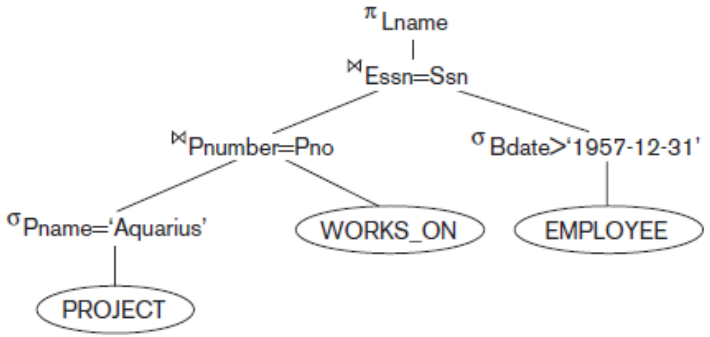
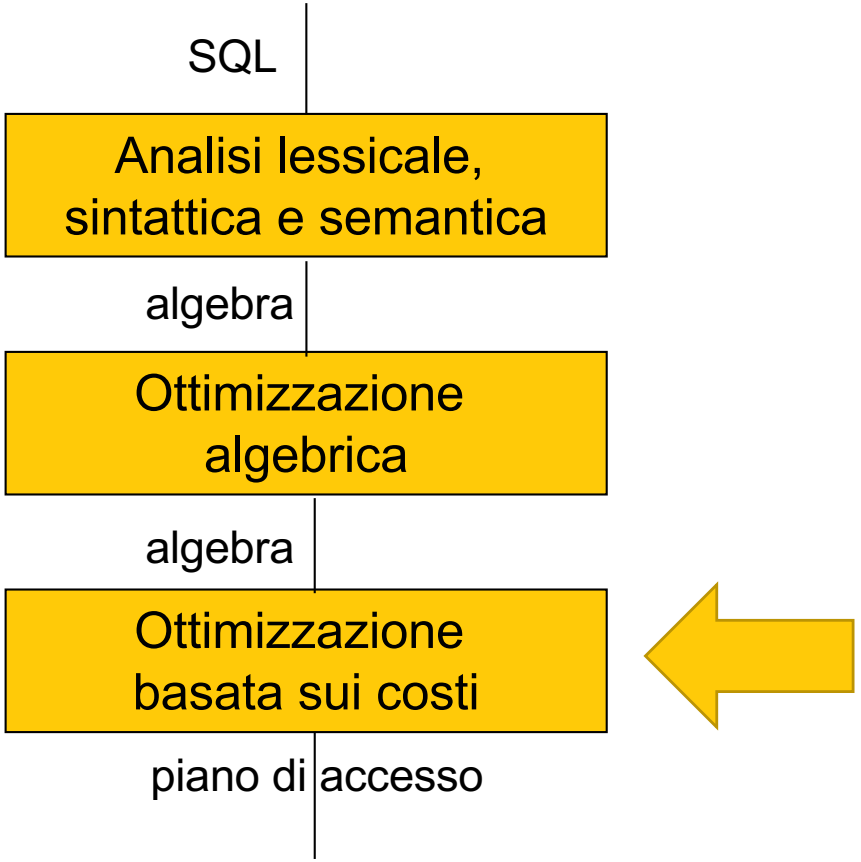
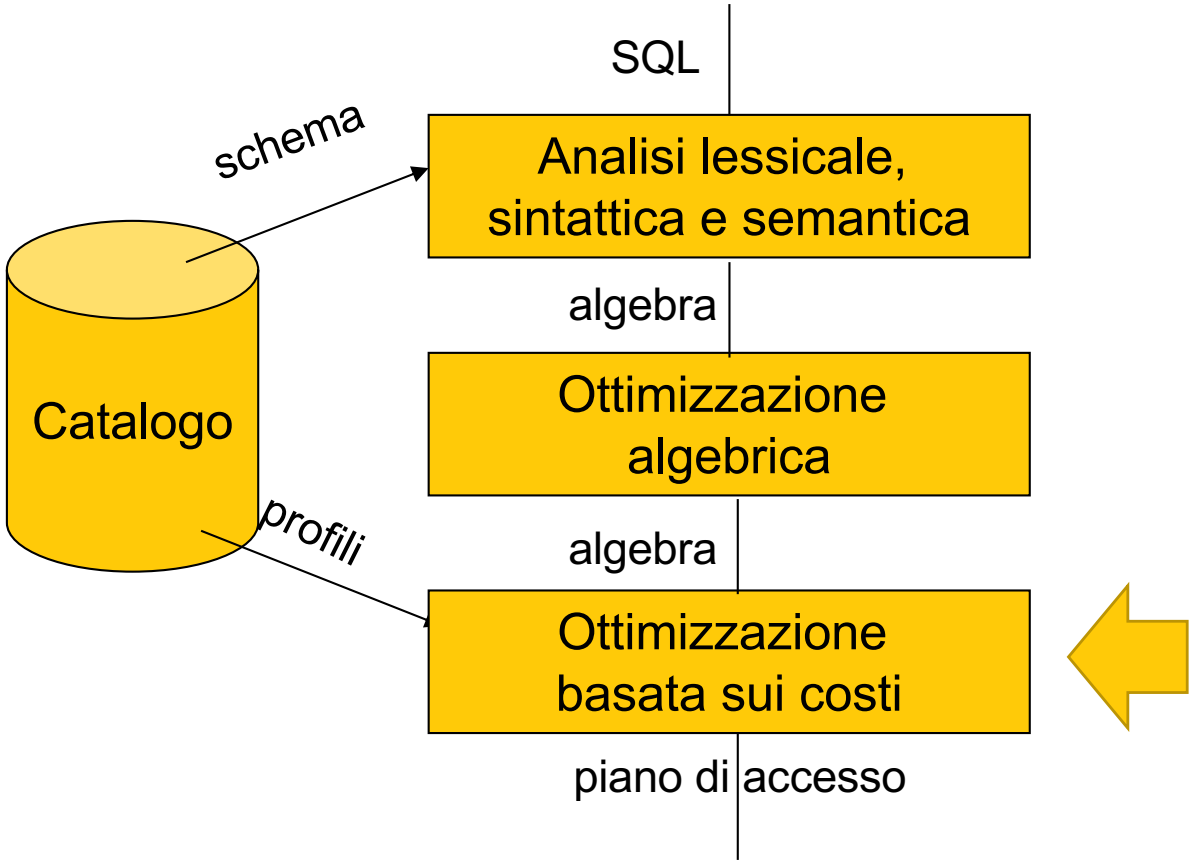
Esecuzione e ottimizzazione delle interrogazioni



- Il DBMS offre più strategie per eseguire gli operatori algebrici:
  - Algoritmi per la selezione
  - Algoritmi per il join
  - Algoritmi per la proiezione
- Molte strategie assumono i dati ordinati:
  - Algoritmo d'ordinamento

