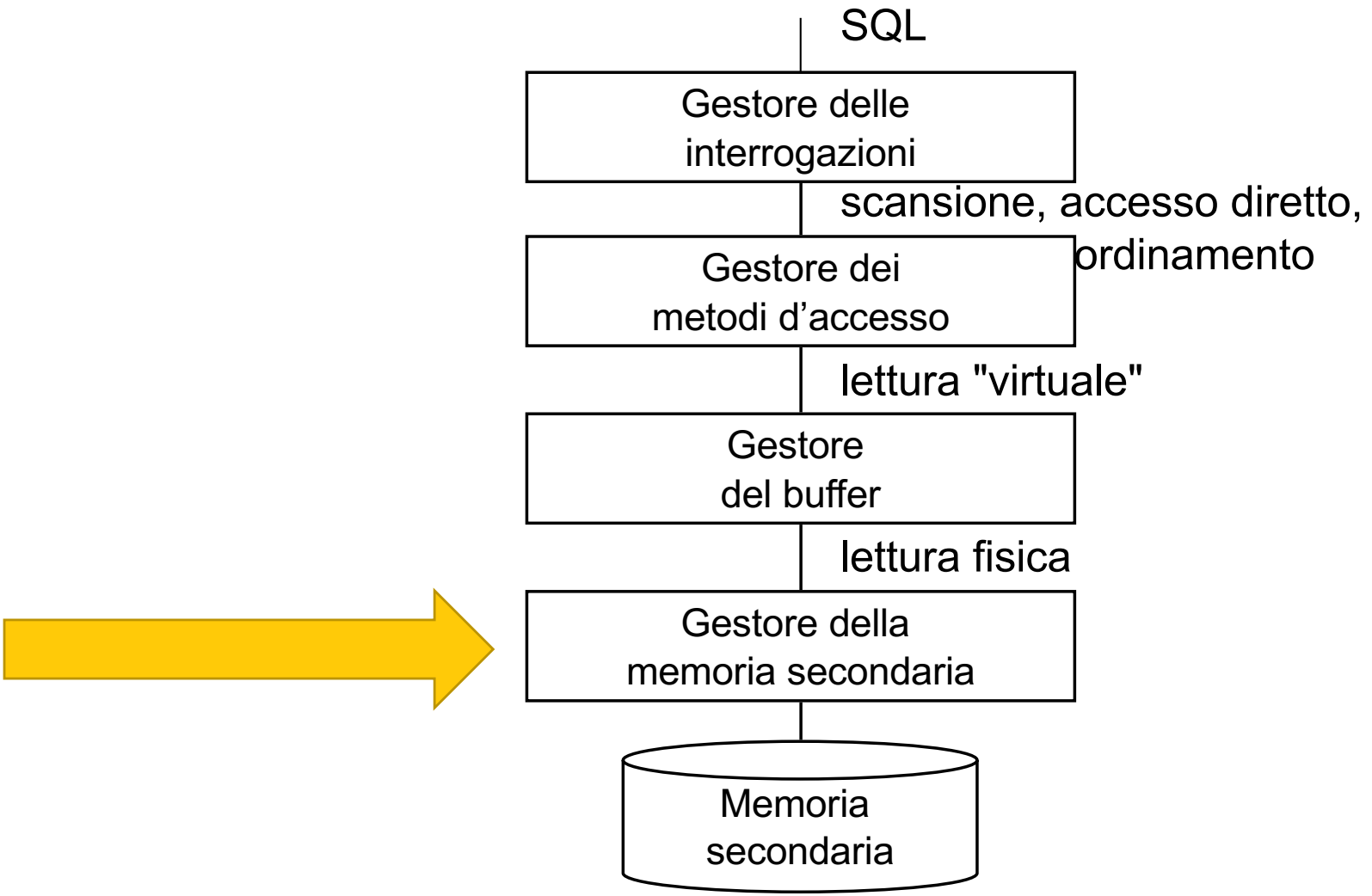
Gestore degli accessi e delle interrogazioni



DBMS e file system

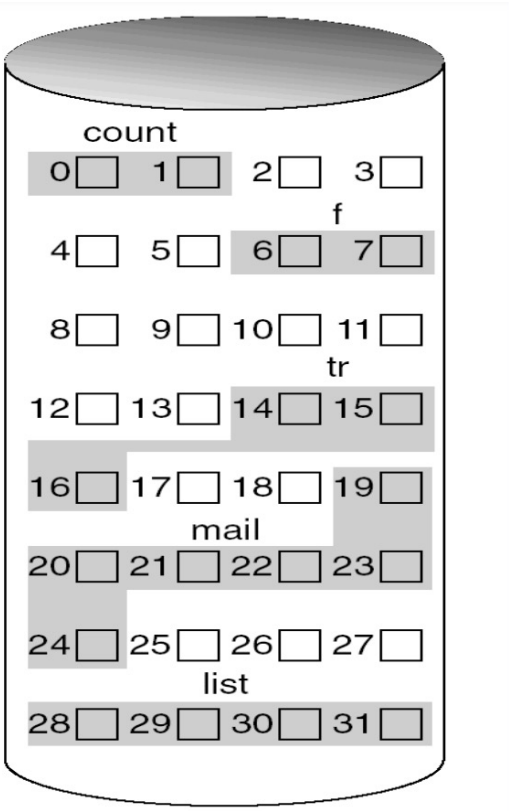
- Il file system è il componente del sistema operativo che gestisce la memoria secondaria
- I DBMS ne utilizzano le funzionalità:
  - per creare ed eliminare file
  - per leggere e scrivere singoli blocchi o sequenze di blocchi contigui

File, blocchi: allocazione

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di lunghezza **fissa**
- L'**allocazione dei blocchi** di un file su disco può essere:

File, blocchi: allocazione

- L'**allocazione dei blocchi** di un file su disco può essere:
  - **contigua**: i blocchi di un file sono allocati consecutivamente;

