

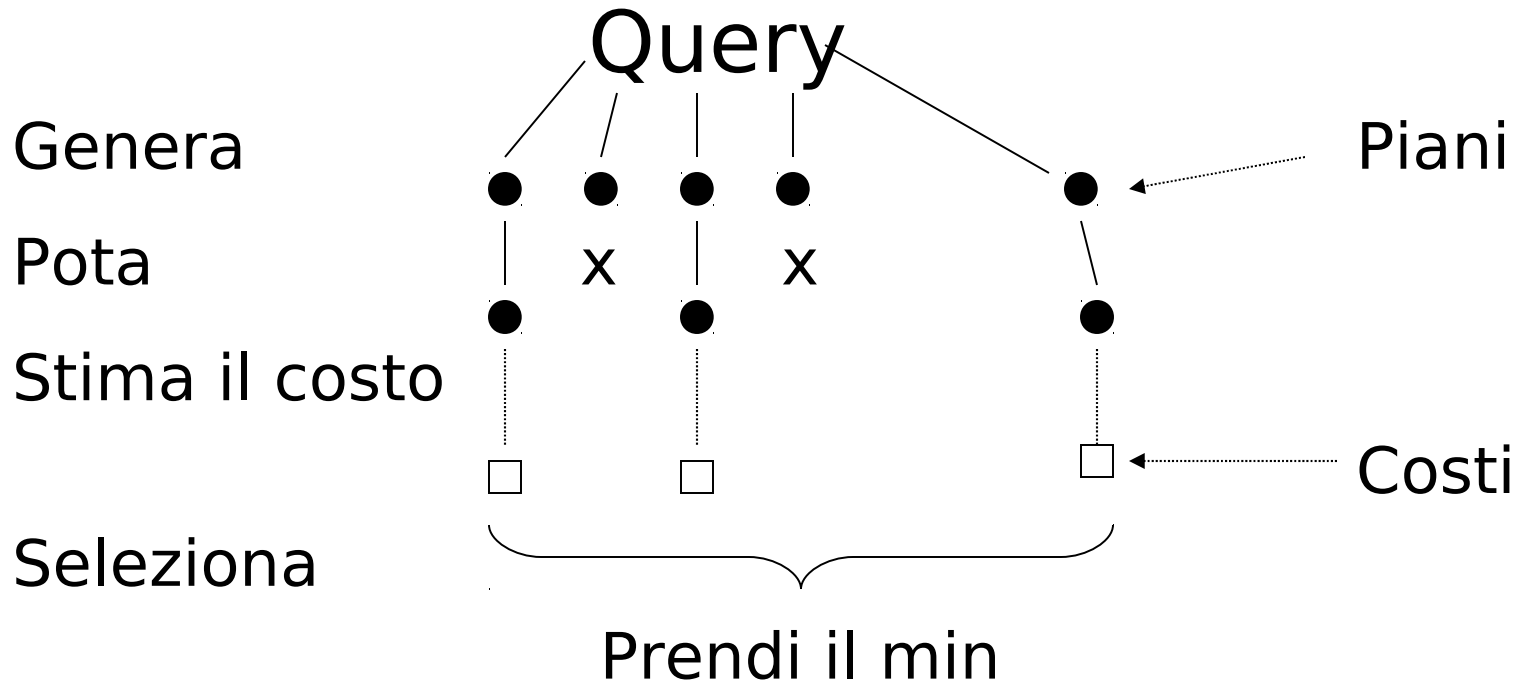
Ottimizzazione delle query

Leggere il capitolo 15 del Garcia-Molina et al.

Lucidi derivati da quelli di Hector Garcia-Molina

Ottimizzazione delle query

--> Generare e confrontare i piani



Per generare i piani si considera:

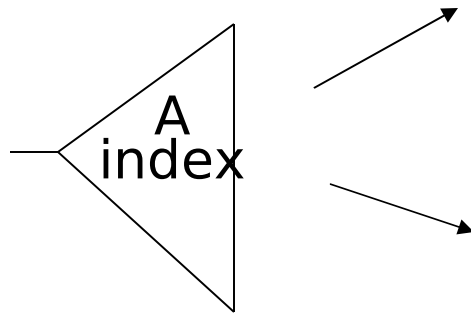
- Trasformazione dell'espressione relazionale
- L'uso di indici esistenti
- Algoritmi per il join

Stima dell'IO:

- Conta il # di blocchi di disco che devono essere letti (o scritti) per eseguire il piano della query
- Parametri:
 - $T(R)$ numero di tuple in R
 - $B(R)$ numero di blocchi occupati da R

Indice primario

Indice su un campo sul quale il file e'
ordinato



A

10	
15	
17	
19	
35	
37	

Operazioni fisiche

- Selezione
- Join
- Costo= numero di operazioni di I/O (ignorando la scrittura dell'output)

Selezione

$\forall \sigma_{A=a}(R):$

- Se non c'è indice su A, scansione di R, costo $B(R)$
- Se c'è un indice su A, accesso tramite indice, costo $3 + \lceil B(R)/V(R,A) \rceil$ se l'indice è primario altrimenti $3 + T(R)/V(R,A)$ (può essere inferiore perché alcune tuple possono fortuitamente capitare sullo stesso blocco)

Selezione

$\forall \sigma_{a < \text{chiave} < b} (R)$

- Se non c'è un indice su A, scansione di R, costo $B(R)$
- Altrimenti, sia $f = (b - a + 1) / (\max - \min + 1)$
 - Se l'indice è primario, costo $3 + \lceil f * B(R) \rceil$
 - Se l'indice è secondario, costo $3 + f * T(R)$

Esempi

- $B(R)=1.000$, $T(R)=20.000$

$\forall \sigma_{A=a}(R)$

- Casi:
 - Senza indice: costo=1000
 - Se $V(R,A)=100$ e l'indice e' primario, costo= $3+1.000/100=13$
 - Se $V(R,A)=100$ e l'indice e' secondario, costo= $3+20.000/100=203$

Esempi

- Se $V(R,A)=10$ e l'indice e' secondario, $\text{costo}=3+20.000/10=2003 \Rightarrow$ e' piu' alto della scansione!
- Se $V(R,A)=20.000$ cioe' A e' una chiave, allora $\text{costo}=3+1=4$

Join

- Vari tipi:
 - Join a un passo
 - Join a un passo e mezzo:
 - Join nested-loop basato sulle tuple
 - Join nested-loop basato sui blocchi
 - Join a due passi:
 - Sort-based join
 - Sort join
 - Index-based join
 - Hash-based join

Join a un passo

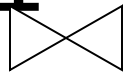
- Join di $R1(A,C)$ con $R2(C,B)$, $R2$ la piu' piccola
- $R2$ sta in memoria interamente

Join a un passo

- Leggi in memoria tutte le tuple di R2 e costruisci con esse una struttura in memoria centrale avente come chiave di ricerca C. Una tabella hash o un albero di ricerca vanno bene. Usa M-1 buffer per questo
- Leggi un blocco alla volta di R1 e mettilo nel rimanente buffer di memoria. Per ciascuna tupla di R1, trova le corrispondenti tuple di R2 che fanno join su C utilizzando la struttura di ricerca. Per ciascun tupla di R2 genera una tupla del risultato

