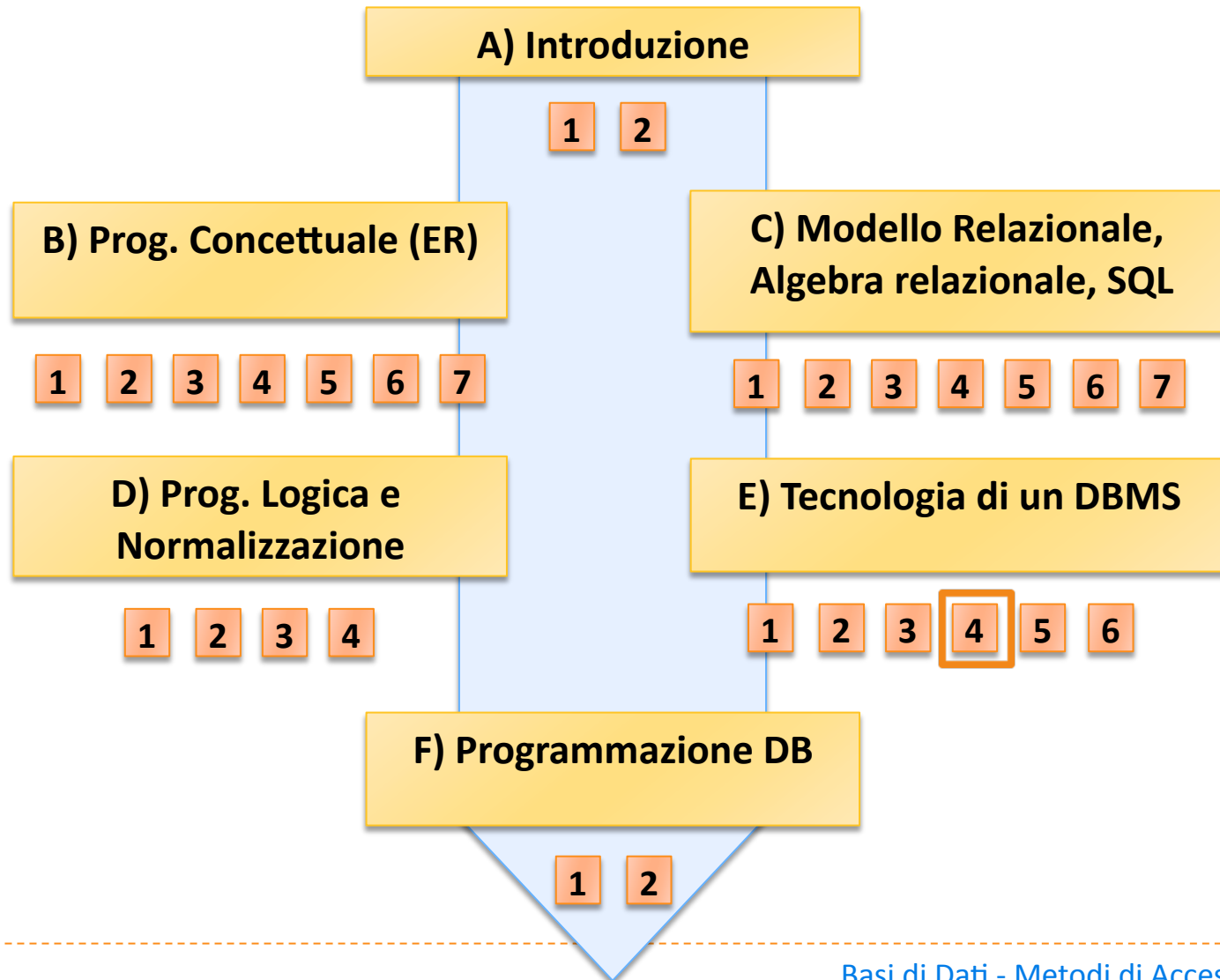


Basi di Dati

Metodi di Accesso (Hash)

Basi di Dati – Dove ci troviamo?



Organizzazioni hash

- ▶ **IDEA DI BASE** : associare ad un file di NB blocchi una funzione che trasformi un valore di chiave k_i in un numero di blocco b_i tra 1 ed NB
 - ▶ una tupla viene memorizzata nel blocco b_i
 - ▶ ad ogni blocco è associato un indirizzo nel disco
 - ▶ un blocco puo' contenere 1 o più tuple

Organizzazioni hash

La **regola** con cui ad una chiave **k** viene associato un numero di blocco **b** si chiama **funzione hash** o **algoritmo di hashing**.

tupla : < chiave, informazione >

↳ $\text{HASH}(\text{chiave}) = b$



b-1



b



b+1

Organizzazioni hash

Il file è ben utilizzato se:

- ci sono pochi spazi vuoti, cioè se il fattore di "packing" (caricamento) è elevato:

$$F_p = m/M < 1, \text{ dove:}$$

M = numero massimo di record per blocco,

m = numero medio di record per blocco

- l'occupazione effettiva dei blocchi è vicina alla media

Fp basso sta a significare che il file è vuoto,

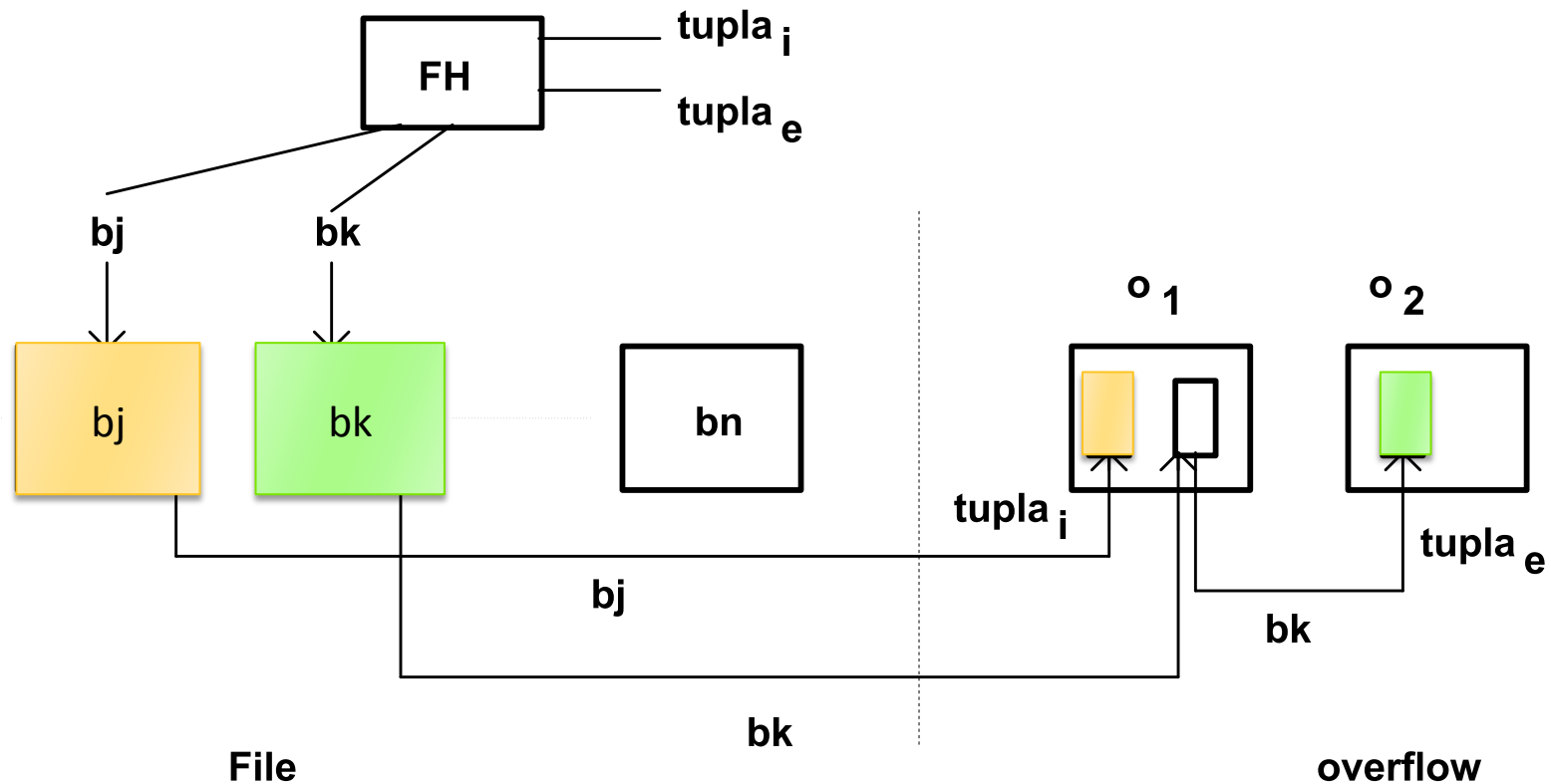
Fp vicino ad 1 da significare che molti blocchi

sono pieni. Con i blocchi pieni si creano problemi di

OVERFLOW

Organizzazioni hash

Quando un blocco (es. b_j) è pieno ed una nuova tupla deve essere inserita poiché **FH** (f. hash) gli ha assegnato quel blocco (b_j) allora si deve creare una **lista (catena) di overflow**.



Organizzazioni hash

L'**OVERFLOW** avviene per due motivi:

- ci sono più tuple con lo **stesso valore di chiave** per cui FH le manda nello stesso blocco
- FH **non è perfetta** ed assegna a due o più valori di chiave diversi lo stesso blocco del file (collisione)

con lunghe liste di **overflow** si degrada l'efficienza

FH deve distribuire le tuple il più **uniformemente** possibile nei blocchi. Osserviamo che:

- più piccolo è il blocco rispetto alle tuple più è alto F_p ma **più** probabile è l'overflow,
- più grande è il blocco rispetto alle tuple più è basso F_p ma è **meno** probabile l'overflow.

Metodi per la funzione Hash

1) MID-SQUARE:

- ▶ La chiave (convertita in numero se alfanumerica) è moltiplicata per se stessa ed i **numeri centrali del quadrato** vengono presi e normalizzati per rientrare nel "range" NB del numero di blocchi del file.
- ▶ ES: se un record ha una chiave di 6 cifre ed il file ha NB=7000 blocchi, il quadrato ha 12 cifre e si prendono le quattro dalla 5^a alla 8^a, se il numero è 172148, il quadrato è 029634933904 e le cifre centrali sono 3493.

Metodi per la funzione Hash

- ▶ Per evitare che il numero così ottenuto sia maggiore di 7000, lo si **normalizza** moltiplicandolo per $7000/10000$, cioè il numero del blocco che risulta è $0.7 \times 3493 = 2445$.
- ▶ I due numeri immediatamente precedenti e successivi danno:

172146	⇒	3224	⇒	2397
172147	⇒	3450	⇒	2415
172149	⇒	3527	⇒	2469
172150	⇒	3562	⇒	2493
- ▶ Due valori di K successivi non danno luogo a blocchi successivi

Metodi per la funzione Hash

2) DIVISIONE

- ▶ La chiave K viene **divisa per il numero primo** più vicino ad NB ed il resto viene preso come numero del blocco

Es. 172148 viene diviso per 6997 (numero primo più vicino < NB) ed il resto è 4220

172146 \Rightarrow 4218
172147 \Rightarrow 4219
172149 \Rightarrow 4221
172150 \Rightarrow 4222

- ▶ Il metodo della divisione non è un "**randomizzatore**", ma assegna valori successivi di K a blocchi successivi.

Metodi per la funzione Hash

3) SHIFTING

- ▶ Le **cifre** della chiave **K** vengono **divise** in due (o più) gruppi, vengono **spostate** in modo da sovrapporsi per un numero di cifre pari al numero di cifre che rappresentano **NB** blocchi, i frammenti di **K** vengono sommati ed il risultato viene normalizzato.

Es: : 17207329 ⇒ 1720 | 7329 ⇒

$$\begin{array}{r} 1720 + \\ 7329 = \\ \hline 9049 \end{array}$$

8 cifre ⇒ 4 cifre per NB

Metodi per la funzione Hash

4) FOLDING

- ▶ I frammenti vengono "ripiegati" (fold) su se stessi e poi sommati, le cifre in eccesso vengono eliminate
- ▶ ES.: 17207329 \Rightarrow 17 | 207 | 329

$$\begin{array}{r} 8 \text{ cifre} \Rightarrow 3 \text{ cifre} \\ 207 + \\ 923 + \\ \hline 710 = \\ 1840 \Rightarrow 840 \end{array}$$

SHIFTING e FOLDING vengono generalmente adoperati per chiavi alfa- numeriche molto lunghe

Metodi per la funzione Hash

- ▶ **Esempio**: sequenza di 22 registrazioni di cui si evidenzia solo la chiave, da memorizzare in un archivio di 13 blocchi la cui capacità è 2.
- ▶ **funzione hash**: i caratteri della chiave vengono convertiti in numeri prendendo i primi 4 bit della codifica BCD, successivamente se la stringa numerica così ottenuta è più lunga di 5 la si spezza in gruppi di 5 a partire da sinistra e poi si sommano i gruppi, infine si divide il numero per 13 e si prende il resto della divisione.