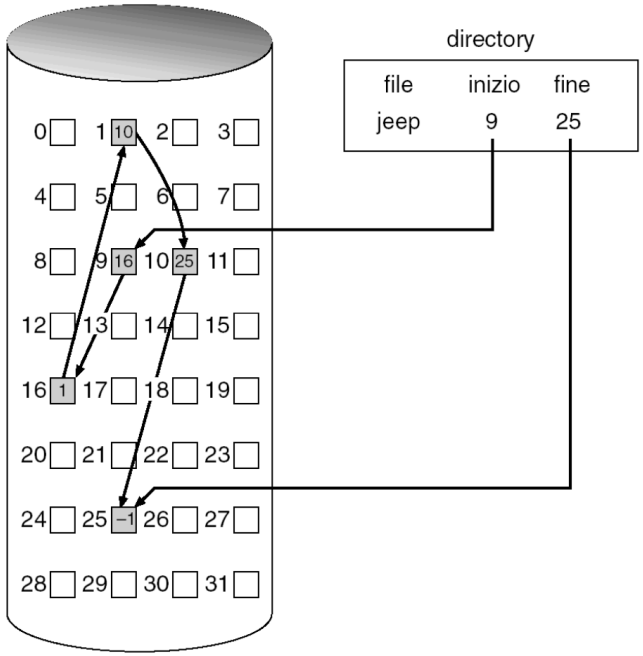


directory

file	inizio	lunghezza
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

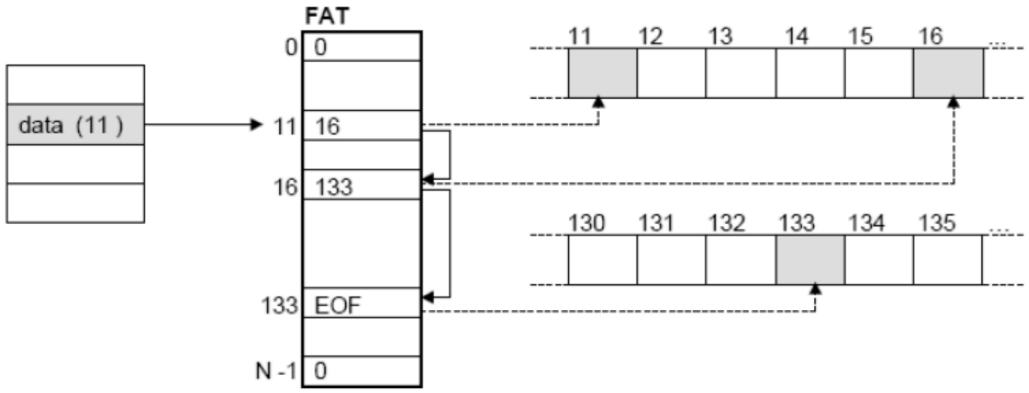
File, blocchi: allocazione

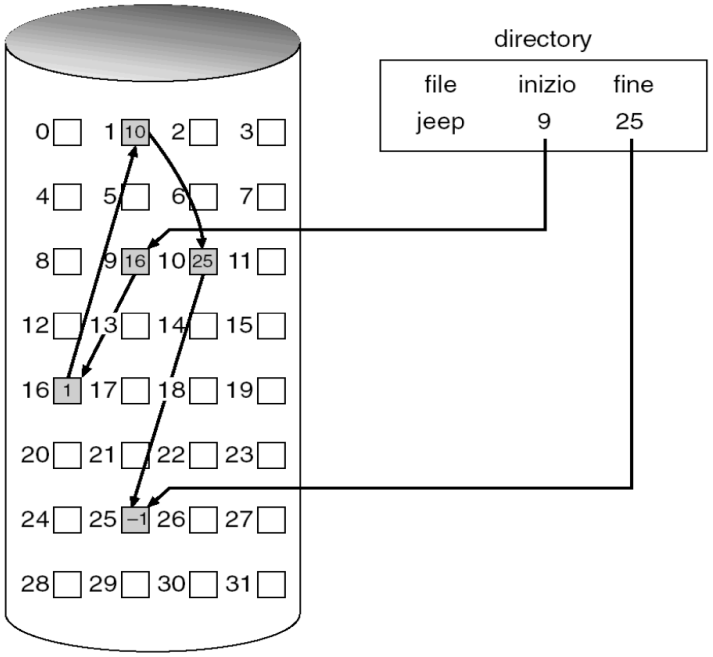
- L'**allocazione dei blocchi** di un file su disco può essere:
  - **contigua**: i blocchi di un file sono allocati consecutivamente;
  - **concatenata**: i blocchi non sono necessariamente consecutivi e sono collegati tra loro mediante puntatori;



File, blocchi: allocazione

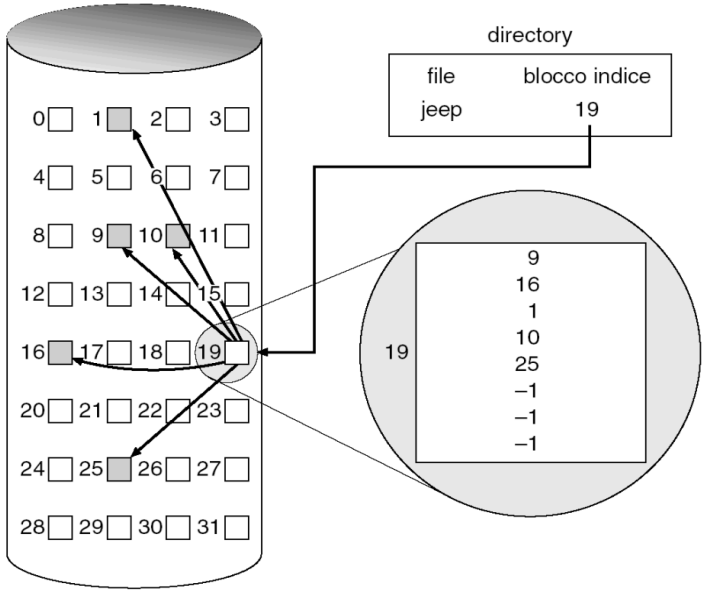
- L'**allocazione dei blocchi** di un file su disco può essere:
  - **contigua**: i blocchi di un file sono allocati consecutivamente;
  - **concatenata**: i blocchi non sono necessariamente consecutivi e sono collegati tra loro mediante puntatori;





File, blocchi: allocazione

- L'**allocazione dei blocchi** di un file su disco può essere:
  - **contigua**: i blocchi di un file sono allocati consecutivamente;
  - **concatenata**: i blocchi non sono necessariamente consecutivi e sono collegati tra loro mediante puntatori;
  - **indicizzata**: uno o più blocchi indice contengono i puntatori ai blocchi di dati del file.



