

# DMA

- **Cycle stealing** = Per ogni parola il DMA deve acquisire il bus e rilasciarlo
- **Modalità burst** = Il DMA acquisisce il bus una volta e può scrivere più parole prima di rilasciarlo

# INTERRUPT

- **Interrupt precisi:**
  1. Il PC è salvato in un luogo ben preciso
  2. Tutte le istruzioni prima di quella puntata dal PC sono state completate
  3. Nessuna istruzione dopo quella puntata dal PC è stata completata
  4. Si conosce lo stato di esecuzione dell'istruzione puntata dal PC
- **Interrupt non precisi:** Non rispettano una delle regole

## LBA → CHS

- $C = \text{LBA} // (\text{Ns} * \text{Nh})$
- $H = (\text{LBA} // \text{Ns}) \% \text{Nh}$
- $S = (\text{LBA} \% \text{Ns}) + 1$

## CHS → LBA

- $\text{LBA} = (C * \text{Nh} + H) * \text{Ns} + (S - 1)$

# PERFORMANCE DISCO

- **Latenza rotazionale massima** = 60 sec / RPM
- **Latenza rotazionale media** = Latenza rotazionale massima / 2
- **Tempo di trasferimento medio** = Latenza rotazionale massima / Numero medio di settori per traccia
- **Tempo di accesso medio** = Tempo medio di seek + Tempo di trasferimento medio + Latenza rotazionale media
- **Tempo medio di posizionamento** = Tempo di seek medio + Latenza rotazionale media
- **Tempo medio di accesso a un settore** = Tempo medio di posizionamento + Tempo di trasferimento medio
- **Capacità di un disco** = Numero di settori di ogni cilindro \* Numero di cilindri \* Dimensione di 1 settore \* Numero di superfici (testine)
- **Massimo tasso di trasferimento** = Byte per traccia / Latenza rotazionale massima

# AFFIDABILITÀ DISCO

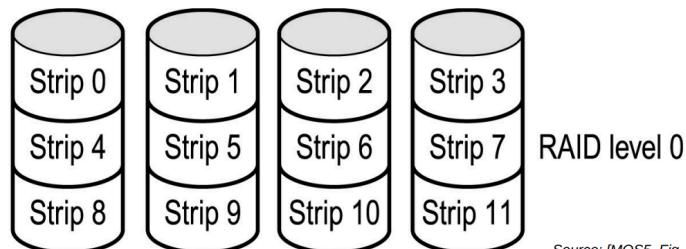
- $MTTF_Y = 1 / AFR$
- $MTTF_H = 1 / HFT = (24 * 365) * MTTF_Y$
- $AFR \lambda_{N Dischi} = N * \lambda_{1 Disco}$

# RAID

- **LIVELLO 0**  
Semplice parallelismo dei dischi. Striping a blocchi.

*N dischi*

- +: Performance,  
Capacità  
-: No tolleranza ai guasti



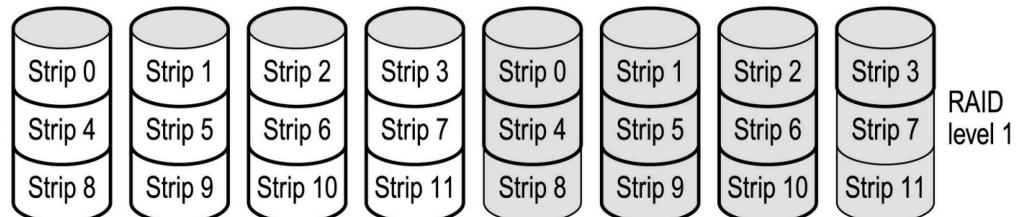
*Source: IMOS5, Fig. 1*

Tolleranza ai guasti = [0]

- **LIVELLO 1**  
Strip dei blocchi che vengono specchiati

*2 N dischi*

- +: Alta tolleranza alle rotture,  
Semplice restaurare i dati,  
Lettura fino a 2 volte più veloce  
-: Costoso (doppio dei dischi),  
Bassa capacità



Tolleranza ai guasti = [>=1, <= n]

- **LIVELLO 2**

### Striping a bit + ECC

Se un disco si rompe i bit dei dati e l'ECC conservati negli altri dischi possono essere usati per ricostruire i dati danneggiati.

*N dischi per i dati + M dischi di parità*

*M dipende dal tipo di ECC usato*

+: Buone performance in lettura,

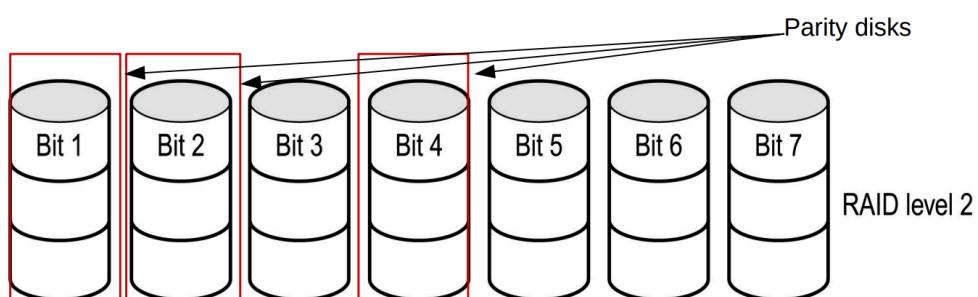
Affidabilità grazie all'ECC

- : Solo 1 richiesta di I/O alla volta per le strip da 1 bit,

Costoso e alto overhead,

Complesso da implementare,

Tutti i dischi devono avere le testine sincronizzate



Tolleranza ai guasti = [ $\geq 1$ ]

- **LIVELLO 3**

Striping a bit + Bit di parità

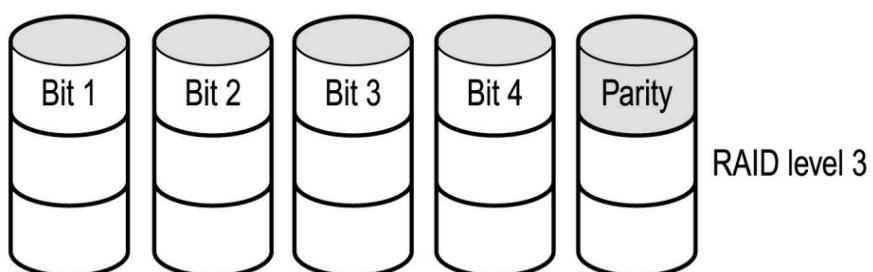
*N dischi per i dati + 1 per la parità.*

Il bit di parità dice se il numero di bit della stringa è pari o dispari.

