ARGOMENTO 12

Piani di esecuzione di query

Il problema (intuizione)

Quando scriviamo una query SQL, come ad esempio:

```
SELECT nome, cognome
FROM studenti
WHERE corso = 'Informatica'
ORDER BY cognome;
```

il DBMS non «esegue» direttamente il testo della query (che dice **cosa** vogliamo ottenere)

Il suo compito è di costruire un piano per dcidere **come** ottenere i dati richiesti nel modo più efficiente possibile

Il ruolo dell'ottimizzatore del DBMS

- Il *query optimizer* del DBMS prende la query e valuta diverse possibili strategie di esecuzione.
- Esempi di scelte che può fare:
  - Come accedere alle tabelle: *full table scan* o *indice*?
  - Come applicare le condizioni (WHERE): filtrare subito o dopo una join?
  - Come eseguire le join: nested loop, hash join, merge join …
  - Come ordinare i dati: usare un indice già ordinato o fare un sort esplicito?

Esecuzione di query SQL

- Per ogni query, esistono almeno due livelli di pianificazione:
  - Pianificazione logica: decidere COSA fare per produrre il risultato
    - Es. selezione, proiezione, join, unione, …
  - Pianificazione fisica: decidere COME eseguire i singoli passi del piano logico
    - file scan, nested loop join, hash join, …
- Ogni piano può avere costi MOLTO differenti
- Domanda: come si fa a calcolare (stimare) il costo dell'esecuzione di un piano?

Esecuzione di query SQL

- Per *stimare* il costo dell'esecuzione di un piano, noi qui ci limiteremo a discutere due aspetti:
  - La stima della **cardinalità** di ogni passo intermedio dell'esecuzione della query
  - Il **costo** di ogni singolo passo in termini di I/O (come visto nelle lezioni precedenti)

- Argomenti più avanzati (che non toccheremo) sono:
  - Lo spazio di ricerca (quali diversi piani considerare)
  - Gli algoritmi di ricerca (come esplorare lo spazio di ricerca)

Cosa dobbiamo sapere per il calcolo

- Come premessa, è necessario conoscere almeno quanto segue:
  - La cardinalità delle relazioni coinvolte (|R|)
  - La presenza o meno di indici (e di che tipo)
  - Il numero di pagine P_R utilizzate per memorizzare ogni relazione (dimensione pagina P / dimensione tuple t_R)
  - Informazione statistica sugli attributi (es. il valore minimo, il valore massimo, il numero di valori diversi, …)

Albero di esecuzione della query

- Per rappresentare i passi di esecuzione di una query, spesso si una rappresentazione ad albero
  - Le foglie sono le relazioni di partenza (con la modalità di accesso, es. full scan o indice)
  - I nodi intermedi corrispondono a operazioni algebriche sulla/e relazioni coinvolte (join, filtri, aggregazioni, …)
  - A ogni nodo intermedio viene associato un costo
  - La radice corrisponde all'output della query (che a sua volta è una relazione)

La base di dati «sailors»

Sailors (sid, sname, srating, age)
Boats(bid, bname, color)
Reserves(sid, bid, date, rname)