Organizzazione interna dei file

- I dati nei file sono generalmente memorizzati in forma di record:
  - un record è costituito da un insieme di valori (campi) collegati
  - ogni tupla di una relazione corrisponde ad un record, ogni attributo ad un campo
- un file è una collezione di record:
  - file con record a lunghezza fissa se tutti i record memorizzati nel file hanno la stessa dimensione (in byte)
  - file con record a lunghezza variabile sono però necessari per:
    - memorizzazione di tipi di record diversi nello stesso file
    - memorizzazione di tipi di record con campi di lunghezza variabile (varchar)
    - memorizzazione di tipi di record opzionali
    - memorizzazione di tipi di record con campi multivalore (es. basi di dati OO o OR)

Organizzazione interna dei file

- Un file può essere visto come una collezione di record
- Tuttavia, poiché i dati sono trasferiti in blocchi tra la memoria secondaria e buffer, è importante assegnare i record ai blocchi in modo tale che uno stesso blocco contenga record tra loro correlati
- Se si riesce a memorizzare sullo stesso blocco record che sono spesso richiesti insieme si risparmiano accessi a disco

Mapping di relazioni a file

- L'organizzazione dei file, sia in termini di distribuzione dei record nei blocchi sia relativamente alla struttura all'interno dei singoli blocchi è gestita direttamente dal DBMS.
- Per DBMS di:
  - piccole dimensioni, una soluzione spesso adottata è di memorizzare ogni relazione in un file separato
  - grandi dimensioni, una strategia frequente è di allocare un unico grosso file, in cui sono memorizzate tutte le relazioni

28

File, blocchi, record, fattore di blocco

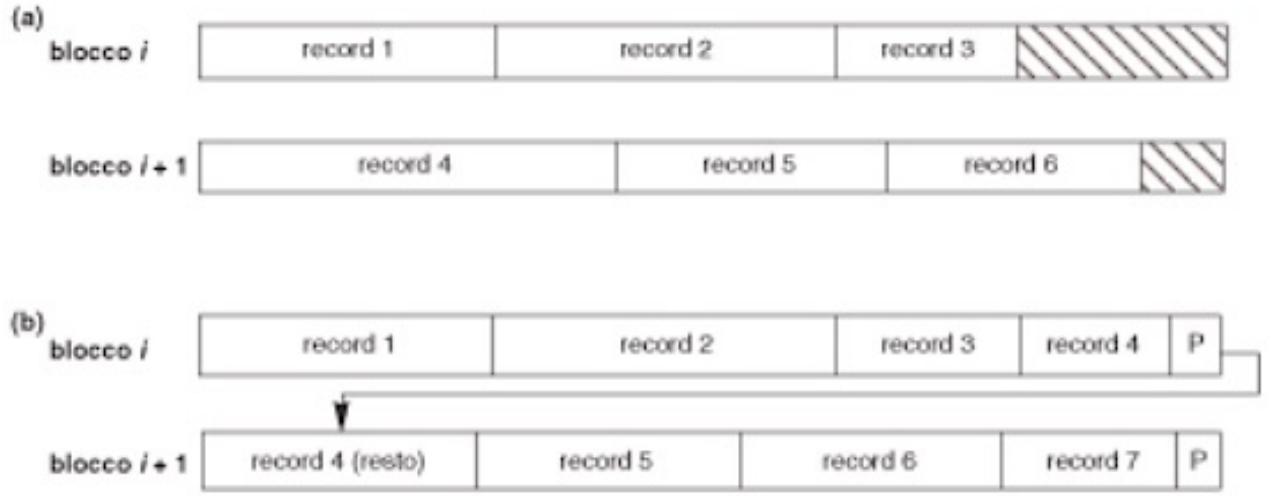
- I blocchi (componenti "fisici" di un file) e i record (componenti "logici") hanno dimensioni in generale diverse:
  - la dimensione del blocco dipende dal file system
  - la dimensione del record dipende dalle esigenze dell'applicazione (DBMS –> tuple), e può anche variare nell'ambito di un file
- Fattore di blocco: numero di record in un blocco
  - L_R: dimensione di un record (nei record a lunghezza fissa, o media della dimensione)
  - L_B: dimensione di un blocco
  - se $L_B > L_R$, possiamo avere più record in un blocco:

$$\lfloor L_B / L_R \rfloor$$

File, blocchi, record, fattore di blocco

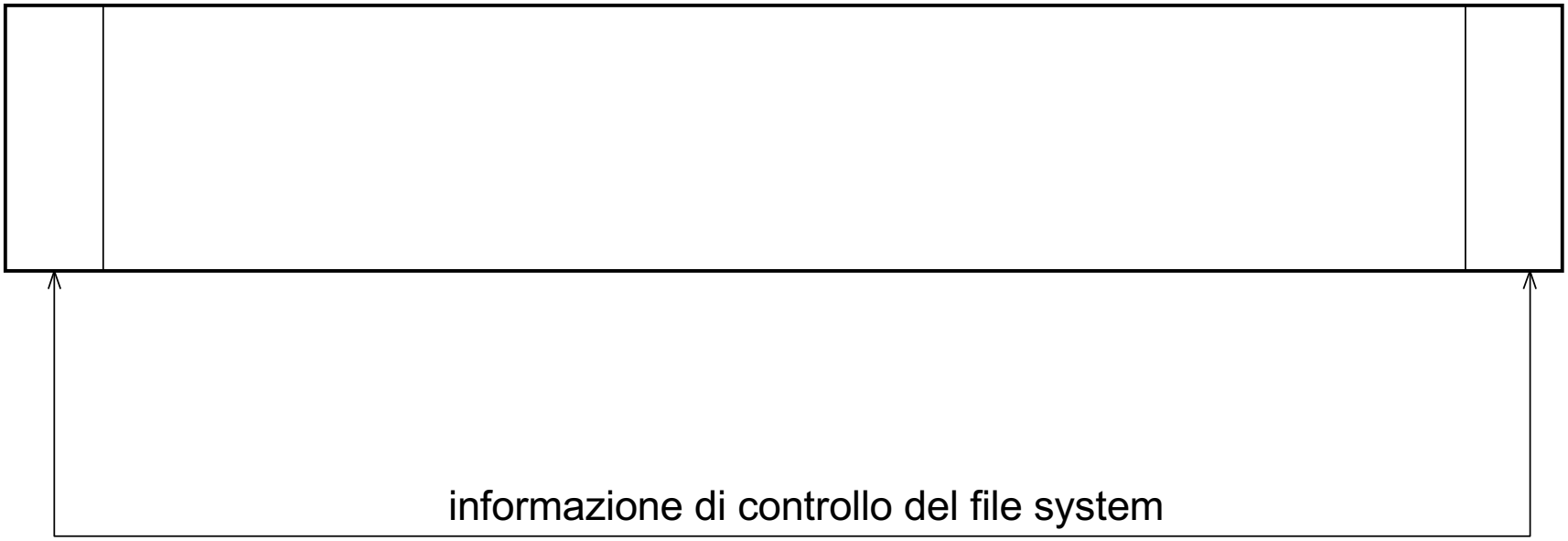
L'**allocazione dei record** nei blocchi può essere

- **unspanned**, in cui ogni record è interamente contenuto in un blocco, oppure
- **spanned**, in cui è possibile allocare un record a cavallo di più blocchi.



