio (**scheduling**)

# Sistemi batch multiprog.: scheduling

SO effettua delle scelte tra tutti i job

- quali job caricare in memoria centrale:  
**scheduling dei job** (*long-term scheduling*)
- a quale job assegnare la CPU: **scheduling della CPU** o (*short-term scheduling*)

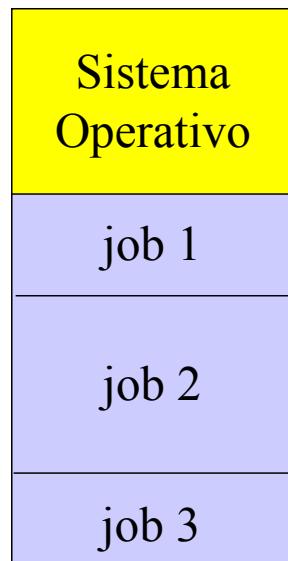


## Sistemi batch multiprogrammati



# Sistemi batch multiprogrammati

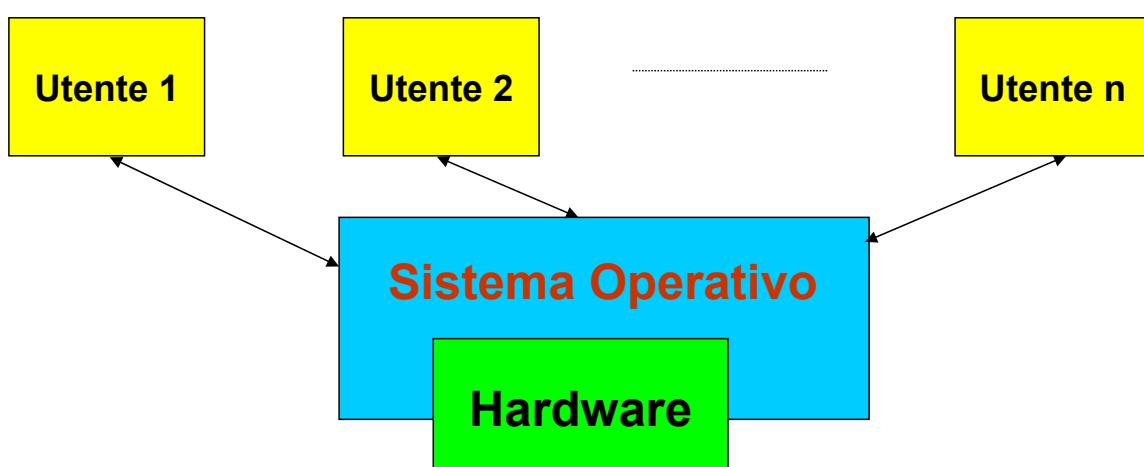
In memoria centrale, ad ogni istante, possono essere caricati più job:



Configurazione della **memoria centrale** in sistemi batch multiprogrammati

**Necessità di protezione**

# Sistemi time-sharing (Multics, 1965)



# Sistemi time-sharing

**Mutiutenza:** il sistema presenta ad ogni utente una **macchina virtuale completamente dedicata** in termini di

- utilizzo della CPU
- utilizzo di altre risorse, ad es. file system

**Interattività:** per garantire un'accettabile velocità di "reazione" alle richieste dei singoli utenti, SO **interrompe l'esecuzione** di ogni job dopo un intervallo di tempo prefissato (**quanto di tempo**, o **time slice**), assegnando la CPU a un altro job

# Sistemi time-sharing

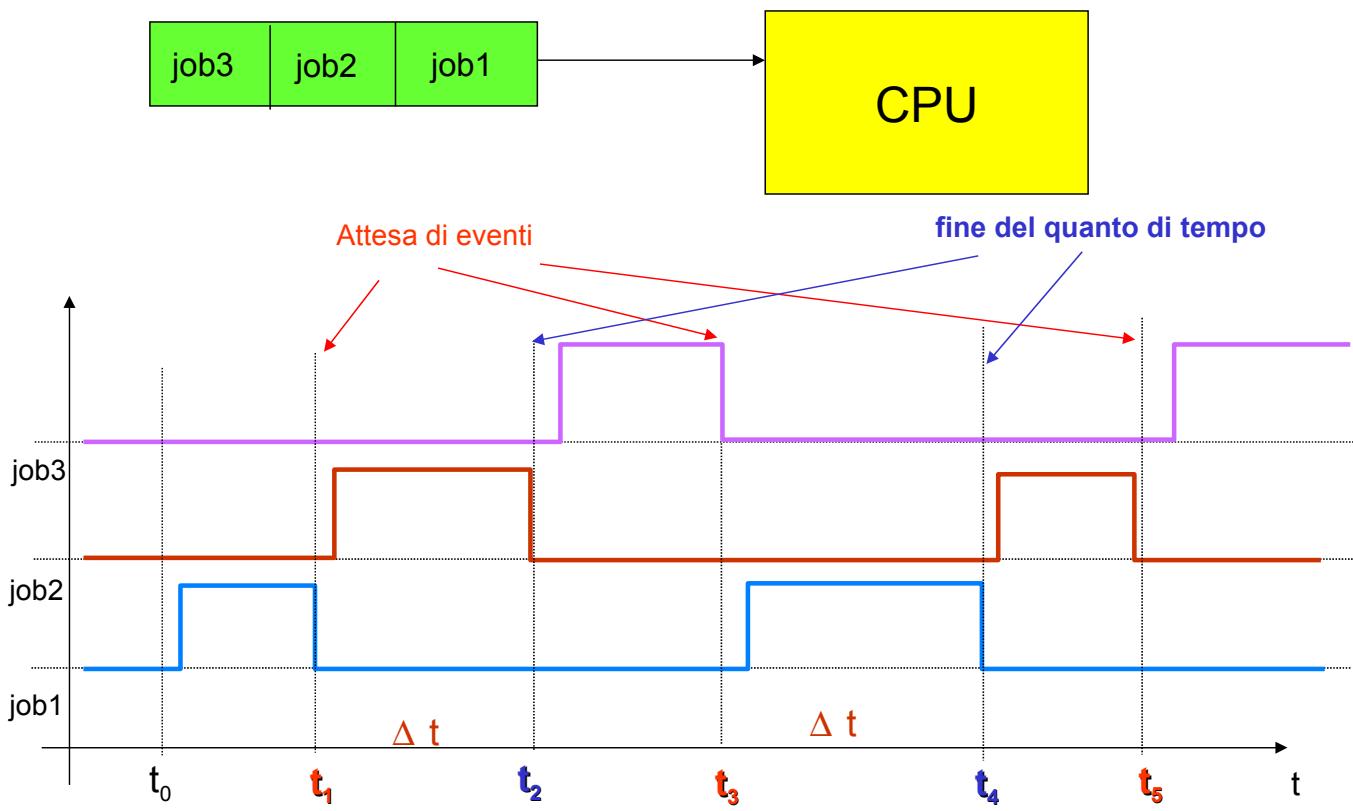
**Sono sistemi in cui:**

- attività della **CPU è dedicata a job diversi** che si alternano **ciclicamente** nell'uso della risorsa
- frequenza di commutazione della CPU è tale da fornire l'illusione ai vari utenti di una macchina completamente dedicata (**macchina virtuale**)

**Cambio di contesto** (*context switch*):

**operazione di trasferimento del controllo da un job al successivo**   costo aggiuntivo (overhead)

# Sistemi time-sharing



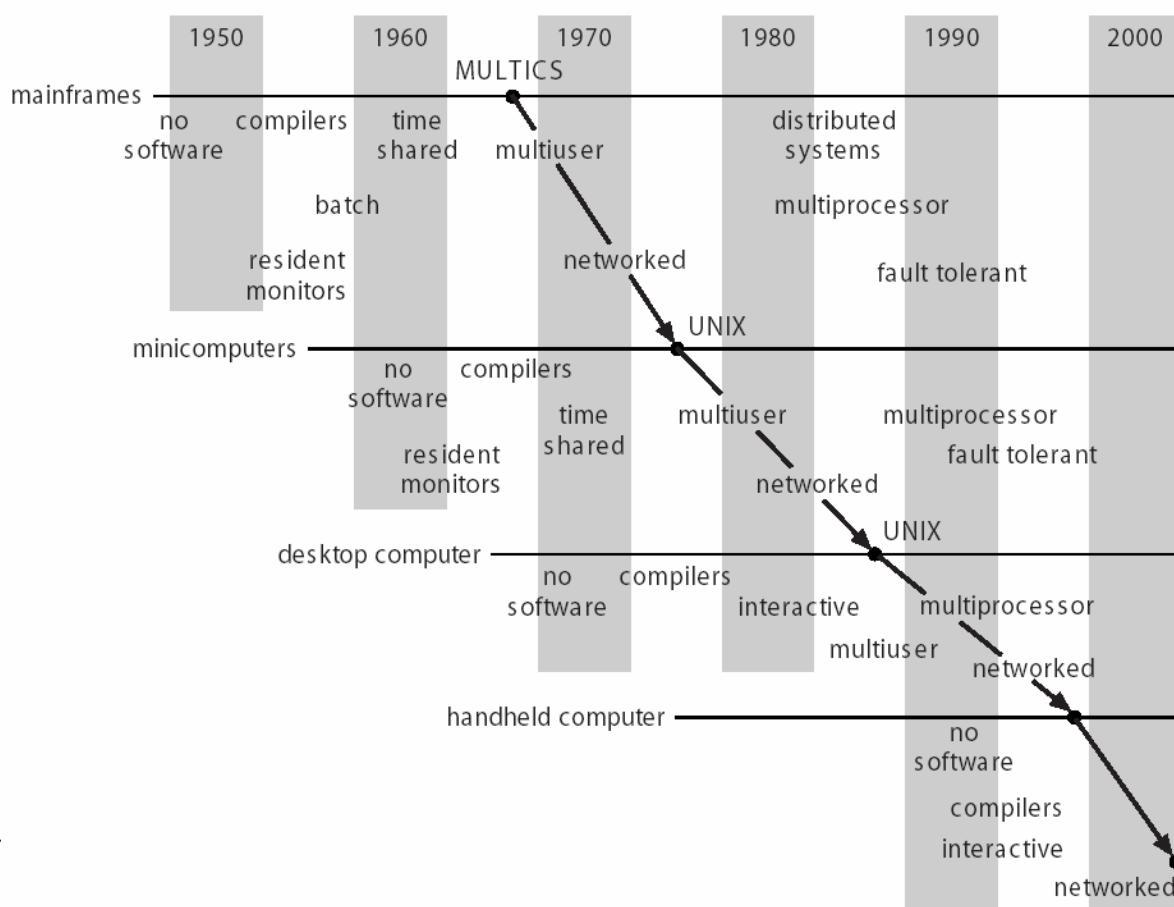
## Time-sharing: requisiti

- **Gestione/protezione** della memoria:
  - trasferimenti memoria-disco
  - **separazione degli spazi** assegnati ai diversi job
  - molteplicità job + limitatezza della memoria  
⇒ **memoria virtuale**
- **Scheduling** CPU
- **Sincronizzazione/comunicazione** tra job:
  - interazione
  - prevenzione/trattamento di blocchi critici (**deadlock**)
- **Interattività**: accesso **on-line** al **file system** per permettere agli utenti di accedere semplicemente a codice e dati

# Esempi di SO attuali

- **MSDOS**: monoprogrammato, monoutente
- **Windows 95/98, molti SO attuali per dispositivi portabili (Symbian, PalmOS)**: multiprogrammato (time sharing), tipicamente monoutente
- **Windows NT/2000/XP**: multiprogrammato, “multiutente”
- **MacOSX**: multiprogrammato, multiutente
- **UNIX/Linux**: multiprogrammato, multiutente

## Evoluzione dei concetti nei SO



# Hardware di un sistema di elaborazione

## Funzionamento a interruzioni:

- le varie *componenti* (HW e SW) del sistema interagiscono con SO mediante **interruzioni asincrone (interrupt)**
- ogni interruzione è causata da un **evento**, ad es.:
  - **richiesta di servizi al SO**
  - **completamento di I/O**
  - **accesso non consentito alla memoria**
- ad ogni interruzione è associata una **routine di servizio (handler)** per la **gestione dell'evento**

## Interruzioni hardware e software

- **Interruzioni hardware:**

dispositivi inviano segnali per richiedere l'esecuzione di servizi di SO



- **Interruzioni software:** programmi

*in esecuzione* possono generare interruzioni SW

- quando tentano l'esecuzione di **operazioni non lecite** (ad es. divisione per 0): **trap**
- quando richiedono l'esecuzione di servizi al SO - **system call**

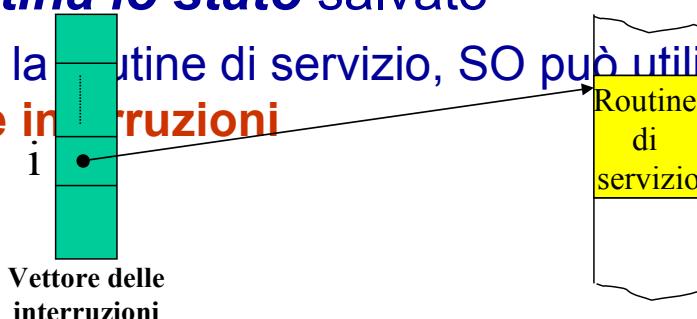


# Gestione delle interruzioni

Alla ricezione di un'**interruzione**, SO (lo vedremo in seguito nel dettaglio per il cambio di contesto):

- 1] interrompe la sua esecuzione => **salvataggio dello stato** in memoria (locazione fissa, stack di sistema, ...)
- 2] attiva la **routine di servizio all'interruzione** (handler)
- 3] **ripristina lo stato** salvato

Per individuare la routine di servizio, SO può utilizzare un **vettore delle interruzioni**



## Input/Output

Come avviene l'I/O in un sistema di elaborazione?

**Controller**: interfaccia HW delle periferiche verso il bus di sistema  
ogni controller è dotato di

- ❑ **un buffer** (ove **memorizzare temporaneamente** le informazioni da **leggere o scrivere**)
- ❑ alcuni **registri speciali**, ove **memorizzare le specifiche delle operazioni** di I/O da eseguire

# Input/Output

Quando un job richiede un'operazione di I/O (ad esempio, **lettura da un dispositivo**):

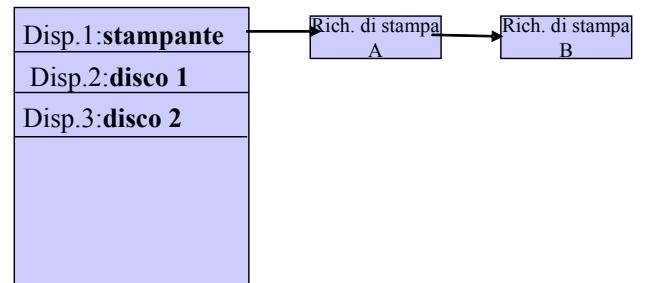
- CPU **scrive nei registri speciali** del dispositivo coinvolto le **specifiche dell'operazione** da eseguire
- controller esamina i registri e provvede a **trasferire i dati richiesti dal dispositivo al buffer**
- invio di **interrupt alla CPU** (completamento del trasferimento)
- CPU esegue l'operazione di I/O tramite la routine di servizio (**trasferimento dal buffer del controller alla memoria centrale**)

# Input/Output

## 2 tipi di I/O

- **Sincrono**: il **job viene sospeso** in attesa del completamento dell'operazione di I/O
- **Asincrono**: il sistema restituisce **immediatamente il controllo al job**
  - se necessario, funzionalità di blocco in attesa di completamento dell'I/O
  - possibilità di più I/O **pendenti**  
-> **tabella di stato dei dispositivi**

I/O asincrono = maggiore efficienza



# Virtualizzazione

## Virtualizzazione

Dato un sistema caratterizzato da un insieme di risorse (hardware e software), **virtualizzare il sistema** significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.

Ciò si ottiene introducendo **un livello di indirezione** tra la vista logica e quella fisica delle risorse.



Gli obiettivi della virtualizzazione possono essere diversi.

# Tecnologie di virtualizzazione

**Obiettivo:** disaccoppiare il comportamento delle risorse di un sistema di elaborazione offerte all'utente dalla loro realizzazione fisica.

## Esempi di virtualizzazione

**Virtualizzazione a livello di processo.** I sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una macchina virtuale (CPU, memoria, dispositivi) dedicata. La virtualizzazione è realizzata dal **kernel** del sistema operativo.

**Virtualizzazione della memoria.** In presenza di memoria virtuale, ogni processo vede uno spazio di indirizzamento di dimensioni indipendenti dallo spazio fisico effettivamente a disposizione. La virtualizzazione è realizzata dal **kernel** del sistema operativo.

## Altri Esempi di virtualizzazione

**Astrazione:** in generale un oggetto astratto (risorsa virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica):

- esibendo le proprietà significative per l'utilizzatore
- nascondendo i dettagli realizzativi non necessari.

