CAPITOLO 2

BREVE STORIA DEI SISTEMI OPERATIVI

L’evoluzione dei S.O.:

- è stata influenzata dai progressi dell’hardware.
- ha guidato il progresso dell’hardware (esempi: gestione degli interrupt, memoria virtuale, protezione della memoria).

Vediamo i S.O. relativi a 4 generazioni di computer:

- Generazione 1, 1945-1955, tubi a vuoto.
- Generazione 2, 1955-1965, transistor.
- Generazione 3, 1965-1980, circuiti integrati.
- Generazione 4, dal 1980, PC.

Siamo interessati alla storia dei S.O. e/o dei calcolatori?

NO! O, per lo meno, non nel corso di S.O.

Però certe scelte/caratteristiche/funzionalità degli attuali sistemi operativi

- sono state proposte già negli anni ’60 (Esempi: interrupt, semafori, …..)
- sono più facili da capire se si ha chiaro il contesto storico nel quale sono stati introdotti
- sono più facili da capire se si ha chiaro quanto accadeva prima che fossero introdotte
- sono più facili da capire se si ha chiaro il perché siano state introdotte.

Generazone 1 – tubi a vuoto.

- Uso: un gruppo di persone progetta, costruisce, programma e manutiene le macchine, usate per calcoli scientifici (niente email, pagine web, basi di dati, documenti testuali, …..).
  I calcolatori occupano interi piani di edifici.

- Hardware: tubi a vuoto + tavole di commutazione.

- Programmazione: solo linguaggio macchina (no assembly o C, C++, Java, Lisp,….), oppure cablando circuiti.

- S.O.: nessuno. Non c'è nessuna macchina virtuale sopra la macchina fisica:
  - gira un solo programma per volta,
  - il programma ha a disposizione tutte le risorse fisiche,
  - il programma deve crearsi le risorse logiche,
  - il programma deve controllare tutti i dispositivi fisici senza intermediari.

Generazione 2 – transistor.

- Uso: Le macchine vengono prodotte in serie, anche se in pochi esemplari, e vendute!

  Distinzione tra progettista, costruttore, operatore, programmatore-utente, manutentore.

  Le macchine sono ancora impiegate prevalentemente per calcoli scientifici, e l'utente è il programmatore.

- Hardware: transistor. I calcolatori occupano stanze.

- Programmazione: in Fortran o assembly, con programmi "incisi" su schede perforate. Esistono il compiler Fortran e l'assembler per tradurre Fortran e assembly in linguaggio macchina.

- S.O.: dopo un periodo di assenza, vengono introdotti i sistemi batch, detti anche monitor residenti o batch monitor (esempi: IB-SYS, Fortran Monitor System).

Dispositivi per macchine di generazione 2: lettori di schede e stampanti (no hard disk, usb, tastiere, monitor, floppy,…….)

Si interagische con la macchina caricando le schede a mano nel lettore di schede e prelevando le stampe.

Esecuzione del programma prima dell'avvento dei S.O.:

- Incisione del programma sorgente sulle schede perforate.
- Caricamento del programma in memoria tramite il lettore di schede. Se il programma è in Fortran/assembly, va caricato tramite il lettore di schede anche il compiler/assembler.
- Se il programma è in Fortran o assembly, esecuzione del compiler o assembler per ottenere l'eseguibile.
- Esecuzione del programma eseguibile.
- Raccolta dell'output dalla stampante.

Sistema **batch**. Il programma su schede è detto **job**, vengono introdotti i nastri magnetici.

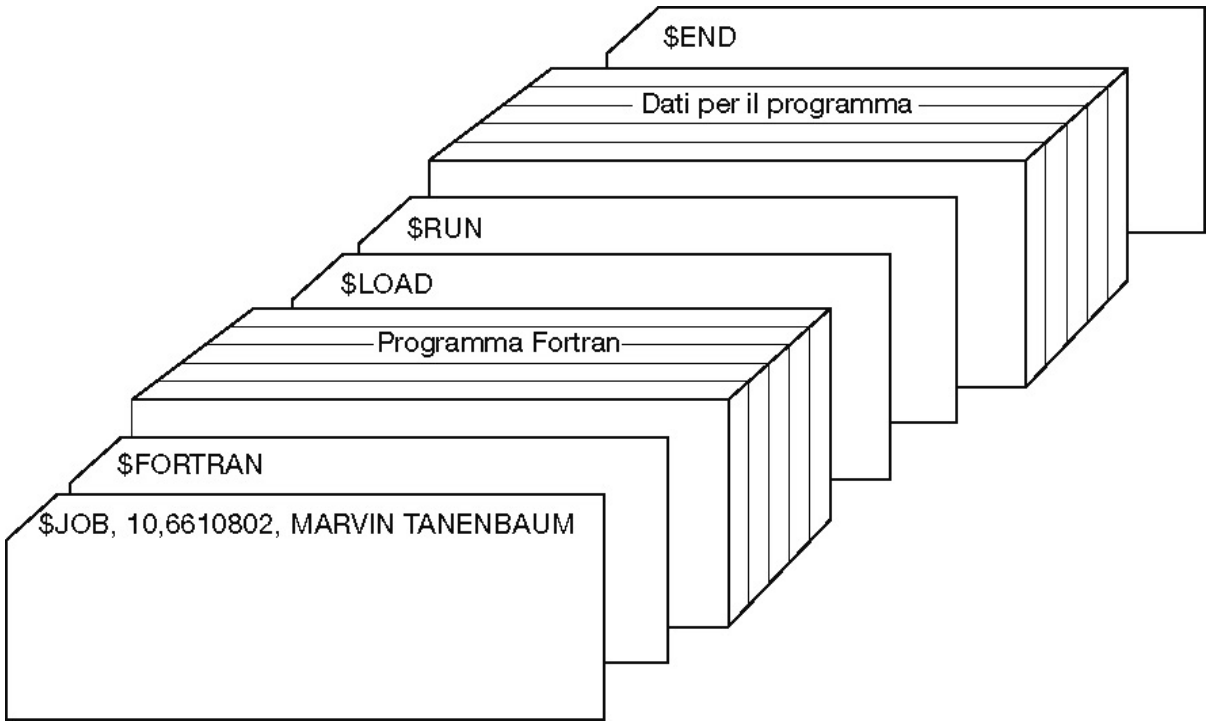
- alcune schede contengono indicazioni al S.O.: "carica il compiler", "carica l'eseguibile", "esegui"… Sono trattate dal command processor del S.O.. Ricordano le system call.