+: Stesse performance e affidabilità del RAID di livello 2 a minor costo

- : Sincronizzazione delle testine,

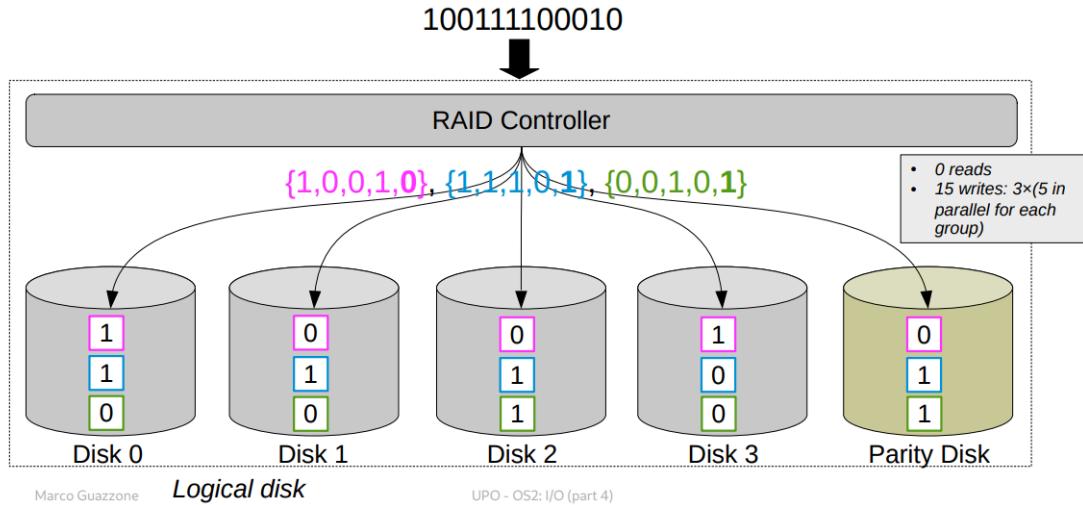
1 richiesta di I/O alla volta



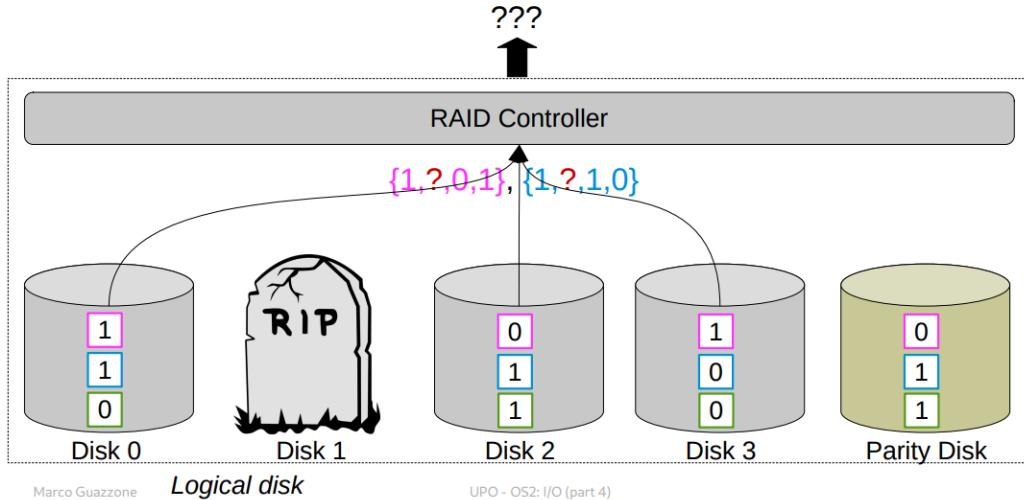
Tolleranza ai guasti = [1]

## **ESERCIZI LIVELLO 3**

- **EXAMPLE:** write request of “100111100010“

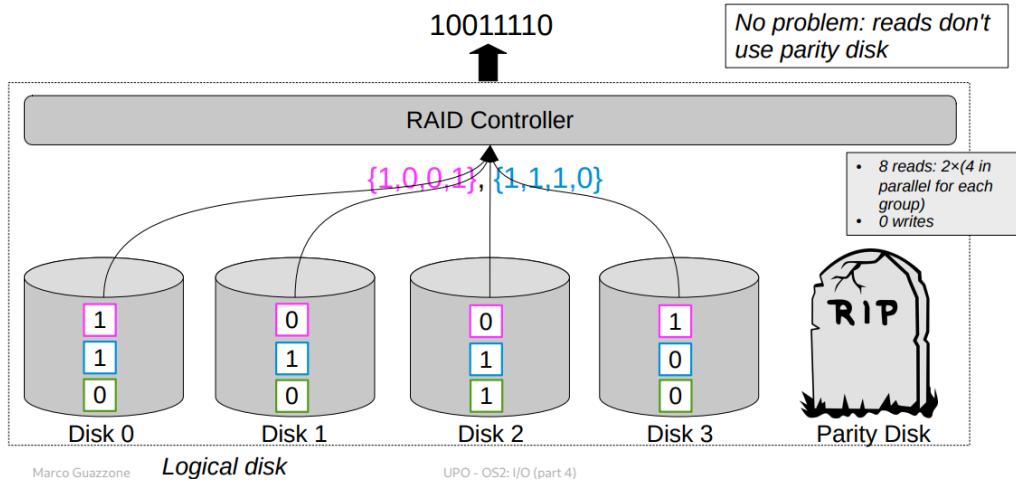


- **EXAMPLE:** disk 1 crashes, read request of 1<sup>st</sup> byte (stripe 0 and 1)



Il bit di parità permette di ricostruire la stringa accedendo alle informazioni dal disco di parità.

- **EXAMPLE:** parity disk crashes, read request of 1<sup>st</sup> byte (stripe 0,1)



- **LIVELLO 4**

Striping a blocchi + Blocco di parità

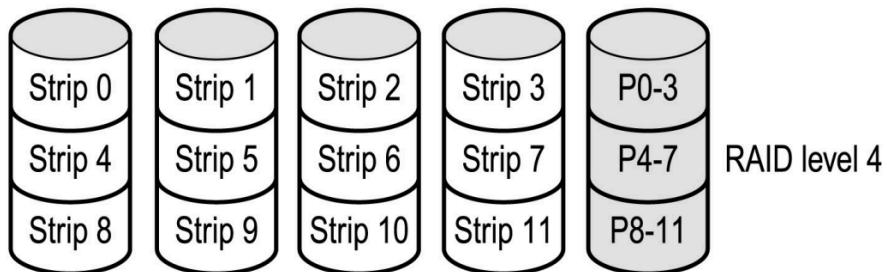
*N dischi per i dati + 1 per la parità*

+: Buona affidabilità,

Buone performance per lettura e scrittura di grandi richieste

- : Grande overhead per piccole scritture,

Disco di parità fa da collo di bottiglia



Tolleranza ai guasti = [1]

### ESERCIZI LIVELLO 4

Viene applicato lo XOR tra strip della stessa stripe.

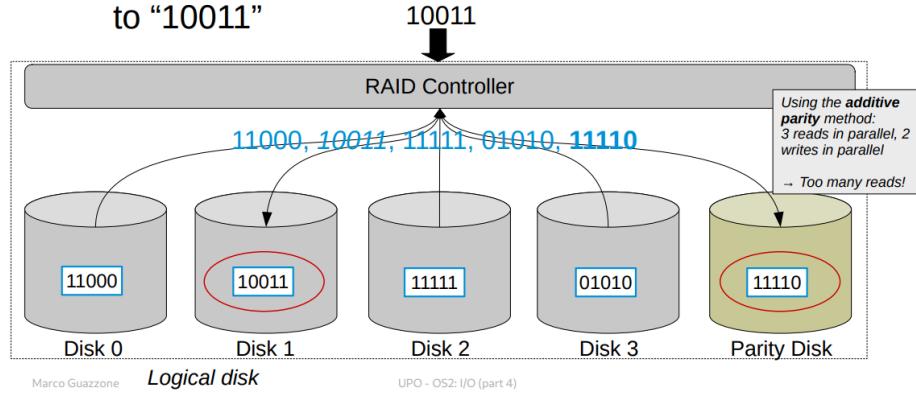
## XOR Truth Table

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

- **Parità additiva** : Leggo i dischi che non vengono modificati (non la parità) e scrivo i dischi modificati e la parità. (Utile per grandi modifiche)