Join a un passo: costo

- $\text{Costo} = B(R1) + B(R2)$
- Requisiti di memoria $B(R2) \leq M-1$
o approssimativamente $B(R2) \leq M$

Esempio R1  R2 sull'attributo

comune C

$$T(R1) = 10.000$$

$$T(R2) = 5.000$$

$$S(R1) = S(R2) = 1/10 \text{ di blocco}$$

Memoria disponibile $M = 501$ blocchi

$$B(R1) = 10.000 \times 1/10 = 1.000$$

$$B(R2) = 5.000 \times 1/10 = 500$$

$$\text{Costo} = 1000 + 500 = 1500$$

Join a un passo e mezzo


- Join nested loops o iterativo basato sulle tuple

for each $r \in R1$ do

for each $s \in R2$ do

if $r.C = s.C$ then output r,s

pair

Esempio R1  R2 sull'attributo
comune C

$$T(R1) = 10.000$$

$$T(R2) = 5.000$$

$$S(R1) = S(R2) = 1/10 \text{ di blocco}$$

Memoria disponibile $M = 101$ blocchi

Esempio Join iterativo $R1 \bowtie R2$

- Abbiamo $\left\{ \begin{array}{l} T(R1) = 10,000 \\ T(R2) = 5.000 \end{array} \right.$
 $S(R1) = S(R2) = 1/10$ blocchi
 $M = 101$ blocchi

Costo: per ogni tupla di $R1$:

[Leggi la tupla + Leggi $R2$]

Totale = $10.000[1 + 5.000] = 50.010.000$ IOs

Totale = $T(R1) * (1 + T(R2))$

• Possiamo far meglio?

Usa la memoria

- (1) Leggi 100 blocchi di R1
- (2) Leggi tutta R2 (usando 1 blocco)
+ join
- (3) Ripeti fino alla fine

$$B(R1)=1000$$

$$B(R2)=500$$

Costo: per ogni “pezzo” di R1:

leggi pezzo: 100 IOs

leggi R2 500 IOs

600

$$\text{Totale} = \frac{1000}{100} \times 600 = 6.000 \text{ IOs}$$

- Possiamo fare meglio?

• Inverti l'ordine: R2 \times R1

$$\text{Totale} = \frac{500}{100} \times (100 + 1.000) =$$

$$5 \times 1.100 = 5.500 \text{ IOs}$$

In generale

- Sia R2 la relazione piu' piccola
- Il numero di iterazioni nel loop piu' esterno e' $B(R2)/(M-1)$ dove M e' il numero dei buffer di memoria
- Ad ogni iterazione, leggiamo M-1 blocchi di R2 e B(R1) blocchi di R1
- Il numero di I/O e' dunque

$$(B(R2)/(M-1))(M-1+B(R1))= \\ B(R2)+B(R1)B(R2)/(M-1)$$

In generale

- Se M , $B(R1)$ e $B(R2)$ sono grandi ma M e' il piu' piccolo, una approssimazione della formula e' $B(R1)B(R2)/M$
- Condizioni sulla memoria: $M \geq 2$
- Join nested-loop basato sui blocchi (algoritmo a un passo e mezzo)

Sort-based join (algoritmo a due passi)

- (1) if R1 e R2 non sono ordinate, ordinale
- (2) Merge R1 e R2. Si usano due buffer, uno per il blocco corrente di R1 e l'altro per il blocco corrente di R2. Ripeti:
 - (a) trova il valore minimo c di C che e' davanti nei blocchi di R1 ed R2
 - (b) se c non appare davanti nell'altra relazione, elimina le tuple con valore c

- (c) altrimenti identifica tutte le tuple da entrambe le relazioni con valore c . Se necessario, leggi blocchi dalle relazioni ordinate $R1$ e $R2$ finché siamo sicuri che non ci siano più c in nessuna relazione. M buffer sono disponibili per questo
- (d) Genera tutte le tuple che possono essere formate dal join delle tuple di $R1$ e $R2$ con il valore comune c
- (e) Se una delle relazioni non ha più tuple non considerate in memoria principale, ricarica il buffer per quella relazione