Es: tipi di dato vs. rappresentazione binaria nella cella di memoria

Il **disaccoppiamento** è realizzato dalle operazioni (interfaccia) con le quali è possibile utilizzare l'oggetto.

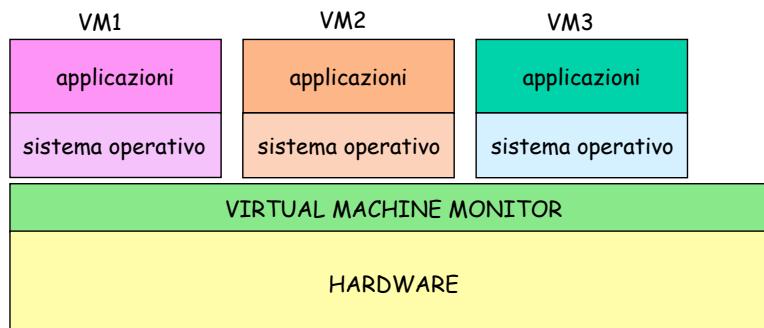
**Linguaggi di Programmazione.** La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura del sistema (S.O. e HW):

- Interpreti (esempio Java Virtual Machine)
- Compilatori

## Virtualizzazione di Sistema.

Una singola piattaforma hardware viene condivisa da più elaboratori virtuali (macchine virtuali o VM) ognuno gestito da un proprio sistema operativo.

Il disaccoppiamento è realizzato da un componente chiamato *Virtual Machine Monitor* (VMM, o *hypervisor*) il cui compito è consentire la **condivisione da parte di più macchine virtuali di una singola piattaforma hardware**. Ogni VM contiene un proprio sistema operativo, che definisce un ambiente di esecuzione distinto e isolato delle altre macchine virtuali, che consente l'esecuzione di applicazioni all'interno di esso.



Il VMM è il **mediatore unico** nelle interazioni tra le macchine virtuali e l'hardware sottostante, che garantisce:

- **isolamento** tra le VM
- **stabilità** del sistema

# Emulazione

Esecuzione di programmi compilati per una particolare architettura (e quindi un **particolare insieme di istruzioni**) su un sistema di elaborazione dotato di un **diverso insieme di istruzioni**.

- Vengono emulate interamente le singole istruzioni dell'architettura ospitata permettendo a sistemi operativi o applicazioni, pensati per determinate architetture, di girare, non modificati, su architetture completamente differenti.
- **Vantaggi:** interoperabilità tra ambienti eterogenei,
- **Svantaggi:** ripercussioni sulle performances (problemi di efficienza).
- L'approccio dell'emulazione ha seguito nel tempo due strade: **l'interpretazione e la ricompilazione dinamica.**

# Interpretazione

- Il modo più diretto per emulare è *interpretare*. L'interpretazione si basa sulla lettura di **ogni singola istruzione** del codice macchina che deve essere eseguito e sulla **esecuzione di più istruzioni sull'host virtualizzante** per ottenere semanticamente lo stesso risultato
- E' un metodo molto generale e potente che presenta una grande flessibilità nell'esecuzione perché consente di emulare e riorganizzare i meccanismi propri delle varie architetture. Vengono, ad esempio, normalmente utilizzate parti di memoria per salvare il contenuto dei registri della CPU emulata, registri che potrebbero non essere presenti nella CPU emulante.
- Produce un sovraccarico mediamente molto elevato poiché possono essere necessarie **molte istruzioni dell'host** per interpretare una singola istruzione sorgente.

# Compilazione dinamica

- Invece di leggere una singola istruzione del sistema ospitato, legge interi **blocchi di codice**, li analizza, li traduce per la nuova architettura **ottimizzandoli** e infine li mette in esecuzione.
- Il vantaggio in termini prestazionali è evidente. Invece di interpretare una singola istruzione alla volta, il codice viene **tradotto e ottimizzato**, utilizzando tutte le possibilità offerte dalla nuova architettura e messo in esecuzione.
- Parti di codice utilizzati frequentemente possono essere **bufferizzate** per evitare di doverle ricompilare in seguito.
- Tutti i più noti emulatori come **QEMU e Virtual PC per MAC (PowerPC)** utilizzano questa tecnica per implementare la virtualizzazione.

## QEMU

**Qemu** è un software che implementa un particolare sistema di emulazione che permette di ottenere **un'architettura nuova e disgiunta** in un'altra che si occuperà di ospitarla (convertire, ad esempio, le istruzioni da 32 bit a 64) **permettendo quindi di eseguire programmi compilati su architetture diverse**

- Questo software è conosciuto grazie alla sua velocità di emulazione ottenuta grazie alla tecnica della *traduzione dinamica*. È simile a Virtual PC, ma più veloce nell'emulazione delle architetture x86.

. **Qemu**, inizialmente, era un progetto che si prefiggeva di emulare solo il microprocessore x86 su un sistema GNULinux, allo scopo di poter eseguire in ambiente linux applicazioni windows (tramite Wine).

Ora, **Qemu** è in grado di emulare sistemi x86, AMD64, Power PC, MIPS e ARM.

## Virtual PC

- Software di emulazione che consentiva a computer con **sistema operativo Microsoft windows o Mac OSX** l'esecuzione di **sistemi operativi diversi**, come varie versioni di Windows o Linux, anche in contemporanea.
- L'emulatore ricrea in forma virtuale un ambiente di lavoro che riproduce quasi integralmente quello di un **PC basato su Intel**.
- è destinato soprattutto a consentire l'**uso di vecchie applicazioni** non più supportate dai moderni sistemi operativi.
- La successiva **introduzione dei processori Intel** nei computer Apple ha tolto interesse pratico all'utilizzo di Virtual PC da parte degli utenti Macintosh. (possibilità di *dual boot* o di uso di software di virtualizzazione, come Parallels, Virtual Box, Vmware Fusion).

## MAME

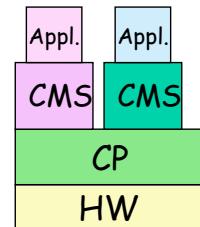
- (acronimo per **Multiple Arcade Machine Emulator**) è un software per personal computer sviluppato inizialmente per MS-DOS e in seguito per quasi tutte le macchine e sistemi operativi in circolazione, in grado di **caricare ed eseguire il codice binario originale delle ROM** dei videogiochi da bar (*arcade*), **emulando l'hardware tipico di quelle architetture**.
- Facendo leva sull'enorme potenza di calcolo degli attuali PC rispetto ai primordiali processori per video giochi dell'epoca, la VM può tranquillamente operare secondo il **paradigma dell'interpretazione** senza compromettere in modo significativo l'esperienza di gioco, almeno per i video giochi più vecchi che non facevano un uso massiccio della grafica.

# Cenni storici

La virtualizzazione non è un concetto nuovo:

## ■ Anni 60: IBM

- CP/CMS sistema suddiviso in 2 livelli:
  - CP (control program): esegue direttamente sull'HW svolgendo il ruolo di VMM, offrendo molteplici interfacce allo strato superiore
  - CMS (conversational monitor system): sistema operativo, interattivo e monoutente, replicato per ogni macchina virtuale
- VM/370: evoluzione di CP/CMS, caratterizzata dalla possibile eterogeneità di sistemi operativi nelle diverse macchine virtuali (IBM OS/360, DOS/360).



## ■ Anni 70:

- sistemi operativi multitasking

## ■ Anni 80:

- Evoluzione della tecnologia (microprocessori, reti)
- Crollo del costo dell'hw
- Migrazione da architetture basate su mainframe verso minicomputer e PC

## ■ Anni 80-90:

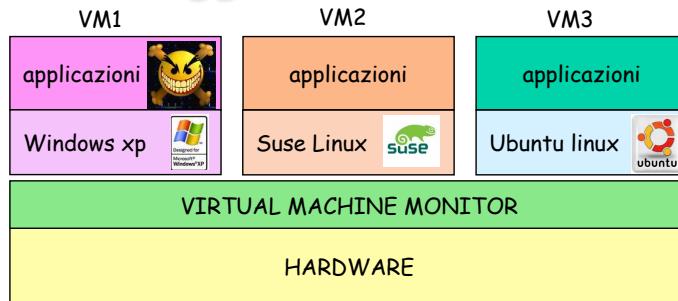
- I produttori di hardware abbandonano l'idea di supportare il concetto di virtualizzazione a livello architetturale (Es. Intel IA-32)
- Paradigma "**one application, one server**"
  - ➔ esplosione del numero di server fisici da configurare, gestire, manutenere, ecc.
  - ➔ Sottoutilizzo delle risorse hardware

## ■ Fine anni 90: necessità di razionalizzazione !

- nuovi sistemi di virtualizzazione per l'architettura Intel x86 (1999, VMware)

## ■ Anni 2010: Cloud Computing.

# Vantaggi della virtualizzazione



- **Uso di piu` S.O. sulla stessa macchina fisica:** più ambienti di esecuzione (eterogenei) per lo stesso utente:
  - Legacy systems
  - Possibilità di esecuzione di applicazioni concepite per un particolare s.o.
- **Isolamento degli ambienti di esecuzione:** ogni macchina virtuale definisce un ambiente di esecuzione separato (*sandbox*) da quelli delle altre:
  - possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM.
  - Sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

# Vantaggi della virtualizzazione

- **Consolidamento HW:** possibilità di concentrare più macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. server farm):
  - Abbattimento costi hw
  - Abbattimento costi amministrazione
- **Gestione facilitata delle macchine:** è possibile effettuare in modo semplice:
  - la creazione di macchine virtuali (virtual appliances)
  - l'amministrazione di macchine virtuali (reboot, ricompilazione kernel, etc.)
  - migrazione a caldo di macchine virtuali tra macchine fisiche:
    - possibilità di manutenzione hw senza interrompere i servizi forniti dalle macchine virtuali
    - disaster recovery
    - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo "autonomico" a situazioni di sbilanciamento

# Vantaggi della virtualizzazione

- **In ambito didattico:** invece di assegnare ad ogni studente un account su una macchina fisica, si assegna una macchina virtuale.
  - possibilità di esercitarsi senza limitazioni nelle tecniche di amministrazione e configurazione del sistema;
  - possibilità di installazione e testing di nuovi sistemi operativi, anche prototipali, senza il rischio di compromettere la funzionalità del sistema.
  - possibilità di testing di applicazioni potenzialmente pericolose senza il rischio di interferire con altri utenti/macchine;
  - possibilità di trasferire le proprie macchine virtuali in supporti mobili (es: penne USB, per continuare le esercitazioni sul computer di casa).

## Realizzazione del VMM

In generale, il VMM deve offrire alle diverse macchine virtuali le risorse (virtuali) che sono necessarie per il loro funzionamento:

- CPU
- Memoria
- Dispositivi di I/O

# Realizzazione del VMM

**Requisiti (Popek e Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures" 1974):**

1. **Ambiente di esecuzione per i programmi sostanzialmente identico a quello della macchina reale.**  
Uniche differenze legate alle dipendenze temporali (più macchine virtuali concorrenti che condividono le risorse HW) → prestazioni.
2. **Garantire un'elevata efficienza nell'esecuzione dei programmi.**  
Quando possibile, il VMM deve permettere l'esecuzione diretta delle istruzioni impartite dalle macchine virtuali → le istruzioni non privilegiate vengono eseguite direttamente in hardware senza coinvolgere il VMM.
3. **Garantire la stabilità e la sicurezza dell'intero sistema.**  
Il VMM deve rimanere sempre nel pieno controllo delle risorse hardware → i programmi in esecuzione nelle macchine virtuali (applicazioni e S.O.) non possono accedere all'hardware in modo privilegiato.

## Realizzazione VMM: parametri e classificazione

- **Livello** dove è collocato il VMM:
  - **VMM di sistema:** eseguono direttamente sopra l'hardware dell'elaboratore (es. vmware esx, xen, kvm)
  - **VMM ospitati:** eseguiti come applicazioni sopra un S.O. esistente (es. vmware player, parallels, virtualPC, virtualbox, UserModeLinux)
- **Modalità di dialogo** per l'accesso alle risorse fisiche tra la macchina virtuale ed il VMM:
  - **Virtualizzazione pura** (Vmware): le macchine virtuali usano la stessa interfaccia (istruzioni macchina) dell'architettura fisica
  - **Paravirtualizzazione** (xen): il VMM presenta un'interfaccia diversa da quella dell'architettura hw.

# VMM di sistema vs. VMM ospitati

## VMM di Sistema.

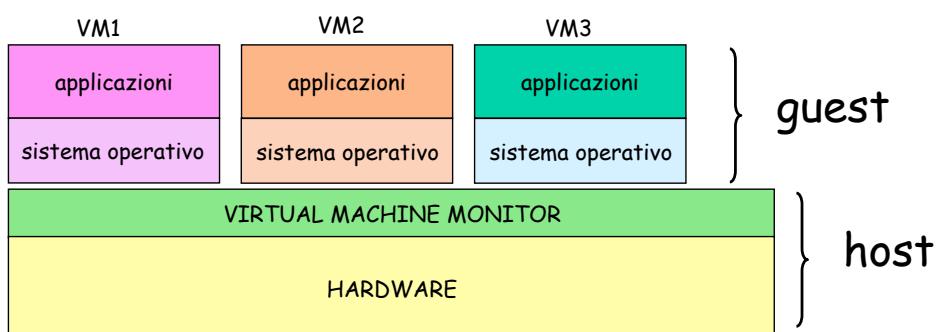
le funzionalità di virtualizzazione vengono integrate in un sistema operativo leggero, costituendo un unico sistema posto direttamente sopra l'hardware dell'elaboratore.

- E' necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche.

Esempi di VMM di sistema: Vmware vsphere, xen, kvm, Microsoft HyperV.

**Host**: piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica, l'eventuale sistema operativo ed il VMM.

**Guest**: la macchina virtuale. Comprende applicazioni e sistema operativo

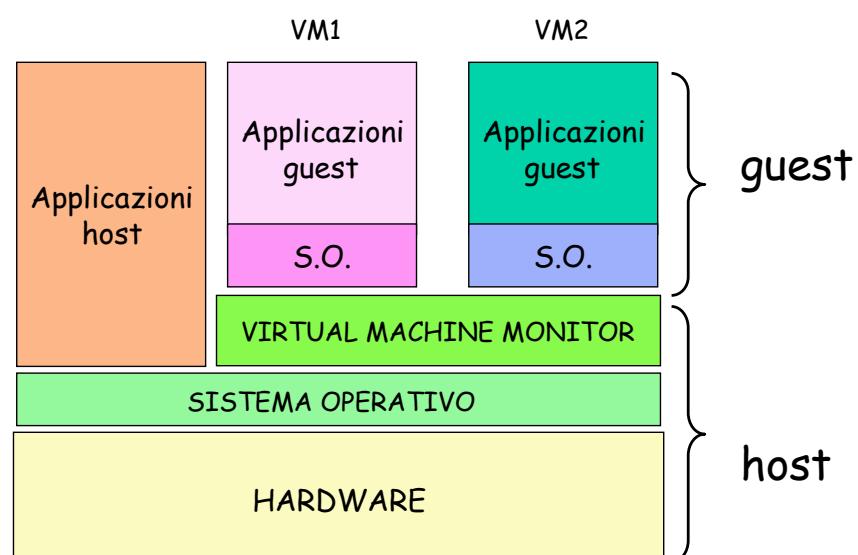


**VMM di Sistema**

## VMM ospitato

- il VMM viene installato come un'applicazione sopra un sistema operativo esistente;
- il VMM opera nello **spazio utente** e accede all'hardware tramite le **system call** del S.O. su cui viene installato.
- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O. (es. scheduling, gestione dei dispositivi, ecc.).
- Peggiora la performance (rispetto al VMM di sistema).