- un **batch** di job viene caricato su un nastro magnetico usando una macchina dedicata (economica).

- il S.O. esegue i job del batch uno alla volta, caricandoli dal nastro. I risultati vengono inseriti in un altro nastro. Il S.O. sfrutta anche un nastro di sistema, che contiene per esempio il compiler Fortran e l'assembler.

- Su un'altra macchina dedicata (economica) i risultati dei job del batch contenuti nel nastro vengono stampati su carta.

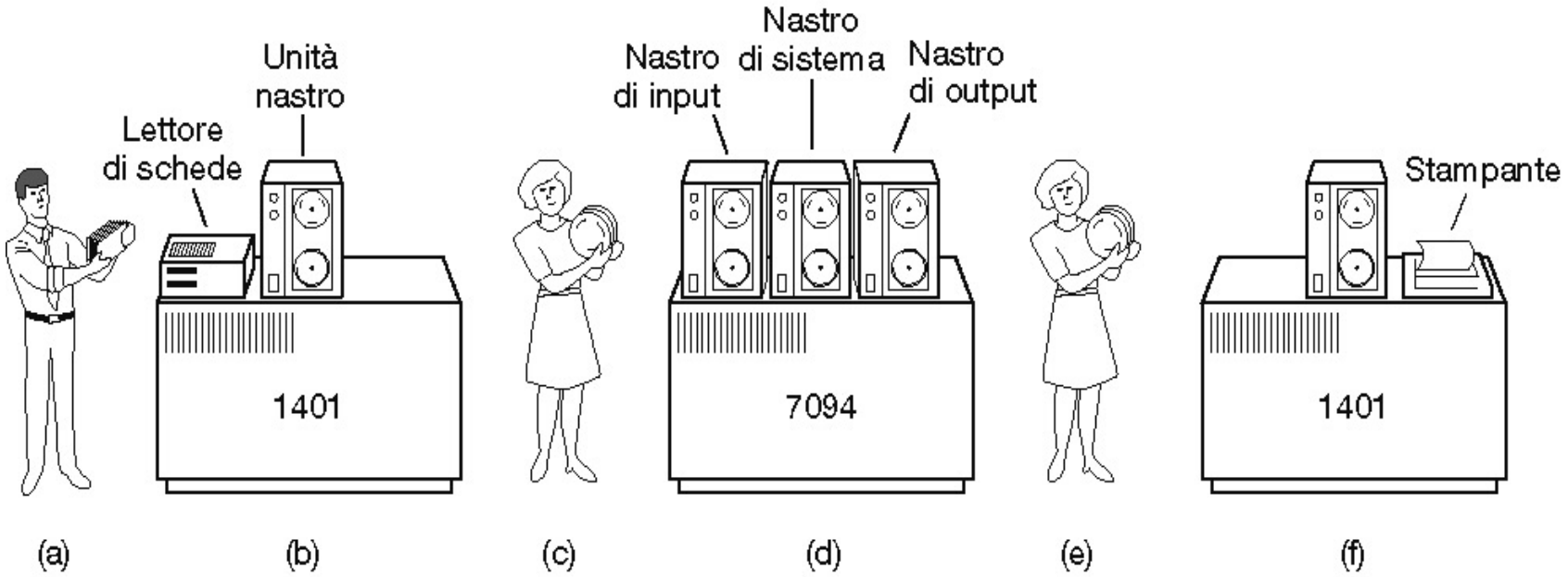
Esempio di programma in Fortran (Tanenbaum - Fig. 1.4)



Indicazioni in Job Control Language per il S.O. Fortran Monitor System:

$JOB: inizio del job
$FORTRAN: carica il compiler
$LOAD: carica l'eseguibile
$RUN: esegui sui dati seguenti
$END: fine del job

Tanenbaum, Fig. 1.3:



Preparazione del batch di job su nastro

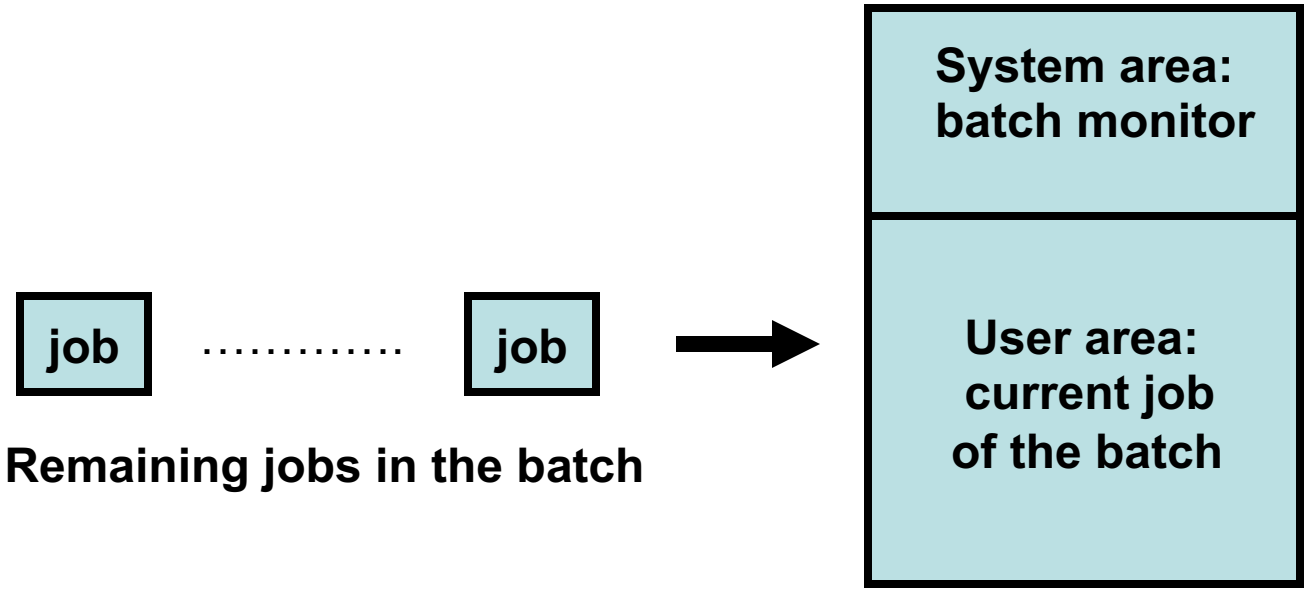
Esecuzione dei job del batch

Stampa dei risultati dei job del batch.

IBM 7094: macchina "potente" introdotta da IBM nel 1961, su cui gira il S.O. batch IB-SYS.

IBM 1401: macchina "economica" introdotta da IBM nel 1959.

Schema di esecuzione del batch di job:



Una zona di memoria, detta system area, è riservata al S.O..
Un'altra zona di memoria, detta user area, ospita un solo job alla volta, cioè il job in esecuzione.

N.B. Il batch non è una struttura di un utente: i job del batch sono indipendenti l'uno dall'altro e, in generale, appartengono ad utenti diversi.

Effetti dell'introduzione dei S.O. batch:

- Grazie ai batch, diminuiscono i tempi di inutilizzo della CPU: i job del batch non vanno caricati a mano uno alla volta dall'operatore, ma vengono caricati dal nastro.

- Per il singolo utente, il tempo di turn-around, cioè il tempo intercorso tra la sottomissione del job e la restituzione dei risultati, può però dilatarsi: i batch vengono preparati quando si ha un numero sufficiente di job.

- Riassumendo: ***i sistemi batch permettono un uso più efficiente della CPU, ma non sono finalizzati a migliorare il servizio all'utente***.

  Si privilegia l'uso efficiente della risorsa CPU, che è la più costosa (almeno nel periodo storico dei sistemi batch), a danno della user convenience.

Funzioni del sistema batch:

- Scheduling (allocazione della CPU ai processi): banale – implicito quando si forma il batch.

- Gestione memoria: banale - al momento del booting, una parte viene riservata al S.O. e l'altra al job corrente.

- Protezione: anche se due job non possono coesistere in memoria, un minimo di protezione è necessaria. Si pensi al caso in cui un job richieda 10 schede di dati e ne vengano fornite solo 6: va evitato che le altre 4 vengano "rubate" al job seguente. Va evitato che un il job acceda alla system area. Come? E.g. tecnica LBR-UBR presentata in seguito.

- Interazione con l'utente: è sufficiente eseguire i comandi inviati al command processor.

Generazione 3 – circuiti integrati:

- Uso: Le macchine vengono ancora usate per calcoli scientifici, che richiedono parecchia CPU, ma anche per elaborazione di dati commerciali, che richiedono meno CPU ma più I/O.

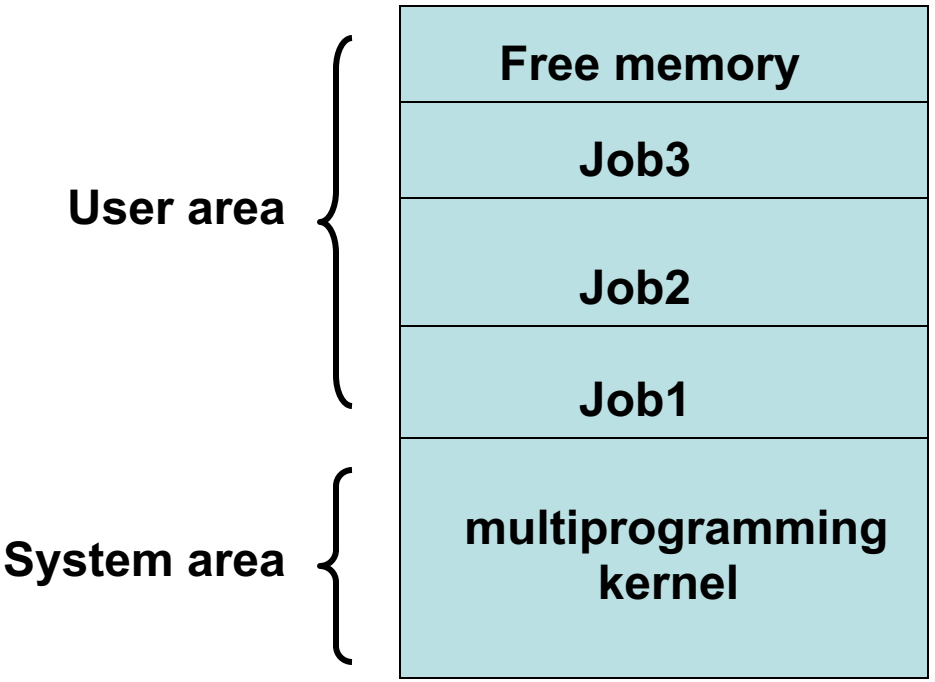
- Hardware: circuiti integrati, introduzione degli hard disk e dei terminali (terminale = tastiera + video).

- Programmazione: linguaggi ad alto livello (C, script di shell,..), favorita dall'introduzione degli editor.

- S.O.: **interattivi**, con **multiprogrammazione** o **timesharing**.