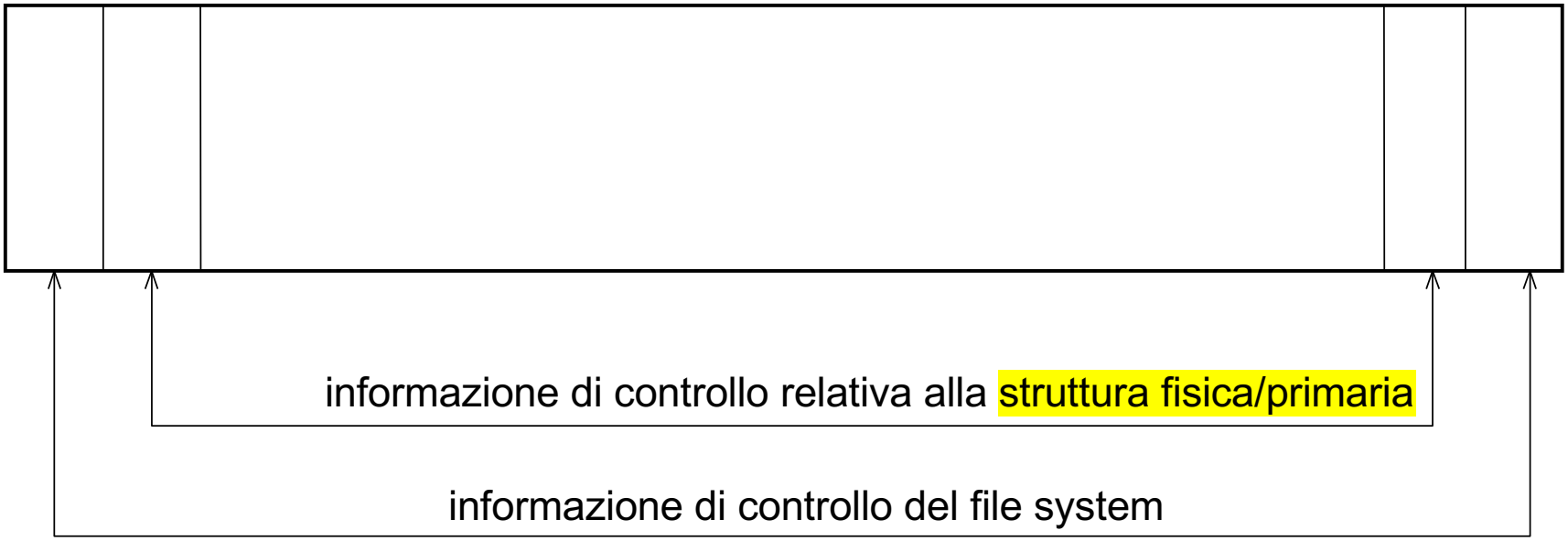
Organizzazione dei record all'intero di un blocco

- Ci sono varie alternative per organizzare i record (tuple) nei blocchi, vediamo una possibilità



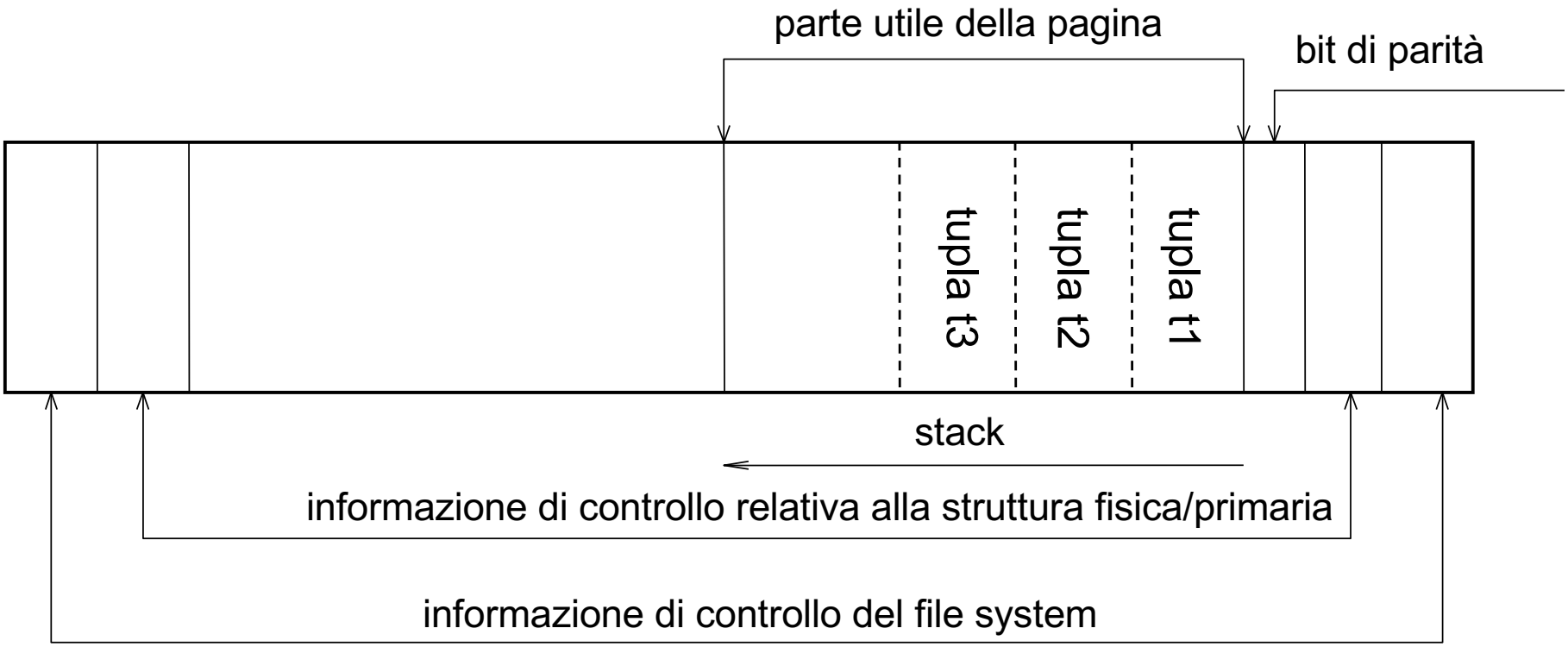
Organizzazione dei record all'intero di un blocco

- Ci sono varie alternative per organizzare i record (tuple) nei blocchi, vediamo una possibilità