**Prodotti:** User Mode Linux, VMware Fusion/player, Microsoft Virtual Server , Parallels, Oracle Virtualbox



**VMM ospitato**

# Implementazione delle architetture di elaborazione

---

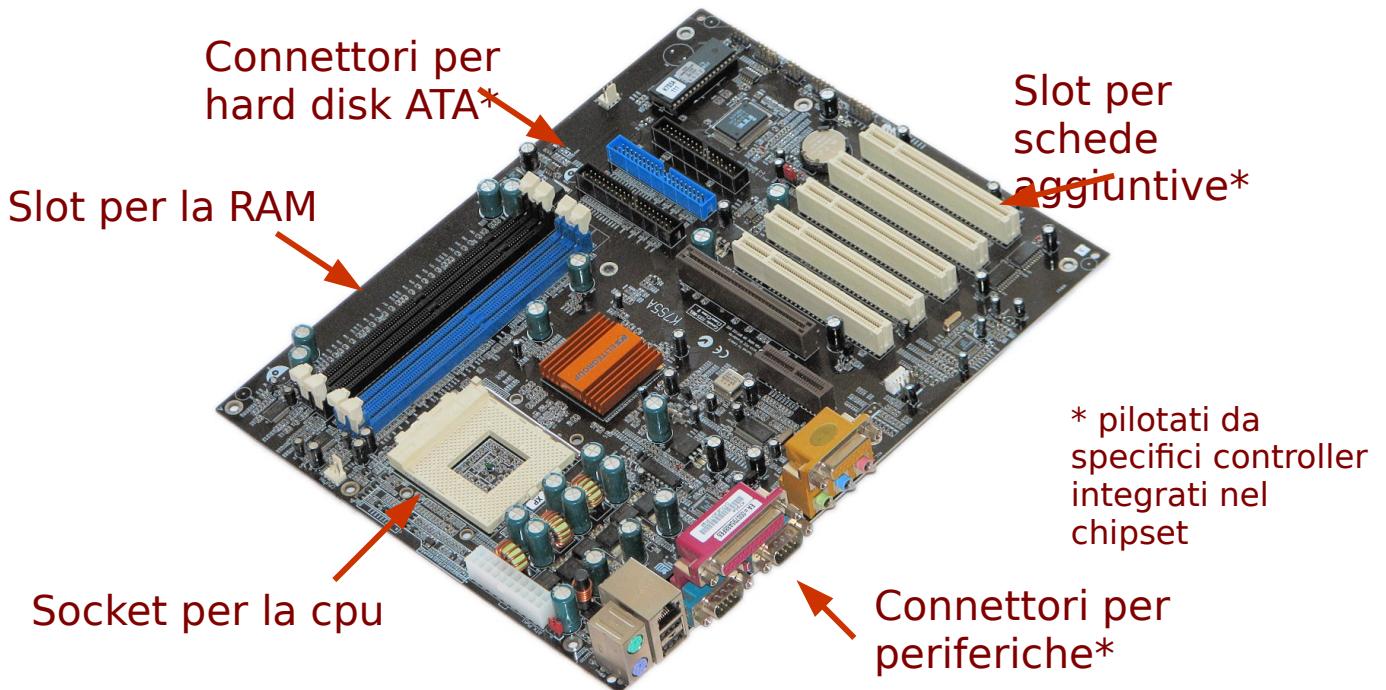
Marco Prandini  
[marco.prandini@unibo.it](mailto:marco.prandini@unibo.it)

---

## CPU e dintorni

- La CPU sa fare solo pochissime operazioni “generiche” per comunicare con l'esterno
    - Riceve il clock
    - Pilota i bus indirizzi e comandi per indicare dove leggere/scrivere
    - Usa il bus dati per inviare/ricevere informazioni
    - Riceve le interruzioni (solo due tipi)
  - Non ha capacità di memorizzazione
  - Servono quindi molti dispositivi di contorno
    - Memorie
    - Sistemi di interfacciamento
  - La *mainboard* è il sistema in cui un *chipset* fornisce le funzioni di base (bus, clock, gestori di interrupt), ospita la memoria e i dispositivi di interfacciamento standard verso altri dispositivi specifici (controller)
-

# Main board / scheda madre



## Memorie

- Sul PC esistono diversi tipi di memoria
- La memoria principale è detta RAM (Random Access Memory)
  - È la più utilizzata dalla CPU sia per recuperare le istruzioni da eseguire che per memorizzare e rileggere i dati
    - Deve essere velocissima e densa (per avere alta capacità e costo relativamente basso)
    - È **VOLATILE** : perde il contenuto allo spegnimento → non può essere usata per avviare il computer!
- Le memorie *non volatili* sono di vari tipi
  - ROM: memorie allo stato solido programmate in fabbrica, inalterabili
    - EEPROM: riprogrammabili a bordo, con circuiteria specifica
  - FLASH: memorie allo stato solido riscrivibili (es. SSD, USB key, SD card per cellulari e fotocamere)
    - Simili alle EEPROM, ma, come la RAM, non richiedono circuiti dedicati per la scrittura
  - Hard Disk: memorie elettromeccaniche magnetiche

# Memorie e avvio

- Appena la CPU si accende, cerca la prima istruzione a un indirizzo predefinito (FFFFFFFFFF0h = 4GB – 16 per i processori Intel dal 386 in poi)



- La mainboard è cablata in modo che a quell'indirizzo si trovi a rispondere un modulo di ROM o FLASH contenente il BIOS, un programma specifico della mainboard, che
  - istruisce il processore a dialogare con tutti gli altri elementi
  - Individua un dispositivo dal quale caricare il sistema operativo
    - Tipicamente un hard disk, magnetico o SSD
    - Può essere altro: un CD/DVD, una memoria esterna USB, un server di rete

## Hard disk

- Gli hard disk “tradizionali” sono composti di uno o più piatti con superficie magnetica, su cui una testina simile al braccio di un giradischi memorizza o legge bit
  - Consumano, scaldano, sono soggetti a (rari) guasti meccanici



- I Solid State Drive (SSD) sono invece totalmente elettronici
  - Più costosi
  - La riscrittura delle celle di memoria può deteriorarle (centinaia di migliaia di cicli, ma è facile raggiungerli)



## Memorie a confronto (ordini di grandezza – dati 2015)

	RAM (DDR3)	SSD	HDD
Capacità tipica	2-8 GB*	32-1024 GB	1-4 TB
Latenza (1)	10 ns	0,1 ms	10 ms
Throughput (2)	8-16 GB/s	400 MB/s	100 MB/s
Costo per GB	6 €	0,4 €	0,03 €

\* capacità di 1 modulo - sulle mainboard più comuni si possono installare fino a 4 moduli ma i sistemi server di fascia alta possono ospitarne decine

(1) il tempo che trascorre tra il comando di lettura e l'arrivo dei dati

(2) la velocità a cui fluiscono i dati letti sequenzialmente, una volta trascorsa la latenza

Per la scrittura valgono le stesse considerazioni (tipicamente con prestazioni inferiori)

## Esigenze diverse, computer diversi

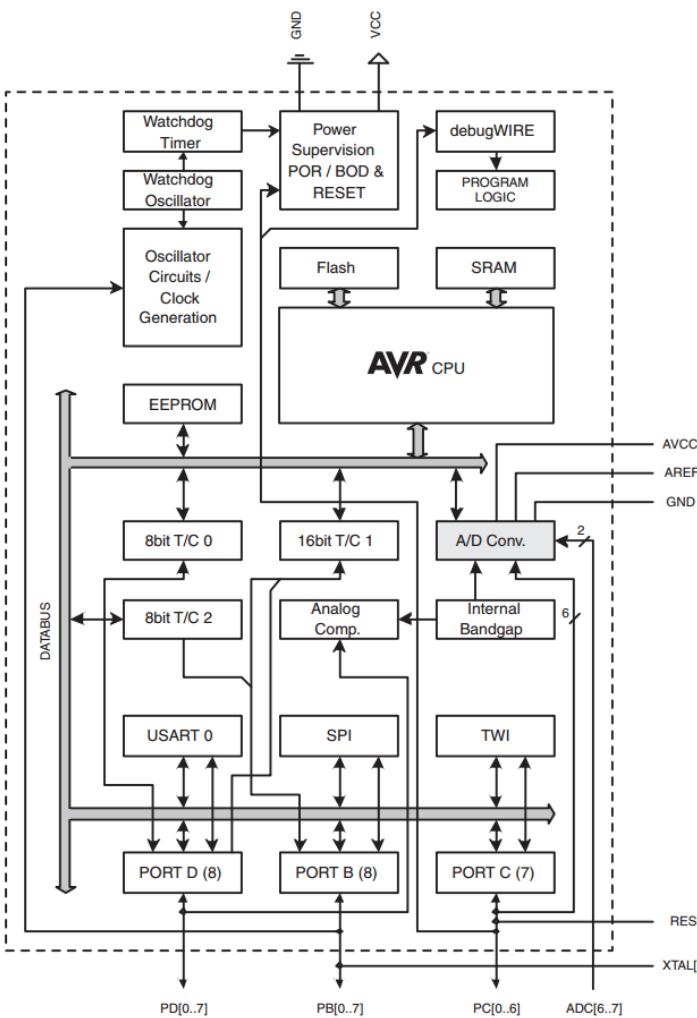
Nel mondo del calcolo in generale, e nella Internet of Things, ci sono esigenze talmente differenziate che sarebbe insensato affrontarle con un'unica architettura. In estrema sintesi:

- “Things”: bassissime esigenze di prestazioni, scarsa necessità di riprogrammarli una volta messi in opera, interfacciamento diretto col mondo esterno, bassissimo costo → **microcontrollori** (es. *Arduino*)
- “Things evolute”: prestazioni contenute, possibilità di interfacciamento tipiche di un PC + interfacciamento diretto col mondo esterno, bassi consumi e ingombri → **single board computer** (es. *Raspberry Pi*)
- Workstation: PC standard, in cui le prestazioni di alcuni sottosistemi possono essere anche molto elevate (es. grafica per CAD e gaming), ma con scarse necessità di multitasking → **PC con pochi core, pochi GB di RAM, dischi standard**
- Server: sistemi pensati per sfruttare l'economia di scala di moltissimi task contemporanei → **molti core, TB di RAM, sistemi di storage parallelo, ridondanza di tutti i componenti per tolleranza ai guasti**

# Microcontrollori

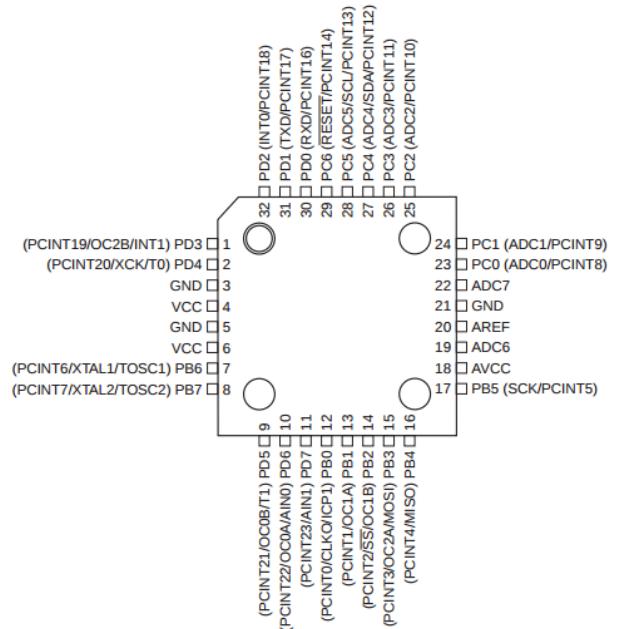
- I microcontrollori condensano in un unico package quasi tutti gli elementi di un sistema CPU-mainboard-memoria-I/O
  - Tipicamente con dimensioni, prestazioni e costi molto più limitati
- Per esempio il uC base per Arduino (a confronto con una CPU i5-4460)
 

<ul style="list-style-type: none"> <li>— Cpu RISC 8 bit a 20MHz</li> <li>— 32KB FLASH</li> <li>— 1KB EEPROM</li> <li>— 2KB RAM</li>   <li>— 8 canali di acquisizione A/D</li> <li>— Porte seriali</li> <li>— 23 pin per I/O digitale direttamente utilizzabile con molti apparati (pulsanti, led, relè...)</li> <li>— 3 €</li> </ul>	<p><b>4 core 64 bit a 3200MHz</b></p> <p>—</p> <p>—</p> <p><b>6MB cache, ma solo come accelerazione della RAM esterna</b></p> <p>—</p> <p>—</p> <p><b>bus a 64 bit (richiede elettronica di interfaccia per poter pilotare dispositivi)</b></p> <p><b>150€</b></p>
--	--



## ATmega328P

<http://www.atmel.com/Images/doc8161.pdf>



# Single-board computer

- Basati sullo stesso principio dei microcontrollori, integrano però componenti tipici dei PC. Ad esempio il Raspberry Pi 2 è basato sull'integrato Broadcom BCM2836, un *System on a Chip (SoC)* con
  - 4 core ARM a 900MHz
  - 1GB di RAM
  - 1 GPU Full HD
  - 1 porte USB (hub 4p sulla scheda)
  - Interfaccia Ethernet
  - 40 pin GPIO
  - Manca memoria non volatile, sostituita da un'interfaccia per schede SD



# Single-board computer

- Sono oggetti in grado di sostituire al 100% un PC per uso domestico o ufficio
  - Su Raspberry Pi sono state portate diverse distribuzioni complete di Linux e Windows Embeddedal costo di poche decine di euro, con un consumo di pochi watt
- I sistemi a microcontrollore, enormemente meno potenti non necessariamente molto più economici, mantengono come vantaggi
  - L'interfacciamento diretto verso il mondo analogico
  - Il consumo 10-20 volte inferiore
- Raspberry ha tracciato la rotta  
<https://www.raspberrypi.org/>
- Molti altri hanno accettato la sfida  
<http://www.computerworlduk.com/galleries/it-vendors/move-over-raspberry-pi-9-single-board-computers-for-geeks-3544497/>

# Dispositivi mobili

- I dispositivi mobili (smartphone, tablet) sono dei single-board computer
  - es. la CPU del Raspberry è esattamente la stessa montata sullo smart watch Samsung Galaxy Gear e su di una decina di telefoni
  - tipicamente gli smartphone sono più potenti di un SBC!
- resi autonomi da
  - un'interfaccia utente (touchscreen, videocamera, speaker)
  - una batteria
  - un set di interfacce wireless (GSM / LTE / WiFi ...)
- e semplificati eliminando GPIO e altre possibilità di espansione diverse dalla porta USB (in qualche variante proprietaria)
- Il sistema operativo è derivato da una versione standard per computer (MacOS → iOS, Linux → Android, Windows → Windows Phone) ottimizzato sull'hardware specifico e quindi legato al produttore

# Workstation

- I PC “desktop” racchiudono in forma abbastanza compatta tutti i componenti necessari, con utili (ma non ampie) possibilità di configurazione ed espansione



# Server (“classici” x86)

- Architetturalmente non sono molto diversi da una workstation, ma sono orientati a

- Prestazioni

- Mainboard multi-CPU (2/4, eccezionalmente 8)

- Espandibilità

- Più slot per RAM
    - Più slot per HDD
    - Montaggio su rack

- Affidabilità

- Doppia alimentazione
    - RAM ECC
    - Multiple interfacce di rete

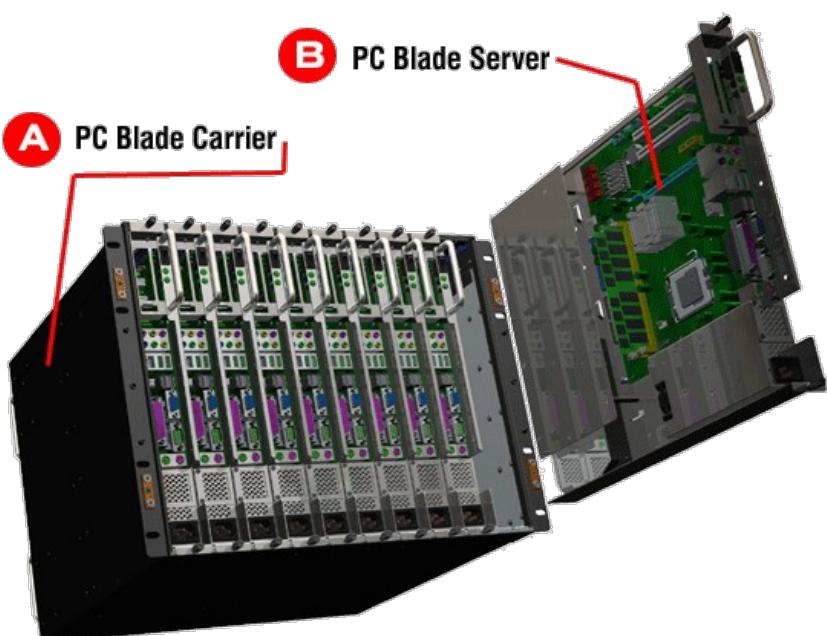


Intel® Server Board S2600WT2 or  
Intel® Server Board S2600WTT

- Supports two Intel® Xeon® processor E5-2600 v3 family
- 24 memory sockets support LR/U/R/NV-DIMMs, up to 1.5 TB total memory (using 64 GB DIMMs)
- 80 PCIe\* Gen 3 and four PCIe Gen 2 lanes available for maximum I/O capacity
- Options to support up to 24 2.5-inch or 12 3.5-inch hot-swap hard drives
- 750 W AC redundant-capable power supply (80 PLUS® Platinum efficiency), 750 W DC redundant-capable power supply (80 PLUS Gold efficiency), or 1100 W AC redundant-capable power supply (80 PLUS Platinum efficiency), second power supply sold separately
- Two full-height half-length PCIe\* Gen 3 x16 slots
- Dual 1 GbE LAN or dual 10 GbE LAN options