- **EXAMPLE: The small-write operation case**

- Each strip is 5 bits long, write request to update strip 1 of stripe 0 from "00101" (see previous slide) to "10011"



- **EXAMPLE: The small-write operation case (cont'd)**

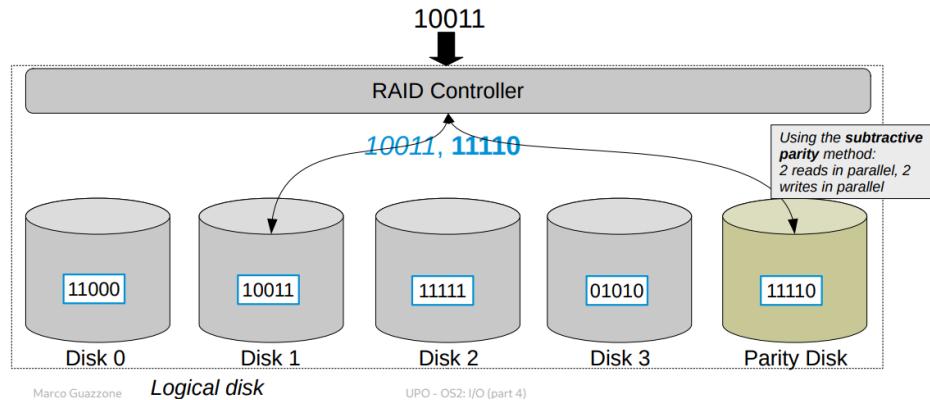
- **Additive parity:**

- Strip0: 11000 →  $1\textcolor{red}{1}0\textcolor{green}{0}\textcolor{magenta}{0}$   $\oplus$
- New Strip1: 10011 →  $\textcolor{red}{1}0\textcolor{green}{0}1\textcolor{magenta}{1}$   $\oplus$
- Strip2: 11111 →  $\textcolor{red}{1}1\textcolor{green}{1}1\textcolor{magenta}{1}$   $\oplus$
- Strip3: 01010 →  $01\textcolor{red}{0}1\textcolor{magenta}{0}$  =
- New Parity Strip: 11110 ←  $\textcolor{red}{1}1\textcolor{green}{1}1\textcolor{magenta}{0}$

**Additive parity** method: 3 reads in parallel, 2 writes in parallel

- **Parità sottrattiva** : leggo le strip da modificare e la vecchia parità, scrivo le nuove strip e la nuova parità. (Utile per piccole modifiche)

- **EXAMPLE:** The small-write operation case (cont'd)



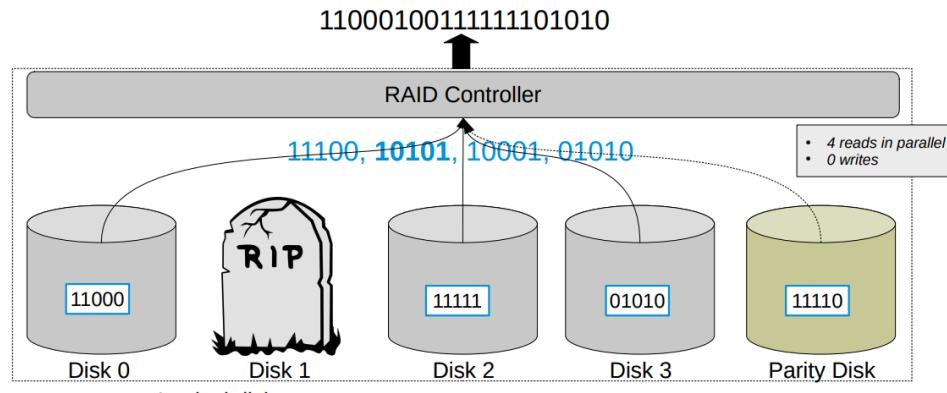
- **EXAMPLE:** The small-write operation case (cont'd)

- **Subtractive Parity:**

- Old Strip1: 00101 →  $\textcolor{red}{0} \textcolor{blue}{0} \textcolor{magenta}{1} \textcolor{red}{0} \textcolor{blue}{1}$  ⊕
- New Strip1: 10011 →  $\textcolor{red}{1} \textcolor{blue}{0} \textcolor{magenta}{0} \textcolor{red}{1} \textcolor{blue}{1}$  ⊕
- Old Parity Strip: 01000 →  $\textcolor{red}{0} \textcolor{blue}{1} \textcolor{magenta}{0} \textcolor{red}{0} \textcolor{blue}{0}$  =
- New Parity Strip: 11110 ←  $\textcolor{red}{1} \textcolor{blue}{1} \textcolor{magenta}{1} \textcolor{red}{0}$

**Subtractive parity:** 2 reads in parallel, 2 writes in parallel.  
In this case, the **subtractive parity** method is more efficient than the **additive parity** method!

- **EXAMPLE:** The disk failure case (cont'd)



11000

**10011**

11111

01010

-----

11110

Calcolo l'inverso dello XOR per ritrovare la stringa mancante

- **LIVELLO 5**

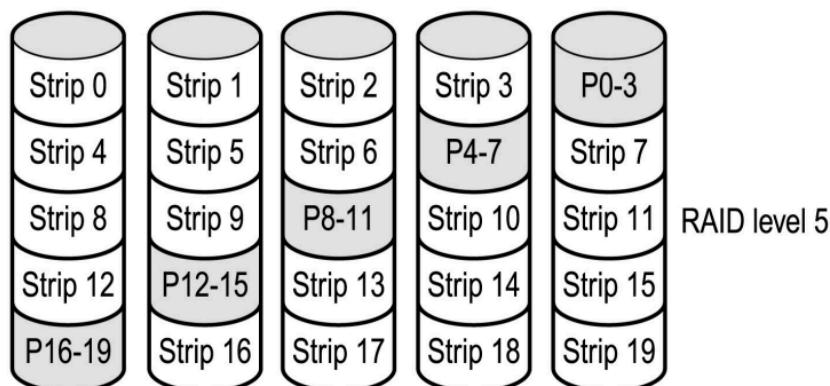
Striping a blocchi + **Blocchi di parità distribuiti**

*N+1 dischi, i dati e i blocchi di parità sono alternati in tutti i dischi*

+: Stessi vantaggi del RAID 4,

Migliorate le richieste di lettura e scrittura

-: Overhead per piccole scritture



Tolleranza ai guasti = [1]

- **LIVELLO 6**

Striping a blocchi + Blocchi di parità doppiamente distribuiti.

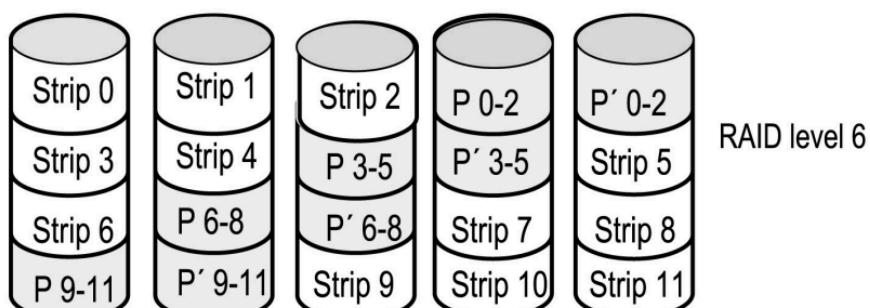
“P” blocchi basati su XOR e “Q” blocchi basati su Reed-Solomon codes.