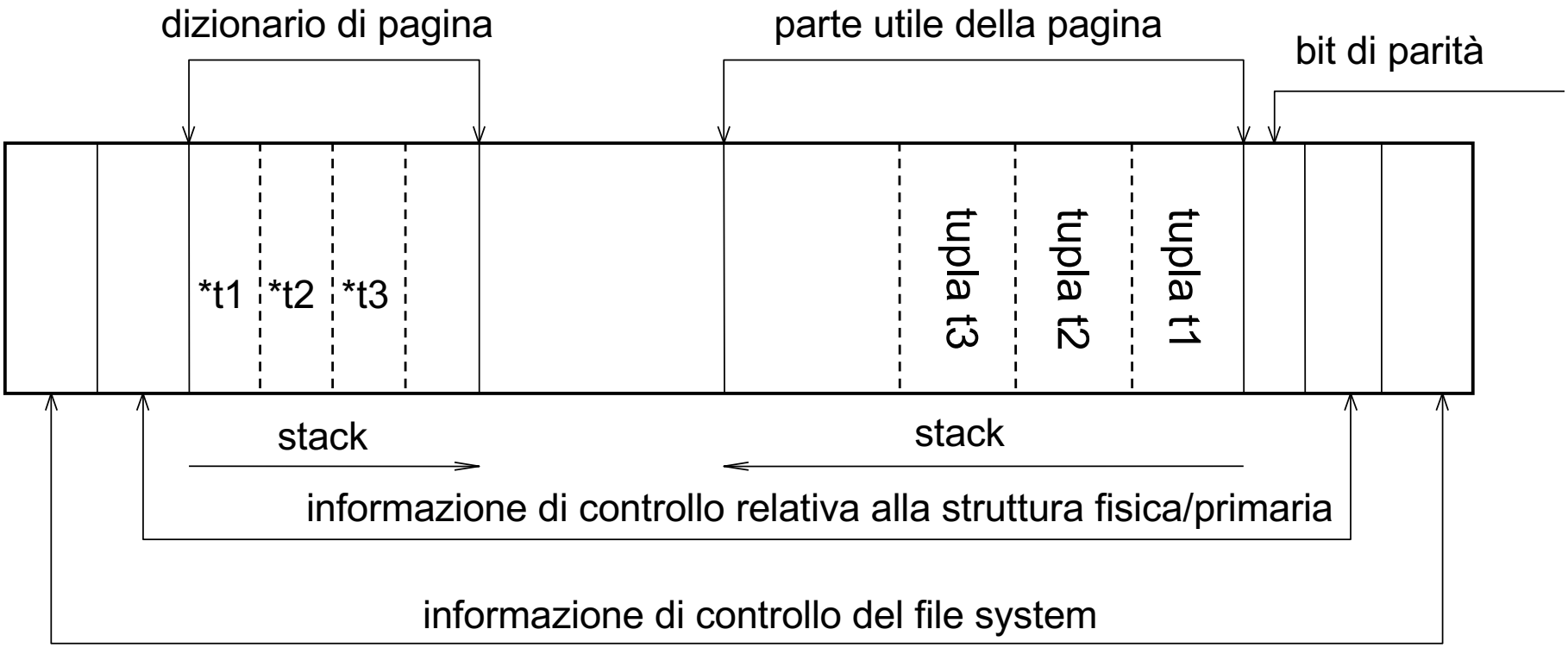
Organizzazione dei record all'intero di un blocco

- Ci sono varie alternative per organizzare i record (tuple) nei blocchi, vediamo una possibilità



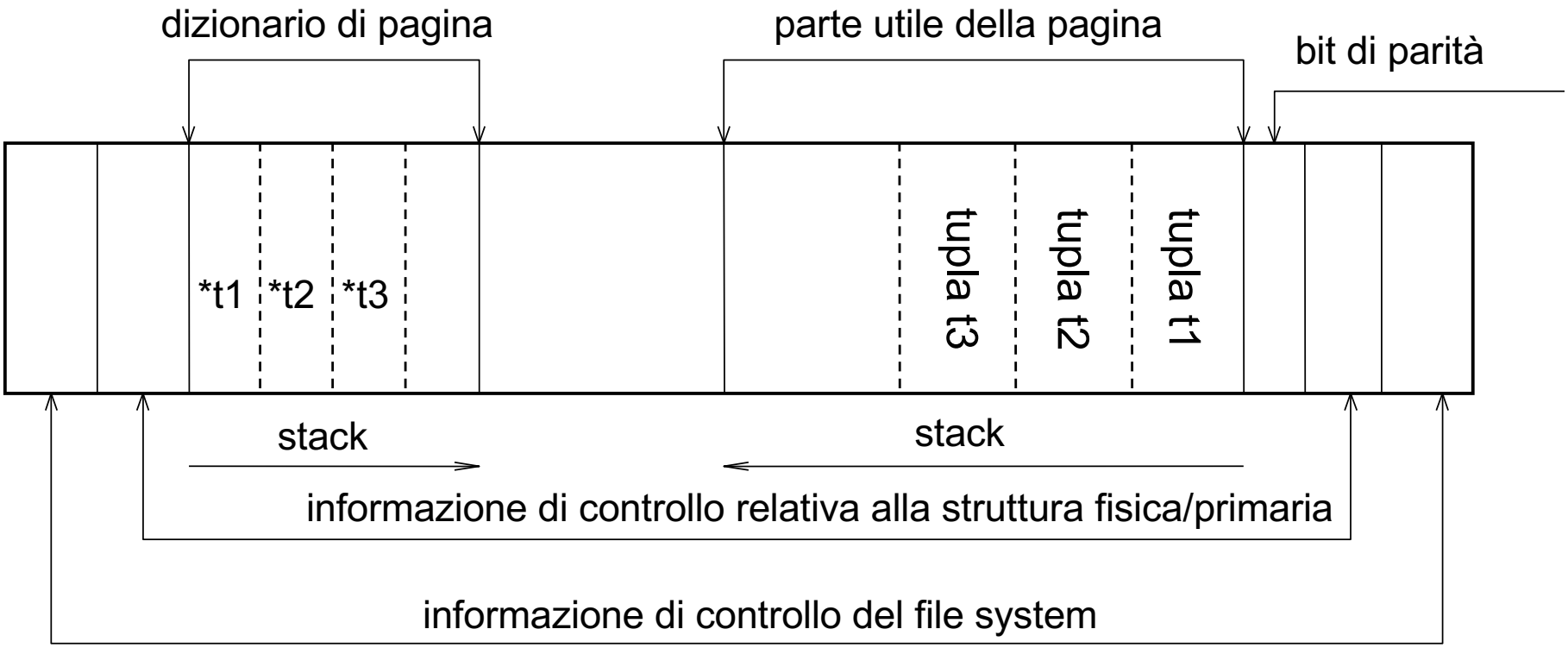
Organizzazione dei record all'intero di un blocco