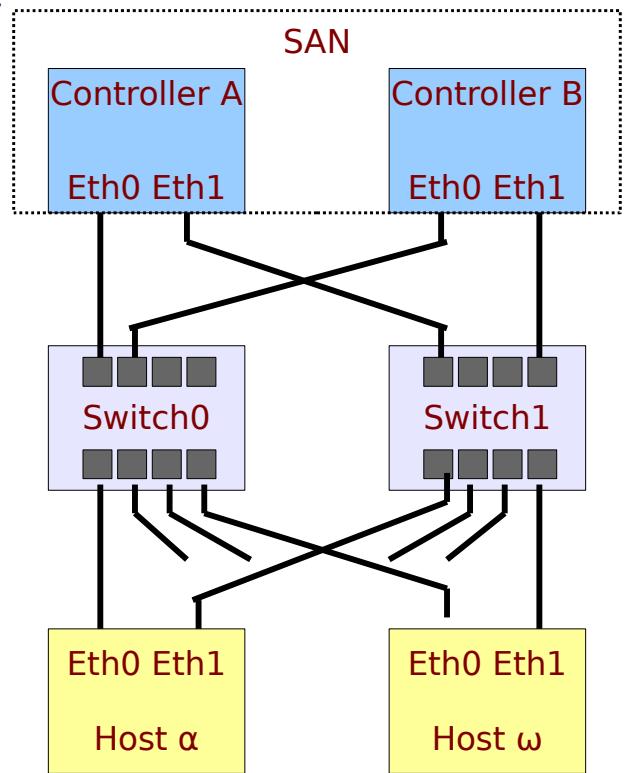
## Server blade

- Mettono a fattor comune alimentazione e connessione KVM (Keyboard, Video, Mouse) tra diverse “lame”, per incrementare la densità



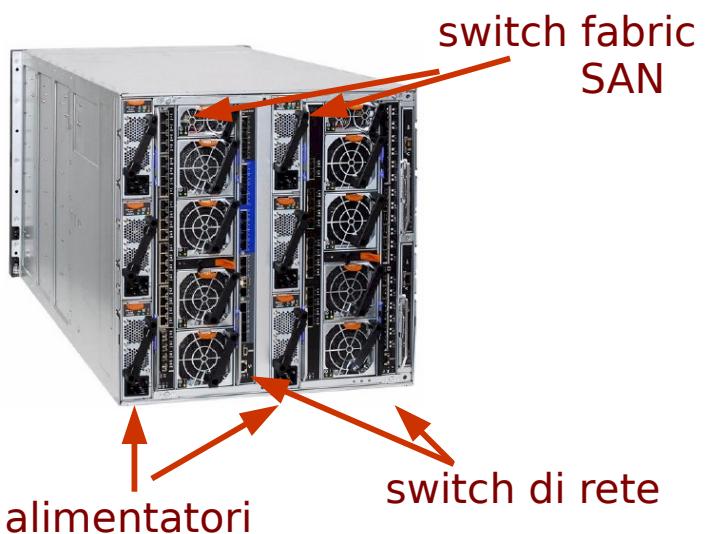
# Cluster

- Per erogare servizi in modo affidabile e con prestazioni molto superiori a quelle ottenibili da un singolo sistema, l'affidabilità, molti computer vengono affiancati per svolgere collettivamente il compito richiesto
- Tipicamente il cuore del cluster è un sistema di storage condiviso + un sistema per la realizzazione di connessioni ridondanti (doppi schede su ogni server, doppi switch, ecc....)
- La parte più complessa è gestire l'accesso concorrente alle risorse e il monitoraggio dei nodi, perché in caso di guasto di uno i servizi vengano riavviati su quelli ancora attivi



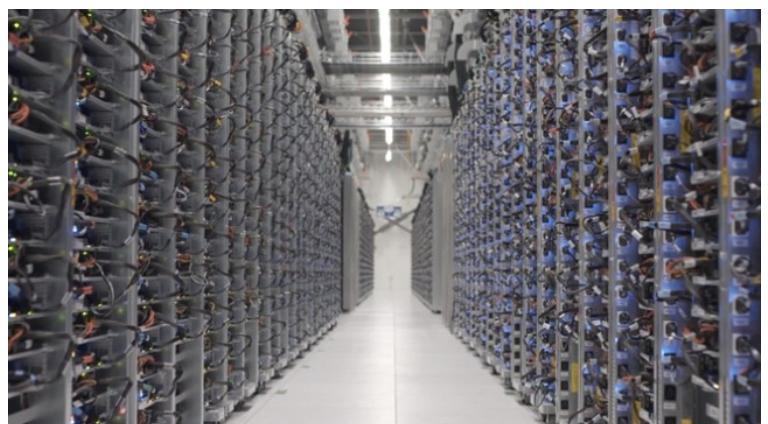
# Datacenter-in-a-box

- Vista la preponderanza dei cluster, i sistemi blade si sono evoluti per ospitare tutti i componenti necessari



# Collocazione dei server

- Gli investimenti in hardware di fascia alta sono ingenti, e poco convenienti se non si riescono a usare i sistemi vicino al 100% della capacità
- Servono significative risorse umane per la gestione
- Per offrire servizi internet ad alta disponibilità, non basta hardware affidabile, va anche collocato in datacenter costosi



## Datacenter

- Resistenza della struttura
  - cause naturali: terremoti, inondazioni, ...
  - cause artificiali: incidenti aerei e ferroviari, terrorismo, ...
  - cause interne: incendi, da controllare con sistemi che consentano l'intervento anche quando l'incendio stesso li danneggia parzialmente
- Sicurezza e controllo degli accessi
- Condizionamento dell'aria
  - gestione di temperatura e umidità con sistemi tolleranti ai guasti e alle interruzioni di erogazione dell'energia elettrica
- Condizionamento dell'alimentazione elettrica
  - erogazione su almeno due linee indipendenti per ogni apparato
  - pulizia della sinusoide per allungare la vita degli apparati
  - sistemi di continuità ad intervento istantaneo e di durata prolungata
    - combinano motogeneratori che possono sopperire a giorni di mancata erogazione, ma richiedono 15-20 minuti per l'avviamento, a sistemi a batteria ad intervento istantaneo ma di bassa capacità
- Connettività di rete
  - connessione tramite provider indipendenti
  - collocazione fisica dei cavi su percorsi indipendenti

# Cloud: IaaS



MA....

- Al giorno d'oggi i cluster sono usati quasi solo per far girare hypervisor su cui istanziare macchine virtuali (VM)
- Le VM sono indipendenti dall'hardware e dalla collocazione (comunque vanno amministrate da remoto)

■ ALLORA.... Perché non lasciare che qualcuno si occupi di datacenter e hardware, e *noleggiare le risorse (CPU, RAM, disco, banda, ....) solo per il tempo necessario?*



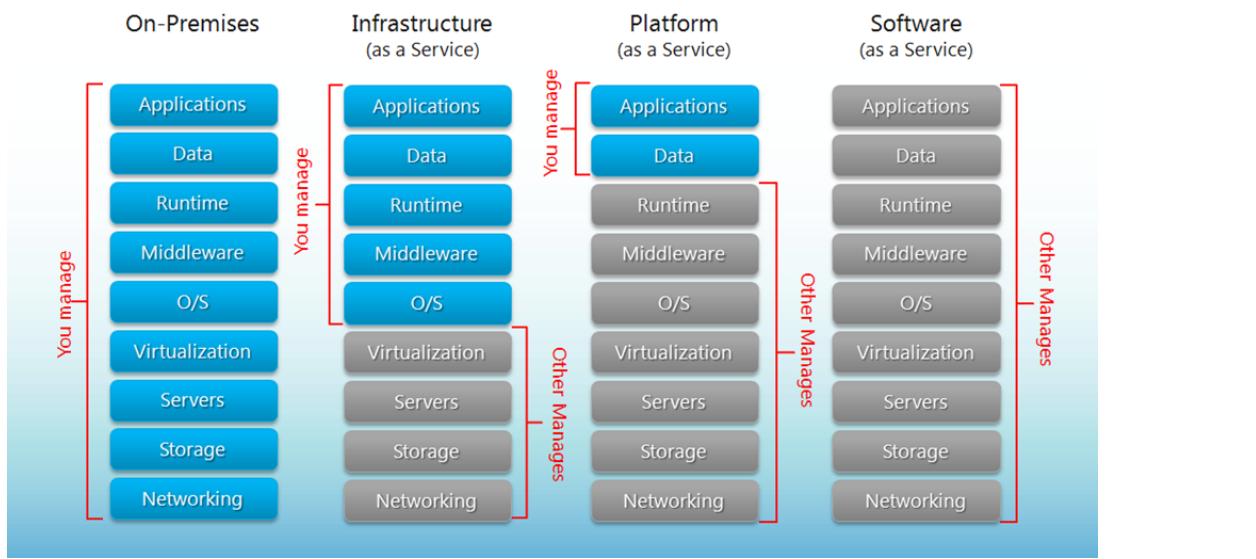
## Infrastructure as a Service

### Un esempio di IaaS: AWS

Compute	Administration & Security	Application Services
<b>EC2</b> Virtual Servers in the Cloud	<b>Directory Service</b> Managed Directories in the Cloud	<b>SQS</b> Message Queue Service
<b>Lambda</b> Run Code in Response to Events	<b>Identity &amp; Access Management</b> Access Control and Key Management	<b>SWF</b> Workflow Service for Coordinating Application Components
<b>EC2 Container Service</b> Run and Manage Docker Containers	<b>Trusted Advisor</b> AWS Cloud Optimization Expert	<b>AppStream</b> Low Latency Application Streaming
Storage & Content Delivery	CloudTrail	Elastic Transcoder
<b>S3</b> Scalable Storage in the Cloud	<b>Config</b> Resource Configurations and Inventory	<b>Elastic Transcoder</b> Easy-to-use Scalable Media Transcoding
<b>Elastic File System</b> Fully Managed File System for EC2	<b>CloudWatch</b> Resource and Application Monitoring	<b>SES</b> Email Sending Service
<b>Storage Gateway</b> Integrates On-Premises IT Environments with Cloud Storage		<b>CloudSearch</b> Managed Search Service
<b>Glacier</b> Archive Storage in the Cloud		
<b>CloudFront</b> Global Content Delivery Network		
Database	Deployment & Management	Mobile Services
<b>RDS</b> MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora	<b>Elastic Beanstalk</b> AWS Application Container	<b>Cognito</b> User Identity and App Data Synchronization
<b>DynamoDB</b> Predictable and Scalable NoSQL Data Store	<b>OpsWorks</b> DevOps Application Management Service	<b>Mobile Analytics</b> Understand App Usage Data at Scale
<b>ElastiCache</b> In-Memory Cache	<b>CloudFormation</b> Templated AWS Resource Creation	<b>SNS</b> Push Notification Service
<b>Redshift</b> Managed Petabyte-Scale Data Warehouse Service	<b>CodeDeploy</b> Automated Deployments	
Networking	Analytics	Enterprise Applications
<b>VPC</b> Isolated Cloud Resources	<b>EMR</b> Managed Hadoop Framework	<b>WorkSpaces</b> Desktops in the Cloud
<b>Direct Connect</b> Dedicated Network Connection to AWS	<b>Kinesis</b> Real-time Processing of Streaming Big Data	<b>WorkDocs</b> Secure Enterprise Storage and Sharing Service
<b>Route 53</b> Scalable DNS and Domain Name Registration	<b>Data Pipeline</b> Orchestration for Data-Driven Workflows	<b>WorkMail</b> Secure Email and Calendering Service
	<b>Machine Learning</b> Build Smart Applications Quickly and Easily	

# Cloud: PaaS / SaaS

- Perché amministrare una VM se l'unica necessità è avere, ad esempio, un web server funzionante sul quale pubblicare un sito? → *Platform a.a.S.*
- E perché realizzare un complesso sito dinamico per gestire, ad esempio, la creazione di documenti condivisi, quando si può utilizzare a consumo un servizio già esistente? → *Software a.a.S.*



# Cloud e IoT

Cosa c'entra tutto questo con la Internet of Things?

- Microcontrollori, single-board computer, dispositivi mobili... miliardi di fonti di dati da raccogliere, memorizzare, analizzare
- I giganteschi datacenter dei fornitori di soluzioni cloud sono praticamente l'unica possibilità di riuscire
- La flessibilità del modello “\*aaS” permette di realizzare rapidamente prototipi software senza investimenti di capitale
- Gli stessi prototipi possono scalare automaticamente a dimensioni quasi illimitate seguendo la crescita del business

# Distribuzioni

**GNU/Linux è un insieme composto di elementi sostanzialmente fissi, reperibili in forma sorgente:**

- kernel
- librerie di sistema
- utilità di base
- software applicativo di uso comune

**Per portare su di un sistema questi componenti però serve**

- o un altro sistema funzionante su cui scaricare e compilare tutti i pezzi
- o una *distribuzione*

---

EnAIP Tecnico Informatico



# Distribuzioni

**Una distribuzione**

- raccoglie in un supporto fisico pratico tutti i componenti, già compilati per l'architettura su cui si vuole installare GNU/Linux
- fornisce gli strumenti per l'avvio autonomo del sistema da installare
- tramite un installer propone la corretta sequenza dei passi di predisposizione del sistema
- fornisce strumenti per la gestione del software installato/installabile: il grande valore aggiunto è la gestione delle dipendenze

---

EnAIP Tecnico Informatico



# Distribuzioni: storia

**La prima distribuzione (SLS) compare nel 1992, ma**

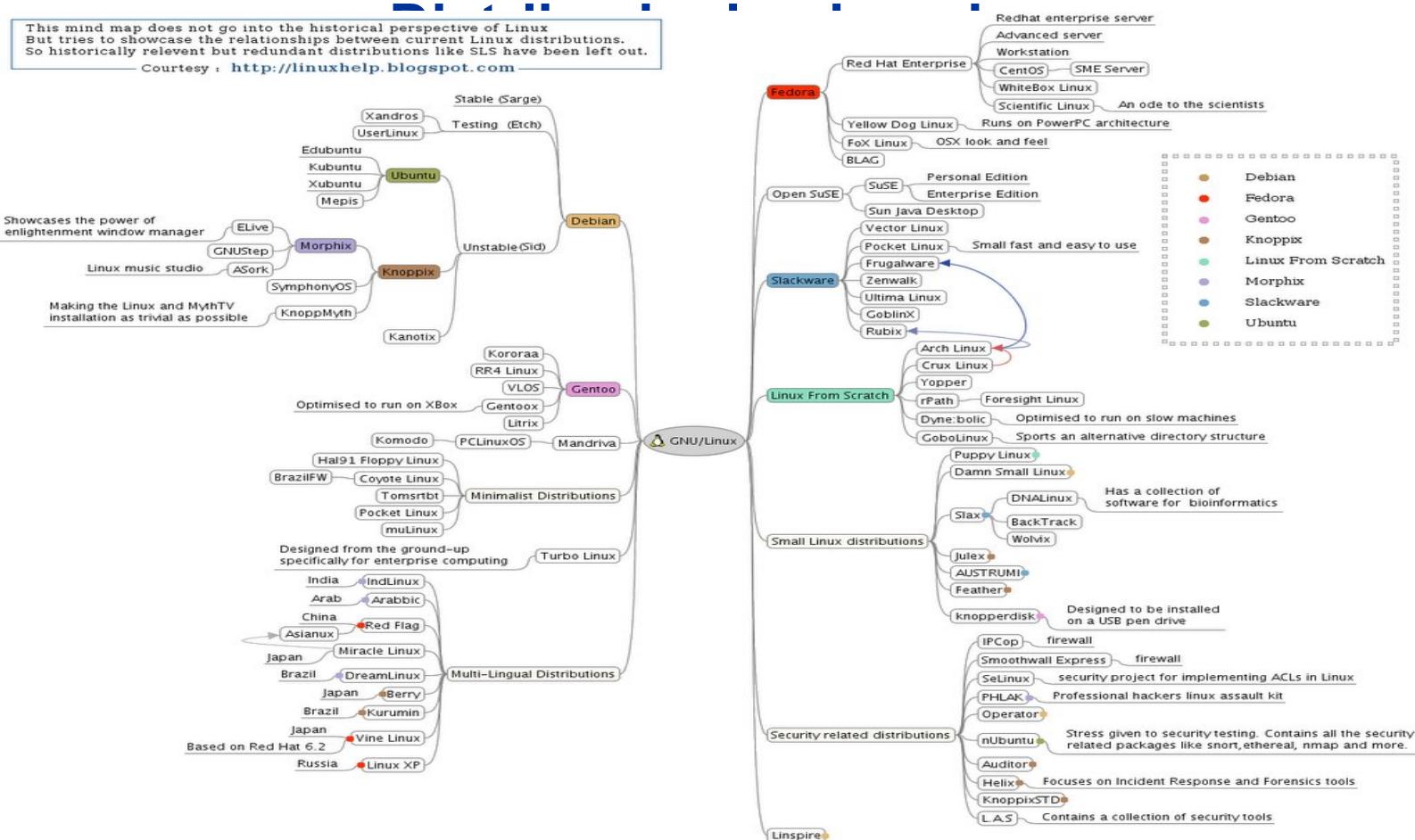
- è poco pratica anche per l'epoca
- contiene un mix di pacchetti con licenze libere e proprietarie che la rende difficile da usare senza grattacapi