*N+2 dischi, ogni stripe ha 2 blocchi di parità distribuiti salvati in dischi differenti*

+: Buone performance (non come RAID 5, lettura uguale, scrittura peggiore),

Alta tolleranza

-: Ogni scrittura colpisce 2 blocchi di parità



Tolleranza ai guasti = [2]

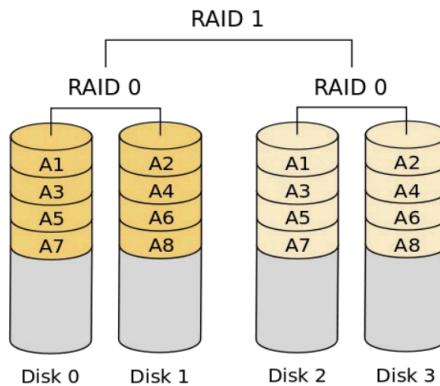
- **LIVELLO 0+1**

2 RAID 0 a specchio

+ : Alte performance in lettura,

Più affidabile di RAID 0

- : Costoso



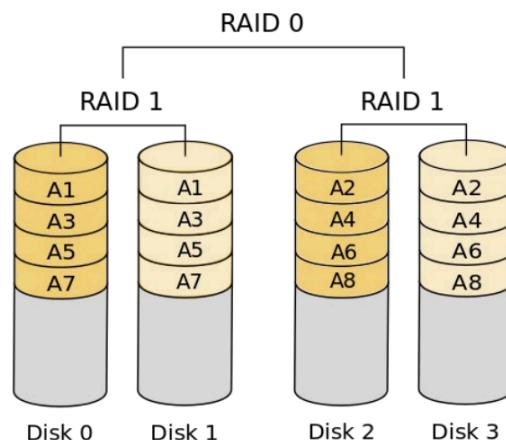
Tolleranza ai guasti = [ $>=1, <=n$ ]

- **LIVELLO 1+0**  
2 RAID 1 in parallelo

+ : Buone performance,

Più affidabile di RAID 0

- : Costoso



Tolleranza ai guasti = [1]

## FORMATTAZIONE A BASSO LIVELLO

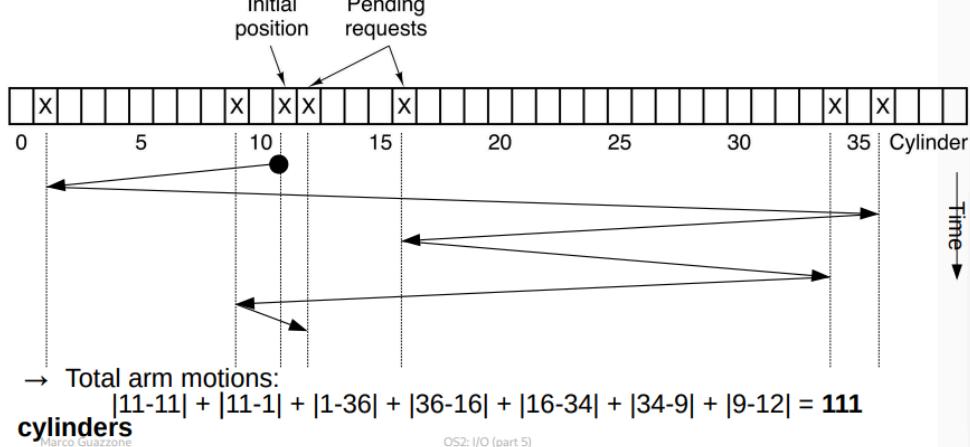
- **Cylinder skew** = [ track-to-track seek time / Tempo di trasferimento medio **in microsec** ]
- **Maximum data rate** = Dimensione della traccia / Latenza rotazionale massima
- **Sector interleaving** = [in-memory sector transfer time / Tempo di trasferimento medio **in microsec** ]

# SCHEDULING DEL BRACCIO

- **FCFS (First Come First Served)** = Richieste servite nell'ordine di arrivo

- **EXAMPLE:** disk with 40 cylinders + FCFS

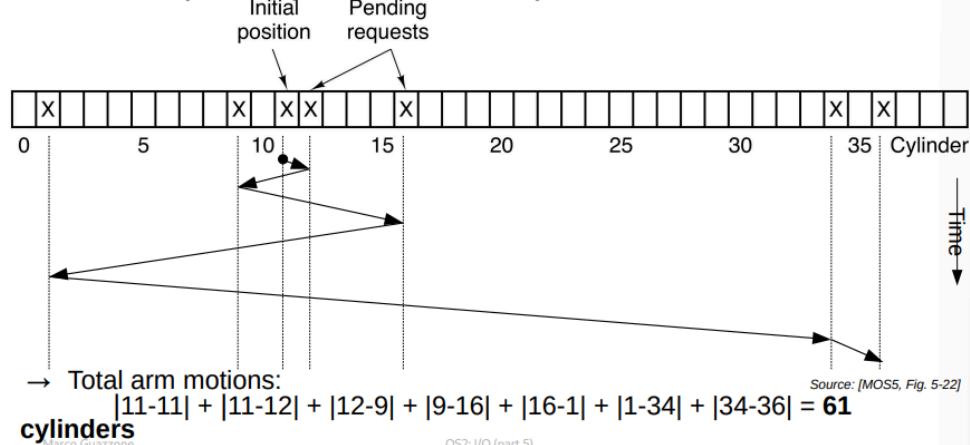
- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



- **SSF (Shortest Seek First)** = Seleziona le richieste con il tempo di seek più corto rispetto alla posizione corrente della testina. Rischio di starvation se le richieste si concentrano tutte in un punto

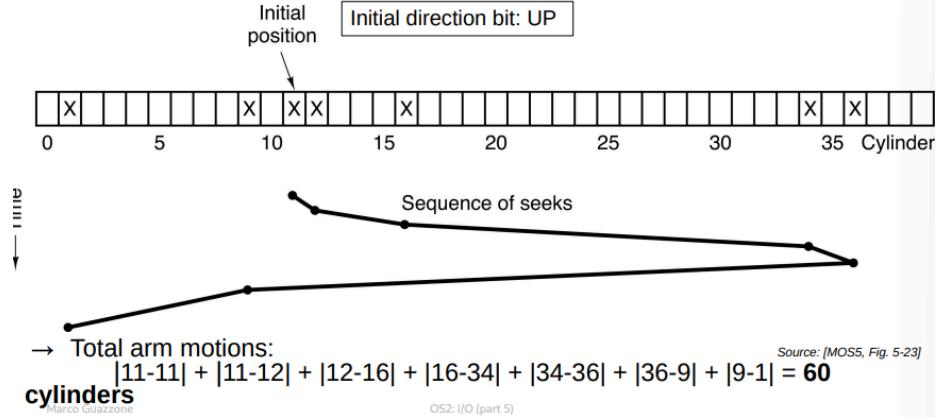
- **EXAMPLE:** disk with 40 cylinders + SSF

- Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



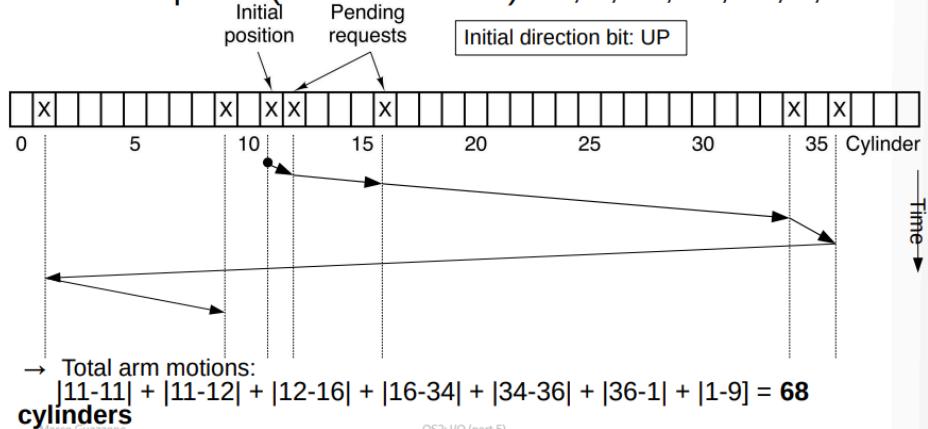
- **LOOK (Elevator Algorithm)** = Il braccio si muove nella stessa direzione finché sono presenti richieste in quella direzione, poi inverte direzione

- **EXAMPLE:** disk with 40 cylinders + LOOK
  - Requests (from first to last): 11, 1, 36, 16, 34, 9, 12



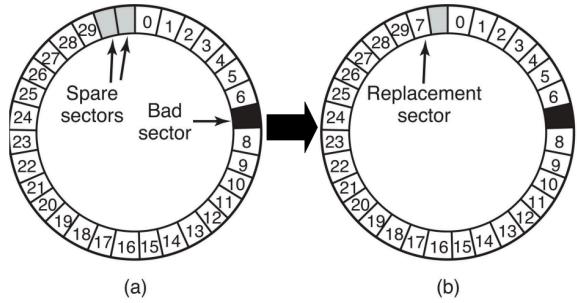
- **C-LOOK (Circular LOOK) =** Quando raggiunge l'ultima richiesta in una direzione torna immediatamente all'inizio dell'altro lato senza servire richieste nel tragitto del ritorno.

- **EXAMPLE:** disk with 40 cylinders + C-LOOK
  - Requests (from first to last): 11, 1, 36, 16, 34, 9, 12

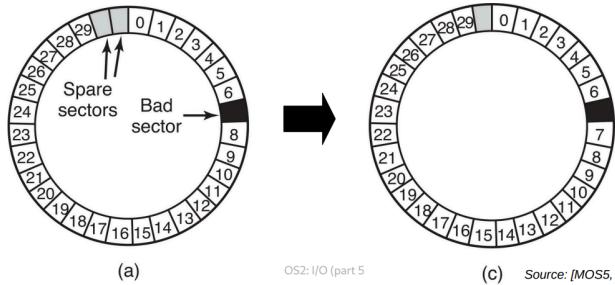


## ERROR HANDLING

- **Linear recording density** = Dimensione traccia / Lunghezza traccia =  
(Numero di settori per traccia \* Dimensione settore in bit) / Circonferenza tracce al centro del disco
- **Approccio a livello del controller**
  - Settori di riserva usati in caso di guasti
  - **Sector forwarding** = il controller mappa su un settore di ricambio il settore danneggiato



- **Sector slipping** = Il controller sposta tutti i settori di uno a partire dal settore danneggiato



- **Approccio a livello del Sistema Operativo**

- Il SO fa una scansione alla ricerca dei settori danneggiati e crea una mappa dei settori utilizzabili

## MOUSE

- **Numero massimo di messaggi al secondo** = Movimenti massimi [in mm/sec] / mickey
- **Max data rate** = Numero massimo di messaggi al secondo \* Numero di byte per messaggio

## HARD DISK POWER MANAGEMENT

- **Break-even condition** =  $E_{sd} + P_s * (T_d - T_{sd} - T_{wu}) + E_{wu} = P_w * T_d$
- **Break-even point  $T_d$**  =  $(E_{sd} + E_{wu} - P_s * (T_{sd} + T_{wu})) / (P_w - P_s)$
- $E_{sd}$  = Energia spesa da stato attivo a basso consumo
- $P_s$  = Consumo istantaneo
- $T_{sd}$  = Tempo transizione a → b
- $T_{wu}$  = Tempo transizione b → a
- $E_{wu}$  = Energia spesa da basso consumo ad attivo
- $P_w$  = Consumo istantaneo stato attivo
- $T_d > T_{sd} + T_{wu}$  Sennò non ha senso fare la transizione di stato

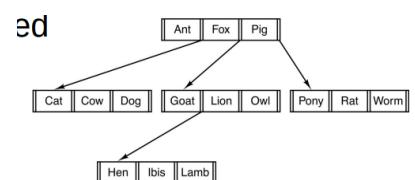
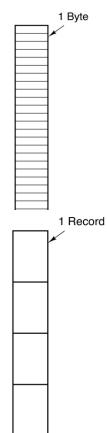
# CPU POWER MANAGEMENT

- $P = P_{\text{dinamico}} + P_{\text{statico}} = k * V^2 * f + P_{\text{statico}}$
- $E_{\text{nocut}} = (P_{\text{dinamico}} * T_{\text{utilizzo}}) + (P_{\text{statico}} * T_{\text{totale}})$
- $E_{\text{cut}} = (P_{\text{dinamico}} / n^2) * T_{\text{utilizzo}} + P_{\text{statico}} * T_{\text{totale}}$
- $\Delta E \% = 100 * \Delta E / E_{\text{nocut}}$
- $V = \text{Voltaggio CPU}$
- $f = \text{frequenza di clock}$
- $k = \text{costante che dipende dal sistema}$
- $P = \text{Consumo istantaneo}$
- $P_{\text{dinamico}} = \text{Consumo operazioni}$
- $P_{\text{statico}} = \text{Consumo fisso anche quando non fa nulla}$

# FILE SYSTEMS

## STRUTTURE DEI FILE

- **Sequenza di byte** = Sequenza non strutturata di byte. Il sistema operativo non sa cosa è presente nel file.
- **Sequenza di record** = Sequenza di record con dimensione prestabilita. Il sistema operativo fornisce funzioni per leggere/scrivere un record alla volta.
- **Albero** = Un file è un albero di record, anche di differente lunghezza. Ogni record ha un campo **chiave**, l'albero è ordinato rispetto alle chiavi.



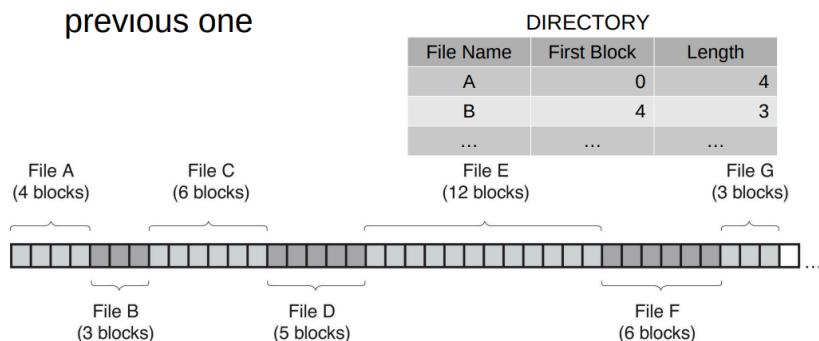
## PATH NAME

- **Assoluti** = Un path assoluto specifica la posizione di un file o una directory a partire dalla radice del file system. La radice è rappresentata da una barra `/`. Un path assoluto inizia sempre con `/`.
- **Canonici** = Un path canonico è una versione semplificata di un path che non contiene parti ridondanti come `.` (directory corrente) o `..` (directory superiore). Viene spesso risolto in un percorso assoluto.