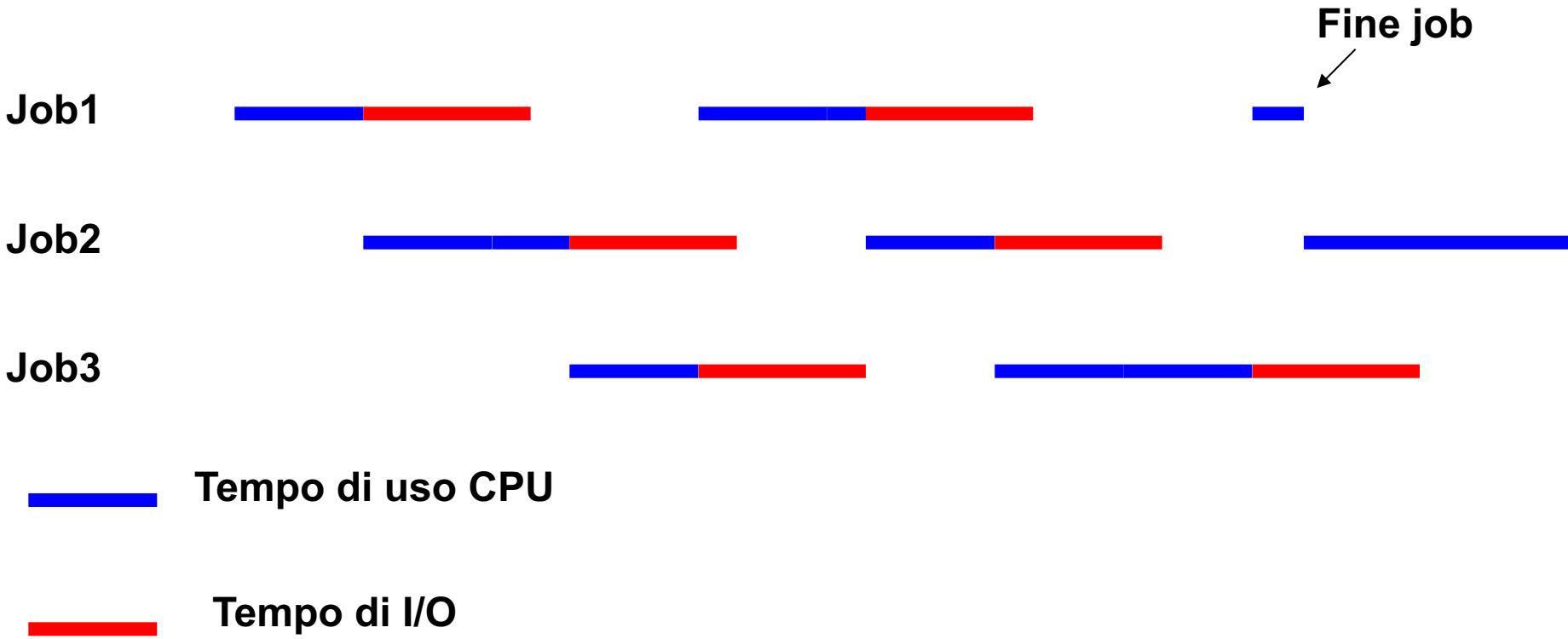
Multiprogramming:

- Più job vengono caricati in memoria, che viene quindi partizionata:



- Quando un job inizia un'operazione di I/O, la CPU non rimane inattiva (idle), ma viene assegnata ad un altro job.
- L'hardware deve consentire a CPU e dispositivi di lavorare in parallelo.

Esempio di esecuzione (imprecisa, vedi Pag. 28)



• Quando un job ottiene la CPU, la mantiene finchè non inizia un'operazione di I/O.
• Questo schema richiede di avere più job contemporaneamente in memoria.

Multiprogrammazione – CPU scheduling:

- Quando il job in esecuzione avvia un'operazione di I/O, la CPU deve essere assegnata ad un job tra quelli in memoria non impegnati in attività di I/O.

- E' necessaria una **politica di scheduling** per scegliere il job da eseguire tra quelli eseguibili, cioè tra quelli caricati in memoria che non stanno eseguendo I/O.

Multiprogrammazione - gestione memoria:

- ogni job deve poter accedere ai propri dati,

- nessun job deve riuscire ad accedere ai dati degli altri. L'hardware deve offrire meccanismi a supporto di tale compito, che studieremo nel Cap. 7 (Gestione Memoria).

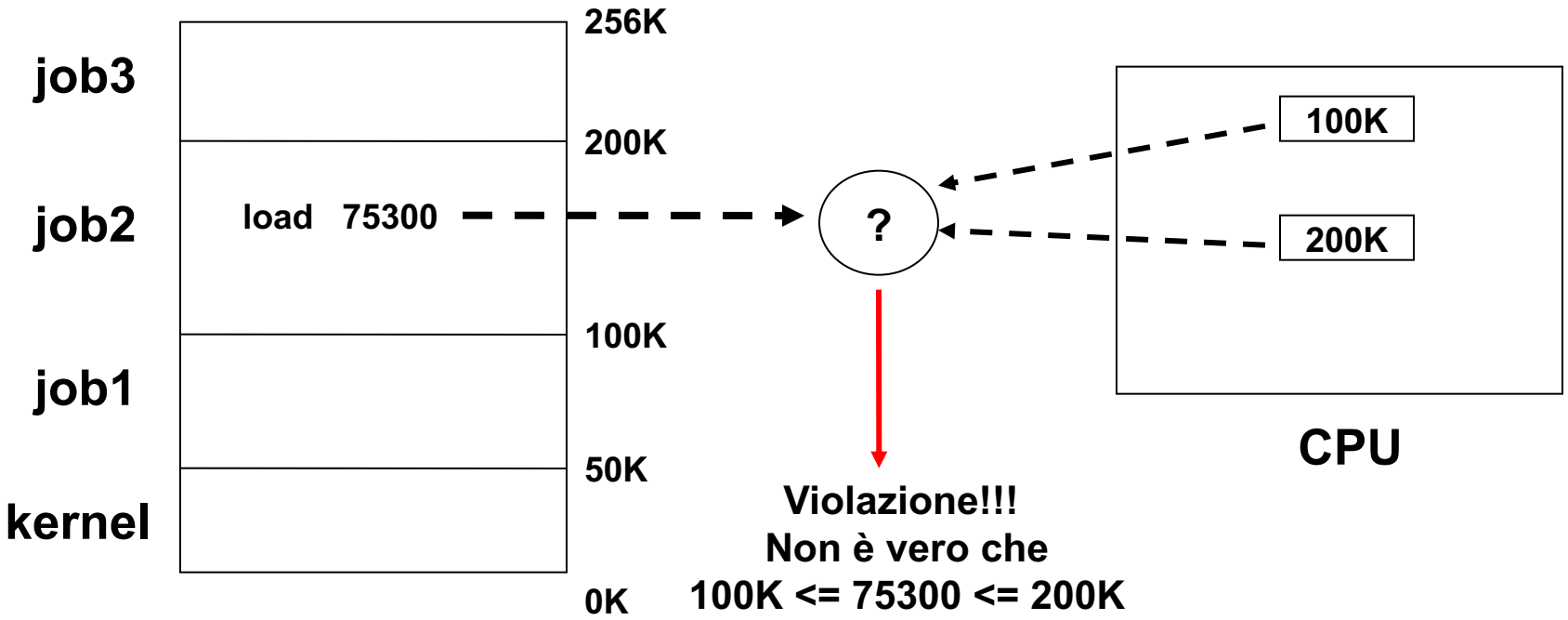
  Esempio di tecnica banale:

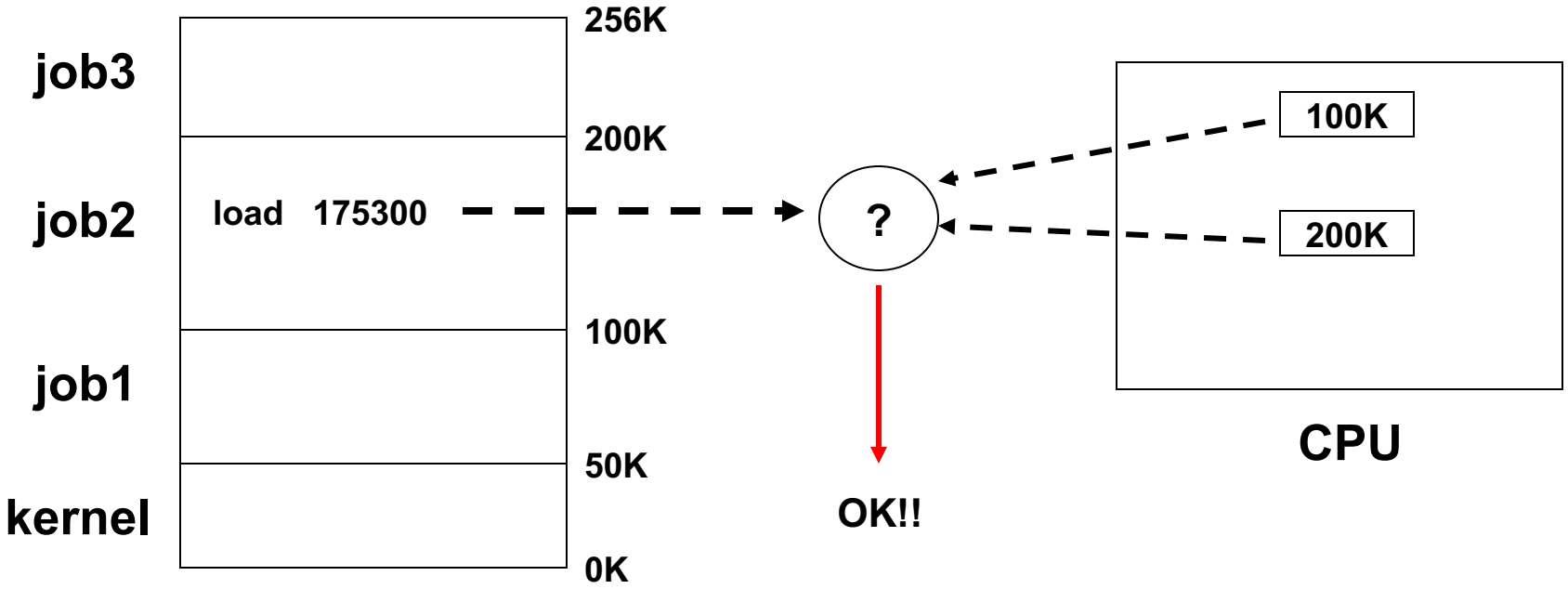
  - la CPU ha i registri **L**ower **B**ound **Re**gister (**LBR**) ed **U**pper **B**ound **R**egister (**UBR**);

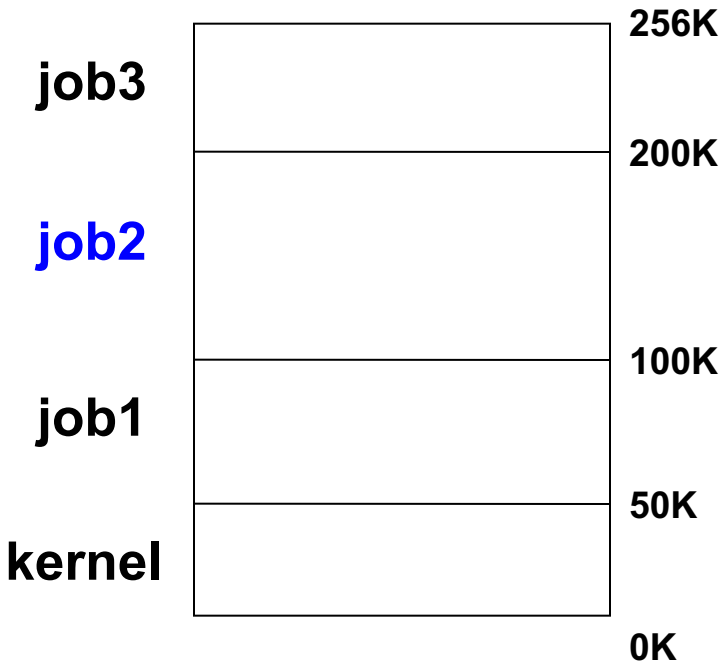
  - gli indirizzi non compresi tra LBR e UBR non sono validi, cioè causano un'***interruzione*** del programma (dettagli nel Cap. 3);