- Ci sono varie alternative per organizzare i record (tuple) nei blocchi, vediamo una possibilità

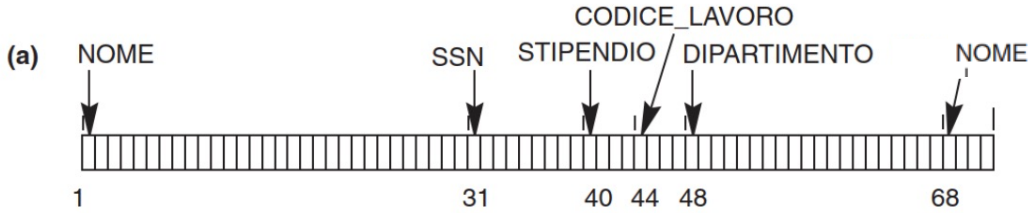


Organizzazione record all'intero di un blocco

- Dizionario di pagina contiene puntatori ai record/tuple
  - Record dim. fissa: …..
  - Record dim. variabile: ….



Organizzazione record all'intero di un blocco:
Dizionario di pagina – record lunghezza variabile

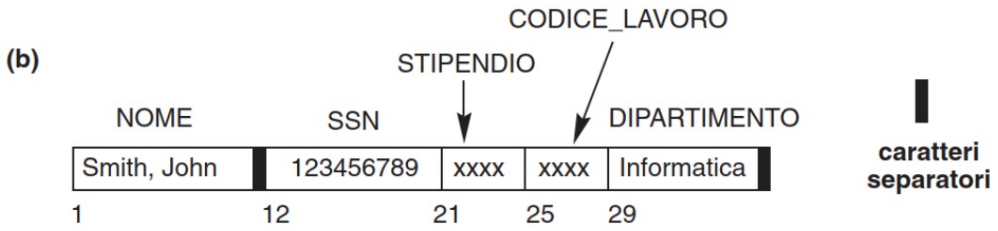



```
struct impiegato {
char nome[30];
char ssn[9];
int stipendio;
int codice_lavoro;
char dipartimento[20];
} ;
```

Puntatore campo nome del 3° record?
Puntatore campo ssn del 5° record?

Organizzazione record all'intero di un blocco: Dizionario di pagina – record lunghezza fissa

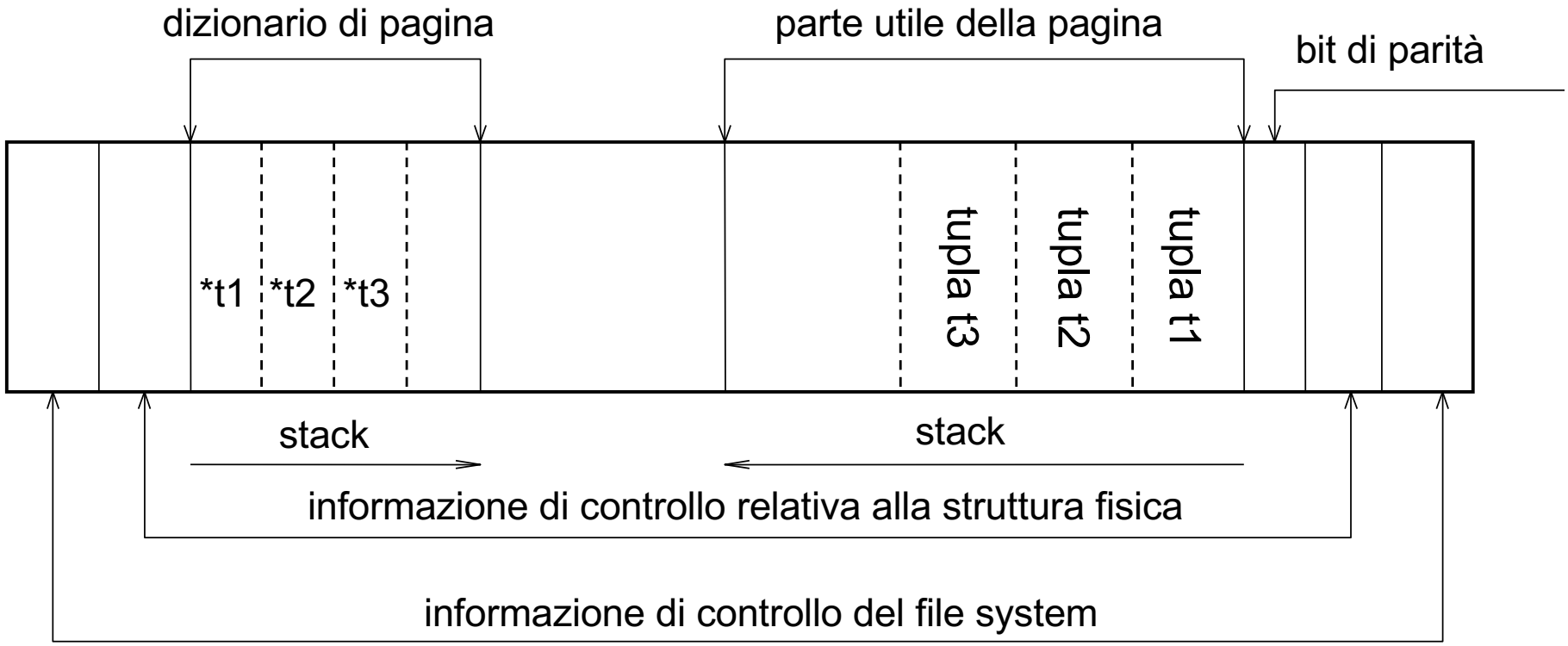
```
struct impiegato {
varchar nome[30];
char ssn[9];
int stipendio;
int codice_lavoro;
char dipartimento[20];
} ;
```



Per ogni record memorizzo l'offset da inizio parte dati
Per ogni campo, memorizzo offset da inizio record

Organizzazione record all'intero di un blocco

- Dizionario di pagina contiene puntatori ai record/tuple
  - Record dim. fissa: facile
  - Record dim. variabile: contiene offset di ogni tupla dall'inizio parte utile, e di ciascun campo da inizio tupla



Organizzazione dei record all'interno del file: strutture primarie

- **Strutture primaria**: Criterio con cui si memorizzano le tuple all'interno del file
- Tecniche principali
  - **Disordinata/seriale/heap***:* ordinamento fisico ma non logico
  - **ordinata**: l'ordinamento delle tuple coerente con quello di un campo
  - **Hash /accesso calcolato***:* posizioni individuate attraverso indici