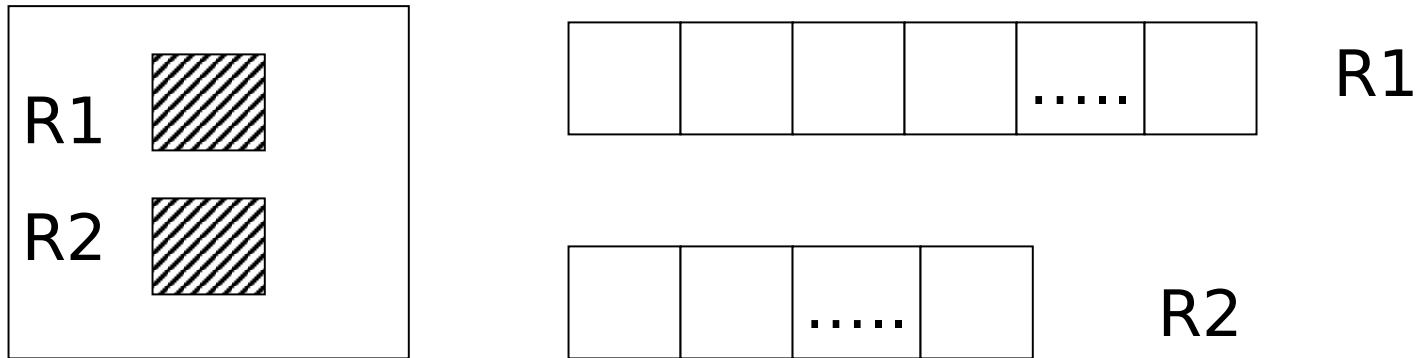
Esempio: 2 record per blocco

R1.C	R2.C
10	5
20	20
20	20
30	30
40	30
	50
	52

Esempio Sort-based Join

- Siano R1, R2 ordinate su C

Memoria



Totale: Leggi R1 + Leggi R2

$$= 1000 + 500 = 1,500 \text{ IOs}$$

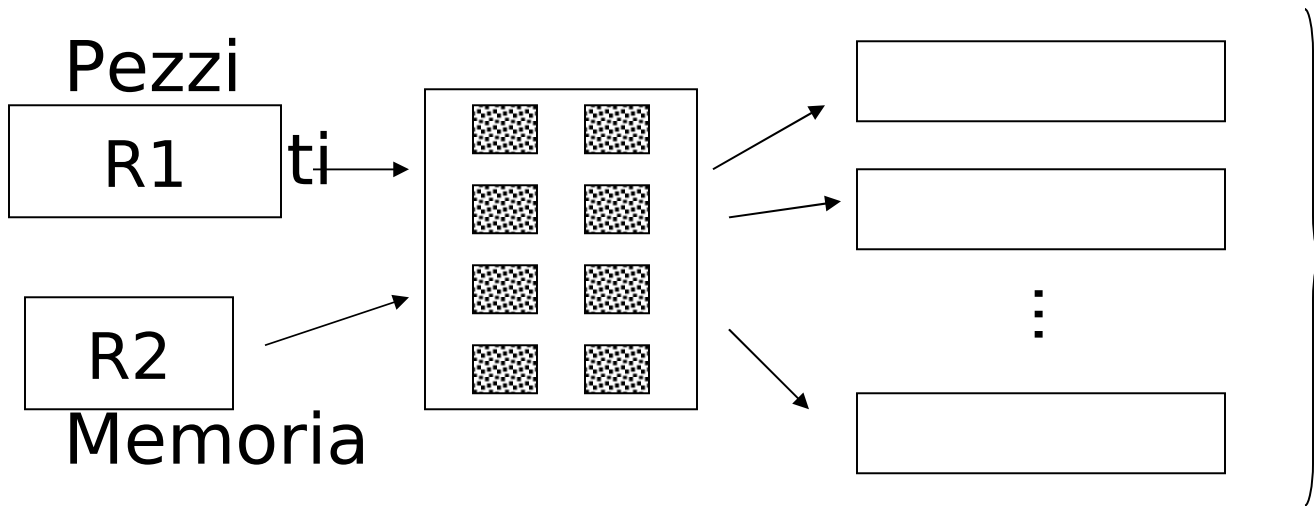
Esempio Sort-based Join

- R1, R2 non ordinate
- > dobbiamo prima ordinare R1, R2
- come?

Un modo per ordinare: Merge Sort

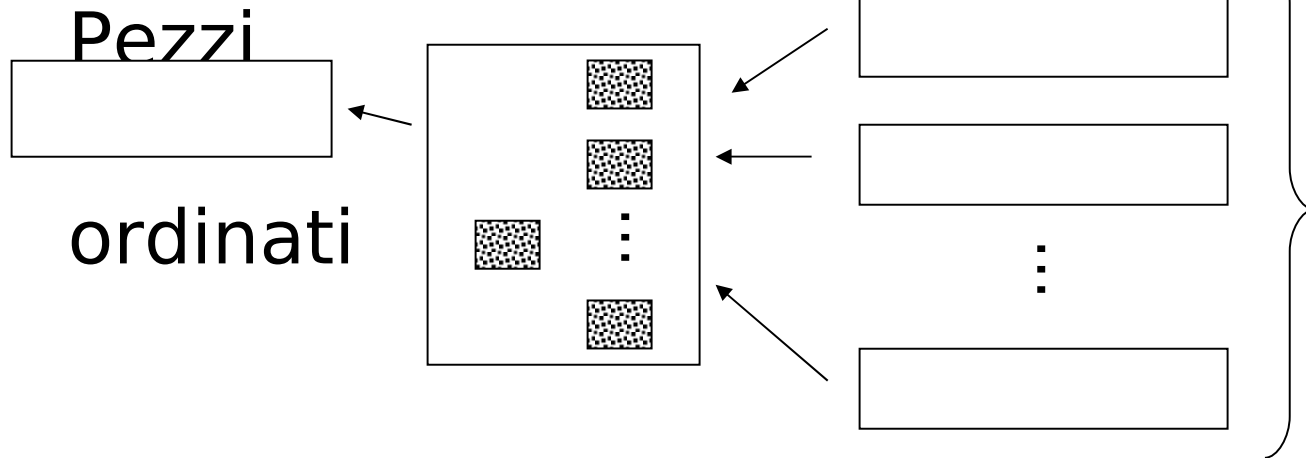
(i) Per ciascun pezzo di R da 100 bl.:

- leggi pezzo
- ordinalo in memoria
- scrivilo su disco



(ii) Leggi tutti i pezzi + fai il merge +
scrivi il risultato

File ordinato Memoria



Costo: Ordinamento

Ogni tupla e' letta,scritta,
letta, scritta

quindi...

Costo del sort di R1: $4 \times 1,000 = 4,000$

Costo del sort di R2: $4 \times 500 = 2,000$

Esempio Sort-based Join (continua)

R1,R2 non ordinate

Costo totale=costo del sort+costo
del join

$$= 6,000 + 1,500 = 7,500$$

Ma: ^{IQS} costo del join iterativo= 5,500
quindi il sort-based join non guadagna

In generale

- Costo del sort-based join:
 $5(B(R1)+B(R2))$

Ma se $R1 = 10,000$ blocchi
 $R2 = 5,000$ blocchi non ordinate

$$\frac{\text{Iterativo: } 5000}{100} \times (100 + 10,000) = 50 \times 10,100 \\ = 505,000 \text{ IOs}$$

$$\underline{\text{Sort based join: } 5(10,000 + 5,000) = 75,000 \text{ IOs}}$$

Sort based join vince! Il join iterativo e' infatti un algoritmo quadratico (dipende dal prodotto $B(R1)B(R2)$)

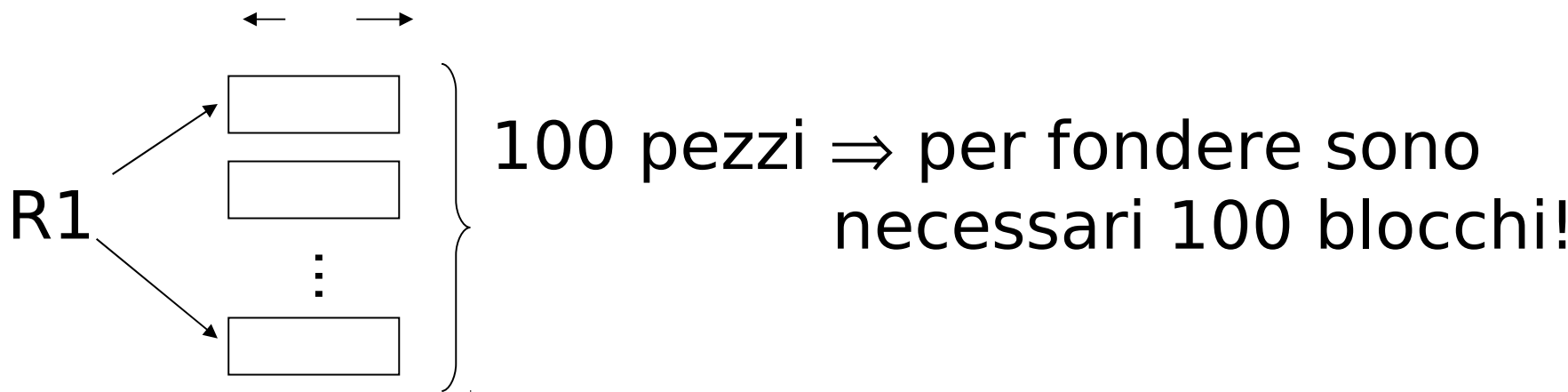
Casi particolari

- Caso in cui le tuple con valore c per C non stanno in memoria centrale:
 - Se le tuple di una relazione con valore c stanno in $M-1$ buffer, allora caricale nei buffer e utilizza il rimanente per caricare le tuple dell'altra relazione
 - Se per nessuna delle due relazioni le tuple con valore c stanno in $M-1$ buffer, utilizza il nested loop join basato sui blocchi per queste tuple

Quanta memoria e' necessaria per il merge sort?