Metodi per la funzione Hash

DATI	NUMERO	RESTO (mod 13)
1 LUISA	34921	4
2 EDVIGE	54602	3
3 ADELE	14535	2
4 AGNESE	17557	8
5 FRANCESCA	74384	12
6 ROBERTA	96290	13
7 NOEMI	56549	13
8 ELENA	53551	5
9 LAURA	31491	6
10 BEATRICE	26074	10
11 DIANA	49151	12
12 IRENE	99555	2
13 PAOLA	71631	2
14 BERENICE	26890	7
15 VANESSA	51573	3
16 SARA	02191	8
17 CARLA	31931	10
18 OLGA	06371	2
19 GIORGIA	79788	8
20 LARA	03191	7
21 CAROLA	31964	11
22 GIOVANNA	80202	6

Metodi per la funzione Hash

indirizzo	ARCHIVIO		puntatore
1			
2	ADELE	IRENE	14
3	EDVIGE	VANESSA	
4	LUISA		
5	ELENA		
6	LAURA	GIOVANNA	
7	BERENICE	LARA	
8	AGNESE	SARA	15
9			
10	BEATRICE	CARLA	
11	CAROLA		
12	FRANCESCA	DIANA	
13	ROBERTA	NOEMI	
14	PAOLA	OLGA	overflow record
15	GIORGIA		overflow record

Come si può vedere i blocchi 1 e 9 non vengono riempiti, mentre 2 ed 8 vanno in overflow.

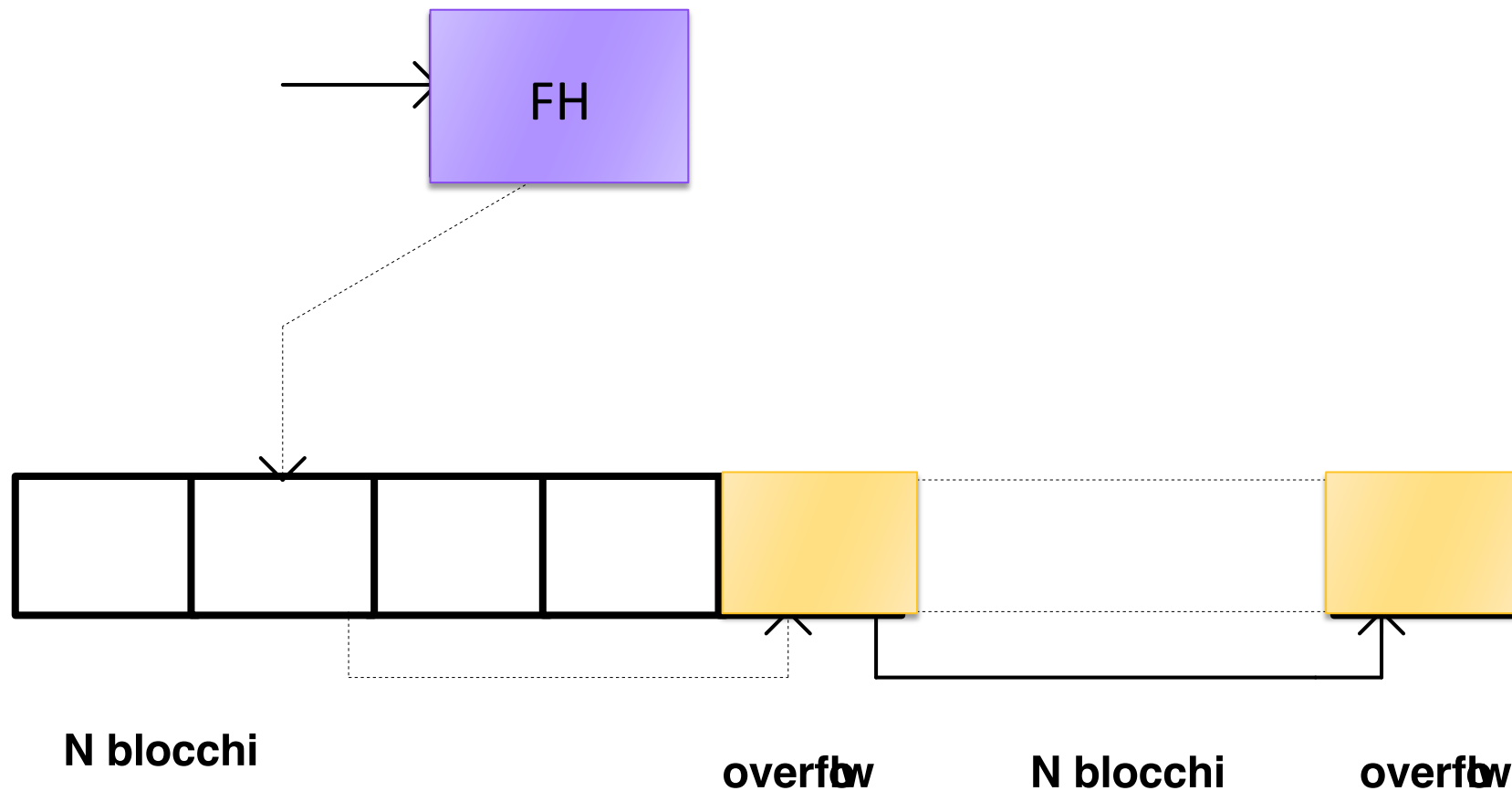
COSA VUOL DIRE "METODO MIGLIORE"?