**Nel 1993 compaiono le grandi capostipiti:**

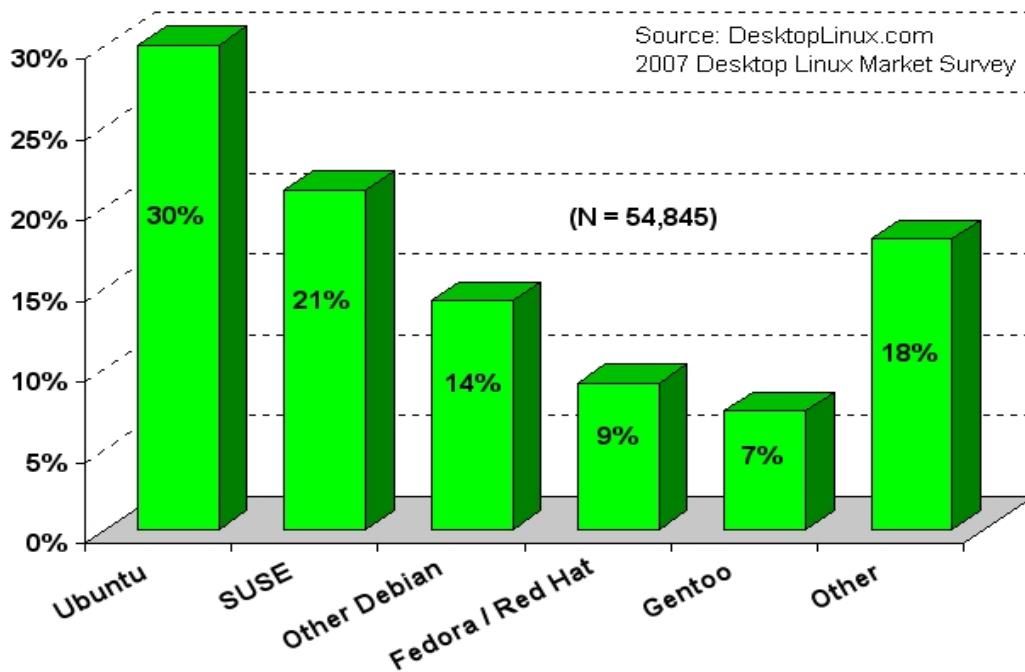
- Slackware, spartana ma efficiente, la più antica ancora in vita
- Debian, con un manifesto simile a quello di GNU

**Nel 1994 viene lanciata RedHat, la prima distribuzione con chiari intenti commerciali**

EnAIP Tecnico Informatico



## Desktop Linux Distributions



EnAIP Tecnico Informatico



## Distribuzioni: criteri per la scelta

### Architetture supportate

- tutte le distribuzioni supportano i processori Intel 32bit, la maggior parte quelli a 64bit, alcune sono disponibili per tutte le varietà di processori su cui è stato portato il kernel
- È bene ricordare che i pacchetti di terze parti potrebbero non essere disponibili per tutte le architetture supportate

### Stabilità vs. Aggiornamento

- il processo di rilascio frequente e continuo del software nel mondo GNU/Linux ha come conseguenza inevitabile che le versioni più aggiornate possano essere meno stabili
- vi sono distribuzioni che hanno come filosofia l'inclusione dei pacchetti più recenti (e quindi con funzionalità maggiori) anche a costo di una minor robustezza, ed altre che garantiscono l'inclusione solo di software ben collaudato

EnAIP Tecnico Informatico

# Distribuzioni: criteri per la scelta

## Supporto e durata

- La disponibilità di supporto garantito è tipica delle distribuzioni commerciali, ma anche con le distribuzioni gratuite più diffuse, in virtù della dimensione della relativa comunità di utenti, è semplice risolvere eventuali problemi
- Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API (tipicamente viene garantito solo il backporting dei security fix, non quello di tutti i bug fix)

## Aampiezza del set di pacchetti

- Si va dai 1500 delle distro minimali ai 26000 di Debian
- Una scelta intelligente mette tutto l'essenziale in 1 CD

---

EnAIP Tecnico Informatico

# Distribuzioni: criteri per la scelta

## Procedura di installazione

- Un installer adatto agli utenti inesperti non deve proporre scelte troppo dettagliate e complesse, che possono invece essere desiderabili per un utente con necessità più avanzate
- Alcuni installer possono contenere strumenti ausiliari, ad esempio per ricavare lo spazio per Linux su di un disco già occupato da altri S.O.

## Strumenti per la gestione dei pacchetti

- Il sistema di packaging e la qualità dei relativi tool per cercare, installare, aggiornare e rimuovere software possono differire notevolmente
- I migliori tool sono in grado di gestire l'upgrade di versione dell'intera distribuzione

---

EnAIP Tecnico Informatico



# Distribuzioni: criteri per la scelta

## Strumenti di configurazione

- Sebbene le configurazioni avanzate siano tipicamente fatte manualmente dai sistemi, la disponibilità di strumenti di ausilio basati su interfaccia testo o grafica può essere un criterio di scelta

## Interfaccia grafica (desktop environment)

- Il sottosistema grafico è tra i più complessi da configurare, e incide moltissimo sulle prestazioni, per cui poter scegliere il desktop environment adatto all'hardware disponibile può essere molto importante
- in ordine approssimativo di "peso" e funzionalità crescenti citiamo: JWM, FluxBox, Xfce, KDE, Gnome

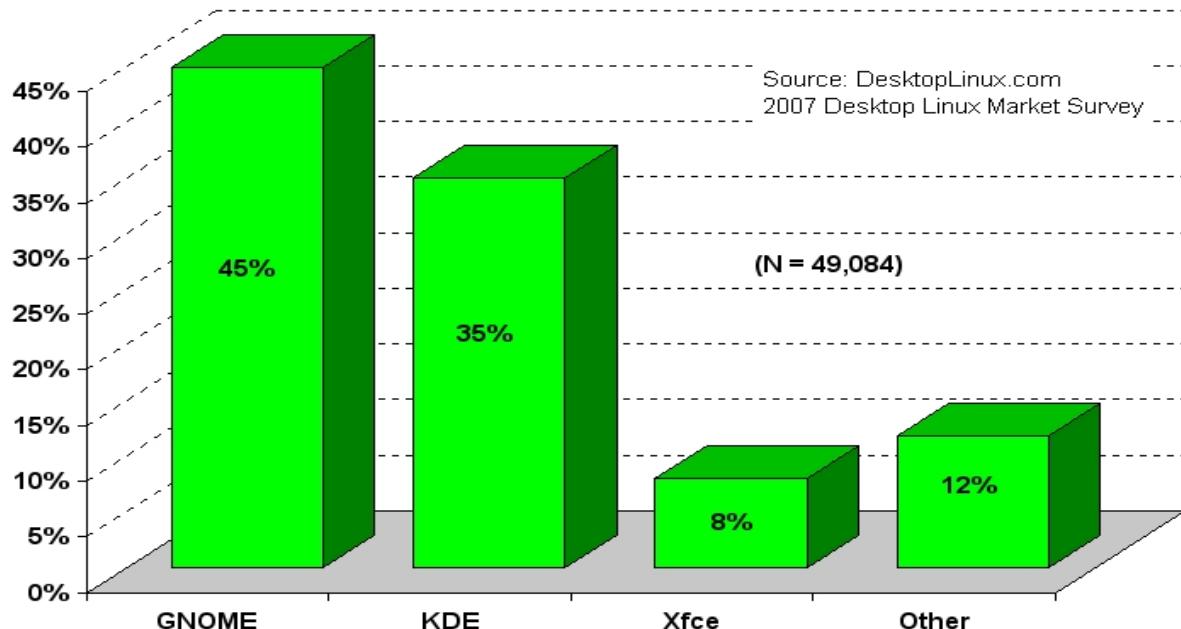
---

EnAIP Tecnico Informatico



# Sul desktop

## Desktop Linux Windowing Environments



---

EnAIP Tecnico Informatico



RegioneEmilia-Romagna



# I.F.T.S. 2020 - 2021

Operazione Rif. PA 2020-14432/RER Approvata con Deliberazione di Giunta Regionale n. 927 del 28/07/2020, cofinanziata con risorse del Fondo Sociale Europeo e della Regione Emilia-Romagna

Progetto 1, edizione 1

**Tecnico per lo sviluppo di applicazioni informatiche  
specializzato in soluzioni web based**

**Periodo di svolgimento:** novembre 2020 – giugno 2021

**Durata:** 800 ore

## DISPENSE DIDATTICHE

**Modulo n°: 6**

**Titolo modulo: AMMINISTRAZIONE DI SISTEMI SERVER  
(Installazione software e gestione processi)**

**Docente: MARCO PRANDINI**

**Coordinatrice del Corso : Sara Forlivesi**



Fondazione En.A.I.P. S. Zavatta Rimini  
Viale Valturio, 4 47923 Rimini  
Tel. 0541.367100 – fax. 0541.784001  
[www.enaiprimini.org](http://www.enaiprimini.org); e-mail: [info@enaiprimini.org](mailto:info@enaiprimini.org)

# Gestione del software

- Ciclo di vita
    - installazione
    - aggiornamento
    - disininstallazione
  - Problematiche
    - prerequisiti hardware/s.o.
    - dipendenze da/di altri componenti software
    - configurazione
- 

1

## Installazione manuale

- Da binari
    - semplice copia nei "posti giusti"
    - verifica manuale della compatibilità con l'architettura
    - verifica manuale del soddisfacimento delle dipendenze
  - Da sorgente
    - necessità di compilazione
    - indipendenza dall'architettura
    - possibile maggior flessibilità nel soddisfacimento delle dipendenze
- 

2

# Installazione manuale

## ■ Dipendenze del componente software da altri

- Nel caso di un'installazione da binari, probabile necessità di disporre non solo dei software indicati come prerequisiti, ma anche che essi siano di una versione specifica
- Nel caso di installazione da sorgente, qualche grado di flessibilità (possibilità che i sorgenti dispongano di diverse interfacce per adeguarsi a cosa si trova sul sistema)
  - Necessità di disporre non solo dei componenti runtime relativi ai software richiesti, ma anche delle librerie di sviluppo (prototipi, interfacce, librerie per collegamento statico, ...)
    - In un sistema “ideale” ho tutti i sorgenti per cui dispongo sempre di tutti questi elementi
    - Nelle distribuzioni, per flessibilità, ogni pacchetto software ha un corrispondente pacchetto -dev o -devel (vedi prossime slide)

---

3

# Installazione manuale tipica in Linux

## ■ Il caso più comune è quello di software

- distribuito per mezzo di un archivio tar.gz
- scritto in C
- predisposto alla compilazione tramite autoconf
  - verifica se sono soddisfatti tutti i prerequisiti
  - rileva le versioni ed le collocazioni dei pacchetti sul sistema
  - accetta dall'utente la specifica di varianti (attivazione/disattivazione di funzionalità, preferenze architetturali, ...)
  - genera i Makefile sulla base delle specificità del sistema e delle scelte operate dall'utente

---

4

# Installazione manuale tipica in Linux

## ■ I passi tipici quindi sono:

- reperimento del software
- estrazione del pacchetto
- esame delle scelte disponibili
- configurazione dei sorgenti
- compilazione
- installazione
  - NOTA: solo quest'ultima operazione può richiedere i diritti di superutente, e quindi si deve evitare di compiere le precedenti come *root*. Sono noti casi di malware che sfruttano proprio la cattiva abitudine di eseguire una o più delle operazioni preliminari con diritti eccessivi.

---

5

# Installazione manuale tipica in Linux

## ■ estrazione del pacchetto

- solitamente si presenta come archivio tar compresso
- è buona prassi determinare una collocazione sensata per i sorgenti ed estrarre in tale directory l'archivio
  - nel caso si stia per affrontare un upgrade sostanziale del sistema, che coinvolga numerose applicazioni, può essere utile raccogliere in modo più chiaro tutti i pacchetti che verranno installati unitariamente
- è prudente testare l'archivio prima dell'estrazione per verificare la gerarchia di directory che genera
- Es:

```
cd /usr/local/src
tar tvzf net-snmp-5.4.tar.gz
tar xvzf net-snmp-5.4.tar.gz
```

---

6

# Installazione manuale tipica in Linux

- esame delle scelte disponibili
  - si entra nella directory generata dall'estrazione e si esamina il contenuto
    - è bene leggere i file README ed INSTALL che di solito accompagnano il software
  - se esiste un eseguibile di nome configure lo si lancia con il parametro --help per ottenere la lista dei parametri di configurazione disponibili
    - scelte comuni riguardano la collocazione del software, l'attivazione o la disattivazione di sottocomponenti, la predisposizione dei componenti attivati come moduli dinamicamente caricabili piuttosto che la loro integrazione statica nel codice, ...

---

7

# Installazione manuale tipica in Linux

- configurazione dei sorgenti
  - si lancia nuovamente configure con i parametri scelti
  - si risolvono i problemi evidenziati da configure (tipicamente assenza di pacchetti necessari come prerequisiti)
    - configure non è a prova d'errore,
- compilazione
  - si lancia make o si seguono le indicazioni presenti nell'output generato dal passo precedente
- installazione
  - si lancia sudo make install

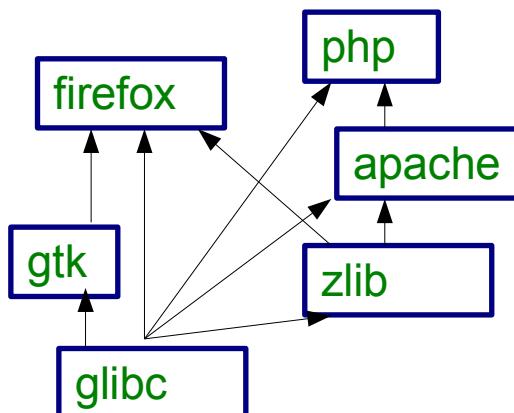
---

8

# Installazione assistita

- Comunemente effettuata per mezzo di software ausiliari
  - package manager specifico della distribuzione Linux (rpm/yum, dpkg/apt, ...)
  - installer per Windows
- Un tool di installazione
  - può farsi carico delle verifiche relative alle dipendenze
  - non può configurare ogni dettaglio del sistema in modo specifico
  - può generare dinamicamente dati specifici

Esempio di grafo delle dipendenze:



A → B significa che A "serve" per B; "serve" può essere una dipendenza tra funzionalità logiche (non ha senso avere un linguaggio di generazione pagine web senza un web server) o fisiche (un binario linkato dinamicamente non gira senza tutte le librerie di cui importa i simboli)

9

## Pacchetti

- Le *distribuzioni* di Linux organizzano il software in *pacchetti* e dispongono di un *package manager* per la loro gestione
- Un pacchetto si presenta sotto forma di singolo file che contiene in forma compatta l'insieme di
  - software precompilato
  - criteri per la verifica della compatibilità e dei prerequisiti
  - procedure di pre/post-installazione
- La garanzia della compatibilità con un determinato sistema può essere data solo a patto di vincolare con precisione alcuni parametri:
  - architettura
  - versione della distribuzione
  - versione del software contenuto nel pacchetto

10

# Debian e Red Hat

- Due distribuzioni capostipite da cui sono state derivate quasi tutte le varianti più diffuse  
<http://upload.wikimedia.org/wikipedia/commons/9/9a/Gldt1009.svg>

- Due sistemi di gestione dei pacchetti con molte somiglianze

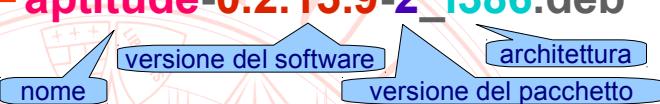
- Tool di basso livello per la gestione dei singoli pacchetti
- Tool intermedi per la gestione coordinata di pacchetti e dipendenze
- Tool per il reperimento automatico da *repository* dei pacchetti necessari

---

11

## Pacchetti

- I pacchetti per le distribuzioni Debian e derivate (es. Ubuntu) sono in formato .deb
  - **aptitude-0.2.15.9-2\_i386.deb**



- I pacchetti per le distribuzioni RedHat e derivate (es. CentOS, Fedora) sono in formato .deb
  - **httpd-2.4.6-45.el7.centos.x86\_64.rpm**

---

12

## Gestione dei pacchetti .deb

database location:	/var/lib/dpkg, /var/lib/apt
sources file:	/etc/apt/sources.list
update sources:	apt-get update
key management:	apt-key
search:	apt-cache search keywords
install:	dpkg -i filename.deb
	apt-get install packagenames
upgrade	apt-get upgrade [packagenames]
remove	dpkg -r packagename
	apt-get remove packagenames

---

13

## Gestione dei pacchetti .rpm

database location:	/var/lib/rpm
sources file:	/etc/yum.conf
update sources:	yum update
key management:	rpm --import keyfile
search:	yum search keywords
install:	rpm -i filename.rpm
	yum install packagenames
upgrade:	yum upgrade [packagenames]
verify integrity:	rpm -V [packagenames a]
remove:	rpm -e packagenames
	yum remove packagenames