Organizzazione dei record all'interno del file: strutture primarie: disordinata (heap)

- I record sono inseriti nel file nell'ordine in cui si presentano (ordine di inserimento nella relazione)
- Operazioni:
  - Inserimento (molto efficiente):
    - Un nuovo record viene inserito nell'ultimo blocco (richiesto solo un rif. all'ultimo blocco)
    - ATTENZIONE: se record hanno vincoli di chiave, è richiesta eseguire ricerca
  - Ricerca (poco efficiente):
    - Ricerca sequenziale. Costo lineare al numero di blocchi (mediamente si deve leggere metà dei blocchi)
  - Cancellazione:
    - Richiedere una ricerca per trovare il record da cancellare
    - Il record da cancellare può essere marcato come cancellato, senza ulteriori modifiche su altri record
      - Potrebbe portare a spreco di memoria e richiede periodica riorganizzazione dei file
  - Modifica:
    - Richiede una ricerca per trovare il record da modificare
    - Modifica
      - Si sovrascrive, se la modifica non cambia la lunghezza del record (aka record lunghezza fissa)
      - Altrimenti, si cancella il vecchio record e si inserisce il nuovo record con la modifica
- È molto diffusa nelle basi di dati relazionali, associata a indici secondari (discussi dopo)

Costo ricerca in termini di numero accessi

- Si supponga un file heap con r=300.000 record memorizzati su disco con blocco B=4096 di lunghezza. I record hanno lunghezza fissa R=100 byte e sono memorizzati in modo unspanned.

Mediamente, quanti accessi sono richiesti per ricerca un record?

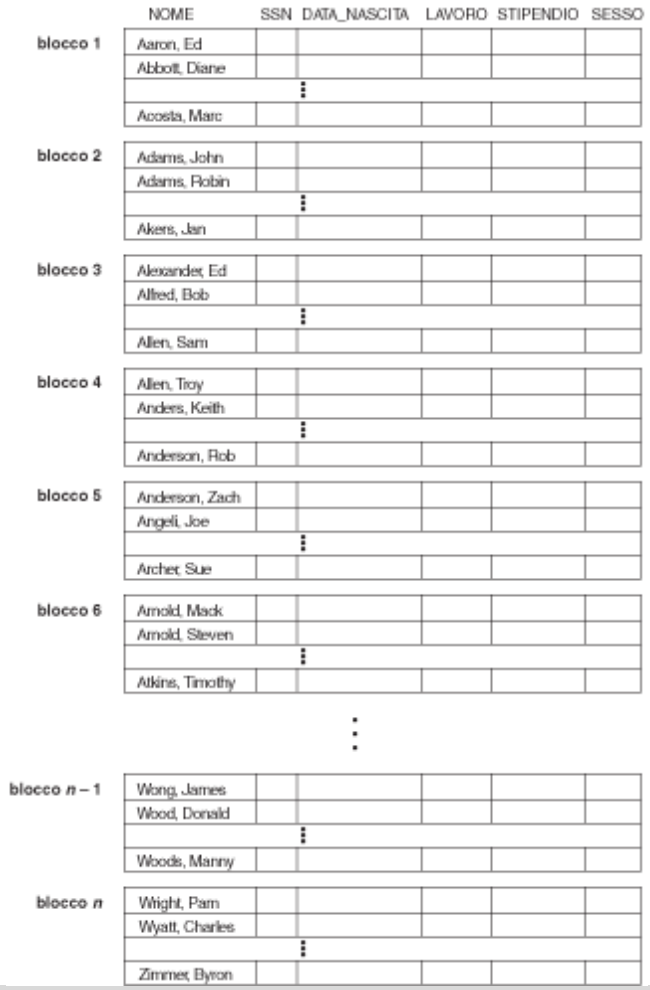
- Fattore di blocco $\lfloor 4096/100 \rfloor = 40$
- Numero blocchi per memorizzare il file: $\lceil 300.000/40 \rceil = 7500$
- Numero di accessi ai blocchi: 7500/2= 3750

Organizzazione dei record all'interno del file: strutture primarie: ordinata (sorted)

- I record sono inseriti rispettando l'ordinamento di un campo (detto campo di ordinamento o chiave di ordinamento)

Organizzazione dei record all'interno del file: strutture primarie: ordinata (sorted)

- I record sono inseriti rispettando l'ordinamento di un campo (detto campo di ordinamento o chiave di ordinamento)



Organizzazione dei record all'interno del file: strutture primarie: ordinata (sorted)

- I record sono inseriti rispettando l'ordinamento di un campo (detto campo di ordinamento o chiave di ordinamento)
- Operazioni:
  - Inserimento (poco efficiente):
    - L'inserimento di un nuovo record deve rispettare l'ordinamento del campo. Questo può richiedere di spostare diversi record per inserirne uno nuovo
  - Ricerca (efficiente su campo ordinato):
    - La lettura dei record secondo l'ordine del campo d'ordinamento è estremamente facile (es. stampo impiegato ordinati per cognome)
    - Ricerca sul campo d'ordinamento ottimizzate
      - Ricerco singolo record -> ricerca binaria
      - Ricerco gruppi record con valori inclusi in un intervallo (range query) -> trovo il primo valore e leggo sequenzialmente
  - Cancellazione
    - Come per heap, con necessità di riorganizzare file
  - Modifica (poco efficiente)
    - Se non è possibile sovrascrivere il record modificato (aka cambia la lunghezza), è necessario riorganizzare il file per garantire l'ordinamento

Esercizio: ricerca binaria sul file ordinati

- Si assuma un file ordinato sul campo ID di B blocchi
- Si scriva il pseudocodice per eseguire una ricerca binaria del record con campo di ordinamento = k

Esercizio: ricerca binaria sul file ordinati

```
Lower=1;
Upper= b;
While (Upper>=Lower)
        i=(Lower+Upper)/ 2
        leggo il blocco i
        Leggo t0,  il primo record del blocco i
        if (k<t0.id)
                Upper=i-1
        else
                leggo tn, l'ultimo record del blocco i
                If (k> t.id)
                        Lower=i+1;
                else
                        trovato / il blocco con il record è   in memoria
                        exit
endWhile
```

Quanti accessi in memoria secondaria sono necessari?

Costo ricerca su file ordinato

- Si assuma un file ordinato di B blocchi: quanti accessi in memoria secondaria sono necessari per una ricerca binaria?