- ▶ Il metodo migliore è quello che **distribuisce il più uniformemente possibile le tuple**, evitando di riempire troppo alcuni blocchi lasciandone vuoti altri. In generale bisogna fare esperimenti e poi eventualmente cambiare funzione Hash e/o il numero di blocchi NB.
- ▶ Il metodo **Mid-square** e quello che più si avvicina ad un generatore di valori **random**. Il metodo della **divisione** è in generale il migliore perché "**distribuisce**" **meglio** le tuple.
- ▶ Per un FILE HASH si devono definire un numero **NB** di blocchi che costituiscono la "**PRIME AREA**" e **NO** per la "**OVERFLOW AREA**"

Overflow distribuito

- ▶ **Un** blocco di overflow ogni **N** blocchi. Se non c'è posto nel primo ov. b. si carica la tupla nel successivo ov. b.
- ▶ Questo metodo si attua quando il file system alloca al file molti cilindri di disco, allora per ogni cilindro di **T** tracce le prime **T-1** contengono i blocchi del file e la **T**-esima contiene l'overflow . Ciò consente di cercare in overflow evitando il più possibile di spostare il pettine di lettura .
- ▶ Sinonimi
 - blocco: **bucket**, **pagina**
 - tupla: **record**, **n-pla**

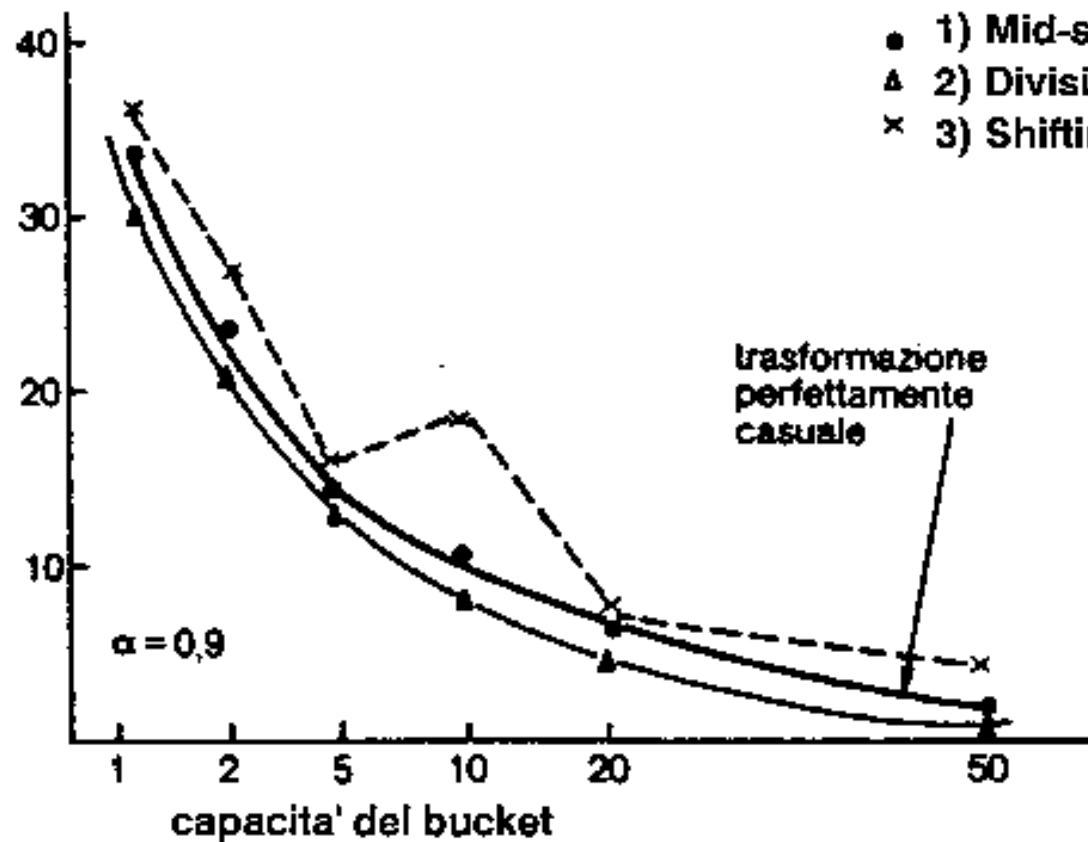
Overflow distribuito



Prestazioni

STATISTICHE SU FILES

percentuale di overflow



Prestazioni

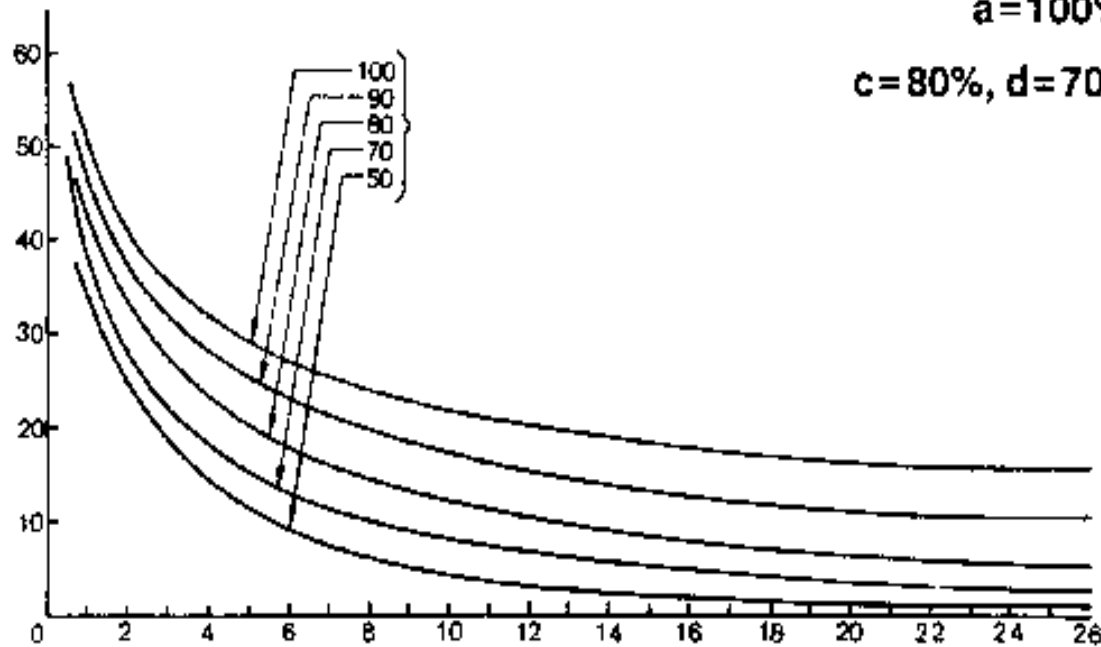
Statistiche generali con prove random

percentuale di overflow

fattore di caricamento:

a=100%, b=90%,

c=80%, d=70%, e=50%.