Es.: Si supponga di avere 10 blocchi di memoria, $B(R1)=1.000$

10



In generale:

Siano M i blocchi di memoria
 x i blocchi della relazione

pezzi = (x/M)

dimensione del pezzo = M

pezzi \leq buffers disponibili per il merge

quindi... $(x/M) \leq M-1$

o $M(M-1) \geq x$ o circa $M^2 \geq x$ o $M \geq \sqrt{x}$

per entrambe le relazioni, cioè

$M \geq \sqrt{\max\{B(R1), B(R2)\}}$

Nel nostro esempio:

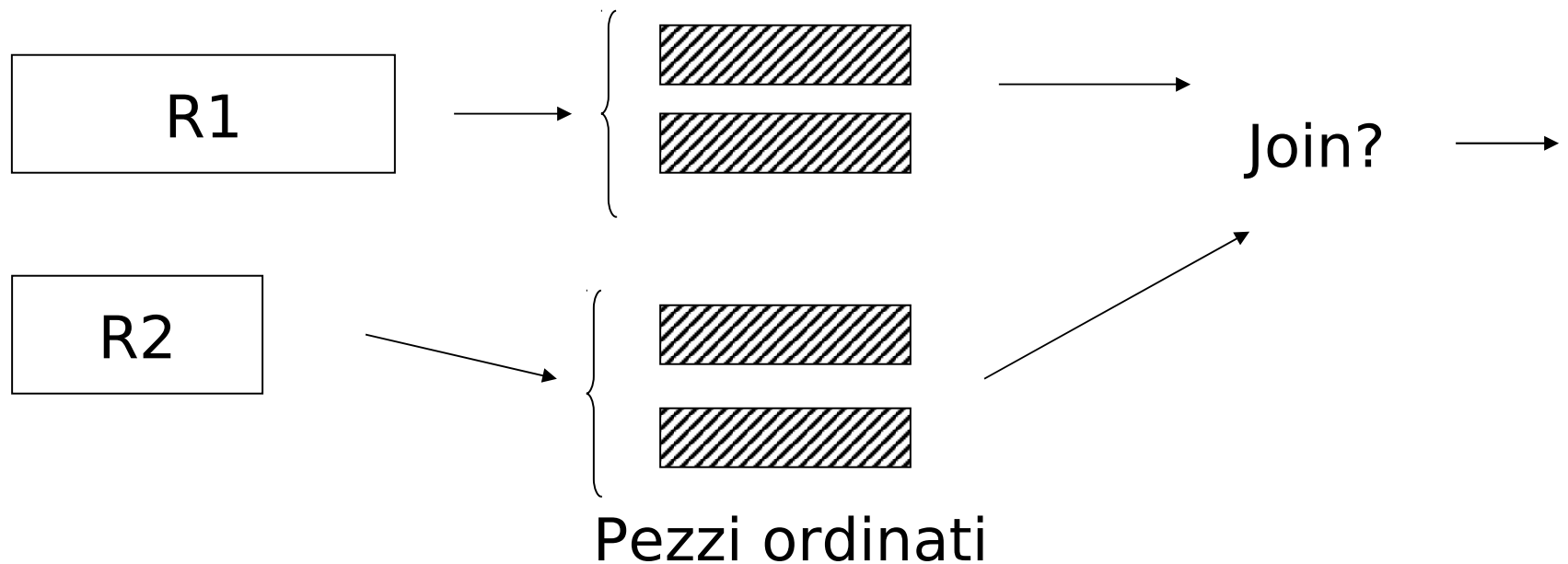
R1 e' 1000 blocchi, $M \geq 31.62$

R2 e' 500 blocchi, $M \geq 22.36$

Sono necessari almeno 32 buffers

Possiamo migliorare?

Suggerimento: abbiamo bisogno dei file ordinati?



Algoritmo sort join

(algoritmo a due passi)

1. Crea sottoliste ordinate su C di dimensione M per $R1$ ed $R2$
2. Porta il primo blocco di ciascuna sottolista in un buffer (si assume che ci siano meno di M sottoliste)
3. Trova il valore c di C minimo tra le sottoliste. Identifica tutte le tuple di entrambe le relazioni con $C=c$. Genera il join delle tuple di $R1$ con quelle di $R2$ che hanno lo stesso valore c . Se il buffer per una delle sottoliste si svuota, riempilo dal disco

Costo del sort join (sort-based join a due passi migliorato):

$$\begin{aligned} C &= \text{Leggi } R1 + \text{scrivi } R1 \text{ in pezzi} \\ &\quad + \text{leggi } R2 + \text{scrivi } R2 \text{ in pezzi} \\ &\quad + \text{merge dei pezzi} \\ &= 2000 + 1000 + 1500 = 4500 \end{aligned}$$

In generale: $\text{costo} = 3(B(R1) + B(R2))$

→ memoria necessaria?

$$B(R1)/M + B(R2)/M \leq M - 1$$

$$\rightarrow B(R1) + B(R2) \leq M^2$$

$$\rightarrow M \geq \sqrt{B(R1) + B(R2)}$$

- Join con indice (concettualmente)

For each $r \in R2$ do

Si assuma un indice su $R1.C$

[$X \leftarrow \text{index}(R1, C, r.C)$

for each $s \in X$ do

output r,s pair]

Nota: $X \leftarrow \text{index}(\text{rel}, \text{attr}, \text{value})$

$\Rightarrow X =$ insieme di tuple di rel con $\text{attr} = \text{value}$

Esempio Join con indice

- Si assuma che esista l'indice R1.C a 2 livelli
- Si assuma R2 non ordinata
- Si assuma che l'indice di R1.C stia nella memoria

Costo:

per ciascun tupla di R2:

- testa l'indice (gratis)
- leggi le tuple di R1

Costo dell'index join

- Costo per reperire le tuple di R2:
 $B(R2)$
- Costo per reperire le tuple di R1:
 - Se l'indice e' primario:
 - $T(R2) \lceil B(R1)/V(R1,C) \rceil$
 - Se l'indice e' secondario
 - $T(R2)T(R1)/V(R1,C)$

Costo totale con il join con indice

$$V(R1,C) = 5000$$

$$\begin{aligned} \text{(a) Indice primario} &= \\ &500 + 5.000 \times \lceil (1.000 / 5.000) \rceil = \\ &500 + 5000 \times 1 = 5.500 \end{aligned}$$

$$\begin{aligned} \text{(b) Indice secondario} &= \\ &500 + 5.000 \times 10.000 / 5.000 = 500 + 5.0 \\ &00 \times 2 = 10.500 \end{aligned}$$

Cosa succede se l'indice non sta in memoria?