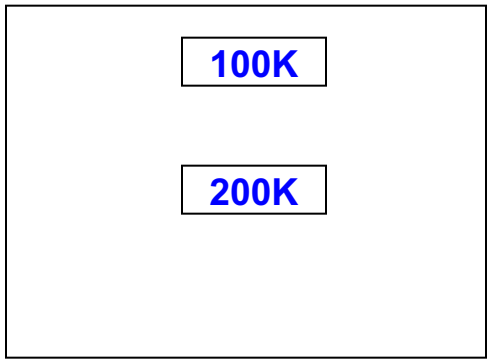
  - le istruzioni per modificare i valori di LBR e UBR sono privilegiate (disponibili solo in modalità kernel);

  - i valori di LBR e UBR vengono modificati dal S.O. quando inizia l'esecuzione di un job (Perché?).

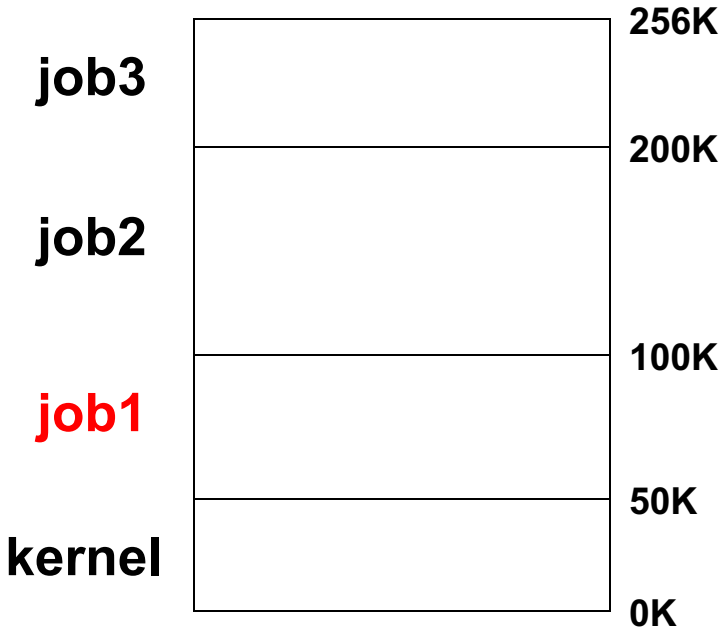

job3
job2
job1
kernel
load 75300
256K
200K
100K
50K
0K
?
100K
200K
CPU
Violazione!!!
Non è vero che
100K <= 75300 <= 200K

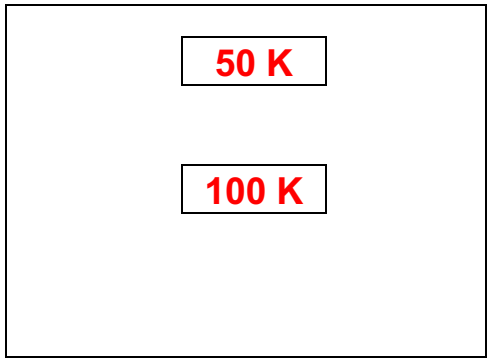

job3
job2
job1
kernel
load 175300
256K
200K
100K
50K
0K
?
100K
200K
CPU
OK!!





CPU
mentre esegue
job 2





CPU
mentre esegue
job 1

Multiprogrammazione - gestione dispositivi.

- ogni job deve poter usare i dispositivi;

- nessun job deve poter interferire sull'uso dei dispositivi da parte degli altri job.

Due modalità principali:

- **partitioning**: ad ogni job vengono assegnati i dispositivi staticamente. Esempio: una porzione riservata del disco. Uso poco efficiente dei dispositivi.

- **pooling**: ad ogni job vengono assegnati i dispositivi dinamicamente.

Uso più efficiente dei dispositivi, ma il S.O. è più complesso.

Multiprogrammazione – requisiti hardware:

- L'hardware deve consentire alla CPU ed ai dispositivi di I/O di lavorare in parallelo (cioè contemporaneamente).

  Ciò si realizza sfruttando il **DMA** (**Di**rect **M**emory **A**ccess), come descritto nel Cap. 3.

- Quando un dispositivo di I/O ha terminato di eseguire un'operazione per conto di un job, il job deve essere inserito nell'elenco dei job schedulabili: ciò si realizza grazie alla tecnica dell'**interrupt**, come descritto nel Cap. 3.

Definizione: Il rapporto tra il numero di programmi eseguiti ed il tempo si chiama **throughput**.

Programma **CPU-bound**: uso intenso CPU, poco I/O.

Programma **I/O bound**: tanto I/O, poco uso CPU.

Per ottimizzare il throughput, è opportuno che lo **scheduler** (la componente del S.O. che effettua lo scheduling) privilegi i programmi I/O bound rispetto a quelli CPU-bound, per far si che siano frequenti gli usi concorrenti di I/O e CPU.

Privilegiare i programmi CPU-bound non consentirebbe di migliorare il throughput rispetto ai sistemi batch (Perché?).

Con i sistemi multiprogrammati vengono introdotti due concetti:

- **program priority**: ad ogni programma è assegnata una priorità, che viene considerata in fase di scheduling per scegliere il programma da schedulare;
- **preemption**: in alcuni casi, la CPU viene tolta forzatamente ad un programma in esecuzione (questo non era previsto nell'esempio a Pag. 14).

Per migliorare il troughput:

- i programmi CPU-bound devono avere priorità inferiore rispetto ai programmi I/O-bound;
- quando un programma I/O-bound termina un'operazione di I/O, se è in esecuzione un programma CPU-bound questo è preempted (subisce la preemption).