Q1: *What are the names of the sailors who have reserved boat with name "100" ?*

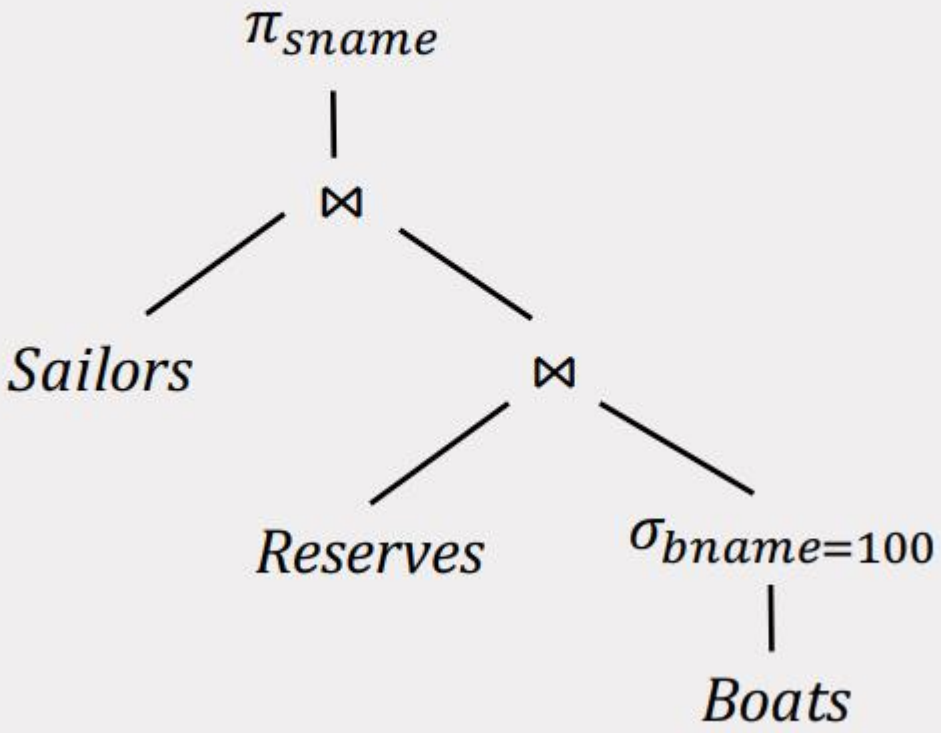
Q2: *What are the names of the sailors who have reserved a red boat ?*

Q3: *What are the names of the sailors who have reserved a green* ***or*** *red boat ?*

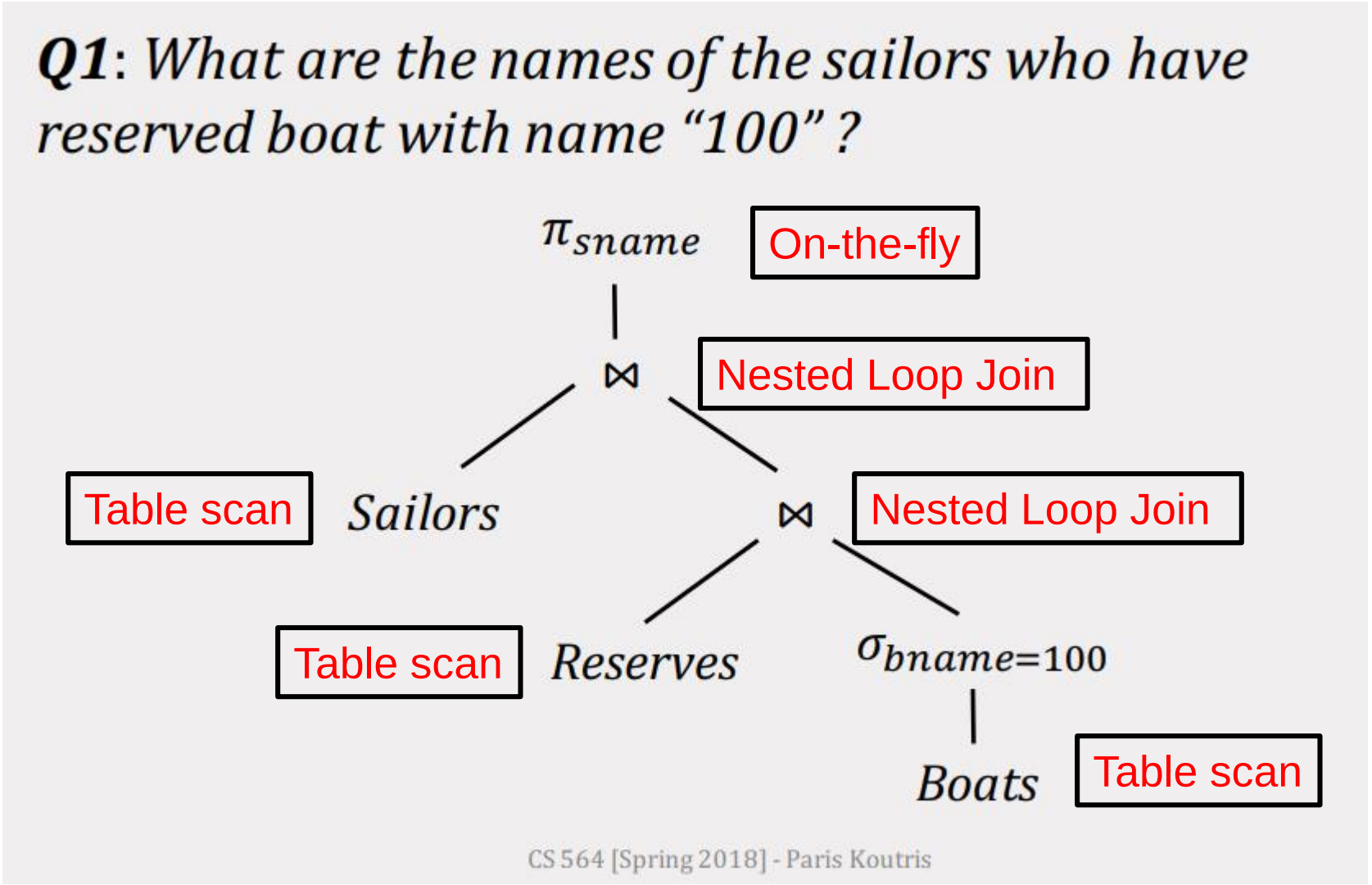
Q4: *What are the names of the sailors who have reserved a green* ***and*** *red boat ?*