---

14

# deb e rpm

## ■ Link per deb

<http://www.debian.org/doc/manuals/debian-reference/ch02.en.html>  
[http://guide.debianizzati.org/index.php/Introduzione\\_all'\\_Apt\\_System](http://guide.debianizzati.org/index.php/Introduzione_all'_Apt_System)

## ■ Link per rpm

<http://yum.baseurl.org/wiki/YumCommands>  
<http://yum.baseurl.org/wiki/RpmCommands>

---

15

# Esempio di pacchetti base e development

## ■ zlib1g

— /usr/lib/libz.so.1.2.3.3

per ogni funzione, es. *compress*:

codice oggetto in formato adatto per il linking dinamico

## ■ zlib1g-dev

— /usr/lib/libz.a  
— /usr/include/zconf.h  
— /usr/include/zlib.h  
— /usr/include/zlibdefs.h

codice oggetto in formato adatto per il linking statico

prototipo per il compilatore

Con questa suddivisione si risparmia (molto) spazio sui sistemi che non sono usati per sviluppare codice basato su questa libreria, nei quali il primo pacchetto fornisce da solo il necessario per usare codice già pronto in forma binaria che **referenzia le funzioni della libreria**

- Su sistemi *deb* → pacchetti “-dev”
- Su sistemi *rpm* → pacchetti “-devel”

---

16

# Esempio di verifica delle dipendenze dinamiche

## ■ ldd /usr/sbin/sshd

```
linux-gate.so.1 => (0xfffffe000)
libwrap.so.0 => /lib/libwrap.so.0 (0xb7ef7000)
libpam.so.0 => /lib/libpam.so.0 (0xb7eed000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0xb7ee8000)
libselinux.so.1 => /lib/libselinux.so.1 (0xb7ed2000)
libresolv.so.2 => /lib/tls/i686/cmov/libresolv.so.2 (0xb7ebf000)
libcrypto.so.0.9.8 => /usr/lib/i686/cmov/libcrypto.so.0.9.8 (0xb7d7c000)
libutil.so.1 => /lib/tls/i686/cmov/libutil.so.1 (0xb7d78000)
libz.so.1 => /usr/lib/libz.so.1 (0xb7d63000)
libnsl.so.1 => /lib/tls/i686/cmov/libnsl.so.1 (0xb7d4a000)
libcrypt.so.1 => /lib/tls/i686/cmov/libcrypt.so.1 (0xb7d1c000)
libgssapi_krb5.so.2 => /usr/lib/libgssapi_krb5.so.2 (0xb7cf3000)
libkrb5.so.3 => /usr/lib/libkrb5.so.3 (0xb7c6b000)
libk5crypto.so.3 => /usr/lib/libk5crypto.so.3 (0xb7c46000)
libcom_err.so.2 => /lib/libcom_err.so.2 (0xb7c43000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0xb7af8000)
/lib/ld-linux.so.2 (0xb7f11000)
libsepol.so.1 => /lib/libsepol.so.1 (0xb7ab7000)
libkrb5support.so.0 => /usr/lib/libkrb5support.so.0 (0xb7aa000)
libkeyutils.so.1 => /lib/libkeyutils.so.1 (0xb7aad000)
```

---

17

# Problematiche di aggiornamento

- Quando si aggiorna un pacchetto software già in uso sul sistema, si deve tener conto di potenziali problemi derivanti da:
  - **prerequisiti**
    - pacchetti che devono esistere perchè il candidato funzioni bene
    - come nel caso dell'installazione
    - potrebbe non essere facile aggiornare i pacchetti-prerequisiti senza causare problemi ad altri software che li utilizzano
  - **configurazione**
    - eventuali modifiche incompatibili apportate al formato delle direttive di configurazione già messe a punto per la versione funzionante

---

18

# Problematiche di aggiornamento

## ■ (continua)

- dipendenze di altri software e test di non regressione
  - modifiche apportate alle interfacce o alle funzionalità del software potrebbero influire sul funzionamento di altri software
  - *configurazione del PATH* per impostare l'ordine di ricerca degli eseguibili nelle directory
  - predisposizione di configurazioni di test per far coesistere le due versioni durante le fasi di verifica
    - es. binding a socket, porte, IP diversi --> problemi di trasparenza per l'utente, licenze, configurazione delle controparti se il software da testare interagisce attraverso interfacce standard

---

19

# Problematiche di aggiornamento

## — (continua dipendenze e test)

- *configurazione del loader* per far convivere differenti versioni di librerie dinamiche, si veda la man page `ld(1)`, specialmente le sezioni sui parametri `-rpath` e `-rpath-link`
  - modifica dei settaggi di default in `/etc/ld.so.conf` (da applicare con `ldconfig`)
  - uso delle variabili `LD_LIBRARY_PATH` in fase di loading e `LD_RUN_PATH` in fase di linking

Esempio:

```
# ldd /usr/sbin/sshd
...
libz.so.1 => /usr/lib/libz.so.1 (0xb7e0c000)
...
# export LD_LIBRARY_PATH=/usr/local/lib
# ldd /usr/sbin/sshd
...
libz.so.1 => /usr/local/lib/libz.so.1 (0xb7dab000)
```

---

20

# Disinstallazione

- Presenta gli stessi problemi dell'aggiornamento in termini di eventuale dipendenza di altri software da quello che si sta per rimuovere
  - in entrambi i casi può essere molto difficile prevedere gli effetti sul sistema se la gestione è manuale
  - il grafo delle dipendenze è quindi il valore aggiunto più significativo dei sistemi a pacchetti
  - può essere molto utile sfruttare la possibilità offerta dai package manager di creare i propri pacchetti, per gestire il software installato manualmente tramite il sistema automatico di verifica delle dipendenze (ma ciò significa censirle con precisione all'installazione)

---

21

# Distribuzioni: criteri per la scelta

## Architetture supportate

- tutte le distribuzioni supportano i processori Intel 32bit, la maggior parte quelli a 64bit, alcune sono disponibili per tutte le varietà di processori su cui è stato portato il kernel
- È bene ricordare che i pacchetti di terze parti potrebbero non essere disponibili per tutte le architetture supportate

## Stabilità vs. Aggiornamento

- il processo di rilascio frequente e continuo del software nel mondo GNU/Linux ha come conseguenza inevitabile che le versioni più aggiornate possano essere meno stabili
- vi sono distribuzioni che hanno come filosofia l'inclusione dei pacchetti più recenti (e quindi con funzionalità maggiori) anche a costo di una minor robustezza, ed altre che garantiscono l'inclusione solo di software ben collaudato

---

22

# Distribuzioni: criteri per la scelta

## Supporto e durata

- La disponibilità di supporto garantito è tipica delle distribuzioni commerciali, ma anche con le distribuzioni gratuite più diffuse, in virtù della dimensione della relativa comunità di utenti, è semplice risolvere eventuali problemi
- Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API (tipicamente viene garantito solo il backporting dei security fix, non quello di tutti i bug fix)

## Aampiezza del set di pacchetti

- Si va dai 1500 delle distro minimali ai 26000 di Debian
- Una scelta intelligente mette tutto l'essenziale in 1 CD

---

23

# Lavorare coi repository

- Un'esigenza molto comune è quella di installare software ben supportato ma non incluso per qualsiasi motivo nei canali ufficiali della distribuzione
- Si aggiunge semplicemente il repository all'elenco

- Apt (deb):

```
/etc/apt/sources.list.d/virtualbox.list :  
deb http://download.virtualbox.org/virtualbox/debian xenial contrib
```

- Yum (rpm):

```
/etc/yum.repos.d/epel.repo :  
[epel]  
name=Epel Linux -  
baseurl=http://mirror.example.com/repo/epel5_x86_64  
enabled=1  
gpgcheck=0
```

---

24

# Gestire la provenienza dei pacchetti

- Si può generare confusione se un pacchetto con lo stesso nome è presente in versioni diverse in repository differenti
- I package manager, di default, scelgono sempre la versione più avanzata
- In alcuni casi anche aggiornamenti nello stesso repo sono indesiderabili
- La situazione va controllata e gestita
  - Controllo della provenienza di un pacchetto
    - Yum: `repoquery -i [package name]`
    - Apt: `apt-cache showpkg [package name]`
  - Elenco dei pacchetti provenienti da un repo
    - Yum: `yum list installed | grep [repo name]`
    - Apt: vari comandi per estrarre manualmente info dai file della cache

---

25

# Limitare le modifiche automatiche

- Per evitare a priori problemi in sistemi con dipendenze complesse (ad esempio mix di pacchetti installati manualmente e via package manager)
  - Version locking/pinning
    - Apt
      - editare `/etc/apt/preferences.d/*`
      - <https://wiki.debian.org/AptPreferences>
    - Yum
      - `yum install yum-plugin-versionlock`
      - poi
      - `yum versionlock [package name]`
      - o editare a mano
      - `/etc/yum/pluginconf.d/versionlock.list`

---

26

# Monitoraggio dei processi

## ■ Comandi essenziali per il monitoraggio

Utenti	File	Processi	Spazio
w last	fuser	ps top uptime	df du free
		<b>lsof</b>	<b>vmstat</b> <b>iostat</b>

## ■ Strumenti essenziali per l'automazione dei task di monitoraggio (ma non solo)

- esecuzione posticipata: **at**
- esecuzione periodica: **cron**



# ps – uptime – free → top

## ■ Comandi che scattano un'istantanea del sistema

- ps: stato dei processi
- uptime: carico del sistema
  - unità di misura = lunghezza media coda runnable
- free: occupazione memoria

## ■ Questi comandi sono interfacce verso proc filesystem

## ■ Comandi di monitoraggio interattivi

- top riassume ps, uptime, free, uso dettagliato cpu
- aggiornato regolarmente
- permette di interagire coi processi
- utile per stima intuitiva dello stato di salute

# top

```
9:31am up 50 min,  2 users,  load average: 0.02, 0.02, 0.04
71 processes: 70 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 4.3% user, 5.2% system, 0.1% nice, 90.2% idle
Mem: 384480K av, 380688K used,   3792K free,   1312K shrd,   51312K buff
Swap: 128516K av,        0K used, 128516K free                  139136K cached

 PID USER      PRI  NI   SIZE   RSS   SHARE STAT %CPU %MEM TIME COMMAND
 1179 root      13   0 3092  3092   2592 S    2.8  0.8 0:50 magicdev
 9299 root      16   0 1044  1040   832 R    2.8  0.2 0:00 top
     1 root      8   0   520   520   452 S    0.0  0.1 0:03 init
     2 root      9   0     0     0   0 SW   0.0  0.0 0:00 keventd
     3 root      9   0     0     0   0 SW   0.0  0.0 0:00 kapm-idled
     4 root     19  19     0     0   0 SWN  0.0  0.0 0:00 ksoftirqd_CPU0
     5 root      9   0     0     0   0 SW   0.0  0.0 0:00 kswapd
     6 root      9   0     0     0   0 SW   0.0  0.0 0:00 kreclaimd
     7 root      9   0     0     0   0 SW   0.0  0.0 0:00 bdflush
     8 root      9   0     0     0   0 SW   0.0  0.0 0:00 kupdated
     9 root     -1 -20     0     0   0 SW<  0.0  0.0 0:00 mdrecoveryd
    71 root      9   0     0     0   0 SW   0.0  0.0 0:00 khubd
   465 root      9   0     0     0   0 SW   0.0  0.0 0:00 eth0
   546 root      9   0   592   592   496 S    0.0  0.1 0:00 syslogd
   551 root      9   0 1124  1124   448 S    0.0  0.2 0:00 klogd
   569 rpc       9   0   592   592   504 S    0.0  0.1 0:00 portmap
  597 rpcuser   9   0   788   788   688 S    0.0  0.2 0:00 rpc.statd
```

# Evoluzione delle risorse

## ■ **vmstat** – uso di memoria, paging, I/O, trap

- utile invocarlo col periodo (in secondi) per monitorare

```
root@Client:~# vmstat 1
procs -----memory----- -----swap-- -----io---- -system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 0 0      0 53404 39008 89588 0 0 12 1 12 31 0 0 100 0 0
 0 0      0 53344 39008 89588 0 0 0 0 19 27 0 1 99 0 0
 0 0      0 53344 39008 89588 0 0 0 0 14 19 0 0 100 0 0
```

## ■ **iostat** - statistiche su uso CPU e I/O

- soprattutto per valutare l'uso dei dispositivi di I/O

```
root@Client:~# iostat /dev/sda1
Linux 3.16.0-4-amd64 (Client) 03/21/18 _x86_64_ (1 CPU)
...
Device:    tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
sda1       0.65     11.62        0.81    123899     8668
```

# Spazio disco

## ■ **df** mostra l'utilizzo dello spazio disco:

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
/dev/hdb1	202220	126789	64991	67%	/
/dev/hdb4	5558076	3916612	1359124	75%	/usr
/dev/hdb3	303336	49822	237851	18%	/var
none	192240	0	192240	0%	/dev/shm
/dev/hda1	10231392	9473248	758144	93%	/win/c
/dev/hda5	9790032	4247000	5543032	44%	/win/d

## ■ **du** permette di calcolare lo spazio occupato dai file (in una directory). Senza opzioni particolari du riporta l'occupazione totale delle dir passate come argomento ed anche di tutte le subdir in esse presenti. Es:

```
# du /tmp
1 /tmp/.font-unix
1 /tmp/.X11-unix
1 /tmp/.ICE-unix
5 /tmp/orbit-root
72 /tmp
```

## ■ **du -s** riporta invece il **summary**, senza dettagli sulle subdir.

# Uso dei file

## ■ Quali file sta usando un processo:

- filesystem speciale **/proc**

<https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

```
# ls -l /proc/2208/fd/
total 0
lrwx----- 1 root      root   64 Apr 26 10:11 0 -> /dev/pts/0
lrwx----- 1 root      root   64 Apr 26 10:11 1 -> /dev/pts/0
lrwx----- 1 root      root   64 Apr 26 10:11 2 -> /dev/pts/0
lr-x----- 1 root      root   64 Apr 26 10:11 3 -> /etc/man.config
```

## ■ Quali processi stanno usando un file:

```
# fuser /etc/man.config
/etc/man.config: 2208 2212 2213 2219
```

- o un intero filesystem:

```
# fuser -m /var
/var:          546  597c  714  714c  879c  898  916  916c
  964  1013  1020  1021  1318  6493  9244  9244m  9249  9249m  9275

c   current directory.
e   executable being run.
f   open file. f is omitted in default display mode.
r   root directory.
m   mmap'ed file or shared library.
```

# Uso globale dei file

## ■ Isof – list open files

- elenca tutti i file impegnati da tutti i processi

## ■ opera su tutti i namespace riconducibili al concetto astratto di file

- regular file
- directory
- block special file,
- character special file
- executing text reference
- library
- stream o network file (socket internet o UNIX domain, NFS file)

## ■ Osservazione: un file cancellato (unlink) dopo l'apertura sarà irreperibile sul filesystem, ma referenziato dal processo e quindi visibile a Isof

COMMAND	PID TID	USER	FD	TYPE	DEVICE SIZE/OFF	NODE NAME
automount	862 870	root	2u	CHR	1,3	0t0 5580 /dev/null
automount	862 870	root	3u	CHR	10,59	0t0 9423 /dev/vboxguest
automount	862 870	root	4uW	REG	0,15	4 12174 /run/vboxadd-service.pid
automount	862 870	root	6u	unix 0xffff88000a746bc0	0t0	12175 socket
rpc.mount	623	root	16u	IPv4	11645	0t0 UDP *:34398
rpc.mount	623	root	17u	IPv4	11794	0t0 TCP *:44328 (LISTEN)
rpc.mount	623	root	18u	IPv6	11798	0t0 UDP *:52821
rpc.mount	623	root	19u	IPv6	11802	0t0 TCP *:36162 (LISTEN)
rpc.mount	623	root	20u	unix 0xffff88000a75ab80	0t0	11809 socket

## Esecuzione posticipata - at

- **atd** è un demone che gestisce code di compiti da svolgere in momenti prefissati. L'interfaccia ad **atd** consiste di 4 comandi:
- **at** [-V] [-q queue] [-f file] [-mldbv] TIME  
pianifica un comando al tempo TIME
- **atq** [-V] [-q queue] [-v]  
elenca i comandi in coda
- **atrm** [-V] job [job...]  
rimuove comandi dalla coda
- **batch** [-V] [-q queue] [-f file] [-mv] [TIME]  
esecuzione condizionata al carico

# Esecuzione posticipata - at

- Se non viene specificato un file comandi per at o batch, verrà usato lo standard input.
- La specifica dell'ora è flessibile e complessa. Per una definizione completa si veda la documentazione in /usr/doc/at-<versione>/timespec. Alcuni esempi:

```
echo 'wall "sveglia"' | at 08:00  
echo "$HOME/bin/pulisci" | at now + 2 weeks  
echo "$HOME/bin/auguri" | at midnight 25.12.2018
```

# Esecuzione periodica - cron

- **crond** è un demone che esamina una serie di file di configurazione ogni minuto, e determina quali compiti specificati nei file debbano essere eseguiti.
- I file di configurazione (**crontab**) sono distinti in due insiemi:
  - Uno per utente (/var/spool/cron/<utente>)
    - si visualizza / edita / sostituisce con  
**crontab -l / crontab -e / crontab <nuova\_tab>**
  - System-wide (/etc/crontab)
    - Solitamente quest'ultimo non fa altro che richiamare l'esecuzione di tutto ciò che trova in alcune directory:  
**/etc/cron.hourly/  
/etc/cron.daily/  
/etc/cron.weekly/  
/etc/cron.monthly/**

ha un campo in più rispetto ai file personali per indicare a nome di che utente eseguire ogni task configurato

# Esecuzione periodica - cron

- Ogni crontab contiene un elenco di direttive nella forma

MINUTO ORA G.MESE MESE G.SETTIMANA <comando>

Es.