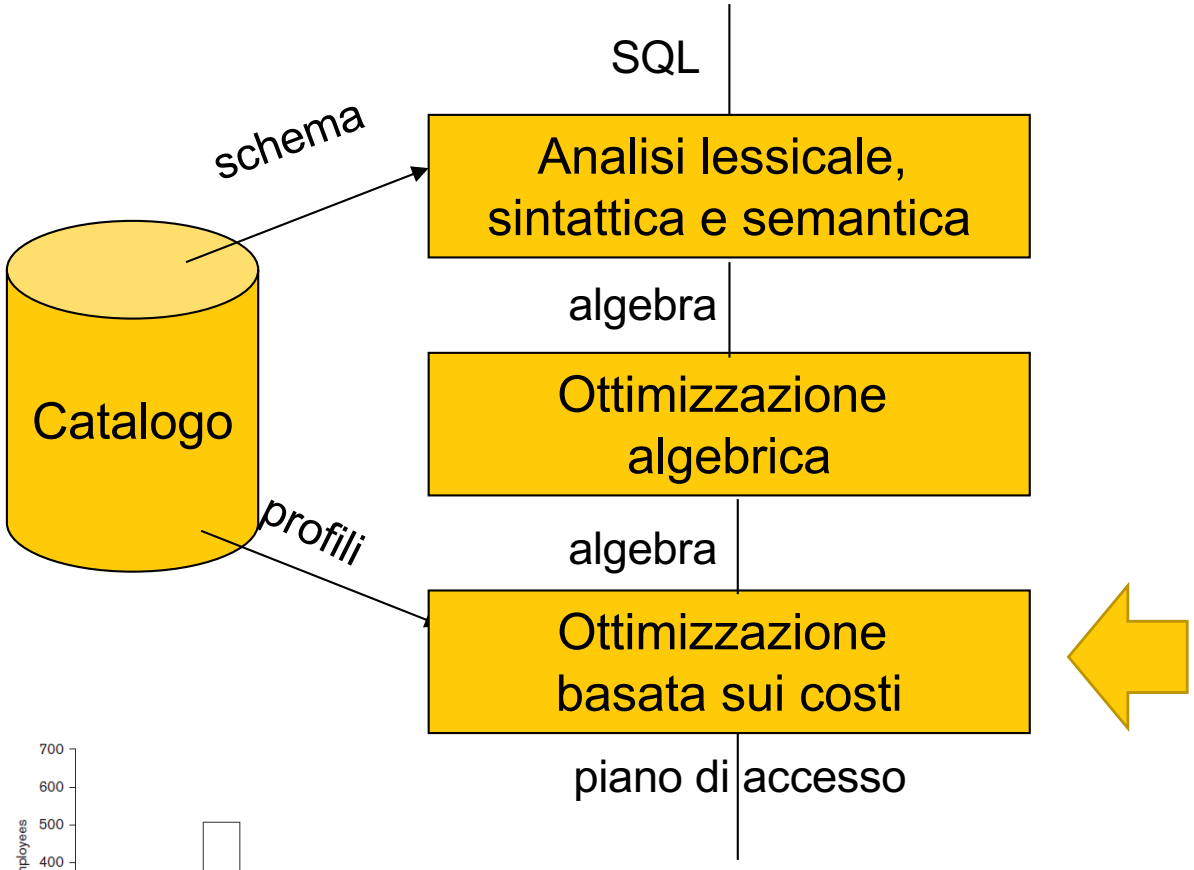


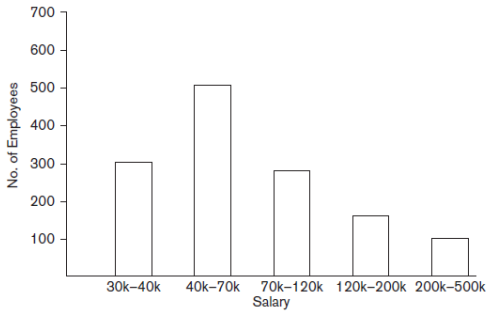
Ottimizzazione basata sui costi



"Profili" delle relazioni

- Informazioni quantitative:
  - cardinalità di ciascuna relazione
  - dimensioni delle tuple
  - dimensioni dei valori
  - numero di valori distinti degli attributi
  - valore minimo e massimo di ciascun attributo
- Sono memorizzate nel "catalogo" e aggiornate con comandi del tipo `update statistics`
- Sono memorizzati anche istogrammi sulla distribuzione dei valori degli attribute più important
- Utilizzate nella fase finale dell'ottimizzazione, per stimare le dimensioni dei risultati intermedi





Algoritmi per l'operatore SELEZIONE

- Selezione
  - Ricerca i record nel file che soddisfano una certa condizione
  - Ricerca dei record
    - Scansione di tutto il file
    - Ricerca basata sull'indice

Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione semplice: $\sigma_{A\ OP\ v}$ OP è =, >,< >=, <=

- S1: Ricerca lineare (brute force)
- S2: Ricerca Binaria
  - $\sigma_{A=v}$
  - file ordinato fisicamente su attributo A
- S3: Uso di indice primario
  - $\sigma_{A=v}$
  - File ordinato fisicamente su attributo A, con indice primario su A
  - NB. restituisce un solo valore
- S4 Uso di indice di clustering (simile al precedente)

- S5: Uso di indice primario per recupero di record multipli
  - $\sigma_{A\ OP\ v}$ OP è >,< >=, <=
  - file ordinato fisicamente su attributo A, con indice primario su A
  - Viene utilizzato l'indice per trovare il record che soddisfa la condizione di uguaglianza e si recuperano i successivi record nel file (ordinato)
- S6: Uso di un indice secondario
  - $\sigma_{A\ OP\ v}$ OP è >,< >=, <=
  - File non ordinato sull'attributo A
  - A è campo di indicizzazione per un indice secondario (e.g., B+ tree)

Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione complesse in congiunzione:

es. $\sigma_{(A\ OP\ v)\ AND\ (A'\ OP\ v')}$

- S7 *Uso di un indice individuale*:
  - uno degli attributi (A o A') ha un indice,
  - si utilizza l'indice per recuperare i record che soddisfanno la condizione semplice
  - si controlla che ogni record recuperato soddisfi le rimanenti condizioni semplici
- S8 *Intersezione di puntatori a record*:
  - Attributi hanno indici che puntano ai record (non solo i blocchi)
  - Si usano gli indici per recuperare l'insieme dei puntatori a record che soddisfano la cond. semplice
  - Si calcola l'intersezione dei puntatori e si accede solo ai record che soddisfano entrambe le condizioni
  - Se solo alcuni degli attributi delle condizioni posseggono indici ai puntatori, si esegue S7
- S9. *Uso di un indice composto*:
  - Esiste un indice composto definito su più attributi della condizione congiuntiva
    - Es. chiave primaria composta da più attributi LAVORA_SU(SSN_I, N_P)
  - Si utilizza l'indice per trovare i valori che soddisfano condizioni semplici (=, >,< >=, <=)

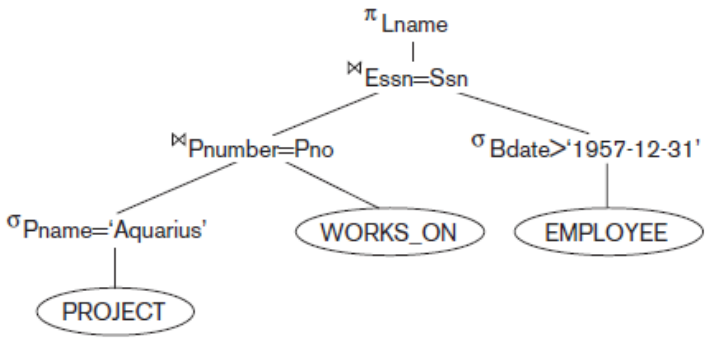
Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione complesse in disgiunzione:

es. $\sigma_{(A\ OP\ v)\ OR\ (A'\ OP\ v')}$

- una **condizione disgiuntiva** è molto più difficile da elaborare
  - i record che soddisfano la condizione disgiuntiva sono l'*unione* dei record che soddisfano le singole condizioni;
- se *una qualsiasi* delle condizioni non ha un percorso d'accesso (indice), si è obbligati a utilizzare l'approccio lineare di forza bruta

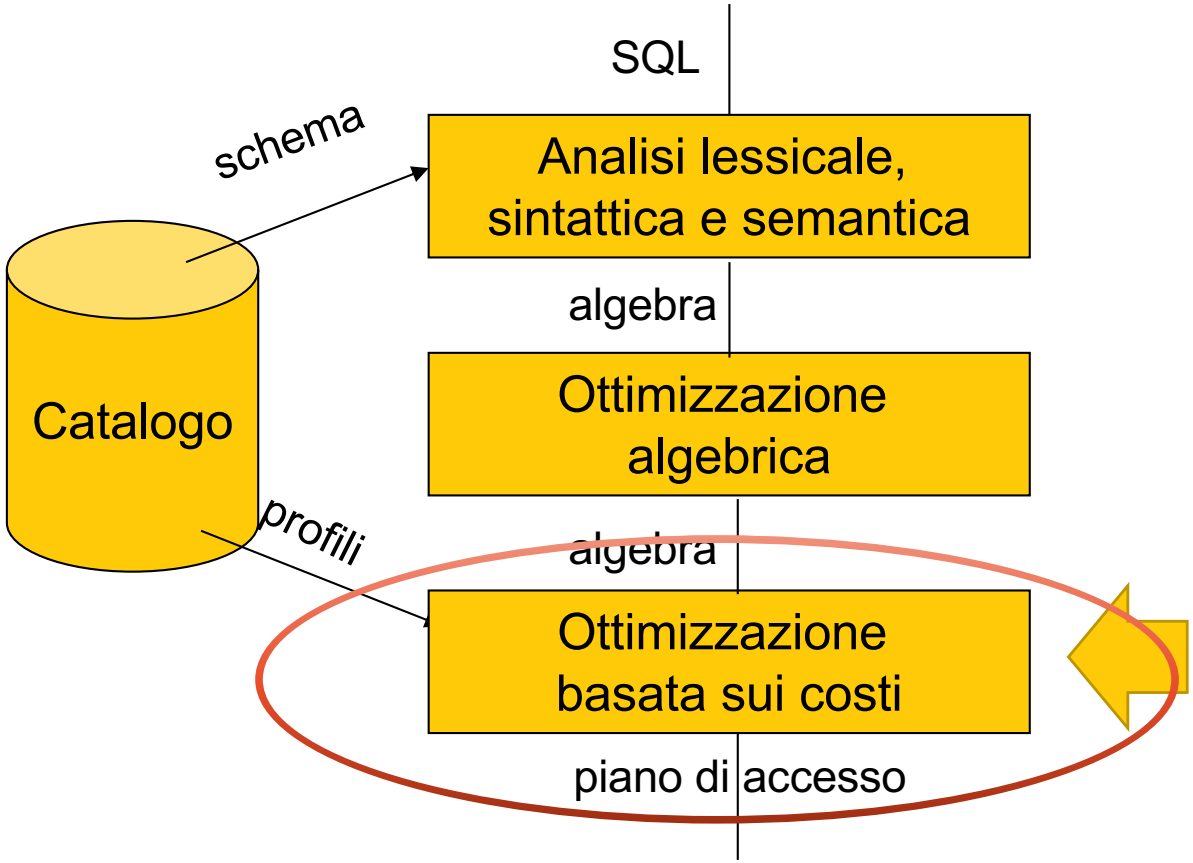
Ottimizzazione basata sui costi



Costi I/O delle

operazioni di SELEZIONE

Stimiamo solo costo di lettura



COSTO I/O: parametri per la stima

- B(R)= numero blocchi di R
- T(R)= numero tuple di R
- B_{fr}= fattore di blocco di R
- sl_A : **selettività** di una condizione su A (sl_A) è definita come il rapporto fra il numero di tuple che soddisfano la condizione e il numero totale di tuple nella relazione
- $Card_\sigma = sl_A * T(R)$ - numero di tuple che soddisfano la condizione

Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione semplice: $\sigma_{A = v}$

- S1: Ricerca lineare (brute force)
  - i) se A è chiave
    - Costo S1 = B(R)/2
  - Ii) se A non è chiave
    - Costo S1 = B(R)
- S2: Ricerca Binaria (file ordinato fisicamente su attributo A)
  - i) se A è chiave
    - Costo S2 = $\log_2 B(R)$
  - Ii) se A non è chiave:
    - E' necessario leggere più tuple, quante?
      - $Card_\sigma = sl_A * T(R)$
    - Quanti blocchi devo leggere per queste tuple?
      - $\lceil Card_\sigma / Bfr \rceil$
    - Costo S2 = $\log_2 B(R) + \lceil Card_\sigma / B_{fr} \rceil - 1$

Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione semplice: $\sigma_{A = v}$

- S3: Uso di indice primario (file ordinato fisicamente su attributo A, con indice primario su A)
  - Costo S3= costo indice primario + 1
    = $\log_2$ (#blocchi indice) + 1
- S4 Uso di indice di clustering (simile al precedente, ma A non chiave)
  - Costo S4= costo indice primario + costo lettura tuple A = v
    = $\log_2$ (#blocchi indice) + $\lceil Card_\sigma / Bfr \rceil$
- S5: si veda dopo
- S6: Uso di un indice secondario: B⁺-tree
  - i) Se A è chiave
    - Costo S6= costo indice + 1
      = #livelli B⁺-tree + 1
  - ii) Se A non è chiave
    - Costo S6= #livelli B⁺-tree + 1 + $\lceil Card_\sigma / Bfr \rceil$

In caso di B⁺-tree costruiti su attributi non chiave è necessario aggiungere un livello:
il puntatore al B⁺tree punta ad blocco contente tutti i puntatori alle varie tuple

Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione semplice: $\sigma_{A\ OP\ v}$ OP è >,< >=, <=

- S1: Ricerca lineare (brute force)
  - Costo S1 = B(R)
- S2: Ricerca Binaria (file ordinato fisicamente su attributo A)
  - Costo S2 = $\log_2 B(R) + \lceil \mathrm{Card}_\sigma / B_{fr} \rceil - 1$
- S3: Uso di indice primario (file ordinato fisicamente su attributo A, con indice primario su A)
  - Costo S3= costo indice primario + $\lceil Card_\sigma / Bfr \rceil$
    = $\log_2$ (#blocchi indice) + $\lceil Card_\sigma / Bfr \rceil$
- S4 Uso di indice di clustering
  - Costo S4 = $\log_2$ (#blocchi indice) + $\lceil Card_\sigma / Bfr \rceil$
- S5: Uso di indice primario per recupero di record multipli
  - Costo S5 = costo indice primario + $\lceil Card_\sigma / Bfr \rceil$
    = $\log_2$ (#blocchi indice) + $\lceil Card_\sigma / Bfr \rceil$
- S6: Uso di un indice secondario: B^+-tree
  - Costo S6 = costo indice B^+tree + costo lettura blocchi nodi foglia B^+Tree + lettura record
    #livelli B^+tree + $\lceil Card_\sigma / Bfrind_{ex} \rceil$ + Card_σ

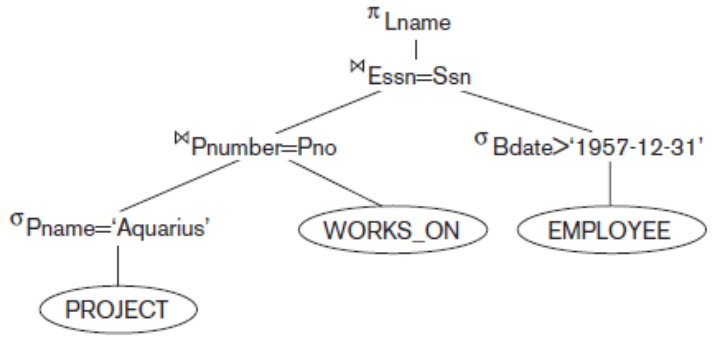
Algoritmi per l'operatore SELEZIONE

Metodi per la selezione con condizione complesse in congiunzione:

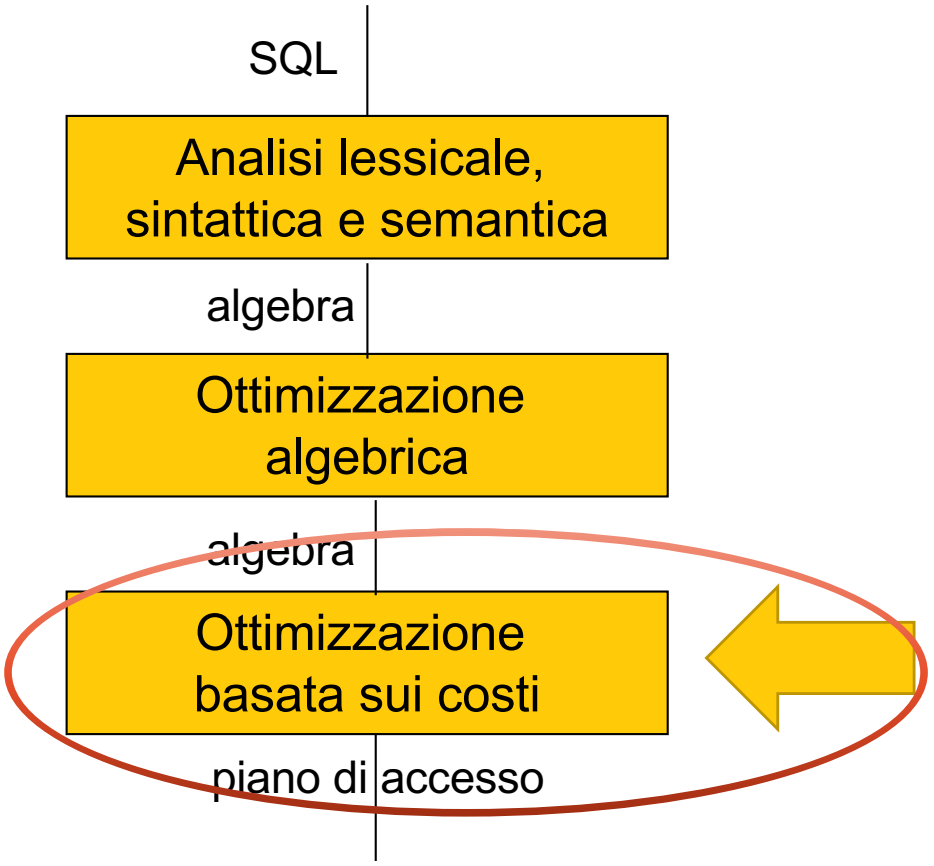
es. $\sigma_{(A\ OP\ v)\ AND\ (A'\ OP\ v')}$

- S7 *Uso di un indice individuale*:
  - uno degli attributi (A o A') ha un indice,
  - si utilizza l'indice per recuperare i record che soddisfanno la condizione semplice
  - si controlla che ogni record recuperato soddisfi le rimanenti condizioni semplici
- S8 *Intersezione di puntatori a record*:
  - Attributi hanno indici puntano ai record (non solo i blocchi)
  - Si usano gli indici per recuperare l'insieme dei puntatori a record che soddisfano la cond. semplice
  - Si calcola l'intersezione dei puntatori
  - Se solo alcuni degli attributi delle condizioni posseggono indici ai puntatori, si esegue S7
- S9. *Uso di un indice composto*:
  - Esiste un indice composto definito su più attributi della condizione congiuntiva
    - Es. chiave primaria composta da più attributi LAVORA_SU(SSN_I, N_P)
  - Si utilizza l'indice per trovare i valori che soddisfano condizioni semplici (=, >,< >=, <=)

Esecuzione e ottimizzazione delle interrogazioni



- Il DBMS offre più strategie per eseguire gli operatori algebrici:
  - Algoritmi per la selezione
  - Algoritmi per il join
  - Algoritmi per la proiezione
- Molte strategie assumono i dati ordinati:
  - Algoritmo d'ordinamento



Algoritmi per l'operatore JOIN

- Operatore di JOIN
  - L'operazione più costosa
  - Consideriamo EQUIJOIN (NATURAL JOIN) tra due relazioni

$$R \bowtie_{R.A=S.B} S$$

- Algoritmi per implementar il join
  - J1: Nested-loop
  - J2: Nested-loop basato su indice
  - J3: Sort-merge join
  - J4: (Partition) hash join