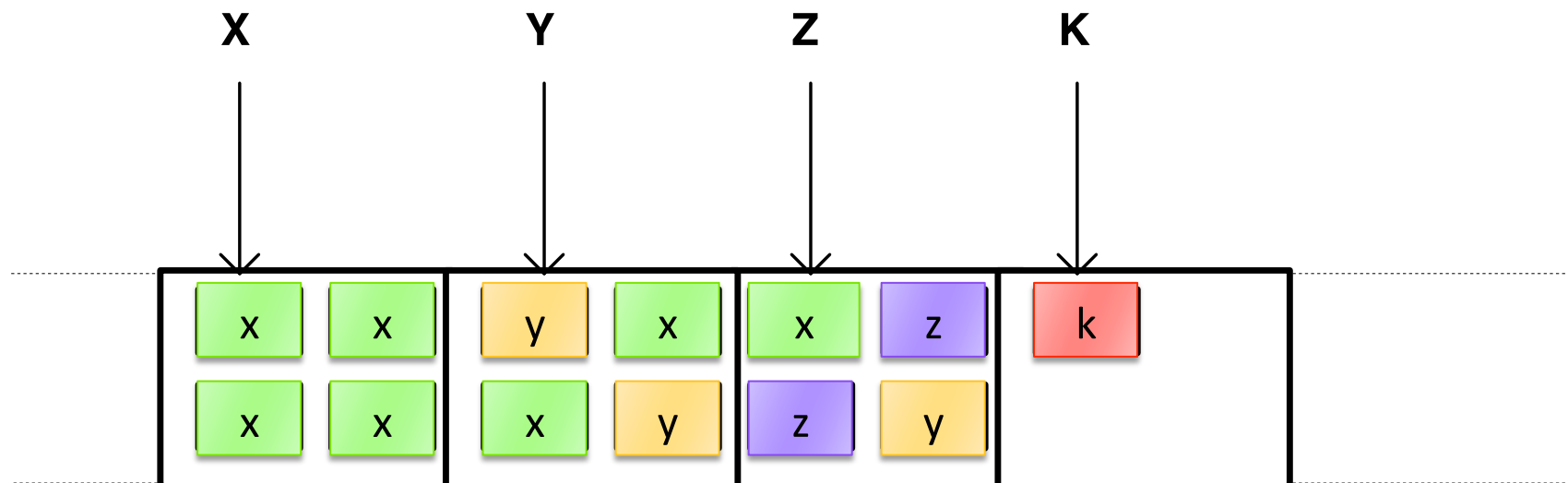
capacita' del bucket

Prestazioni

- ▶ **Es.:** 10000 tuple di 200 bytes con blocchi di 4096, capacità = 20
usando le curve del grafico si ha che:
- ▶ caricamento del **100% overflow** al **17%**,
NB= $10000/20=500$, NO = 85, tot.= 585
 $C_{hash} = 1$ nel 83% dei casi ≥ 2 nel 17%
- ▶ caricamento del **70% overflow** al **3%**,
NB= $10000/14=715$, NO= 22, tot. = 737
 $C_{hash} = 1$ nel 97% dei casi ≥ 2 nel 3%

Metodo open addressing

- Non ci sono blocchi di overflow.
 - I blocchi sono dimensionati con molta area vuota.
 - Se un blocco va in overflow il record viene messo nel primo blocco successivo con area disponibile.
- (NB va in overflow in 1)



4 record per blocco

Metodo open addressing

- ▶ **Prime area consecutive spill**: il record viene posto nel blocco successivo
- ▶ **Prime area skip spill**: il record viene posto in un blocco a distanza prefissata.
- ▶ **Non ci sono puntatori.**
- ▶ La **ricerca** di un record parte dal blocco indirizzato da FH e poi segue in modo **sequenziale** fintanto che viene trovata una posizione vuota, se il record fosse stato presente sarebbe infatti stato posto nella prima posizione vuota disponibile.

Metodo open addressing

- ▶ Questo metodo, poiché si basa sulla ricerca dei record fintanto che non si trova un "vuoto", non può rendere "vuoti" i record eliminati.
- ▶ I record eliminati vengono marcati come "non attivi" poiché servono alla ricerca dei successivi. I record marcati possono però venire utilizzati per accogliere nuovi record attivi.
- ▶ In conclusione ci sono 3 tipi di record: **ATTIVI, NON ATTIVI, VUOTI**
- ▶ Le **prestazioni** degradano per fattori di caricamento >60%

Gestione dell'overflow

Riassumendo...

- ▶ **Strategie di gestione dell'overflow:**
 - ▶ **a concatenamento (con blocchi di overflow)**
 - ▶ Blocchi di overflow alla fine dello spazio di indirizzamento
 - ▶ Blocchi di overflow distribuiti
 - ▶ **open addressing**
 - ▶ Prime area consecutive spill
 - ▶ Prime area skip spill

Hash dinamico

LINEAR HASHING (1980)

Scopi:

- ▶ **Contenimento dell'area di overflow**
- ▶ **Crescita controllata del file**
- ▶ **Riorganizzazione automatica**

I metodi statici si fermano all'esaurimento dell'overflow, devono essere monitorati e ristrutturati in tempo aumentando NB e facendo il re-hash con un altro divisore o altra FH