Esempio: si supponga che l'indice su R1.C sia 201 blocchi

- Mantieni la radice + 99 nodi foglia in memoria
- Il costo atteso di ciascun test e'

$$E = \frac{(0)99}{200} + \frac{(1)101}{200} \approx 0.5$$

Costo totale (inclusi i test, indice secondario)

$$= 500 + 5000 \text{ [test+ ottieni records]}$$

$$= 500 + 5000 \text{ [0.5+2]}$$

$$= 500 + 12,500 = 13,000$$

Costi e requisiti

- Join con indice primario
 - Costo = $B(R2) + T(R2) \lceil B(R1) / V(R1, C) \rceil$
 - Blocchi necessari=2
- Join con indice secondario
 - Costo = $B(R2) + T(R2) T(R1) / V(R1, C)$
 - Blocchi necessari=2

Fin qui

R2 ⋈ R1

Nested-loop basato sui blocchi
5500

Sort-based Join (rel ord) 1500

Sort-based Join 7500

Sort-Join 4500

Join con indice primario 5500

Join con indice secondario
10500

Considerazioni

- Sembra che l'index join non convenga
- In realta', se $R2$ e' molto piccola rispetto a $R1$ e $V(R1,C)$ e' grande, la maggior parte di $R1$ non verra' mai esaminata dall'algoritmo mentre il sort join, l'hash join e il nested loop esaminano ogni tupla di $R1$ almeno una volta

- Hash join (algoritmo a due passi)
Funzione hash h , range $0 \rightarrow k$
 - Buckets per R1: G_0, G_1, \dots, G_k
 - Buckets per R2: H_0, H_1, \dots, H_k

Algoritmo

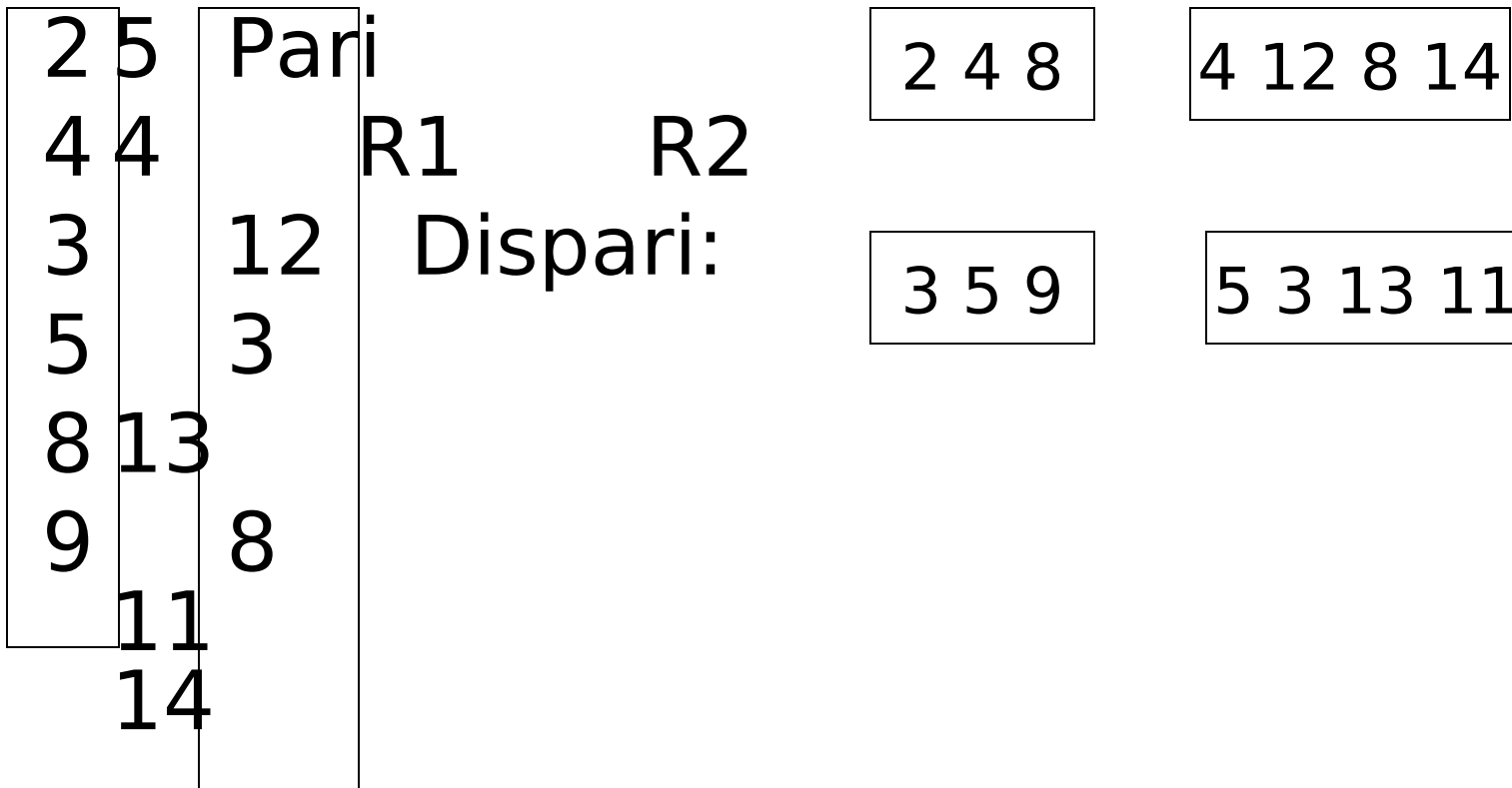
- (1) Manda le tuple di R1 nei bucket G
- (2) Manda le tuple di R2 nei bucket H
- (3) For $i = 0$ to k do
 - unisci le tuple nei bucket G_i, H_i

Semplice esempio

hash:

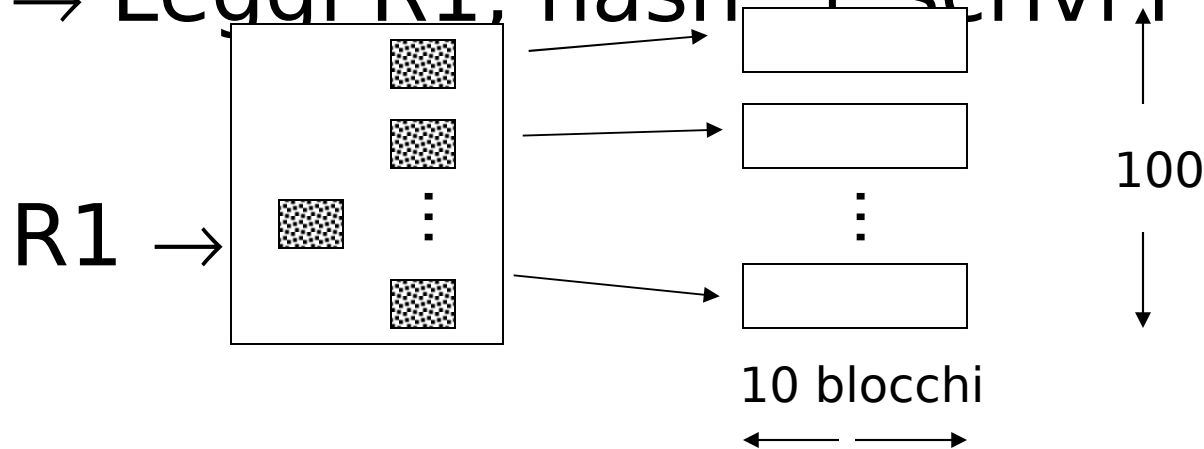
pari/dispari

R1 R2 Buckets

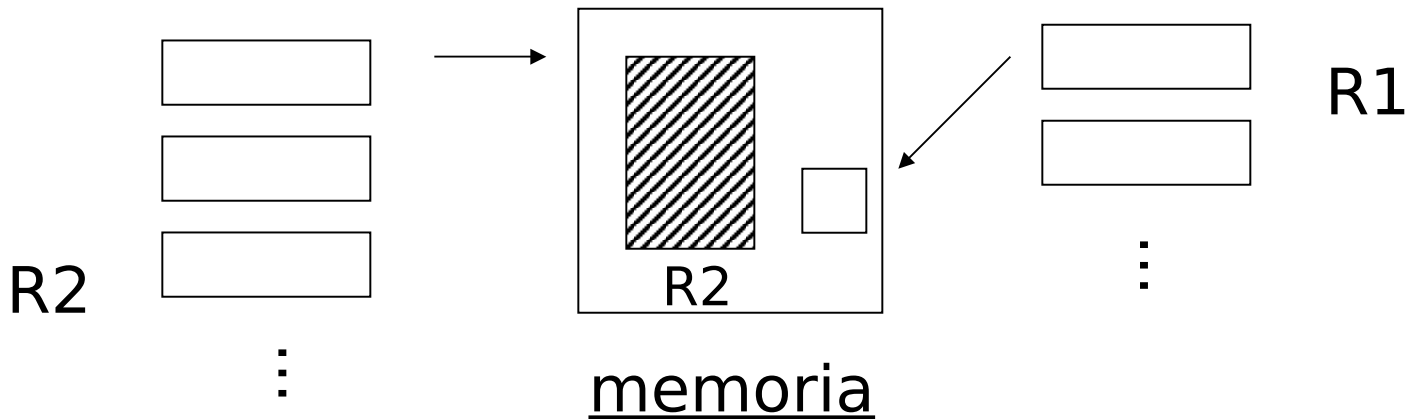


Esempio Hash Join

- R1, R2 non ordinate, $M=101$
- Usa 100 buckets, un blocco per bucket
- Leggi R1, hash + scrivi i buckets



- > Lo stesso per R2
- > Leggi un bucket di R2; costruisci una tabella hash in memoria
- > Leggi il corrispondente bucket di R1 un blocco alla volta + reperisci le tuple di R2 corrisp. con la funzione hash



~ Ripeti per tutti i buckets

Costo:

“Bucketizza:” Leggi R1 + scrivi

 Leggi R2 + scrivi

Join: Leggi R1, R2

Costo totale = $3 \times [1000 + 500] = 4500$

In generale: $\text{costo} = 3 \times (B(R1) + B(R2))$

Nota: questa e' un'approssimazione perche' i buckets variano in dimensione e dobbiamo arrotondare ai blocchi

Memoria necessaria:

Se R_2 è la più piccola,
dimensione del bucket di $R_2 = B(R_2) /$
 $(M-1)$

... $B(R_2) / (M-1) < M-1$
approssimando

$$M > \sqrt{B(R_2)}$$

Hash join ibrido

- Sia R_2 la relazione piu' piccola
- Si divide R_2 in k bucket
- Nel fare cio' si puo' decidere di tenere m dei k bucket interamente in memoria
 - 1 blocco per gli altri $k-m$ bucket
- E' necessario che M sia tale che
 - $mB(R_2)/k + k - m \leq M - 1$

Hash join ibrido

- Quando si leggono le tuple di R1 per suddividerle nei bucket, si tiene in memoria:
 - Gli m bucket di R2
 - Un blocco per ciascuno dei $k-m$ bucket di R1 i cui corrispondenti blocchi di R2 furono scritti su disco
- Se una tupla t di R1 viene mandata in uno dei primi m buckets, viene immediatamente fatto il join con le tuple di R2
- Se viene mandata in uno dei $k-m$ buckets, viene mandata nel corrispondente blocco di memoria per poi essere migrata su disco

Hash join ibrido

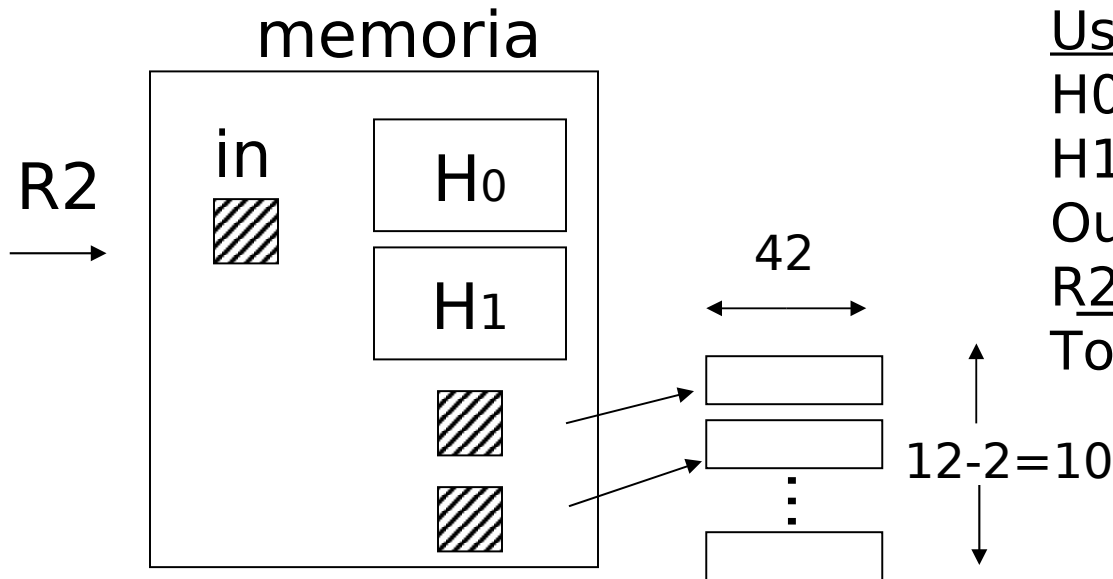
- Nel secondo passo, si fa il join tra i bucket corrispondenti di R1 ed R2 escluso i primi m

Esempio

Es. Numero di bucket $k=12$

Dimensione bucket di R2 = $\lceil 500/12 \rceil = 42$ blocchi

tenine 2 in memoria ($m=2$)