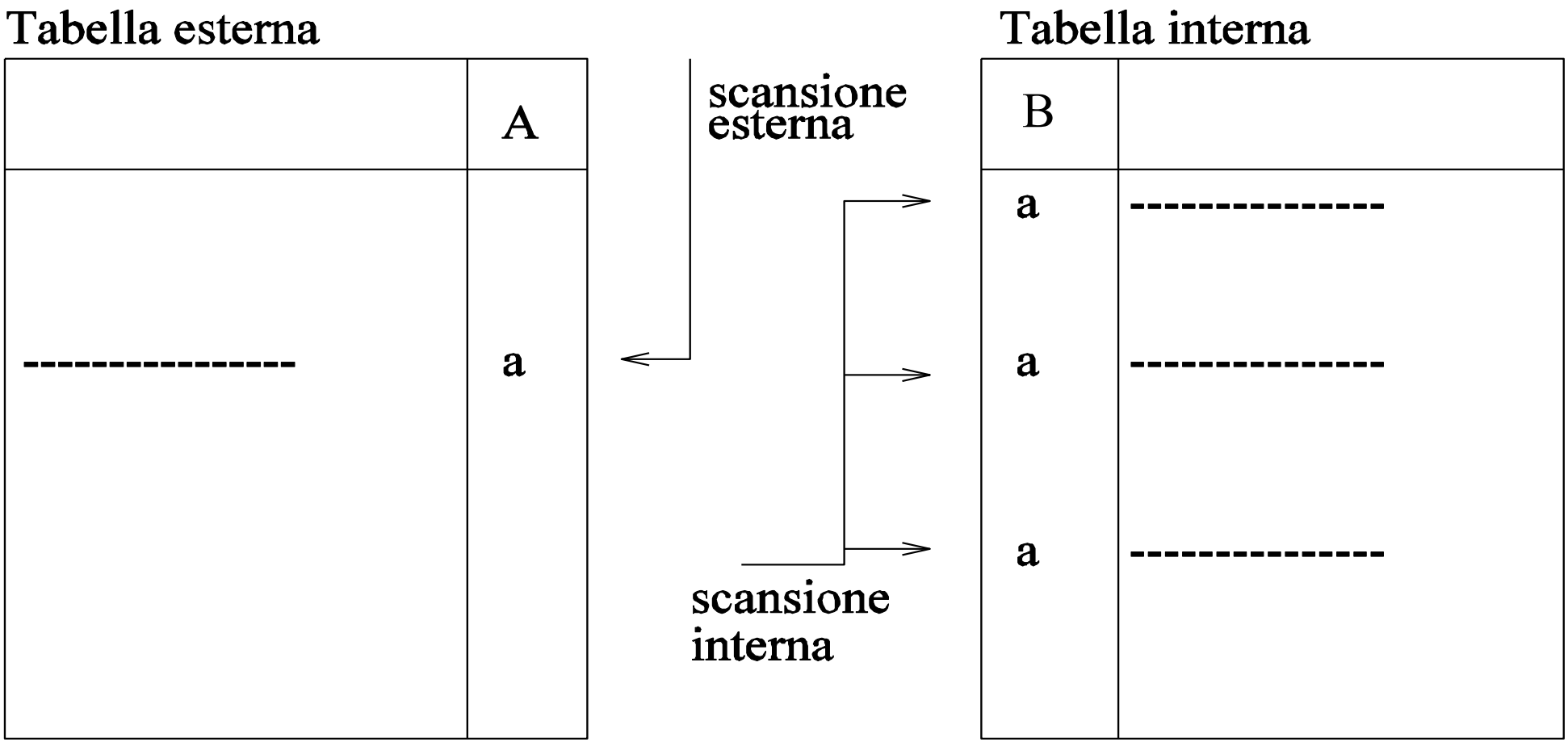
Algoritmi per l'operatore JOIN:
Join nested -loop

- E' l'algoritmo predefinito, di forza bruta, perché non richiede alcun percorso d'accesso speciale a nessuno dei due file del join
- Per ogni record *r* in *R* (ciclo esterno),
  - si recupera ogni record *s* da *S* (ciclo interno)
  - Se se i due record soddisfano la condizione di join, *r.A* = *s.B,* viene composto il join

```
∀ r in R
   ∀ s in S
         If  r.A=s.B
              output  rxs
```

Join Nested-loop

$R \bowtie S$



Algoritmi per l'operatore JOIN:
Join nested –loop su blocchi

Il buffer dedica 3 blocchi al join (2 input e 1 output)

```
Leggo un blocco di R
  Leggo un blocco di S
    output dei possibili join tra i blocchi
```

Stima del costo per la lettura dei blocchi: Costo I/O =

B(R) +B(R)*B(S)

- Es. R in B(R)=1000 e S in B(S)=500 blocchi

  Costo = B(R) * B(S) + B(M) = 1000*500+1000= 501.000 I/Os

  con 10ms/IO, costo = 1,4 ore

Algoritmi per l'operatore JOIN:
Join nested –loop su blocchi

Il buffer dedica 3 blocchi al join (2 input e 1 output)

```
Leggo un blocco di R
  Leggo un blocco di S
    output dei possibili join tra i blocchi
```

Stima del costo per la lettura dei blocchi: Costo I/O =

B(R) +B(R)*B(S)

Quale relazione dovrebbe essere esterna (*outer)*?

la più piccola (in termini di blocchi)

Algoritmi per l'operatore JOIN:
Join nested –loop su blocchi

Il buffer dedica B blocchi per il nested join + 1 blocco per l'output

- Associa ***B***–1 blocchi di buffer alla relazione outer (R) e uno alla relazione inner (S)

```
Leggo B-1 blocchi di R
  Leggo un blocco di S
    output dei possibili join tra i blocchi
```

Costo I/O=

$B(R)+\lceil B(R)/(B-1)\rceil * B(S)$

- Es. R in B(R)=1000 e S in B(S)=500 blocchi, con con 10 blocchi di buffer

  Costo = B(R) + [B(R) /(***B***–1)]* B(S) = 1000 + 112*500= 56.500 I/O

  con 10ms/IO, costo = 565 sec= 9,4 min

Algoritmi per l'operatore JOIN:
Join nested –loop su blocchi

Il buffer dedica B blocchi per il nested join + 1 blocco per l'output

- Associa ***B***–1 blocchi di buffer alla relazione outer (R) e uno alla relazione inner (S)

```
Leggo B-1 blocchi di R
  Leggo un blocco di S
    output dei possibili join tra i blocchi
```

Costo I/O=

$B(R)+\lceil B(R)/(B-1)\rceil * B(S)$

- Es. R in B(R)=1000 e S in B(S)=500 blocchi, con 100 blocchi di buffer

  Costo = B(R) + [B(R) /(***B***–1)]* B(S) = 1000 + 11*500= 6.500 I/O

  con 10ms/IO, costo = 1 min

Algoritmi per l'operatore JOIN:
Join nested –loop su blocchi

Il buffer dedica B blocchi per il nested join + 1 blocco per l'output

- Associa ***B***–1 blocchi di buffer alla relazione outer (R) e uno alla relazione inner (S)
- Se B(R)<=B-1? La relazione R è tutta in memoria

Costo I/O=

B(R) + B(S)

- Es. R in B(R)=100 e S in B(S)=500 blocchi, con 101 blocchi di buffer

  Costo = B(R) + B(S) = 600 I/O

  con 10ms/IO, costo = 6 sec

Algoritmi per l'operatore JOIN:

Join nested-loop basato su indice

- Variante del precedente
  - Si applica nel caso esista un indice su uno dei due attributi di join
- Ipotizziamo esista indice su B di S. In tal caso:
  - Per ogni record *r* in *R* (ciclo esterno):
    - Si utilizza la struttura di accesso (indice) per recuperare direttamente da *S* tutti i record *s* che soddisfano *r.A*= s.B

```
∀ r in R
        s=getIndex_S.B(r.A)  –funzione per recuperare s con s.B=r.A
          If   s NOT empty
                output   rxs
```

Costo I/O=

B(R) + T(R) * c

c: costo di ricerca nell'indice – (Dipende dall'indice utilizzato)

Algoritmi per l'operatore JOIN:
Sort-merge join

- Si può applicare nel caso i record di *R* e *S* siano *fisicamente ordinati* sugli attributi di join R.*A* e S.*B*
- Si esegue una scansione concorrente di entrambi i file secondo l'ordine degli attributi di join confrontando i record che posseggono i medesimi valori per R.*A* e *S*.B

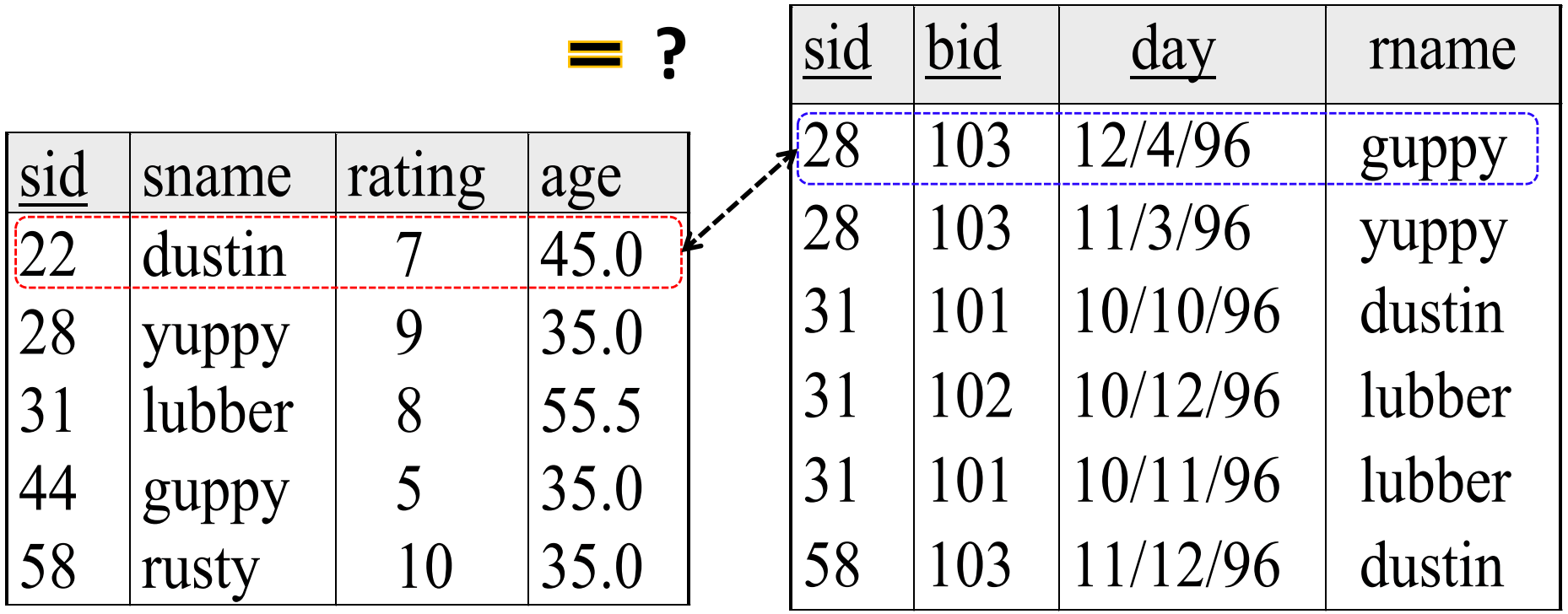
Ann	1
Ben	2
Cal	4
Dan	5
Emi	7
Fin	9

1	8377
4	9476
6	2735
7	1884
8	9546
9	4572

Algoritmi per l'operatore JOIN:
Sort-merge join

- Si può applicare nel caso i record di *R* e *S* siano *fisicamente ordinati* sugli attributi di join R.*A* e S.*B*
- Si esegue una scansione concorrente di entrambi i file secondo l'ordine degli attributi di join confrontando i record che posseggono i medesimi valori per R.*A* e S.*B*.
- Il confronto avviene tra coppie di blocchi dei file:
  - vengono copiati in ordine nei buffer di memoria
  - i record di ogni file sono scanditi solo una volta per il confronto con l'altro file