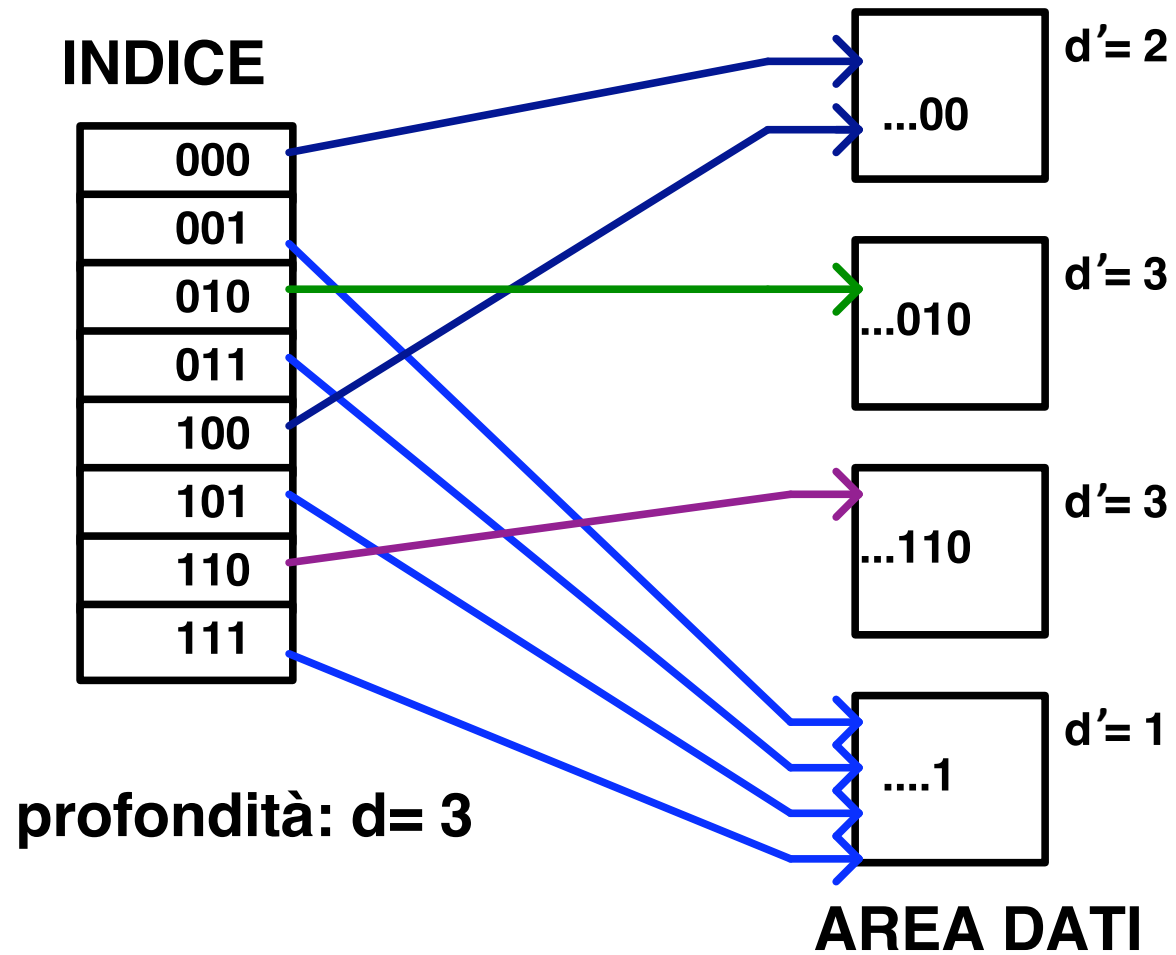
Extendible hashing

- ▶ **L'extendible hashing** è un metodo di hash dinamico che usa una struttura dati ausiliaria, fu proposto da Fagin (IBM) nel '79.
- ▶ Nello schema extendible hashing l'archivio è strutturato in due livelli : un **indice** e **l'area dati** composta da NB blocchi.
- ▶ Si suppone che **l'indice** possa in generale essere **contenuto in memoria centrale** e che quindi influisca in modo poco rilevante sul tempo di accesso all'archivio.

Extendible hashing

- ▶ L'**indice** è composto di coppie **<valore, puntatore >**:
 - ▶ il **valore** è costituito da **d bit**,
 - ▶ il **puntatore** indirizza un blocco nell'area dati,
 - ▶ **d** è chiamato "**profondità dell'indice**",
 - ▶ l'**indice** è costituito da **$P = 2^d$** puntatori che puntano agli NB blocchi con **$P \geq NB$** ,
 - ▶ i **puntatori** non sono necessariamente tutti distinti, ma se vi sono più puntatori allo stesso blocco essi sono **2^q** con **$q < d$** , in questo caso si dice che il blocco ha profondità locale **$d' = d - q$** .
- ▶ Poiché i valori sono sempre un numero **p** potenza di 2 **non c'è bisogno di memorizzarli**.

Extendible hashing



Extendible hashing

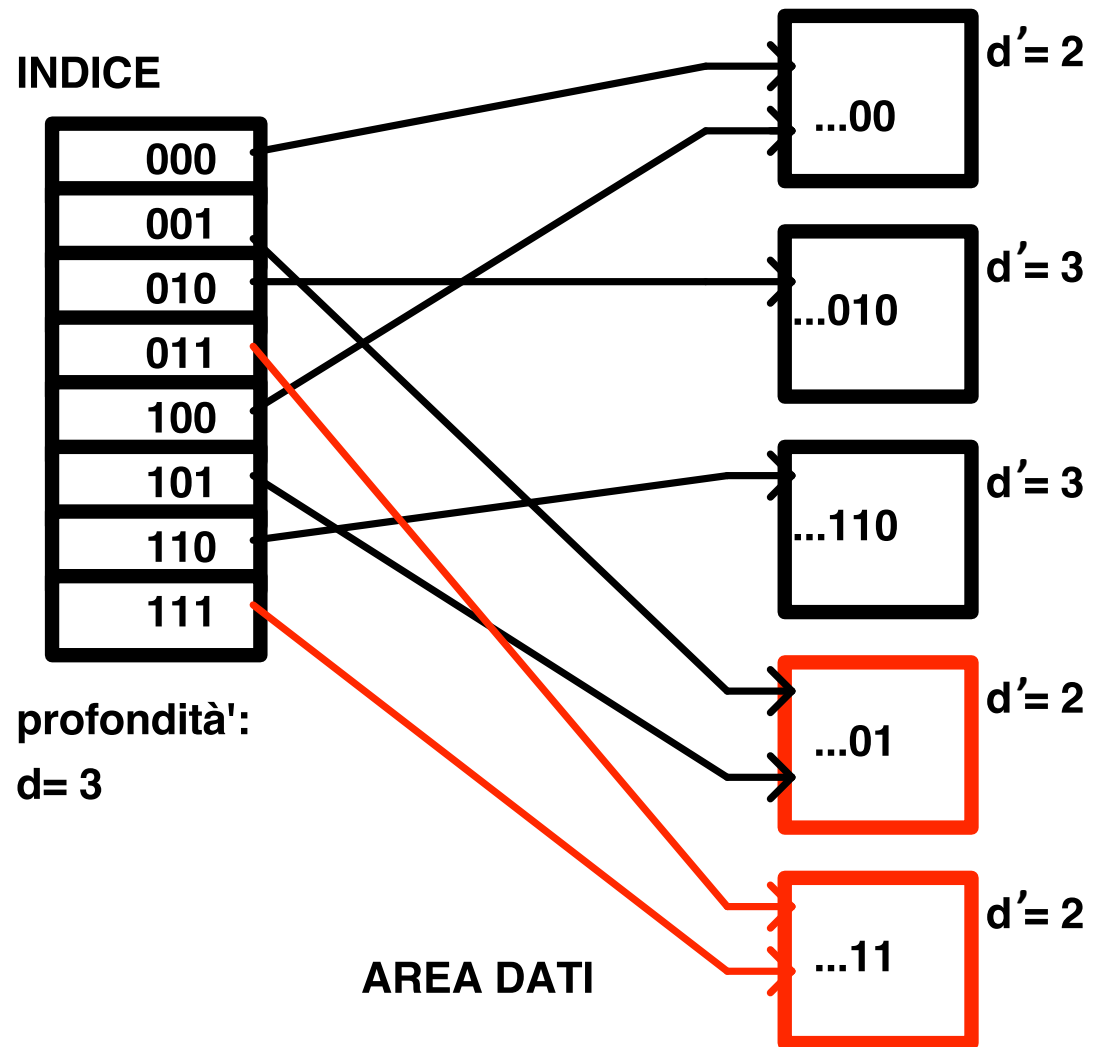
- ▶ La **ricerca** di un record di chiave **K** avviene in due passi:
 - 1 **K** viene trasformata dalla funzione Hash nella chiave hash **H**, si **estrae da H un suffisso S** (prefisso) costituito dai primi **d** bit a partire da destra (o sinistra), **con S si cerca nell'indice e si trova il valore corrispondente**;
 - 2 si estrae il **puntatore**, si accede al blocco indirizzato e si cerca il record con chiave **K**.