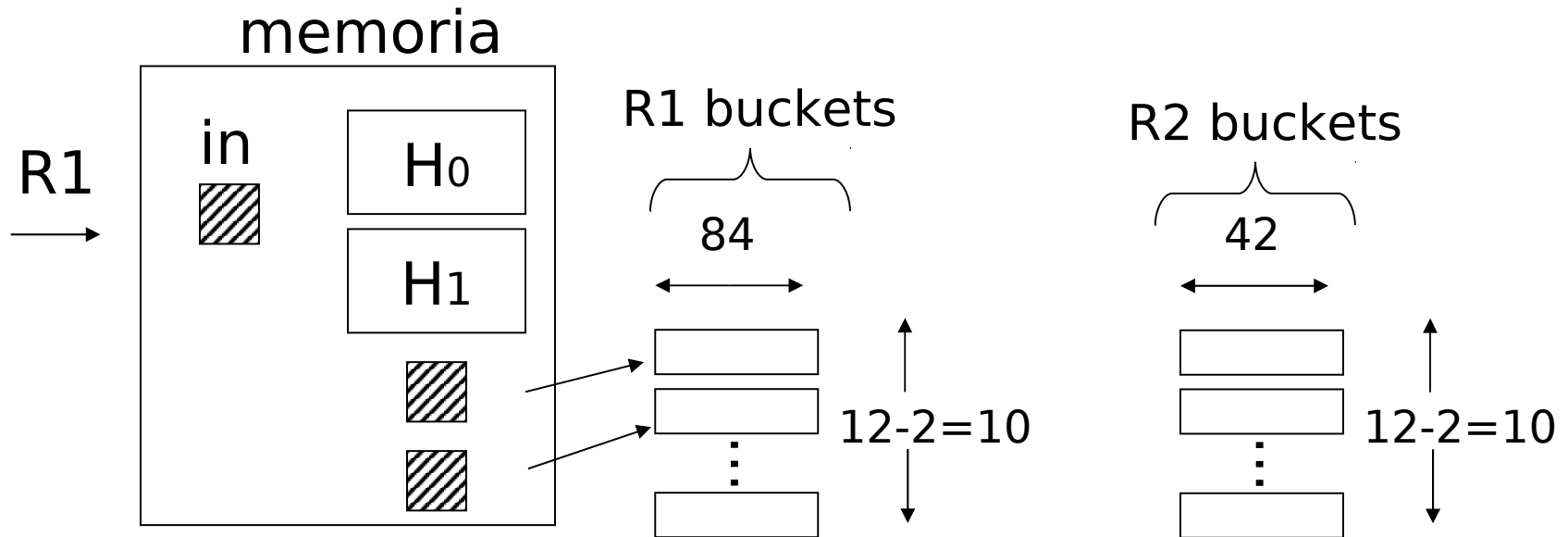
Uso della memoria:

H0	42 buffers
H1	42 buffers
Output	12-2 buffers
R2 input	1
Total	95 buffers

6 buffers inutil.!!

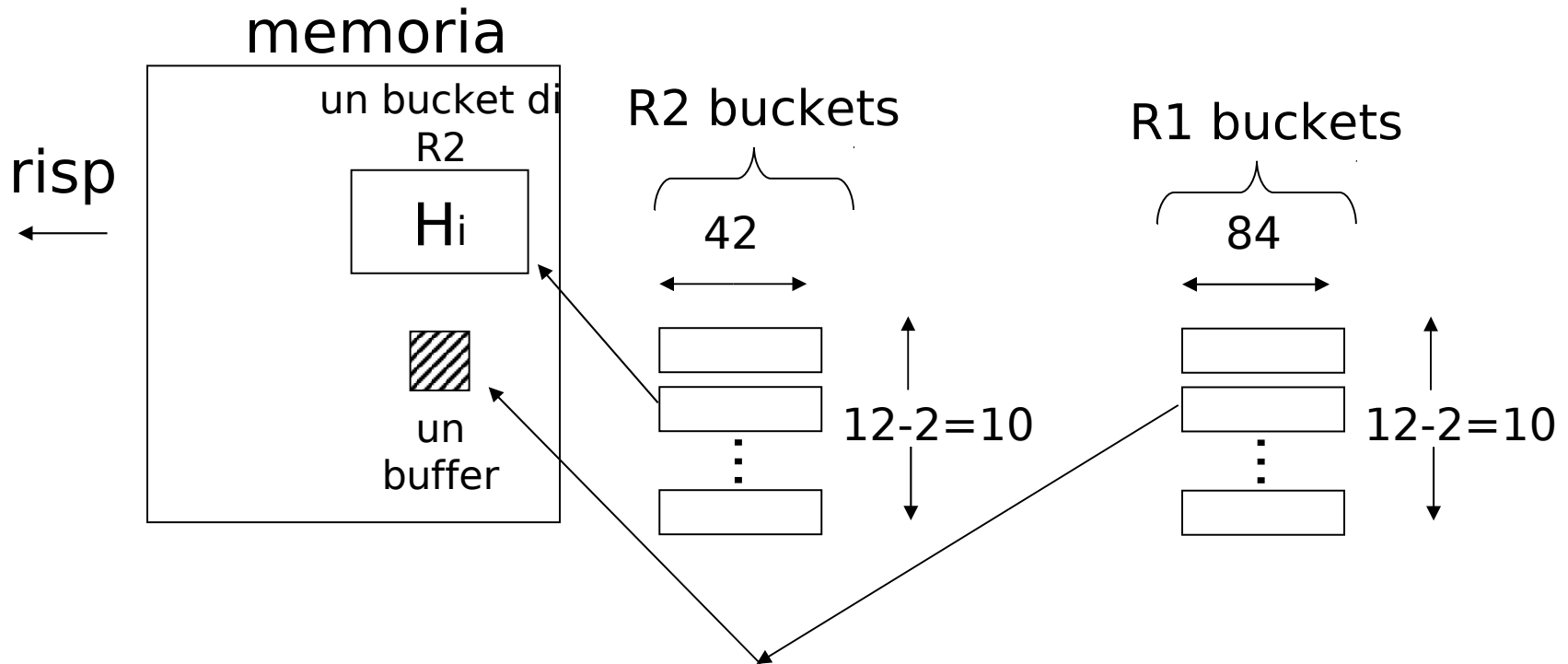
Poi: Bucketizza R1

- Buckets di R1 = $\lceil 1000/12 \rceil = 84$ blocchi
- Due dei buckets di R1 uniti immediatamente con H0, H1



Infine: unisci i bucket rimanenti

- Per ciascun coppia di bucket:
 - Leggi uno dei bucket in memoria
 - Unisci con il secondo bucket



Costo

- Bucketizza R2 = $500+10\times 42=920$
- Per bucketizzare R1, scrivi solo 10 buckets:
quindi, costo = $1000+10\times 84=1840$
- Per il join finale (2 buckets già fatti) leggi
 $10\times 42+10\times 84=1260$