*	*	27	*	*	\$HOME/bin/paga
30	8-18/2	*	*	1-5	\$HOME/bin/lavora
00	00	1	1	*	/usr/sbin/auguri
30	4	1,15	*	6	/bin/backup

- L'azione è eseguita quando l'ora corrente corrisponde a tutti i selettori di una riga (campi in AND logico)
- ECCEZIONE: se sono specificati (diversi da \*) entrambi i giorni (settimana e mese), i due campi sono considerati in OR logico

# Log di sistema

- I log (diari) tenuti dal sistema sono indispensabili per la diagnostica in generale, e in particolare per rilevare attività malevole o sospette
- La loro stessa sicurezza va garantita!
  - Usare appropriatamente un integrity checker
  - Replicarli su macchine remote
- Logging su server remoto
  - Vantaggio aggiuntivo: centralizzazione
  - Implementazioni avanzate: shadow loggers
  - Problema: diventa un bersaglio appetibile
    - DoS



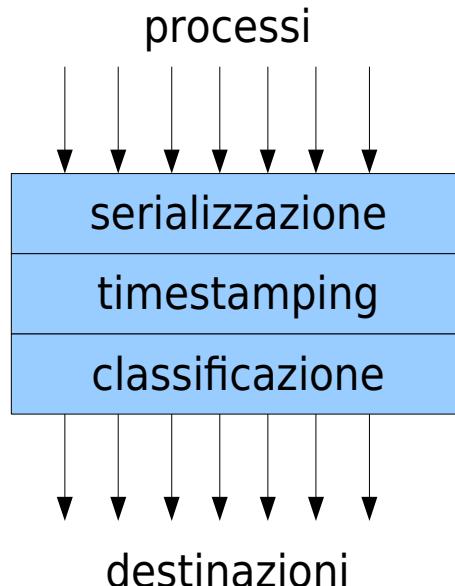
# Linux logging

- Soluzioni comuni
  - Tipicamente producono file di testo
  - Nessuna garanzia di uniformità di formato a parte la marcatura temporale
  - BSD syslog (obsoleto)
    - klogd
  - Rsyslog
  - Syslog-ng
- In prospettiva integrato in systemd
  - Journal
  - Attivo dal boot, non dipende dall'avvio di altri servizi
  - Formato binario, visualizzabile con **journalctl**



# syslog

## ■ Principi di base mantenuti anche dalle evoluzioni



## syslog: selettori e destinazioni

### ■ Ogni messaggio è etichettato con una coppia

**<facility>.<priority>**

- Facility = argomento
  - auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, **local0..local7**
- Priority = importanza in ordine decrescente:
  - emerg, alert, crit, err, warning, notice, info, debug

### ■ Le destinazioni possibili sono

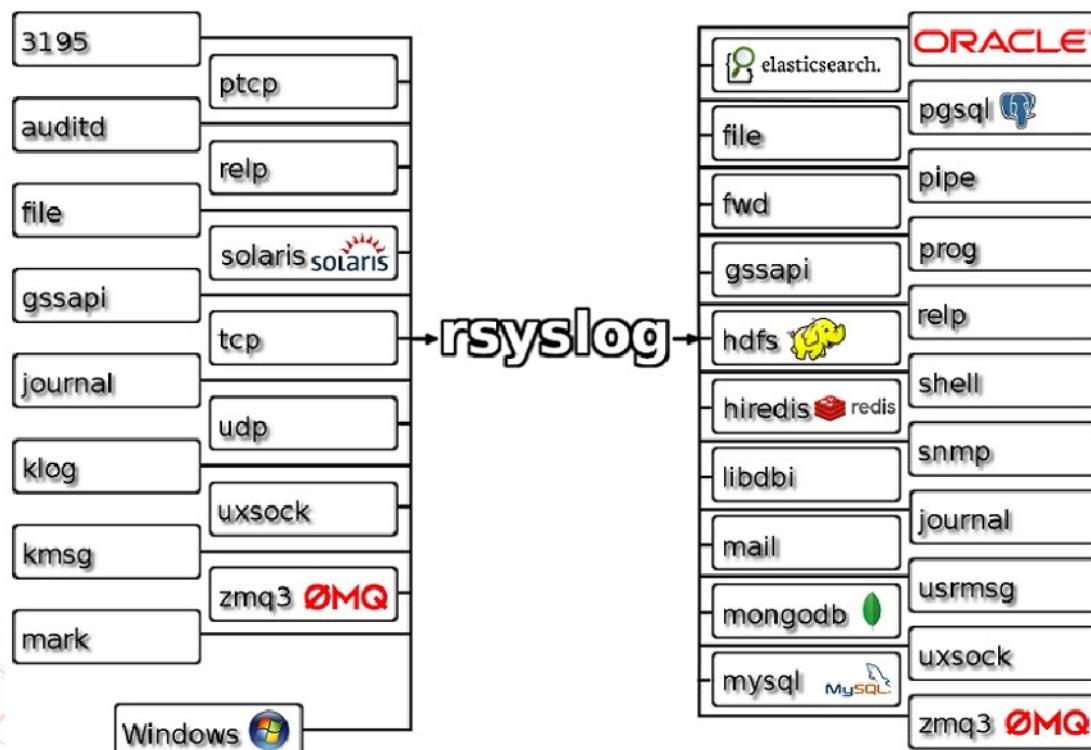
- File: identificato da path assoluto
- STDIN di un processo: identificato da una pipe verso il programma da lanciare
- Utenti collegati: username, o \* per tutti
- Server syslog remoto: @indirizzo o @nome
  - La comunicazione avviene di default su UDP, porta 514

# syslog: selettori

- **/etc/syslog.conf** contiene le regole di smistamento dei messaggi
- Ogni riga = una regola
  - [etichetta di interesse] [destinazione]
  - Parsate tutte, quindi un messaggio può finire su più destinazioni
- Trattamento delle priority
  - Soglia: una regola che specifica una priority fa match con tutti i messaggi di tale priority e superiori a meno che non sia preceduta da “=”
  - Priority speciale *none*: serve per ignorare i messaggi con la facility specificata prima del punto
- Es:

kern.*	/dev/console
*.info;mail.none;	/var/log/messages
*.emerg	*
kern.crit	"/usr/bin/alerter"
*.=warning	@loghost

# rsyslog



# rsyslog

## ■ Struttura modulare per caricare solo le funzioni necessarie

- es. attivazione della ricezione di messaggi via rete (v8.4 / v8-16):

```
$ModLoad imudp          /      module(load="imudp")
$UDPServerRun 514        /      input(type="imudp" port="514")
```

- es. integrazione del kernel logging

```
$ModLoad imklog          /      module(load="imklog")
```

## ■ File di configurazione modulare

- Direttive globali in **/etc/rsyslog.conf**
- Direttive specifiche in file separati sotto **/etc/rsyslog.d**

## ■ Scarto di messaggi (per evitare che vengano catturati da troppi selettori)

- Basta mettere **~** come destinazione

# rsyslog – modalità di output evolute

## ■ Template per definire canali di output

- Possono sostituire le destinazioni in modo più flessibile

- Es:

```
$template apacheAccess,"/var/log/external/%fromhost%/apache/
%msg:R,ERE,1,ZERO:imp:( [a-zA-Z0-9\-.]+)\.--end%-access.log"
local6.notice ?apacheAccess
```

Segnaposto che verrà sostituito per mezzo di una elaborazione del messaggio fatta via regex

Segnaposto che verrà sostituito dal nome dell'host che origina il messaggio

## ■ TCP logging

- Per evitare perdita di messaggi (finché non ci sono crash!)

<http://blog.gerhards.net/2008/04/on-unreliability-of-plain-tcp-syslog.html>

- \*.\* @indirizzo

## ■ Shell execute

- Passa il messaggio come parametro a un programma

- \*.\* ^programma;template

# rsyslog – selettori evoluti

- Rsyslog seleziona i messaggi in tre modi
  - i tradizionali facility.priority
  - Filtri basati su proprietà
  - Filtri basati su espressioni
- Filtri basati su proprietà
  - :property, [!]compare-operation, "value"
  - Es: :msg, !contains, "error" /var/log/good.log
- Filtri basati su espressioni
  - Ancora in evoluzione!
  - if expr then destinazione
  - Diventeranno un sistema completo di scripting, che consentirà di eseguire programmi arbitrari per determinare la destinazione



# rsyslog – moduli

- Troppi per citarli esaustivamente  
<http://www.rsyslog.com/doc/v8-stable/>
- Tra i più interessanti:
  - RELP logging - per garanzia totale di consegna

```
$ModLoad omrelp
*.* :omrelp:indirizzo
```
  - Output su tabelle di database

```
$ModLoad ommysql
```
  - Acquisizione diretta dei messaggi del kernel (sostituisce klogd)

```
$ModLoad imklog
```



# syslog-ng

<https://syslog-ng.org/>

## ■ Flessibile

- Input compatibile con
  - Formati standard syslog (RFC3164, RFC5424)
  - JSON
  - Journald (systemd)
- Output verso molteplici destinazioni
  - Tutti i DB SQL più diffusi
  - DB NOSQL (es. MongoDB)
  - Cloud databases (es. Redis)
- Varietà di protocolli
  - Client-server
  - Message-based (AMQP, STOMP)
- Capacità di elaborazione del contenuto dei messaggi

## ■ E se non basta, estendibile con plugin

- In C, Python, Java, Lua, o Perl

# syslog-ng

## ■ Configurazione:

- Definizione di una **source**, che può unificare più ingressi fisici

```
source s_two {  
    network(ip(10.1.2.3) port(1999));  
    network(ip(10.1.2.3) port(1999) transport("udp"));  
};
```

- Definizione di una **destination**, con relative opzioni

```
destination d_file {  
    file("/var/log/${YEAR}.${MONTH}.${DAY}/messages"  
        template("${HOUR}:${MIN}:${SEC} ${TZ} ${HOST} [${LEVEL}] ${MSG}\n")  
        template-escape(no));  
};
```

- Attivazione di un canale di log

```
log { source(s_two); destination(d_file); };
```

# Analisi e gestione dei log

■ Non basta scrivere gli eventi da qualche parte

■ Analisi

- estrazione del significato dei messaggi
- serie temporali
- correlazione file multipli
- reazione in tempo reale

"Vedi la foresta, ma anche gli alberi"  
- splunk.com

■ Gestione

- spazio
- archiviazione



## Software basici di analisi dei log

	Logwatch	Swatch
Regular expression support	yes	yes
Real-time monitoring	no	yes
Support for multiple log files	yes	no
Good preconfiguration	yes	no
Modular configuration	yes	no
Reactive	no	yes
Interactive	no	no

Logwatch (System log analyzer and reporter):  
<http://www.logwatch.org/>

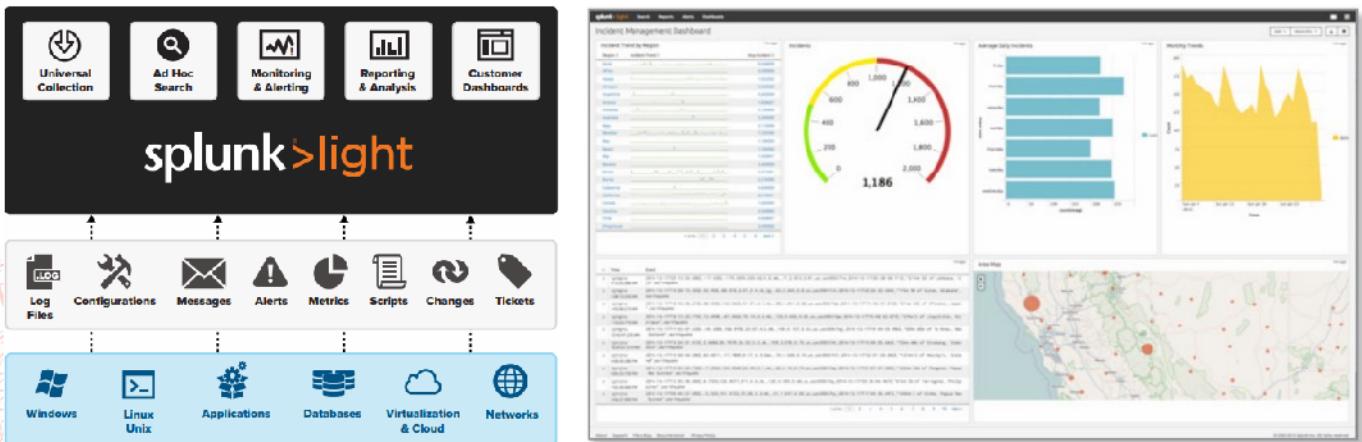
Swatch (Simple WATCHer of Logfiles):  
<http://swatch.sourceforge.net/>



# Sistemi commerciali

## ■ Stanno evolvendo verso gestione integrata

- qualsiasi tipo di "dato macchina"
- algoritmi di machine learning
- reportistica avanzata
- on premise o SaaS



# Analisi dei log di ispirazione cloud

## ■ Stack Elastic, Open Source

- <https://www.elastic.co/>
- Raccolta e esplorazione dei dati di log

## ■ Composto dai tre programmi:

- **Logstash**: Pipeline di elaborazione dei log
- **Elasticsearch**: Database Nosql
- **Kibana**: Visualizzatore web-based per documenti in Elasticsearch



# Gestione dei processi

## ■ Dopo un'installazione "minimale"...

```
milk:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root         1  0.0  0.0  1948   468 ?        Ss May15 0:02 init [2]
[... kernel processes ...]
root      1753  0.0  0.0  2704   392 ?        S<s May15 0:00 udevd --daemon
daemon    2953  0.0  0.0  1688   408 ?        Ss May15 0:00 /sbin/portmap
root      3231  0.0  0.0  1624   568 ?        Ss May15 0:26 /sbin/syslogd
root      3237  0.0  0.0  1576   340 ?        Ss May15 0:00 /sbin/klogd -x
bind     3251  0.0  0.1 39732  1964 ?        Ssl May15 0:00 /usr/sbin/named
root      3266  0.0  0.0 39500   944 ?        Ssl May15 0:00 /usr/sbin/lwres
root      3339  0.0  0.0  1572   444 ?        Ss May15 0:00 /usr/sbin/acpid
103       3344  0.0  0.0  2376   760 ?        Ss May15 0:00 /usr/bin/dbus-d
106       3352  0.0  0.1  6116  1972 ?        Ss May15 0:03 /usr/sbin/hald
root      3353  0.0  0.0  2896   716 ?        Ss May15 0:00 hald-runner
106       3359  0.0  0.0  2016   472 ?        Ss May15 0:00 hald-addon-acpi
106       3367  0.0  0.0  2020   480 ?        Ss May15 0:00 hald-addon-keyb
root      3387  0.0  0.0  1808   360 ?        Ss May15 14:15 hald-addon-stor
root      3414  0.0  0.0  1864   396 ?        Ss May15 0:00 /usr/sbin/dhcdb
root      3421  0.0  0.1  3984  1164 ?        Ss May15 0:00 /usr/sbin/Netwo
avahi     3433  0.0  0.1  2936  1424 ?        Ss May15 4:14 avahi-daemon: r
avahi     3434  0.0  0.0  2552   180 ?        Ss May15 0:00 avahi-daemon: c
root      3441  0.0  0.0  2908   536 ?        Ss May15 0:00 /usr/sbin/Netwo
root      3457  0.0  0.0  1752   452 ?        Ss May15 0:02 /usr/sbin/inetd
root      3477  0.0  0.0  4924   512 ?        Ss May15 0:02 /usr/sbin/sshd
ntp      3507  0.0  0.0  4144   764 ?        Ss May15 0:00 /usr/sbin/ntp
root      3521  0.0  0.0  1976   724 ?        Ss May15 0:02 /sbin/mdadm --m
daemon    3540  0.0  0.0  1828   280 ?        Ss May15 0:00 /usr/sbin/atd
root      3547  0.0  0.0  2196   720 ?        Ss May15 0:00 /usr/sbin/cron
root      3590  0.0  0.0  1572   372 tty2    Ss+ May15 0:00 /sbin/getty 384
root      3591  0.0  0.0  1576   372 tty3    Ss+ May15 0:00 /sbin/getty 384
root      3592  0.0  0.0  1572   372 tty4    Ss+ May15 0:00 /sbin/getty 384
root      3593  0.0  0.0  1572   372 tty5    Ss+ May15 0:00 /sbin/getty 384
root      3595  0.0  0.0  1576   372 tty6    Ss+ May15 0:00 /sbin/getty 384
```