Sort-Merge Join: un esempio



sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

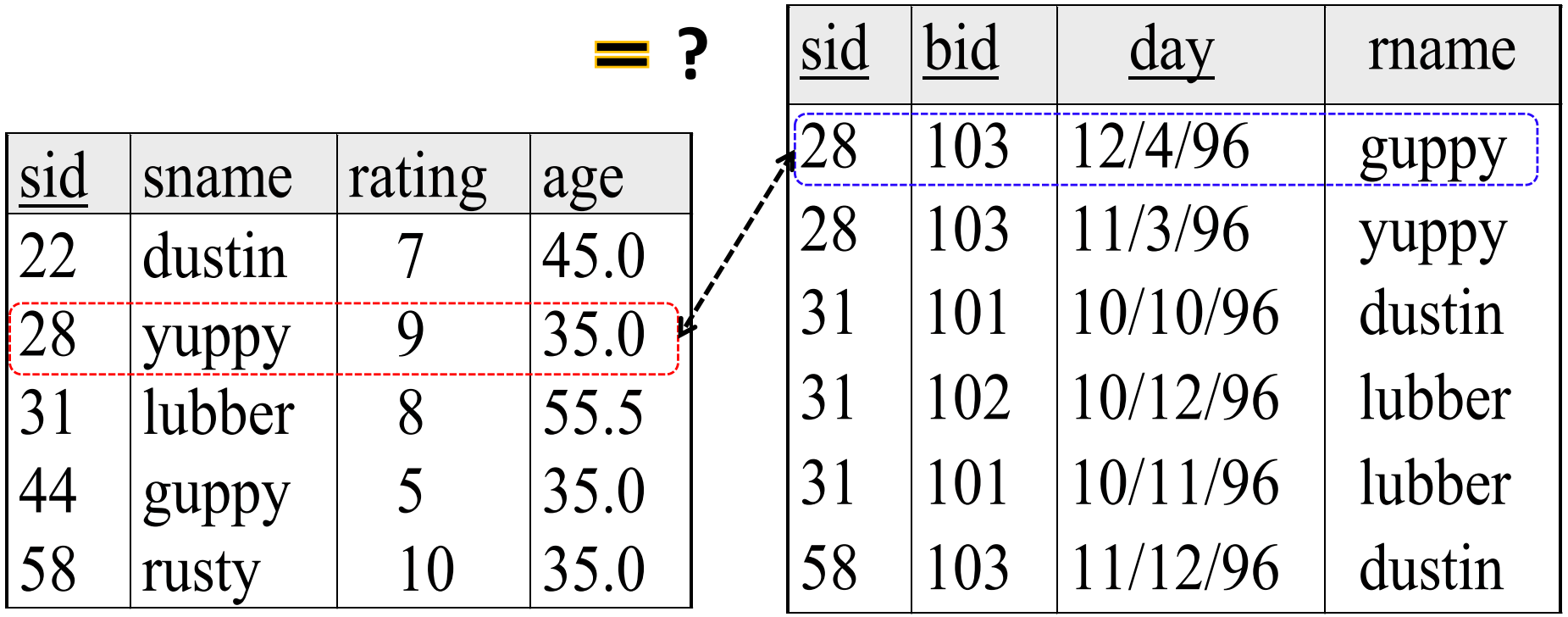
Sort-Merge Join: un esempio

= NO

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

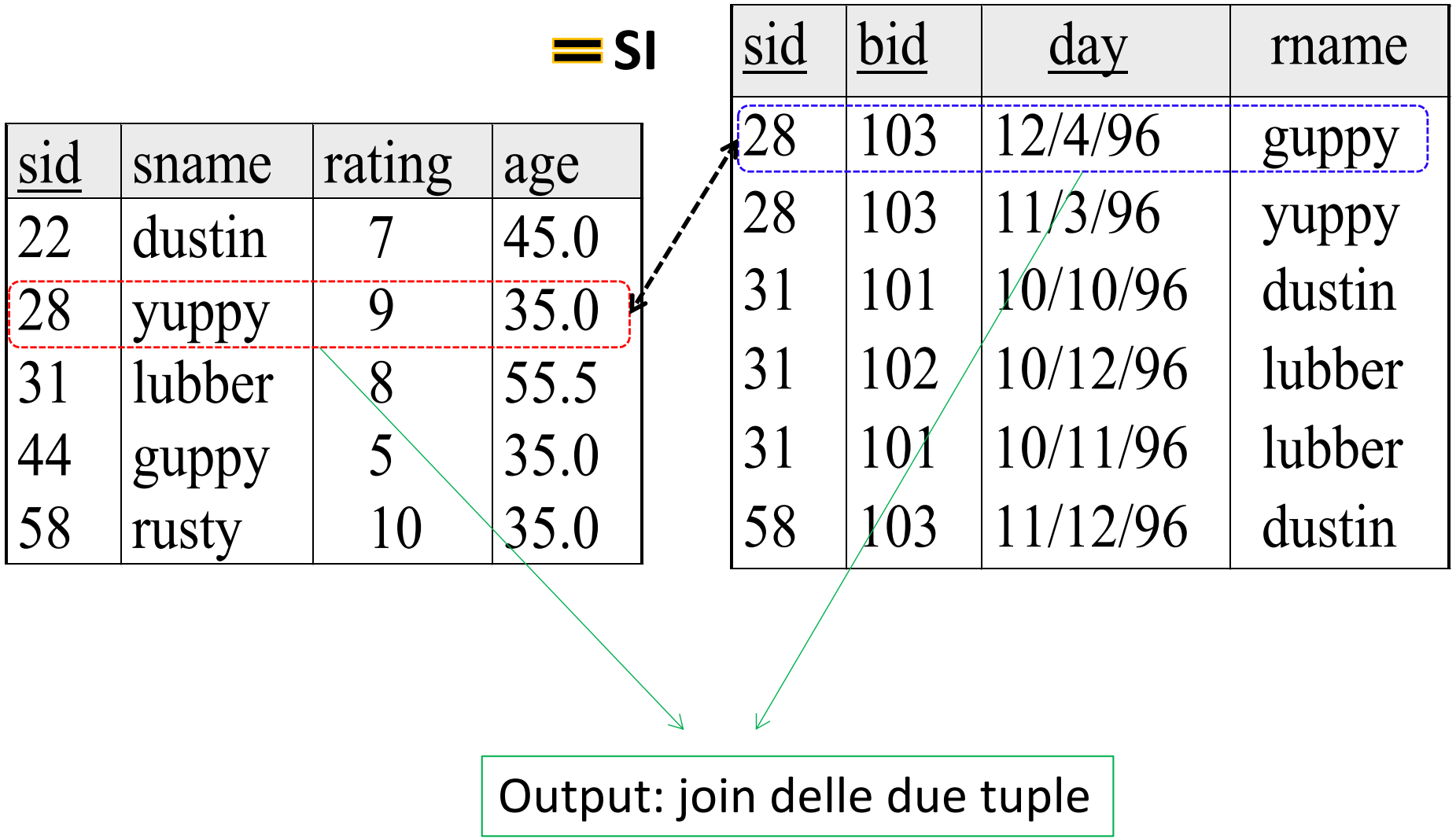
Sort-Merge Join: un esempio



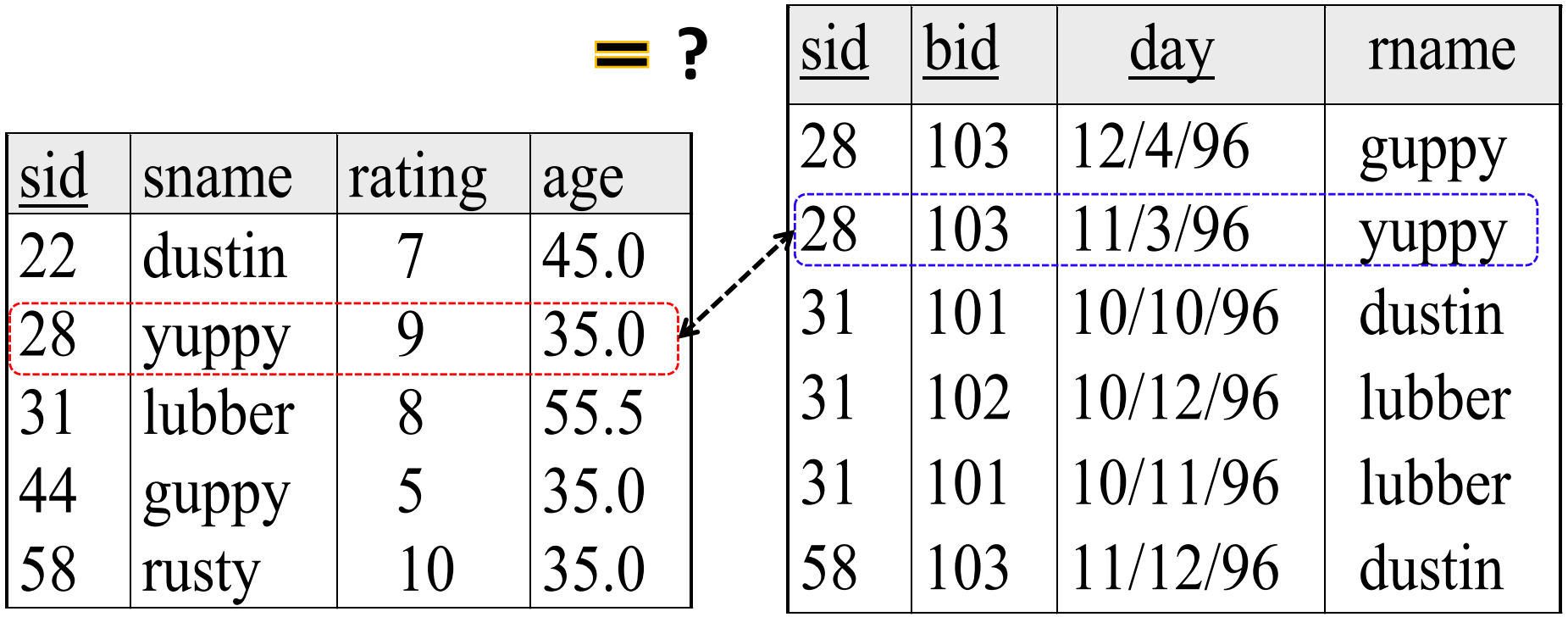
sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: un esempio