- **Relativi** = Un path relativo specifica la posizione di un file o una directory rispetto alla directory corrente. Non inizia con `/`.
- `'.'` = directory corrente
- `'..'` = directory superiore
- **Linking**
  - **Hard link** = i file e tutti i suoi link puntano alla stessa struttura interna usata dal sistema operativo per tenere traccia dei file linkati
  - **Soft link** = il link è conservato in un piccolo file separato di tipo LINK che contiene il path per il file linkato

## IMPLEMENTAZIONI DEI FILE

- **Allocazione contigua**
  - Salva ogni file come una sequenza contigua di blocchi del disco
  - Le directory contengono, in aggiunta al nome del file, l'indirizzo del disco del primo blocco e il numero di blocchi del file



Siano:

- $F'$  = dimensione stimata del file
- $F$  = dimensione reale del file
- $B$  = dimensione del blocco del disco
- $N = \lceil F' / B \rceil$     Numero di blocchi allocati
- **Overhead** = 0%
- $\text{Wasted\%} = (1 - F / (N * B)) \%$     Spazio sprecato

Conviene convertire tutto in Byte

- **Allocazione basata su extent**
  - Ogni file consiste di uno o più regioni non contigue di blocchi contigui (extents)
  - Ogni file necessita di una extent table per tenere traccia degli extents allocati

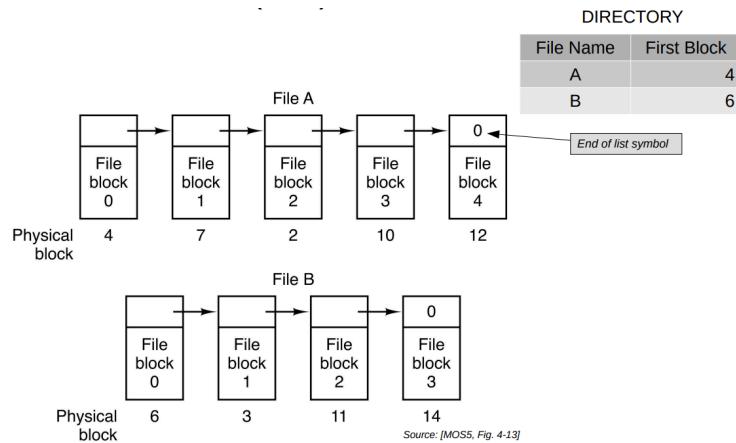
- Le directory contengono l'indirizzo del blocco in cui è archiviata la tabella precedente

Siano:

- $\rightarrow F$  = dimensione del file
- $\rightarrow B$  = dimensione del blocco del disco
- $\rightarrow P$  = dimensione del puntatore al blocco del disco (rappresenta l'indirizzo del blocco)
- $\rightarrow E$  = numero di blocchi per ogni extents
- $\geq M = \lceil F / (E * B) \rceil$  Numero di extents allocati
- $\geq T = \lceil (M * P) / B \rceil$  Numero di blocchi necessari per salvare la extent table
- $\geq N = M * E + T$  Numero di blocchi allocati (blocchi extents+extent table)
- $\geq \text{Overhead\%} = ((T * B) / (N * B)) \% = (T/N) \%$
- $\geq \text{Wasted\%} = (1 - (M * P + F) / (N * B)) \%$

- Allocazione a liste concatenate**

- I blocchi di un file sono organizzati in liste concatenate
- Ogni blocco contiene dei dati e un puntatore al blocco successivo
- Le directory contengono solo il puntatore al primo blocco



Siano:

- $\rightarrow F$  = dimensione del file
- $\rightarrow B$  = dimensione del blocco del disco
- $\rightarrow P$  = dimensione del puntatore al blocco del file
- $\geq N = \lceil F / (B - P) \rceil$  Numero di blocchi allocati
- $\geq \text{Overhead\%} = (N * P / (N * B)) \% = (P / B) \%$
- $\geq \text{Wasted\%} = (1 - (N * P + F) / (N * B)) \%$

- Allocazione basata su cluster**

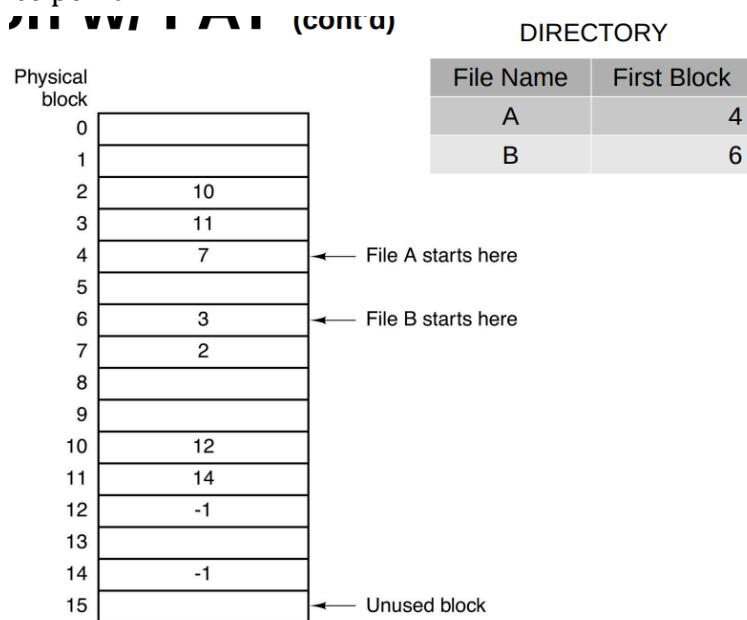
- Raccoglie blocchi contigui in cluster di dimensione fissa e linka i cluster invece che i blocchi. Ogni cluster contiene un puntatore al cluster successivo
- Non usa una tabella separata per conservare i puntatori
- Le directory contengono i puntatori al primo cluster del file associato

Siano:

- $F$  = dimensione del file
  - $B$  = dimensione del blocco del disco
  - $P$  = dimensione del puntatore al blocco del file
  - $C$  = numero di blocchi per cluster
- >  $M = \lceil F / (C * B - P) \rceil$  Numero di cluster allocati  
 >  $N = M * C$  Numero di blocchi allocati  
 >  $\text{Overhead\%} = (M * P / (N * B))\% = (P/(C*B))\%$   
 >  $\text{Wasted\%} = (1 - (M * P + F) / (N * B))\%$

- **Allocazione basata su liste concatenate con FAT**

- Il puntatore al blocco successo è conservato in una File Allocation Table (FAT)
- Ogni entry della tabella contiene il puntatore al blocco successivo o un simbolo speciale per caratterizzare l'ultimo blocco del file o un blocco non allocato
- Le directory contengono il puntatore al primo blocco di un file che è usato come indice per la FAT



Siano:

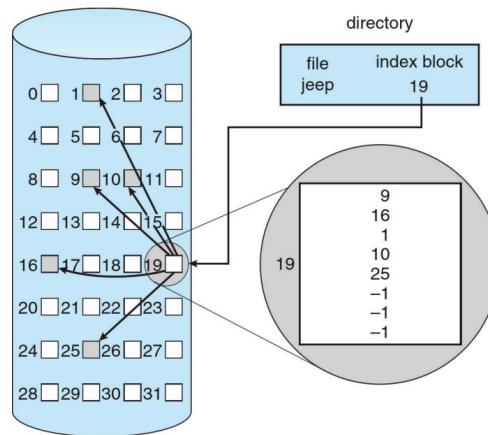
- $D$  = dimensione della partizione del disco