Costo totale = $920+1840+1260 = 4060$

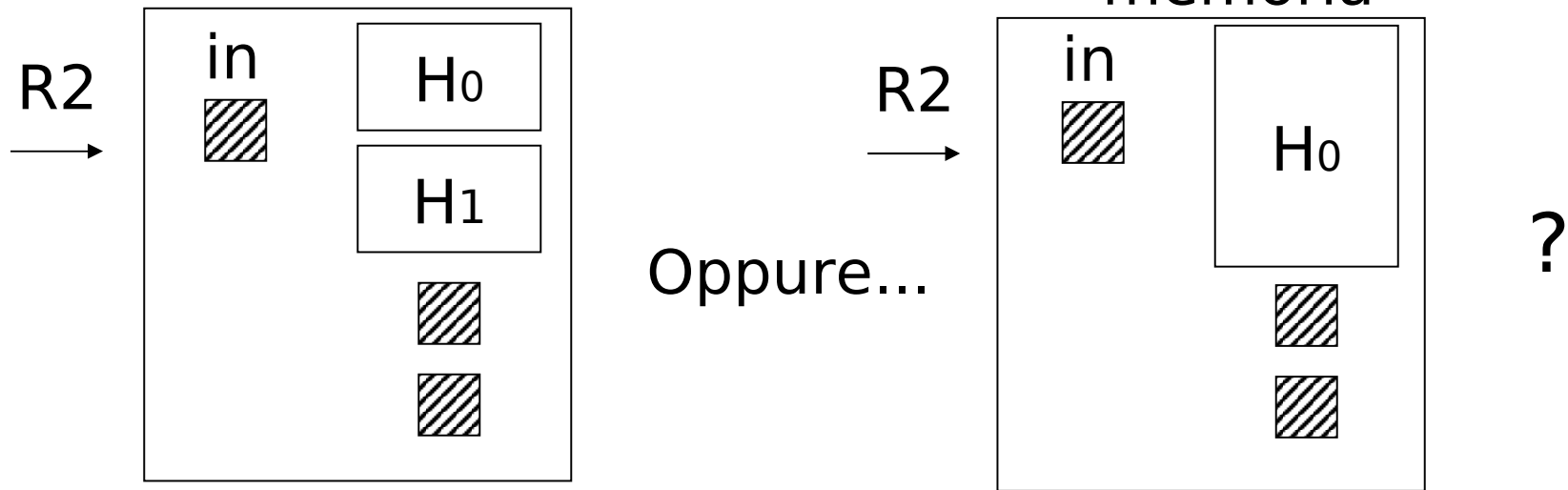
Costo dell'hash join ibrido

- Il risparmio in I/O e' due per ogni blocco dei bucket di R2 che rimangono in memoria e per i corrispondenti bucket di R1.
- Dato che m/k dei bucket sono in memoria, m/k dei blocchi rimangono in memoria e il risparmio e' $2m/k(B(R1)+B(R2))$
- Quindi il costo e'

$$(3-2m/k)(B(R1)+B(R2))$$

- Quanti buckets in memoria?

Bisogna massimizzare m/k sotto il vincolo
 $mB(R2)/k+k-m \leq M-1$
 memoria



Il valore migliore si ha per m e' 1 e per k il piu' piccolo possibile

Perche'

- Tutti gli M buffer possono essere usati per tenere le tuple di R_2 nella memoria tranne $k-m$.
- Vogliamo quindi minimizzare k
- Lo si fa' rendendo ciascun bucket grande quanto la memoria centrale
- Cioe' i buckets sono di dimensione M e $k \approx B(R_2)/M$
- In questo caso, c'e' posto solo per un bucket nella memoria, cioe' $m=1$

Costo per il caso $m=1$

- In questo caso, assumendo per semplicità che k sia circa $B(R2)/M$ si ha

$$(3-2M/B(R2))(B(R1)+B(R2))$$

Esempio

- $k=500/101=5$ dimensione bucket R2
 $500/5=100$ blocchi, non riusciamo a farli stare in memoria insieme ad altri 4 blocchi
- Scegliamo $k=6$
- Hashing di R2: 5 buffer per 5 buckets + $\lceil 500/6 \rceil=84$ buffer per il sesto bucket (tot 89 buffer)
- Bucket di R1 = $\lceil 1000/6 \rceil=167$ blocchi
- Bucketizza R2 = $500+500-84=500+416$
- Bucketizza R1 = $1000+1000-167=1000+833$
- Secondo passo: $416+833=1249$
- Totale = $916+1833+1249=3998$

Esempio

- Utilizzando la formula:

$$(3-2 \times 101/500)(1000+500)=$$

$$(3-0,404) \times 1500 =$$

$$2,596 \times 1500 = 3894$$

Se avessimo usato

$$(3-2m/k)(B(R1)+B(R2))=$$

$$(3-2 \times 1/6)1500 = 2,666 \times 1500 = 4000$$

Un altro trucco dell'hash join:

- Scrivi nei bucket solo coppie $\langle \text{val}, \text{ptr} \rangle$
- Quando c'è una corrispondenza nella fase di join bisogna reperire le tuple

- Per illustrare il calcolo del costo, si assuma:
 - _ 100 coppie <val,ptr> /blocco
 - _ Numero atteso di tuple nel risultato e' 100
- Costruisci una hash table per R2 in memoria
 - 5000 tuple $\rightarrow 5000/100 = 50$ blocchi
- Leggi R1 e unisci
- Leggi ~ 100 tuple di R2

$$\begin{array}{r}
 \underline{\text{Costo totale}} = \text{Leggi R2:} \quad 500 \\
 \text{Leggi R1:} \quad 1000 \\
 \text{Ottieni le tuple:} \quad \underline{\quad 100} \\
 \hline
 1600
 \end{array}$$

Fin qui:

Nested-loop basato sui blocchi	5500
Sort based join (rel ord)	1500
Sort based join	7500
Sort join	4500
R1.C index	5500
Hash join	
normale	4500
ibrido, R2 prima, m=2	4060
ibrido, R2 prima, m=1	3998
Hash join, puntatori	1600

Riassunto Join

Sia R_2 la relazione piu' piccola, M la dim della memoria

Algoritmo	# minimo buffer richiesti	# di I/O
Join a un passo	$B(R_2)$	$B(R_1)+B(R_2)$
Nested-loop join basato sulle tuple	1	$T(R_1)T(R_2)$
Nested-loop join basato sui blocchi	2	$B(R_2)+B(R_1)B(R_2)/M$
Sort-based join	$\sqrt{B(R_1)}$	$5(B(R_1)+B(R_2))$
Sort join	$\sqrt{(B(R_1)+B(R_2))}$	$3(B(R_1)+B(R_2))$

Riassunto join

Algoritmo	# minimo buffer richiesti	# di I/O
Join con indice primario *	2	$B(R2) + T(R2) \lceil B(R1)/V(R1,C) \rceil$
Join con indice secondario *	2	$B(R2) + T(R2)T(R1)/V(R1,C)$
Hash join	$\sqrt{B(R2)}$	$3(B(R1) + B(R2))$
Hash join ibrido	$\sqrt{B(R2)}$	$(3 - 2M/B(R2))$ $(B(R1) + B(R2))$

* Indice interamente in memoria

Riassunto

- Iterazione ok per “piccole” relazioni (rispetto alla dimensione della memoria)
- Per l’equi-join, quando le relazioni non sono ordinate e non esiste un indice, l’hash join e’ normalmente il migliore

- Sort(-based) join buono per non-equi-join (ad es., $R1.C > R2.C$)
- Se le relazioni sono ordinate, usa il sort-based join
- Se esiste un indice, puo' essere utile (dipende dalla stima della dimensione del risultato)