L'introduzione degli hard disk consente alla multiprogrammazione di arricchirsi con lo **spooling**:

SPOOL: **S**imulaneous **P**heripheral **O**peration **O**n Line.

- I programmi possono essere caricati su disco: non appena un job termina e libera una partizione di memoria, il S.O. può assegnarla ad uno dei job su disco, che diventa pertanto eseguibile.

  Quale job andrebbe scelto? Per migliorare il troughput, è bene avere sempre sia programmi CPU-bound sia programmi I/O bound in memoria.

- Anche i dispositivi di output (e.g. stampanti) possono essere gestiti in modo analogo.

Di fatto, non servono più le macchine dedicate alla preparazione dei batch ed alla stampa dei risultati.

Timesharing.

- Estensione della multiprogrammazione: ad ogni job viene assegnato un quanto di tempo (**time slice**), allo scadere del quale la CPU viene assegnata ad un altro job anche in assenza di I/O o preemption.

- Adatto per sistemi dotati di più terminali su cui lavorano utenti diversi: Ogni utente ha l'impressione di interagire continuamente con la macchina, in quanto nessun job di altri utenti può monopolizzare la CPU. (Ricordiamo che siamo in un periodo storico in cui la CPU è estremamente costosa e va condivisa da vari utenti).

- Il S.O., con l'aiuto dell'hardware, deve garantire la protezione dei dati di ogni singolo utente.

- Il timesharing penalizza il troughput, ma migliora i tempi di risposta offerti agli utenti interattivi, per esempio utenti che compilano ripetutamente programmi e li testano inserendo dati da tastiera. Si va a privilegiare la user convenience rispetto all'efficienza dell'uso delle risorse.

Lo scheduler non può essere basato sulle priorità, per evitare che i programmi di alcuni utenti vengano penalizzati. I programmi vanno schedulati introducendo il concetto di **turno**, con la tecnica del **round robin.** Non indaghiamo oltre.

Riassumiamo con parole chiave:

Multiprogrammazione: priorità, preemption, troughput.

Timesharing: round robin, time slice, tempi di risposta.

Osservazione:

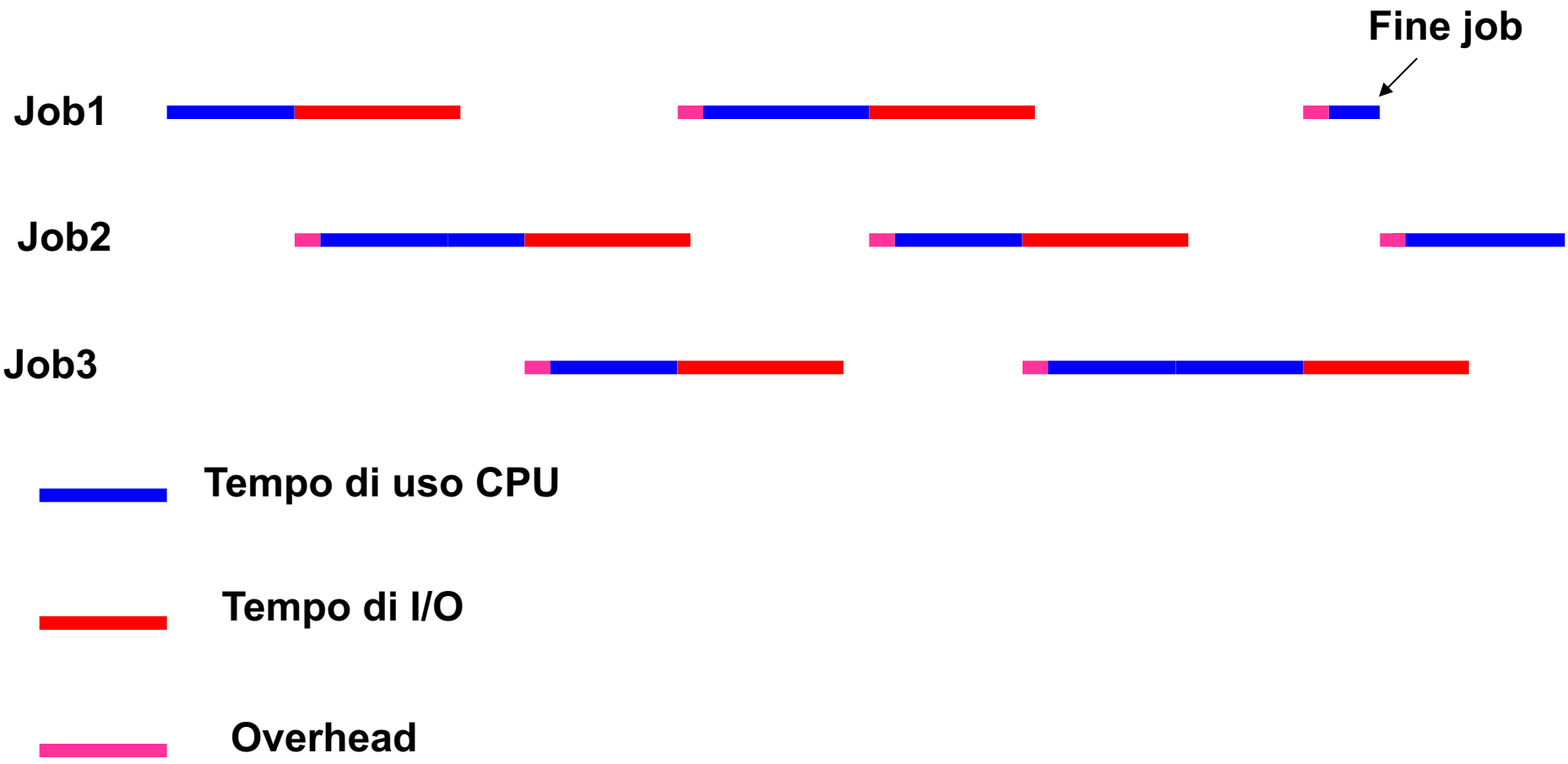
Sottrarre la CPU ad un job ed assegnarla ad un altro (**context switch**) è un'operazione costosa in termini di tempo. E' necessario:

- schedulare il job, tenendo conto delle priorità/turno
- effettuare alcune operazioni che studieremo nel dettaglio nel Cap. 4.