- F = dimensione del file
- B = dimensione del blocco del disco
- P = dimensione del puntatore al blocco del file

- $M = \lfloor D / B \rfloor$  Numero di entries per la FAT
- $S = M * P$  Dimensione della FAT
- $N = \lceil F / B \rceil$  Numero di blocchi allocati per file
- $\text{Overhead\%} = (S / D) \%$
- $\text{Wasted\%} = (1 - (F / (N * B))) \%$

- **Allocazione con index block**

- Ogni file ha la sua tabella di allocazione (index block) conservata in uno o più dischi separati, contiene l'indirizzo del blocco del disco
- Le directory contengono l'indirizzo della index block



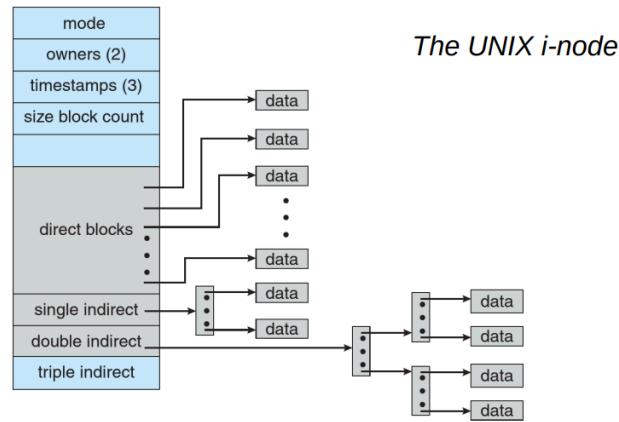
Siano:

- F = dimensione del file
  - B = dimensione del blocco del disco
  - P = dimensione del puntatore al blocco del file
  - I = numero di blocchi del disco allocati nella index block
- $S = I * B$  Dimensione index block
  - $M = \lceil F / B \rceil$  Numero di blocchi allocati solo per i dati
  - $N = I + M$  Numero di blocchi allocati per file
  - $\text{Overhead\%} = (S / (N * B)) \% = (I / N) \%$
  - $\text{Wasted\%} = (1 - (P * M + F) / (N * B)) \% \geq (1 - (S + F) / (N * B)) \%$

- **Allocazione con i-node**

- C'è un index-block principale chiamato i-node
- Le prime n entries dell'i-node sono puntatori diretti a blocchi, contengono gli indirizzi dei blocchi dati del file

- Le restanti entries sono puntatori indiretti a blocchi, contengono gli indirizzi degli index block che rappresentano gli indici multilivello



Siano:

- N = puntatori diretti
- M = puntatori indiretti
- L = lunghezza index block
- B = dimensione index block

$$\begin{aligned} > A = N + \sum_{i=1}^M L^i && \text{Numero massimo di blocchi indirizzabili} \\ > S = A * B && \text{Dimensione file massima} \end{aligned}$$

## BLOCCHI LIBERI

- Liste concatenate**
  - Ogni nodo della lista contiene il numero di puntatori che il blocco del disco può contenere. Inoltre contiene il puntatore al nodo successivo
  - Numero di puntatori a blocco per nodo** =  $\lfloor \text{dimensione blocco}/\text{dimensione puntatore} \rfloor - 1$
  - Numero di blocchi nel disco** =  $\lfloor \text{dimensione disco}/\text{dimensione blocco} \rfloor$
  - Numero massimo di nodi nella lista** =  $\lceil \text{blocchi su disco}/\text{numero di puntatori} \rceil$
  - Dimensione massima lista** = numero nodi \* dimensione blocco
- Bitmap**
  - Un bit per ogni blocco del disco. 1 se il blocco è libero 0 altrimenti o viceversa
  - Numero di blocchi su disco** =  $\lfloor \text{dimensione disco}/\text{dimensione blocco} \rfloor$

- **Dimensione bitmap** = numero di blocchi di disco\*8
- **Numero di blocchi per allocare bitmap** = [dimensione bitmap/dimensione blocco]

# BACKUP

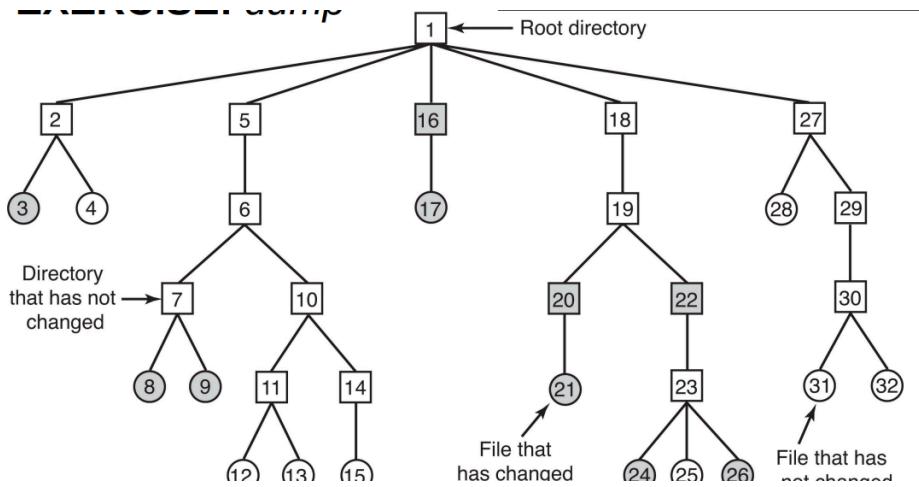
- ***Backup differenziale***
  - Periodicamente eseguito un backup completo
  - Periodicamente eseguito un backup differenziale che salva solo i file che sono cambiati dall'ultimo backup completo

➔ **Recovery:**

  - ◆ Prima ripristinare l'ultimo backup completo, poi ripristina il backup differenziale che corrisponde alla data richiesta
- ***Backup incrementale***
  - Periodicamente eseguito un backup completo
  - Periodicamente eseguito un backup incrementale che salva solo i file che sono cambiati dall'ultimo backup (che sia incrementale o completo)

➔ **Recovery:**

  - ◆ Prima ripristinare l'ultimo backup completo, poi ripristina tutti i backup incrementali a partire dal backup completo fino al giorno richiesto
- ***Dump fisico***
  - Inizia al primo blocco del disco
  - Scrive tutti i blocchi del disco sul disco di backup in ordine
  - Si ferma quando l'ultimo blocco è stato copiato
- ***Dump logico***
  - Inizia da uno o più directories specifiche e ricorsivamente scarica tutti i file e directories trovati
  - Include tutti i file o i file che sono cambiati da una certa data di base
  - Deve includere tutte le directories del path



- **Fase 1 :** Per ogni file modificato e directory segna il suo i-node nella bitmap

Source: [MOS5, Fig. 4-27]																																
End of Phase 1: all directories (either changed or not) and changed files are marked.																																
Phase 1 (a)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

- **Fase 2 :** Deseleziona tutte le directory che non hanno file o directories modificate al di sotto

Phase 1 (a)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
End of Phase 2: only changed files/directories and their parent directories are marked.																																
Phase 2 (b)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

- **Fase 3 :** Scansiona gli i-nodes in ordine numerico e scarica tutte le directories che sono segnate, le deseleziona una volta scaricate

Phase 2 (b)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
End of Phase 3: all marked directories have been dumped and are unmarked.																																
Phase 3 (d)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Source: [MOS5, Fig. 4-28]

- **Fase 4 :** Scansiona gli i-nodes in ordine numerico e scarica tutti i file segnati, gli deseleziona una volta scaricati