Sort-Merge Join: un esempio



sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

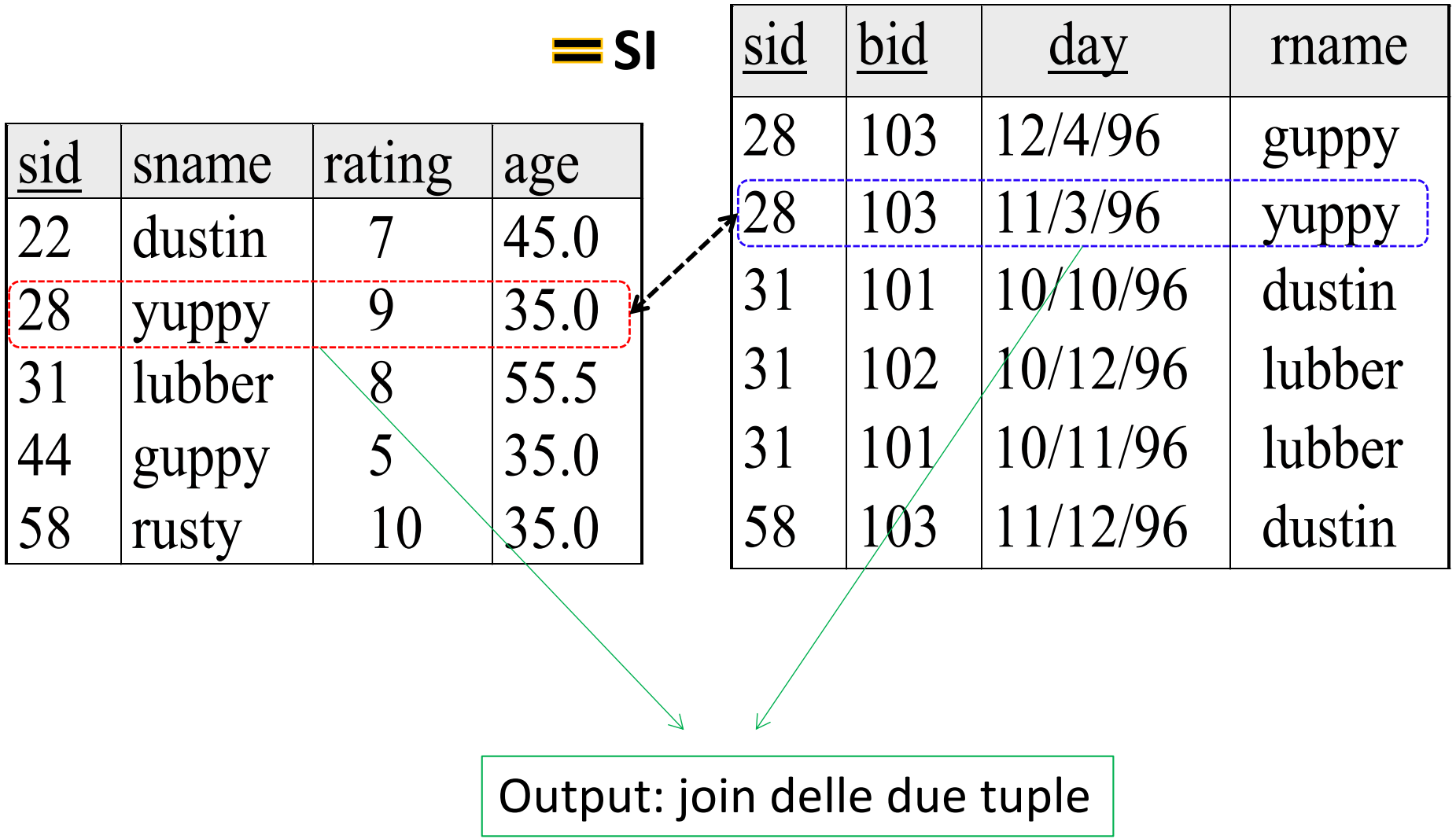
Sort-Merge Join: un esempio

= SI

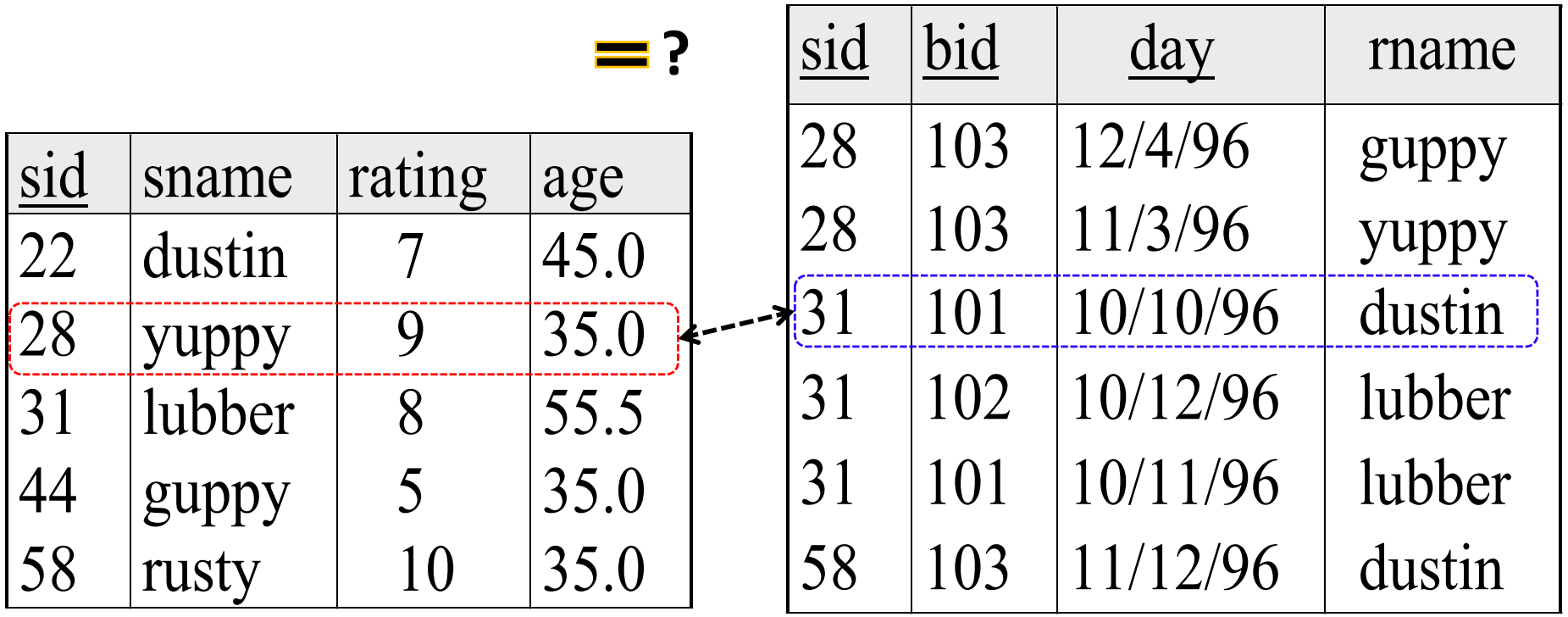
sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: un esempio



Sort-Merge Join: un esempio



sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: un esempio

= NO

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: un esempio

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Sort-Merge Join: un esempio

= SI

sid	sname	rating	age
22	dustin	7	45.0
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	bid	day	rname
28	103	12/4/96	guppy
28	103	11/3/96	yuppy
31	101	10/10/96	dustin
31	102	10/12/96	lubber
31	101	10/11/96	lubber
58	103	11/12/96	dustin

Si continua cosi

Output: join delle due tuple

Algoritmi per l'operatore JOIN:
Sort-merge join

- Se R e S sono fisicamente ordinati, è necessario prima eseguire un ordinamento esterno su entrambi
- Costo I/O = B(R) + B(S)

Algoritmi per l'operatore JOIN:
Sort-merge join

- Se R e S **non** sono fisicamente ordinati, è necessario prima eseguire un ordinamento esterno su entrambi
- Costo I/O = ?
- Avendo a disposizione B blocchi di buffer