Extendible hashing

Inserimento di un record di chiave K:

- ▶ si cerca il blocco dove inserirlo come nel caso della ricerca, se il blocco è saturo si possono avere **due casi**:
 - 1 il blocco ha una **profondità locale $d' < d$** , allora si acquisisce un nuovo blocco, si aumenta **d'** di 1, si distribuiscono i vecchi ed il nuovo record tra il vecchio ed il nuovo blocco sulla base della differenziazione creata dal **nuovo bit del suffisso(0/1)** e si aggiornano i puntatori che puntavano al vecchio blocco.

Extendible hashing



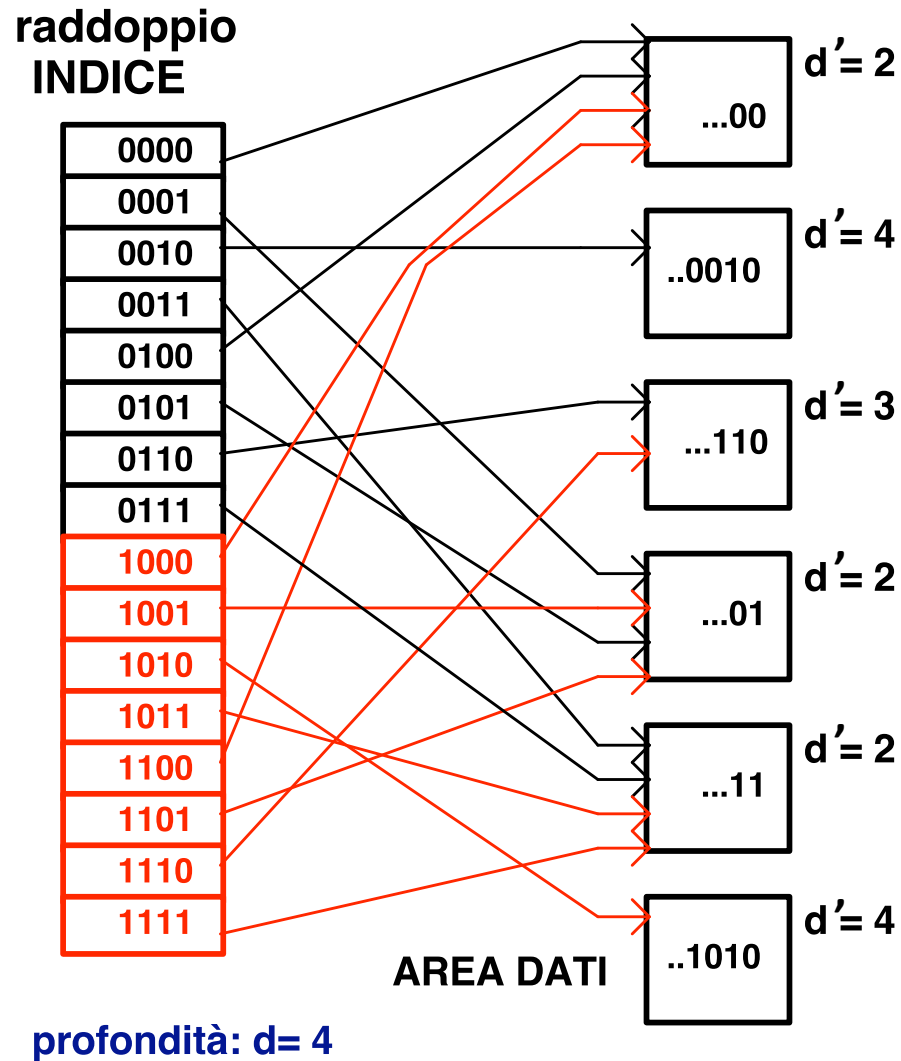
Extendible hashing

- 2 il blocco ha **profondità d** , allora si acquisisce un nuovo blocco, si aumenta di 1 la profondità **d** dell'indice, si ridistribuiscono i vecchi ed il nuovo record sulla base della differenziazione creata dal **nuovo bit del suffisso(0/1)** e

si ricostruisce l'indice raddoppiandolo (passa da 2^{d-1} a 2^d puntatori).

- ▶ In seguito a **cancellazioni**, in caso di svuotamento parziale o totale di blocchi, possono essere fusi due blocchi che si differenziano per l'ultimo bit del suffisso e l'indice può essere contratto.