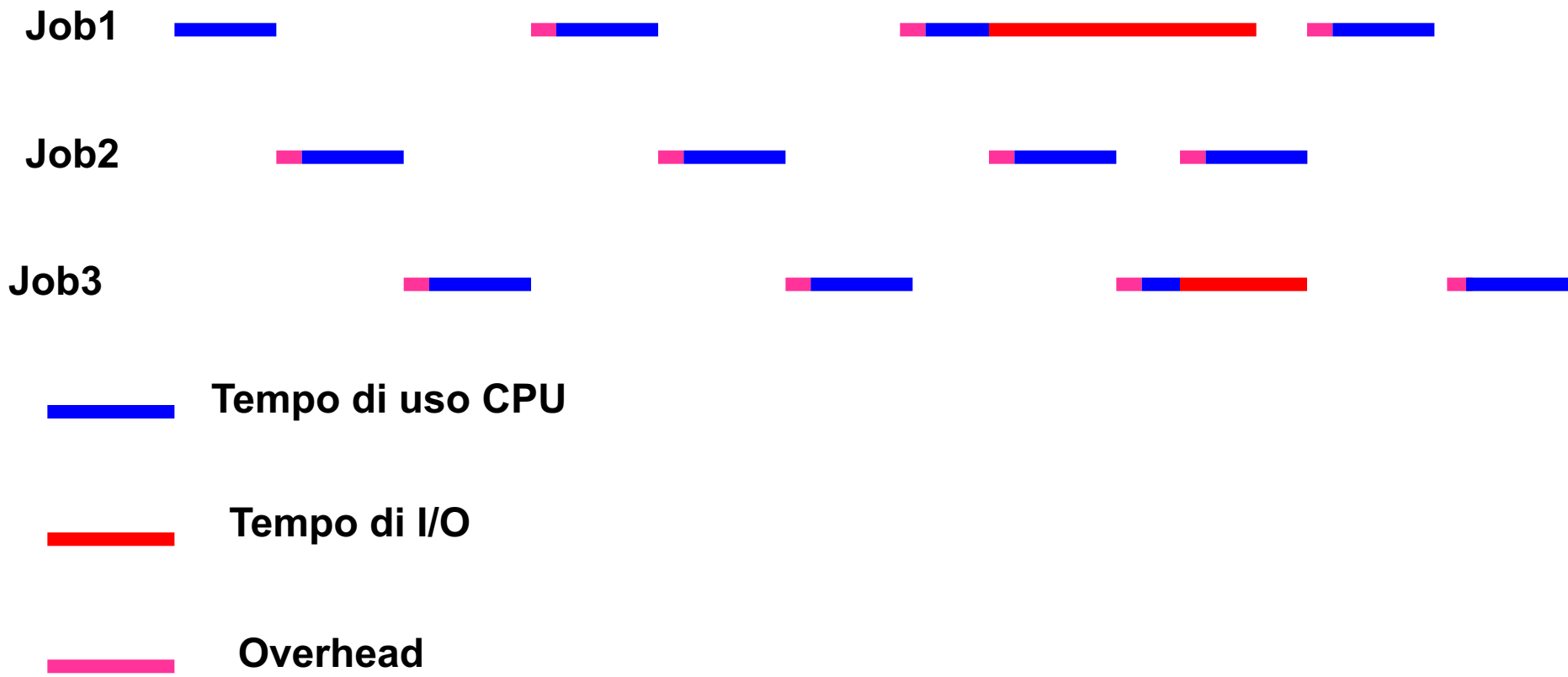
Durante il context switch la CPU non lavora per nessun job, quindi sotto un certo punto di vista non effettua lavoro utile. Si parla di **overhead**.

Avere context switch frequenti garantisce maggior interattività agli utenti ma accresce l'overhead a danno del troughput: è necessario trovare un equilibrio.

L'esempio di esecuzione con multiprogramming di Pag. 15 andrebbe corretto tenendo conto dell'overhead:



Esempio di esecuzione di 3 job con timesharing:



Esempi di S.O. con multiprogrammazione:

- **OS/360**, prodotto da IBM per la famiglia IBM 360, introduce la multiprogrammazione.

- **CTSS** (Compatible Time Sharing System): progettato dal MIT nel 1962 per l'IBM 7094, introduce il timesharing.

- **MULTICS** (MULTiplexed Information and Computing Service): prodotto da MIT, General Electric, Bell Labs (di proprietà AT&T) nel 1965, pensato per sistemi multiutente, introduce il concetto di **processo**.

- **UNIX**: prodotto dai Bell Labs (Ken Thompson) nel 1970 per sistemi monoutente, derivato da CTSS e MULTICS, gira inizialmente su PDP-7, poi viene scritto in C e diventa portabile.

Generazione 4: circuiti integrati LSI/VLSI.

- Hardware: circuiti integrati LSI/VLSI.

- Dagli anni ’80 si diffondono i PC: la facilità d’uso diventa un elemento determinante, per questo vengono introdotte le **G**raphical **U**ser **I**nterface (GUI).

- Dagli anni ’80 si diffondono anche i sistemi real time per la gestione delle applicazioni real time, cioè programmi che rispondono ad un ambiente esterno entro scadenze temporali fissate dall’ambiente medesimo. Sono richieste tecniche di scheduling ad hoc combinate alla possibilità di avere multiprogramming all ’ interno di una singola applicazione.

- Dagli anni ’90 si diffondono i sistemi operativi distribuiti per la gestione dei sistemi distribuiti. Diventano centrali i concetti di controllo distribuito, trasparenza delle risorse/servizi, remote procedure call (RPC).

Nei sistemi operativi moderni vengono combinate le varie tecniche menzionate nelle slide precedenti, al fine di soddisfare richieste di natura diversa da parte degli utenti.