  Costo I/0 = costo ordinamento R + costo ordinamento S + B(R) + B(S)

  $$= 2B(R)*(1+\log_{B-1}\lceil B(R)/B\rceil) + 2B(S)*(1+\log_{B-1}\lceil B(S)/B\rceil) + B(R) + B(S)$$

- Es. R in B(R)=1000 e S in B(S)=500 blocchi, con con 10 blocchi di buffer

  Costo = $2*1000*(1+\log_9\lceil 1000/10\rceil) + 2*500*(1+\log_9\lceil 500/10\rceil) +1000+500$
  Costo=2000*(1+3) + 1000*(1+2) +1000+500= 12.500
  con 10ms/IO, costo = 125 sec

Algoritmi per l'operatore JOIN:
Sort-merge join

- Se R e S **non** sono fisicamente ordinati, è necessario prima eseguire un ordinamento esterno su entrambi
- Costo I/O = ?
- Avendo a disposizione B blocchi di buffer

  Costo I/0 = costo ordinamento R + costo ordinamento S + B(R) + B(S)

  $$= 2B(R)*(1+\log_{B-1}\lceil B(R)/B\rceil) + 2B(S)*(1+\log_{B-1}\lceil B(S)/B\rceil) + B(R) + B(S)$$

- Es. R in B(R)=1000 e S in B(S)=500 blocchi, con con 100 blocchi di buffer

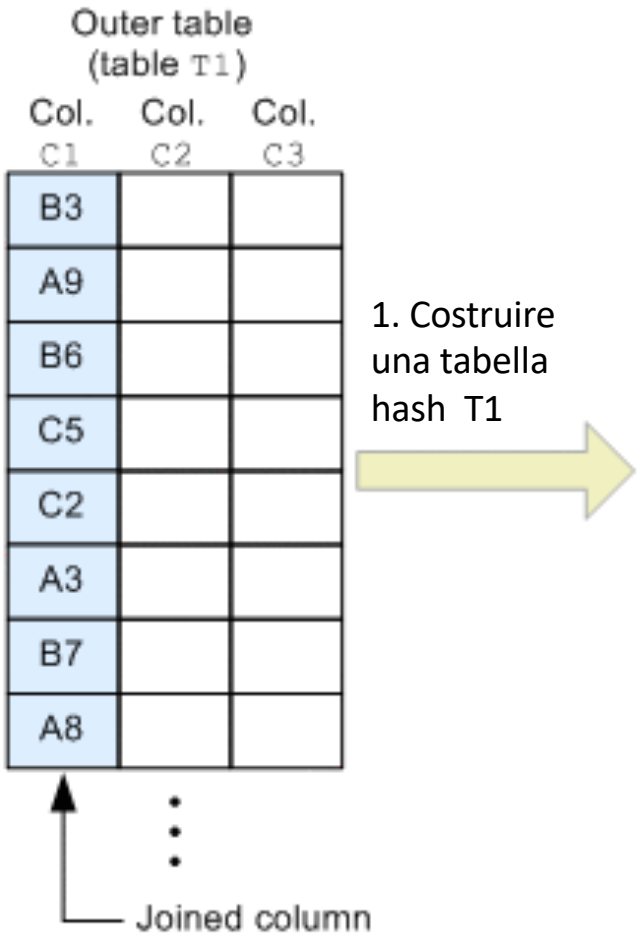
  Costo = $2*1000*(1+\log_{99}\lceil 1000/100\rceil) + 2*500*(1+\log_{99}\lceil 500/100\rceil) +1000+500$
  Costo=2000*(1+2) + 1000*(1+1) +1000+500= 9.500
  con 10ms/IO, costo = 95 sec

Algoritmi per l'operatore JOIN:
Hash join

- Si utilizza una funzione di hash per creare la corrispondenza tra record di R e S aventi lo stesso valore per l'attributo di join
- hash join (semplice):
  - Si applica una funzione hash **h()** ai valori degli attributi di join di una delle due relazioni (e.g., R.A), generando una tabella hash per i suoi record
  - si scandisce la seconda relazione (S) e, per ogni record, si applica h() al suo attributo di join (e.g., S.B), si cerca nella tabella di hash i valori di R corrispondenti

Algoritmi per l'operatore JOIN:

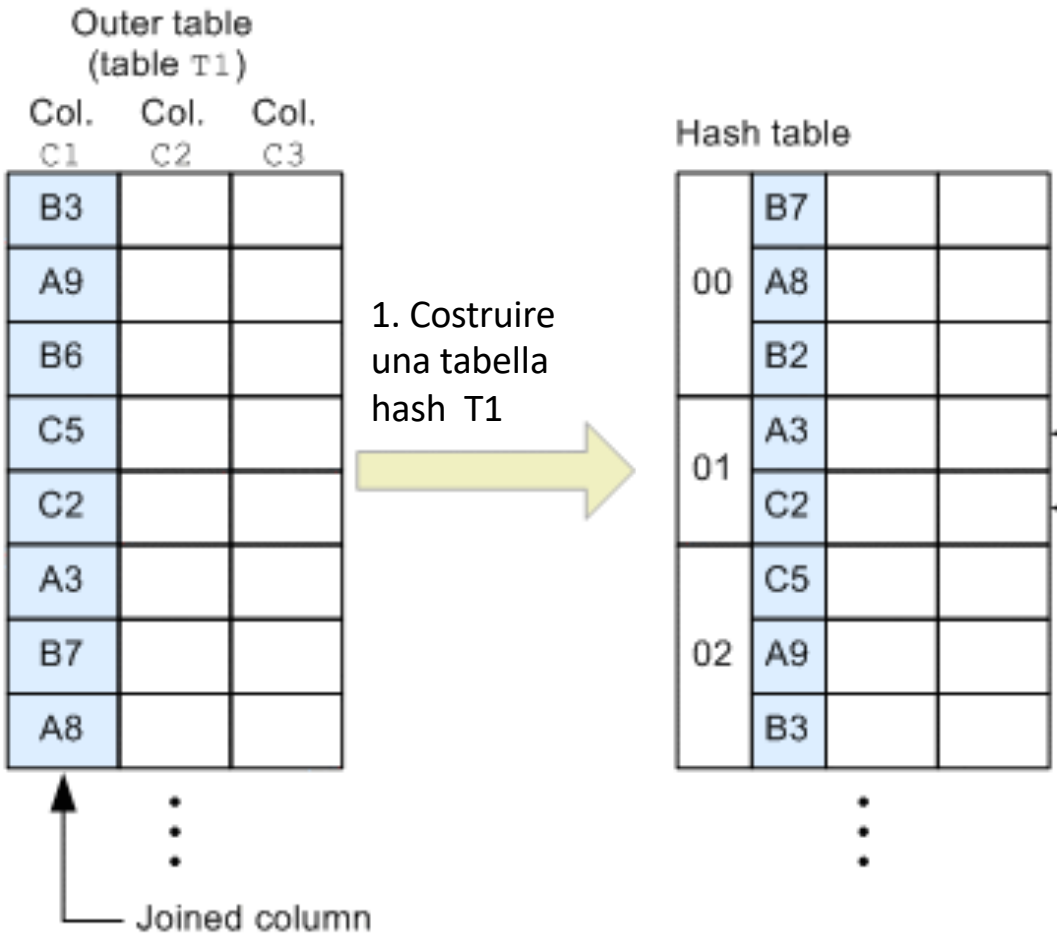
Hash join



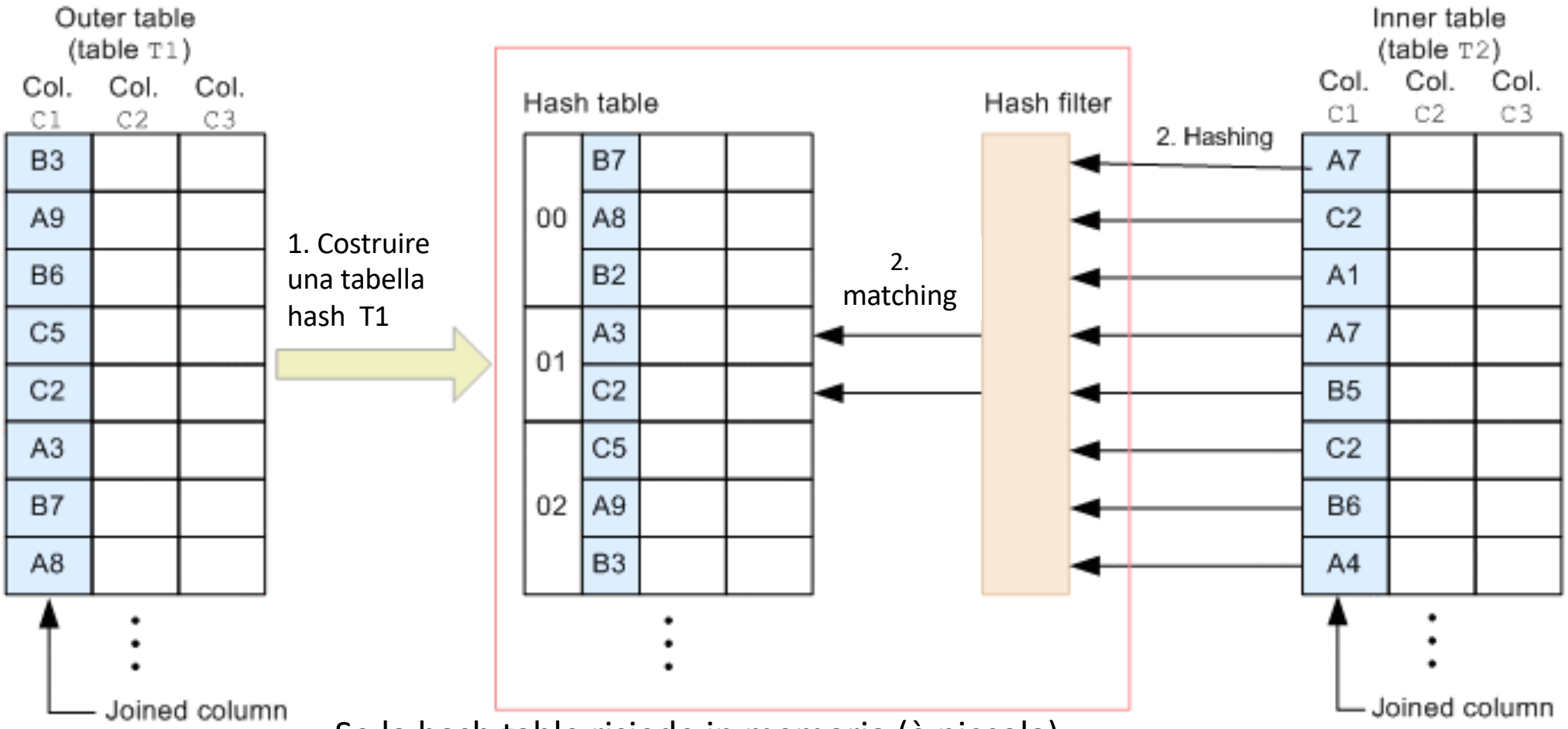
1. Costruire una tabella hash T1

Algoritmi per l'operatore JOIN:

Hash join



Algoritmi per l'operatore JOIN:
Hash join

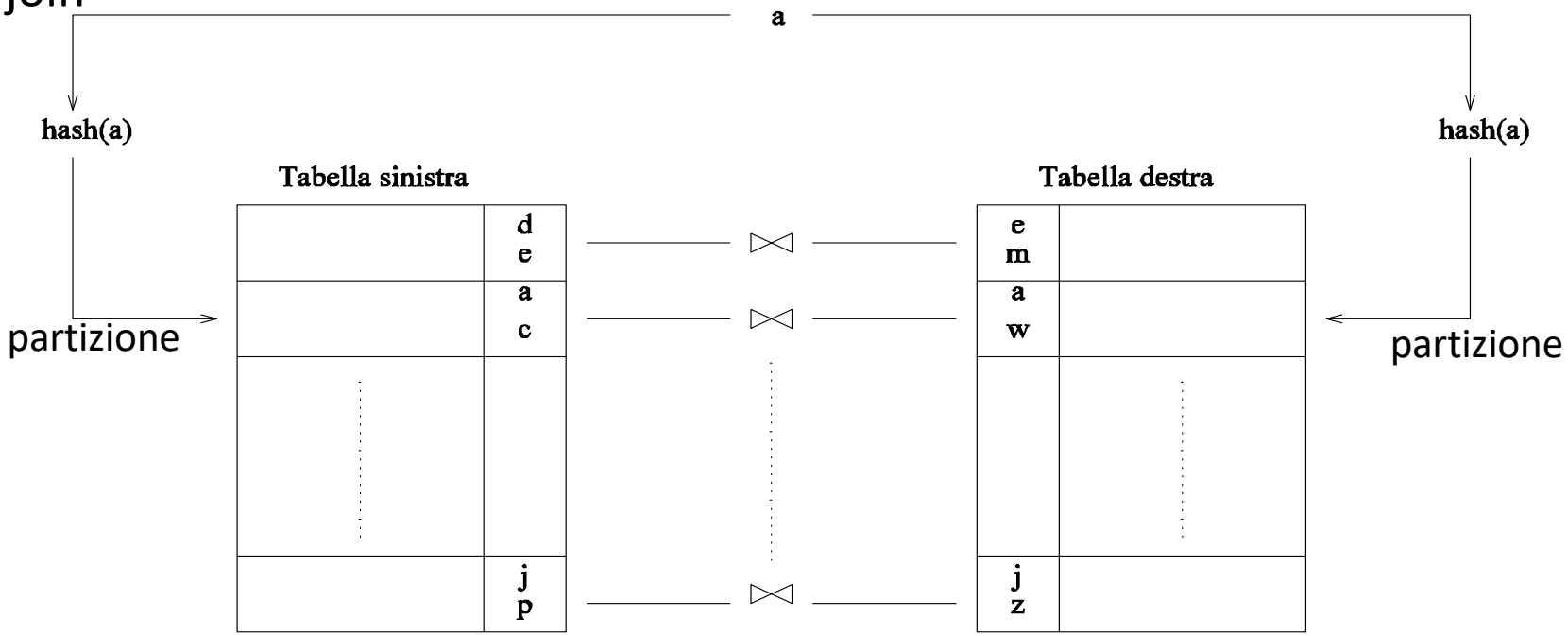


Se la hash table risiede in memoria (è piccola),
costo I/O= # blocchi della seconda relazione

Algoritmi per l'operatore JOIN:

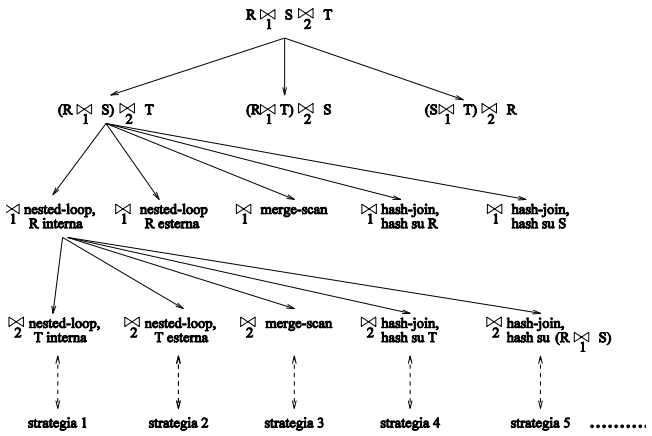
Partioned -Hash join

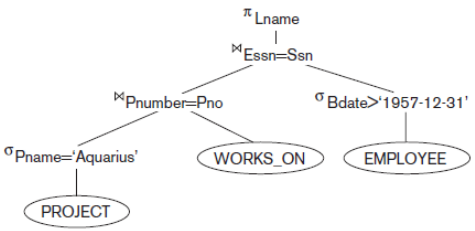
- Si utilizza una funzione hash per dividere le tuple in partizioni disgiunte
- Le tuple di una partizione della prima relazione possono essere in match solo con le tuple della corrispondente partizione della seconda relazione, si diminuiscono i confronti da effettuare
- Per ogni coppia di partizione si può usare un simple hash join o qualunque altro metodo di join

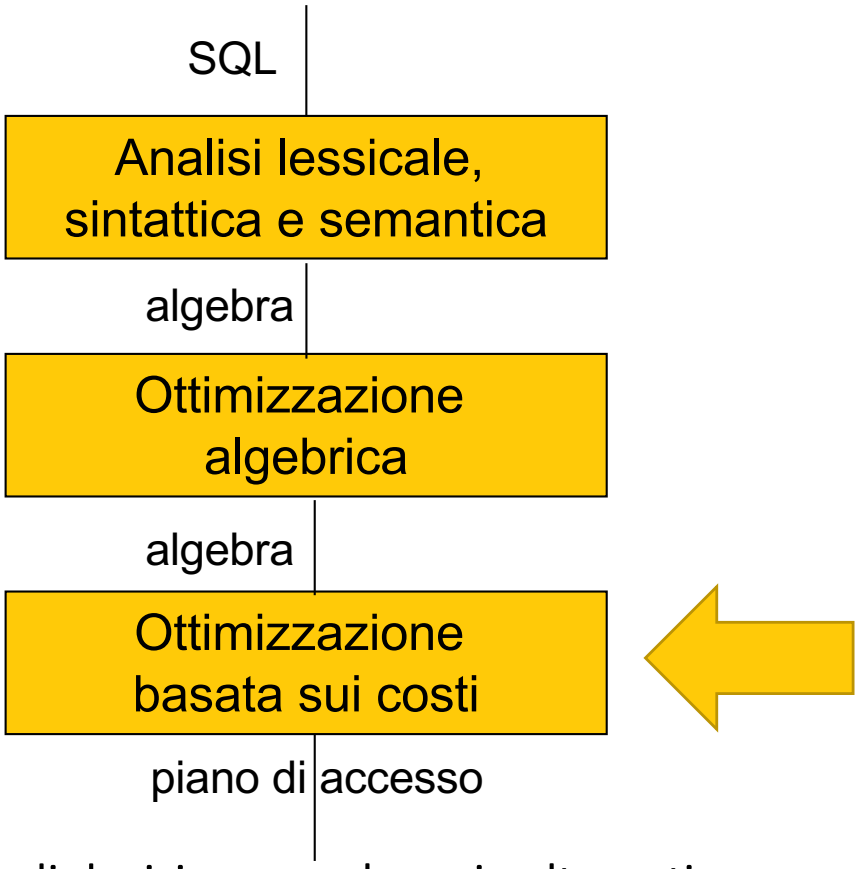


Esecuzione e ottimizzazione delle interrogazioni

SELECT E.Lname
FROM EMPLOYEE E, WORKS_ON W, PROJECT P
WHERE P.Pname='Aquarius' **AND** P.Pnumber=W.Pno **AND**
E.Essn=W.ssn **AND** E.Bdate>'1957-12-31'







- Ogni operatore può essere eseguito con diversi algoritmi
- Per scegliere la combinazione migliore si costruisce un albero di decisione con le varie alternative ("**piani di esecuzione**")
- Si valuta il costo di ciascun piano e si sceglie il piano di costo minore

Esempio albero di decisione per R ⋈ S ⋈ T