Extendible hashing

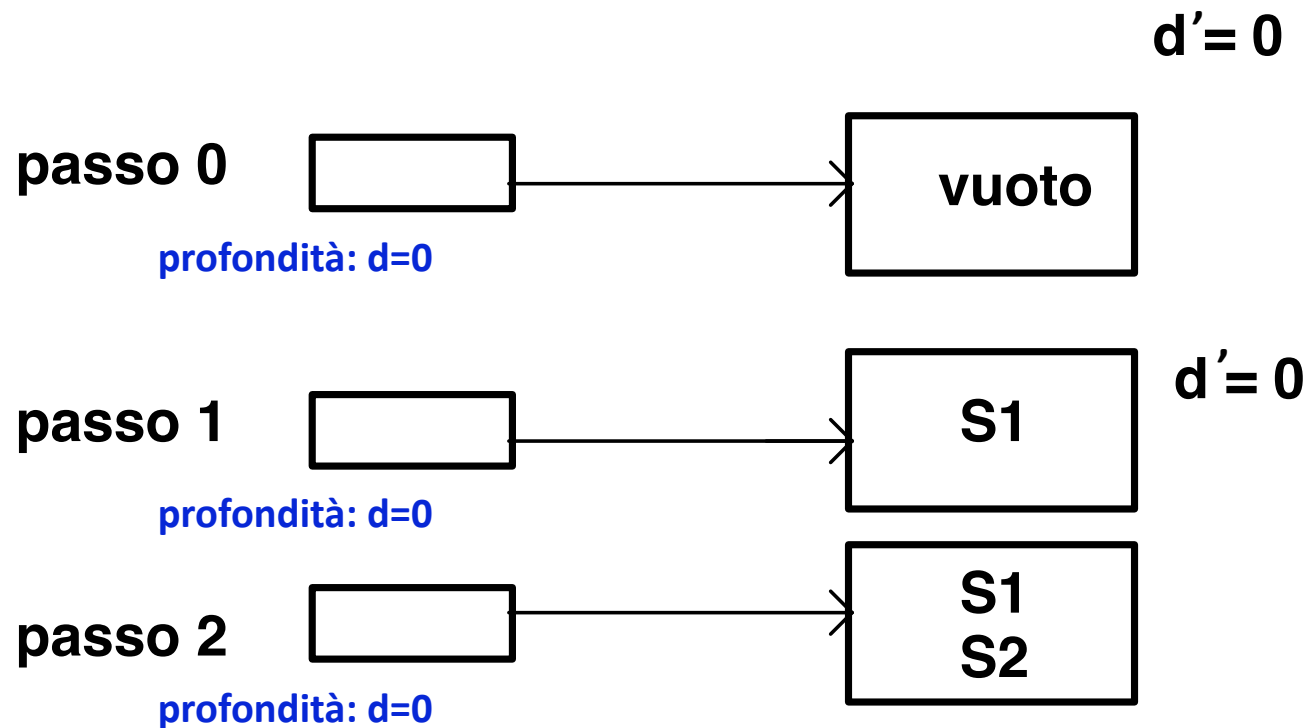


Extendible hashing: costruzione

capacità del blocco= 2

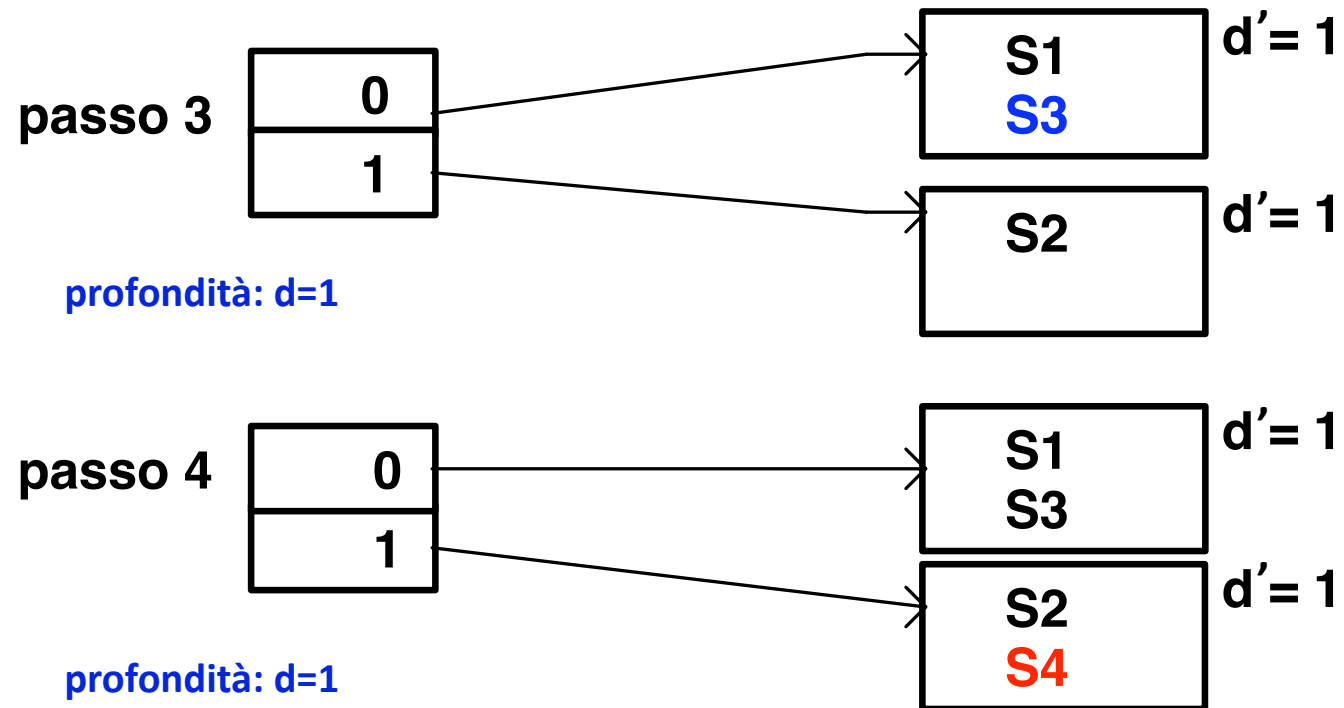
S1= 00011110

S2= 11010001



Extendible hashing: costruzione

S1= 00011110
S2= 11010001
S3= 00111100
S4= 11000011



Extendible hashing: costruzione