Numero accessi: $\lceil \log_2 B \rceil$

Costo ricerca in termini di numero accessi

- Si supponga un file ordinato su campo ID con r=300.000 record memorizzati su disco con blocco B=4096 di lunghezza. I record hanno lunghezza fissa R=100 byte e sono memorizzati in modo unspanned.

Mediamente, quanti accessi sono richiesti per ricerca sul campo di ordinamento?

- Fattore di blocco $\lfloor 4096/100 \rfloor = 40$
- Numero blocchi per memorizzare i record: $\lceil 300.000/40 \rceil = 7500$
- Numero di accessi ai blocchi per ricerca binaria: $\lceil \log_2 7500 \rceil = 13$

Costo ricerca in termini di numero accessi

r=300.000 record memorizzati su disco con blocco B=4096 di lunghezza. I record hanno lunghezza fissa R=100 byte e sono memorizzati in modo unspanned.

- File heap: 3750 accesi
- File ordinato sul campo di ricerca: 13 accessi

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato

- I record sono inseriti in base al valore di un campo chiave (non per forza chiave primaria)
- La locazione dei dati dipende dal **valore** -> *accesso associativo*

ESEMPIO

Si consideri un azienda dove gli unici impiegati hanno matricola da 1 a 100.

- *memorizzo i record impiegati ordinati per il campo matricola*
- *Il **valore della matricola** indica la posizione del record e del blocco*
  - *Es. con fattore blocco 2, il record dell'impiegato con matricola 11 è il primo record del 6° blocco*

Abbiamo un accesso diretto ad un insieme di record sulla base del valore di un campo (**chiave**)

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato

- Se il numero dei valori della chiave è paragonabile al numero di record effettivi
  - Il valore della chiave rappresenta l'indice di un array.
  - Es. matricola=11, elemento 11 dell'array
- Se il numero dei valori della chiave è molto più ampio dei record effettivi?
  - Matricola impiegati codificata in 6 cifre (1 milione di possibili valori), ma solo 100 impiegati
  - Array sparso!
- Volendo continuare ad usare qualcosa di simile ad un array, ma senza sprecare spazio, possiamo pensare di trasformare i valori della chiave in possibili indici di un array:

Funzione di hash

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato (hash)

- **Funzione di hash**
  - associa ad ogni valore della chiave un "indirizzo", in uno spazio di dimensione paragonabile (leggermente superiore) rispetto a quello strettamente necessario
  - poiché il numero di possibili chiavi è molto maggiore del numero di possibili indirizzi esiste la possibilità di collisioni (chiavi diverse che corrispondono allo stesso indirizzo)

Un esempio

- Matricola 6 cifre
- 40 record

M
60600
66301
205751
205802
200902
116202
200604
66005
116455
200205
201159
205610
201260
102360
205460
205912
205762
200464
205617
205667

M
200268
205619
210522
205724
205977
205478
200430
210533
205887
200138
102338
102690
115541
206092
205693
205845
200296
205796
200498
206049

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50

M	M mod 50
60600	
66301	
205751	
205802	
200902	
116202	
200604	
66005	
116455	
200205	
201159	
205610	
201260	
102360	
205460	
205912	
205762	
200464	
205617	
205667	

M	M mod 50
200268	
205619	
210522	
205724	
205977	
205478	
200430	
210533	
205887	
200138	
102338	
102690	
115541	
206092	
205693	
205845	
200296	
205796	
200498	
206049	

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - Collisioni?

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

In caso di collisione di n elementi, l'accesso al primo costa 1, quello al secondo costa 2, cosi via fino al costo pari a n per l'n-esimo.

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

In caso di collisione di n elementi, l'accesso al primo costa 1, quello al secondo costa 2, cosi via fino al costo pari a n per l'n-esimo.

Numero medio di accessi alla struttura dati per un generico elemento?

1,425

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Un esempio

- Matricola 6 cifre
- 40 record
- Array di 50 elementi
- Funzione hash: Mod 50
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

Quanto costano le collisioni in termini di accessi?

In caso di collisione di n elementi, l'accesso al primo costa 1, quello al secondo costa 2, cosi via fino al costo pari a n per l'n-esimo.

Numero medio di accessi alla struttura dati per un generico elemento?

1,425

M	M mod 50
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

Organizzazione dei record all'interno del file: strutture primarie: accesso calcolato (hash)

- Funzione hash applicate all'organizzazione di file:
  - IDEA: Il valore hash del campo chiave di un record indica il blocco in cui il record è memorizzato
  - Record con stesso valore hash sono memorizzati nello stesso blocco
- Per gestire future collisioni, un blocco viene riempito solo parzialmente (fattore riempimento) in modo da lasciare spazio libero
- Dato
  - T il numero di record previsto
  - F il fattore di blocco
  - f il fattore di riempimento (0,1)

  il file avrà un numero di blocchi $B = \lceil T/(f \times F) \rceil$

  Funzione hash applicata al campo chiave restituisce un valore da 0 a B-1

- In questo modo si "ammortizzano" le probabilità di collisione
- Quando uno spazio relativo ad un blocco viene esaurito, viene allocato un ulteriore blocco collegato al precedente (catena di overflow)

Un esempio

- Matricola 6 cifre
- 42 record

- file hash con fattore di blocco 10;
- Servono 5 blocchi
- Funzione hash per indirizzare 5 blocchi
  - Mod 5

M
60600
66301
205751
205802
200902
116202
200604
66005
116455
200205
201159
205610
201260
102360
205460
205912
205762
200464
205617
205667

M
200268
205619
210522
205724
205977
205478
200430
210533
205887
200138
102338
102690
115541
206092
205693
205845
200296
205796
200498
206049

M
220256
220258

Un file hash

file hash con fattore di blocco 10
5 blocchi con 10 posizioni ciascuno:
due soli overflow!

Numero medio di accessi alla struttura dati per un generico elemento? 1,05

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260	220256	205617	205693	206049
102360		205667	200498	
205460		210522	220258	
200430		205977		
102690		205887		

205845

206092

Un file hash

file hash con fattore di blocco 10
5 blocchi con 10 posizioni ciascuno:
due soli overflow!

Numero medio di accessi alla struttura dati per un generico elemento? 1,05

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260	220256	205617	205693	206049
102360		205667	200498	
205460		210522	220258	
200430		205977		
102690		205887		
205845		206092		

File hash, osservazioni

- È l'organizzazione più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza (accesso puntuale):
  - costo medio di poco superiore all'unità
- Non è efficiente per ricerche basate su intervalli (né per ricerche basate su altri attributi)
- Le collisioni (overflow) sono di solito gestite con blocchi collegati
- Non adatto per file che crescono tanto (oltre al fattore di riempimento)