33

# Gestione dei processi

## ■ Anche se tutti questi processi fossero utili, sarebbe importante

- sapere che origine hanno
- sapere come terminarli, evitando che ricompaiano
- **processi inutili non consumano solo risorse, offrono anche opportunità di attacco**

## ■ Banali e fondamentali:

- *man* è il vostro migliore amico, seguito da Internet.
- *ps, top, kill, ...* sono efficaci per individuare e risolvere problemi istantanei, ma non garantiscono che non si ripresenteranno

## ■ Ci sono tre fonti primarie di processi (oltre agli utenti)

- Pianificatori periodici e sporadici
- Demoni di gestione degli eventi
- **Procedure di avvio del sistema**

# Esecuzioni pianificate

## ■ L'esecuzione periodica di programmi è compito di cron

- ogni utente ha la propria **cron table** (**crontab**),
  - guardate in **/var/spool/cron** per trovarle
- i task di sistema sono spesso raccolti in **/etc/crontab**
  - tipicamente preconfigurato per l'esecuzione di script a periodicità di uso comune
  - **/etc/cron.hourly**, **/etc/cron.daily**, **/etc/cron.weekly**,  
**/etc/cron.monthly**
- **/etc/crontab** si può editare direttamente, per le tabelle utente meglio usare  
**crontab -e [-u username]**

## ■ L'esecuzione singola in un istante preciso è compito di at

- **atq** per elencare i job in attesa
- **atrm** per rimuoverli

35

# Event managers / IPC systems

## ■ Dbus è un'architettura di Inter-Process Communication

- Nata per uniformare la comunicazione tra elementi delle interfacce desktop
- Curiosate in **/etc/dbus-1/** per vedere i file di configurazione
- In **/etc/dbus-1/event.d** sono collocati gli script di avvio dei sottosistemi gestiti

## ■ Udev ha rimpiazzato devfs come event manager per la creazione istantanea dei device special file quando un nuovo dispositivo viene connesso; ora è parte di systemd

- In **/etc/udev/rules.d** sono configurate le regole evento → azione

- Es. **70-persistent-net.rules**

```
# PCI device 0x10ec:0x8168 (r8169)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", 
ATTR{address}=="d0:67:e5:18:d9:e4", ATTR{dev_id}=="0x0", ATTR{type}=="1",
KERNEL=="eth*", NAME="eth0"
```

alla comparsa nel subsystem **net** di una scheda con **MAC=d0:67:e5:18:d9:e4** le assegna nome **eth0**

36

# Inizializzazione e attività in background (demoni)

## ■ ***init*** è il primo processo avviato dal kernel

- Gestisce i *runlevel*
  - stati di funzionamento del sistema definiti dal sottoinsieme di servizi attivi
- Orchestra la sequenza corretta di eventi per raggiungere un runlevel
- Intercetta e gestisce alcuni eventi
  - es. ctrl-alt-canc, terminazione anomala di processi,
- Spegne il sistema in modo ordinato

## ■ Tre varianti principali

- (storico) SystemV-style initialization
- Upstart (Canonical, 2006-2014)
- Systemd (ispirazione RedHat, 2010-oggi)

utile da conoscere  
per l'attuale mix  
imprevedibile di  
distribuzioni moderne  
e tradizionali

37

## sysvinit

### ■ **/sbin/init** dell'originale SystemV Unix

- configurato dal file **/etc/inittab**
- *inittab* specifica il default runlevel
  - **id:2:initdefault:**
- ma se la keyword **single** viene passata come parametro al kernel dal boot loader, questo settaggio è scavalcato e il sistema parte in **single user mode** (runlevel 1)
  - **~~:S:wait:/sbin/sulogin**
- *init* avvia i virtual terminal e i gestori delle console su linea seriale (può sembrare un arcaismo, ma nel mondo IoT è tornato alla ribalta)
  - **1:2345:respawn:/sbin/getty 38400 tty1**
  - **2:23:respawn:/sbin/getty 38400 tty2**
  - **...**
  - **T0:23:respawn:/sbin/getty -L ttys0 9600 vt100**
  - **T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttys3**

38

## processi avviati da sysvinit

- **init** è in senso astratto responsabile per tutti i processi che girano sul sistema, ma in particolare due attività possono essere direttamente ricondotte ad esso

- linee del tipo

```
10:0:wait:/etc/init.d/rc 0
```

pilotano il processo di avvio **System-V-style**

- wait = esecuzione sequenziale
  - quando il runlevel obiettivo è 'N', **rc** esegue
    - ogni programma con nome che inizia per 'S' in `/etc/rcN.d/` col parametro **start**
    - ogni programma con nome che inizia per 'K' in `/etc/rcN.d/` col parametro **stop**
  - per evitare l'inutile duplicazione degli script di avvio e arresto dei demoni, questi sono tutti raccolti in `/etc/init.d/`, e symlinked dalle 7 directory `/etc/rcN.d/`

- linee del tipo

```
x:5:respawn:/usr/X11/bin/gdm
```

avviano il programma specificato come 4° campo, e **init** monitora il processo per riavviarlo (**respawn**) se termina

39

## Controllo del sistema con sysvinit

- Configurazione persistente (applicata automaticamente all'avvio)
  - **chkconfig** (RedHat) o **update-rc.d** (Debian) configurano i runlevel gestendo i symlink nelle 7 directory
- Verifica del runlevel attivo
  - **runlevel**
  - restituisce il precedentemente attivo e l'attuale
- Cambio di runlevel
  - **telinit N**
- Avvio, arresto e verifica dello stato dei singoli servizi
  - `/etc/init.d/scriptname {start|stop|status}`
  - supportati da alcuni script **reload**, **restart**, **condrestart**, ...

40

# Upstart (principalmente Ubuntu)

- Un rimpiazzo per *init* basato sulla logica a eventi
  - Inizializzazione dei sottosistemi parallela e non bloccante
  - Gestione omogenea di tutti gli eventi asincroni
    - Aggiunta e rimozione di hardware
    - Avvio e arresto di processi
  - Inizializzazione multi-stadio (es. rilevazione hardware → caricamento firmware → attivazione device → rilevazione delle caratteristiche del device)
  - In prospettiva, integrazione dei pianificatori (cron, at)
- Distribuzioni principali che lo adotta(va)no
  - Ubuntu 6.10 – 14.10
  - Fedora 9 – 14
  - Debian (opzione)
  - Nokia's Maemo platform
  - Palm's WebOS
  - Google's Chromium OS
  - Google's Chrome OS

41

## Qualche concetto di base su upstart

- Filosofia (dal sito):
  - Task e Servizi sono avviati e arrestati in seguito a eventi
  - Il completamento di un avvio/arresto genera a sua volta un evento
  - Gli eventi possono essere ricevuti da qualsiasi processo sul sistema
  - I Servizi possono essere riavviati se terminano inaspettatamente
  - La supervisione e il riavvio di un demone è gestita anche nel caso sia un processo figlio separato dal progenitore
  - La comunicazione avviene via D-Bus
- Operativamente
  - La directory `/etc/init` contiene un file per definire ogni attività
  - Il demone `init` continua ad essere l'orchestratore del sistema
    - ogni modifica ai file di configurazione è rilevata via inotify e applicata in tempo reale
  - Il comando `initctl` interagisce con le attività mandando segnali appropriati (documentati nei sorgenti in `event.h`) a `init` (via sotto-comandi):
    - `start / stop / status`
    - `list / emit / reload-configuration`

42

# Systemd (ispirato da RedHat – ora molto diffuso)

## ■ Che aspetti affronta systemd?

- Dipendenze tra servizi
- Avvio a richiesta di servizi
- Logging precoce
- Conservazione dell'output dei demoni
- Tracciamento dei cgroup (per controllo preciso risorse hardware)
- Tracciamento e gestione dei mount point
- Snapshots di sistema e loro ripristino
- Gestione delle impostazioni globali come hostname, locale, ecc.
- Ambiente deterministico di esecuzione dei servizi
- Aggiornamenti del sistema offline (al riavvio)
- Processo di boot più rapido e senza shell interattive

43

# Systemd

## ■ Systemd si propone di sostituire

- init (etc.)
- udev
- pm-utils
- inetd
- acpid
- crond/atd
- ConsoleKit
- automount
- watchdog
- syslog

44

# Systemd – termini

- Diversi tipi di **[control] unit** i cui nomi seguono la convenzione `name.type`
- **type** può essere:
  - **Service**: controllo e monitoraggio dei demoni
  - **Socket**: attivazione di canali IPC di ogni tipo (file, net socket, Unix socket)
  - **Target**: gruppo di unit che **rimpiazza il concetto di runlevel**
  - **Device**: punti di accesso ai dispositivi, creati dal kernel in seguito a interazioni con l'hardware
    - filesystem-related: **Mounts**, **Automounts**, **Swap**
  - **Snapshots**: stato salvato del sistema
  - **Timers**: attività legate al tempo (→ cron, at)
  - **Paths**: monitoraggio del contenuto di una directory via inotify
  - **Slices**: gestione delle risorse via cgroup
  - **Scopes**: raggruppamento di processi per miglior organizzazione

45

# Systemd – dove trovare le definizioni delle unit

- “libreria” di definizioni di riferimento
  - `/lib/systemd/system`
- File forniti dai mantainer dei diversi pacchetti software
  - Quasi sempre link alle definizioni di riferimento
  - `/usr/lib/systemd/system`
- File con le personalizzazioni
  - prioritari rispetto alle definizioni di sistema sopra elencate
  - `/etc/systemd/system`

46

# Systemd – operazioni base

## ■ Controllo a run time dei servizi

- `systemctl {start|stop|status|restart|reload} servicename`
  - ...intuitivo
  - output molto descrittivo dello stato
    - stato corrente ed elenco dei passi fatti per raggiungerlo
    - process tree
    - righe di log rilevanti
  - “`-H [hostname]`” si connette a un host remoto via ssh

## ■ Configurazione persistente dei servizi al boot

- `systemctl {enable|disable|mask|unmask} servicename`
  - `disable` lascia disponibile la possibilità di usare manualmente `start`
  - `mask` "neutralizza" l'intera definizione della unit, impedendo anche il controllo manuale

47

# Systemd – verifica della configurazione

## ■ Solo qualche esempio

- `systemctl list-units`
  - mostra tutte le *unit* gestite (di tutti i tipi elencati!)
- `systemctl -t type`
  - es.: `systemctl -t timers`
  - mostra tutte le *unit* attive del tipo specificato
- `systemctl list-unit-files [-t type]`
  - es.: `systemctl list-unit-files -t services`
  - mostra tutte le *unit* installate del tipo specificato
- `systemctl --state state`
  - es.: `systemctl --state failed`
  - mostra tutte le *unit* che si trovano nello stato specificato

48

# Systemd – avvio

## ■ I runlevel sono rimpiazzati dai target

/etc/inittab non è più utilizzato

- il target di default è visualizzabile/impostabile con
  - `systemctl get-default`
  - `systemctl set-default [target]`
  - es.: `systemctl set-default graphical.target`

## ■ Equivalenze

- Esplorate /lib/systemd/system

Runlevel	Systemd Target	Description
0	poweroff.target, runlevel0.target	System halt
1	rescue.target, runlevel1.target	Single user mode
3 (2,4)	multi-user.target, runlevel3.target	Multi-user, non graphical
5	graphical.target, runlevel5.target	Multi-user, graphical
6	reboot.target, runlevel6.target	System reboot

49

# Systemd – avvio

## ■ Cosa fa un target? Dalla man page `systemd.target` (5):

“Target units [...] exist merely to group units via **dependencies** (useful as boot targets), and to establish **standardized names** for synchronization points used in dependencies between units.”

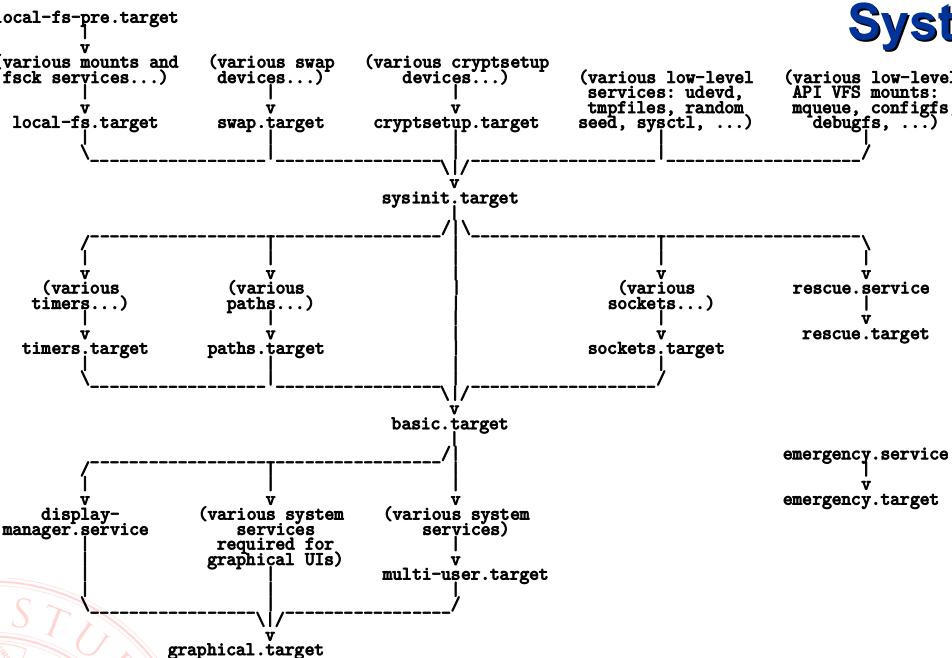
## ■ Dipendenze = automazione robusta

- Sysvinit = sequenziale → lento, nessuna gestione errori
- Systemd = parallelo condizionato → ogni unit parte non appena sono rispettati i vincoli espressi dalle direttive:
  - `Requires`
  - elenco di altre unit da avviare quando questa è avviata/fermata: se l'avvio di tali unit fallisce, questa viene arrestata; si può configurare la relazione temporale (dopo, prima, simultaneamente)
  - `Wants`
  - versione più soft di Requires (il fallimento delle dipendenze non blocca l'avvio di questa unit)
  - `Conflicts`
  - vincolo negativo per rendere unit mutuamente esclusive
  - `OnFailure`
  - unit da avviare quando questa fallisce
  - `RequiredBy / WantedBy`
  - crea automaticamente entry Requires/Wants nelle unit elencate quando questa viene installata
  - `Restart`
  - riavvia il servizio in caso di terminazione; è l'equivalente del respawn di inittab, ma con una varietà ricca di ulteriori sotto-parametri per controllare sotto quali condizioni effettivamente eseguire il riavvio

<https://www.freedesktop.org/software/systemd/man/systemd.service.html>

50

# Systemd – avvio



## Unit speciali

- Alcune unit sono predefinite con nomi fissi e funzioni fondamentali
  - Principalmente target, e alcune slice (vedi [systemd.special\(7\)](#) e [bootup\(7\)](#))
  - Es. punti di controllo della sequenza di boot, che punta a [default.target](#)  
default.target sarà un link a uno dei "veri" target disponibili

51

## Cheat sheet

	SysVinit <small>(Debian) (RedHat)</small>	Upstart	Systemd
<b>Start service</b>	/etc/init.d/name start	service name start	systemctl start name
<b>Stop service</b>	/etc/init.d/name stop	service name stop	systemctl stop name
<b>Status check</b>	/etc/init.d/name status	service name status	systemctl status name
<b>Enable service start at boot</b>	update-rc.d name enable chkconfig name on	rm /etc/init/name.override	systemctl enable name
<b>Inhibit service start at boot</b>	update-rc.d name disable chkconfig name off	echo manual > /etc/init/name.override	systemctl disable name
<b>List installed services</b>	ls /etc/init.d chkconfig --list	service --status-all && initctl list	systemctl list-unit-files -t services
<b>List services starting at boot</b>	ls /etc/rcX.d/S* chkconfig --list   grep X:on	Give up and upgrade to Systemd.	systemctl list-unit-files -t services --state=enabled

X = runlevel di default

`service` e `systemctl` sono stati introdotti dai rispettivi sistemi ma sono wrapper retrocompatibili (in alcuni sistemi c'è un mix di demoni gestiti in 2 o tutti i 3 modi) es. se `systemctl start name` non trova la unit name, prova `service name start`, così come questo proverebbe `/etc/init.d/name start` in caso di assenza di configurazione upstart

Assunzione standard:  
i servizi installati sono configurati per partire al boot