Phase 3 (d)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
End of Phase 4: all changed file have been dumped and the bitmap is completely unmarked.																																
Phase 4 (e)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

## FSCK

- **Consistenza dei blocchi**

- Si usano 2 tavole
  - Tabella dei blocchi in uso : tenere traccia di quanto spesso ogni blocco è utilizzato in un file
  - Tabella dei blocchi liberi : tenere traccia di quanto spesso ogni blocco è presente nella struttura dei blocchi liberi

Block number															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	0	1	1	1	1	0	0	1	1	1	0	0
Blocks in use															
0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	1
Free blocks															

→ **Blocchi mancanti**

- ◆ Il blocco ha 0 in entrambe le tabelle
- ◆ Fsck lo aggiunge alla struttura dei blocchi liberi

→ **Blocchi contemporaneamente liberi e usati**

- ◆ Il blocco ha 1 in entrambe le tabelle
- ◆ Fsck lo rimuove o dalla struttura dei blocchi liberi o dagli i-node

→ **Blocchi liberi duplicati**

- ◆ Il blocco ha un valore >1 nella tabella dei blocchi liberi
- ◆ Fsck ricostruisce la struttura dei blocchi liberi

→ **Blocchi usati duplicati**

- ◆ Il blocco ha un valore >1 nella tabella dei blocchi usati
- ◆ Fsck alloca tanti blocchi liberi quanti sono i duplicati -1, poi copia il contenuto del blocco duplicato nel nuovo blocco libero, in fine per ogni file a cui è associato il blocco duplicato, fsck gli assegna una delle copie allocate all'inizio del blocco duplicato e dealloca da esso il blocco duplicato

- **Consistenza dei file**

- Si usa 1 tabella
  - Tabella dei file in uso : tenere traccia di quanto spesso ogni file appare nell'albero delle directory
- Per ogni file il file system è consistente se il contatore della tabella dei file in uso è concorde con il contatore dei link nel corrispettivo i-node

→ **Il contatore dei link nell'i-node è troppo alto**

- ◆ Contatore dei link nell'i-node > contatore nella tabella
- ◆ Fsck cambia il contatore dell'i-node a quello corretto trovato nella tabella

→ **Il contatore dei link nell'i-node è troppo basso**

- ◆ Contatore dei link nell'i-node < contatore nella tabella
- ◆ Fsck cambia il contatore dell'i-node a quello corretto trovato nella tabella

- **Controllo dello stato degli i-node**

- Controlla ogni i-node se sono corrotti o hanno altri problemi
- In caso di sospetti fsck risolve il problema

- **Controllo delle directory**

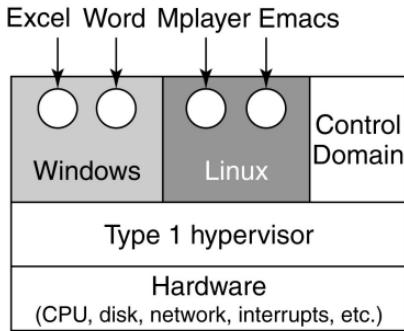
- Controlla che ogni i-node in una directory abbia un numero legale e sia allocato, ‘.’ e ‘..’ presenti, strani permessi da riportare
- **Controllo dei superblocchi**
  - Controlla se il superblocco sembra ragionevole
  - Potrebbe decidere di usare una copia alternativa del superblocco
- **Controllo dei blocchi difettosi**
  - Controlla la presenza di puntatori a blocchi difettosi
  - Rimuove dall'i-node il puntatore e lo aggiunge ai blocchi difettosi per prevenire che venga allocato

## CACHING

- ***LRU (Least Recently Used)***
  - La vittima è il blocco meno recentemente utilizzato
- ***MRU (Most Recently Used)***
  - *La vittima è il blocco usato più di recente*
- ***Write-back caches***
  - Quando un blocco in cache viene modificato è segnato come “dirty” e scritto in memoria successivamente
- ***Write-through caches***
  - Appena un blocco in cache viene modificato viene immediatamente scritto in memoria
- ***Block read-ahead (prefetching)***
  - Caricare i blocchi in cache prima che siano necessari
  - Quando il blocco k di un file è letto il SO controlla se i blocchi k+1, k+2 ecc sono in cache

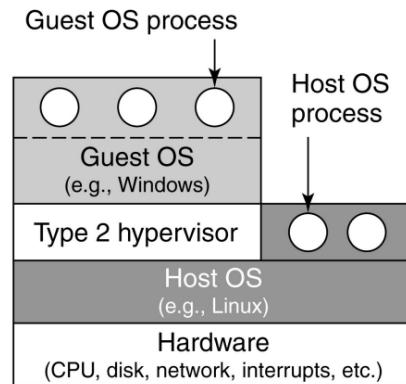
## VIRTUALIZZAZIONE

- ***Hypervisors***
  - ***Bare-metal hypervisor***
    - Un sottile strato di Software che gira nativamente e direttamente sull'hardware. Host OS e VMM sono lo stesso componente



- ***Hosted hypervisor***

- Un altro programma che è avviato e gestito dall'host OS. L'host OS non sa della virtualizzazione



- ***Come il VMM esegue le istruzioni guest***

- ***Interpretazione (emulazione)***
  - VMM interpreta in SW ogni istruzione guest
- ***Trap-and-emulate***
  - VMM configura l'ambiente hardware ma lascia che le istruzioni della VM siano eseguite direttamente sulla CPU.
  - Istruzioni non privilegiate : eseguite direttamente sull'hardware
  - Istruzioni privilegiate : causano una trap che viene gestita dal VMM
- ***Dynamic binary translation***
  - Istruzioni di un guest user-level : VMM esegue direttamente in hardware
  - Istruzioni privilegiate : VMM usa Trap-and-emulate
  - Istruzione sensibile : VMM riscrive parte del guest OS codice binario per sostituire l'istruzione con una sequenza che emula in qualche modo l'istruzione
- ***Paravirtualization***
  - Full virtualization : il guest OS è completamente astratto dal livello HW dal livello virtualizzazione
  - Guest OS e VMM cooperano per rendere l'HW virtualizzabile
- ***HW-assisted Virtualization***

- Full virtualization con trap-and-emulate dove l'hypervisor può avere il vantaggio dell'estensione della virtualizzazione sull'HW
- ***Come mappare un frame della pagina guest su un frame della pagina host?***
  - ***Shadow paging***
    - Per ogni processo di ogni VM, il VMM crea una shadow page table che mappa: guest virtual page → host page frame
  - ***Paravirtualization***
    - Il guest OS è a conoscenza della virtualizzazione e sa che quando finisce di cambiare la sua gPT deve informare il VMM
  - ***Nested paging***
    - La gPT mappa gli indirizzi virtuali dei guest sugli indirizzi fisici.
    - La Nested page table (nPT) mappa gli indirizzi fisici dei guest sugli indirizzi fisici dell'host
- ***Allocare e reclamare memoria***
  - ***Page sharing***
    - Il VMM condivide le stesse pagine dell'host sulle VM
    - Usa la tecnica copy-on-write per gestire i cambiamenti. Quando una VM vuole cambiare una pagina condivisa il cambiamento non deve essere visibile dalle altre VMs. Il VMM rimpiazza la pagina condivisa con una nuova copia che verrà assegnata alla VM.
  - ***Ballooning***
    - Un modulo balloon è caricato in ogni VM come un pseudo driver
    - Può gonfiarsi richiedendo più memoria e sgonfiarsi rilasciandola