Piano logico

Q1: *What are the names of the sailors who have reserved boat with name "100" ?*

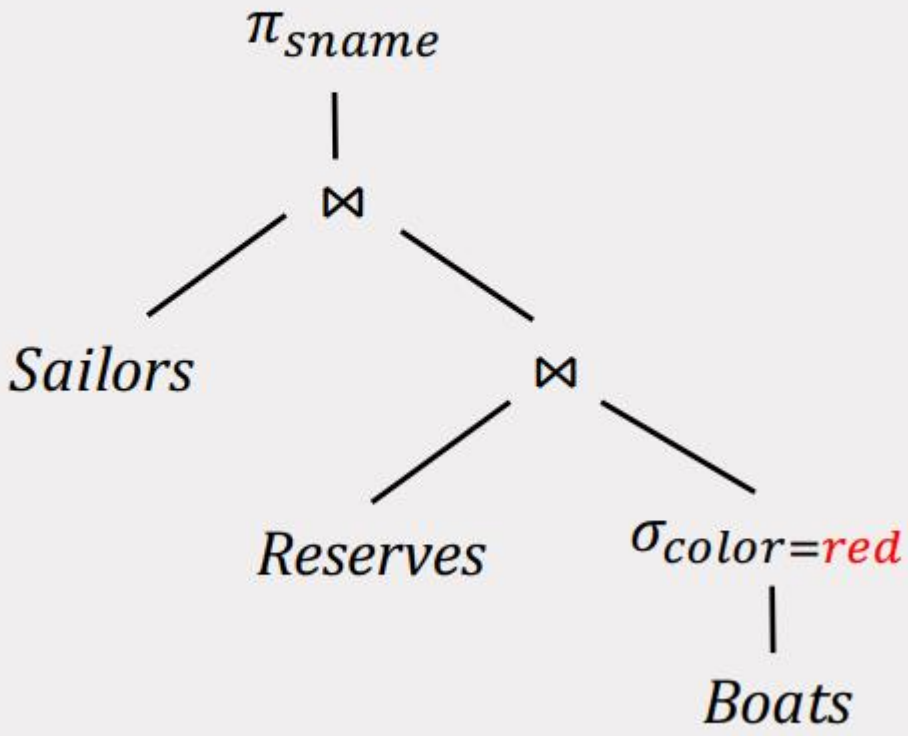


Piano fisico (esempio)



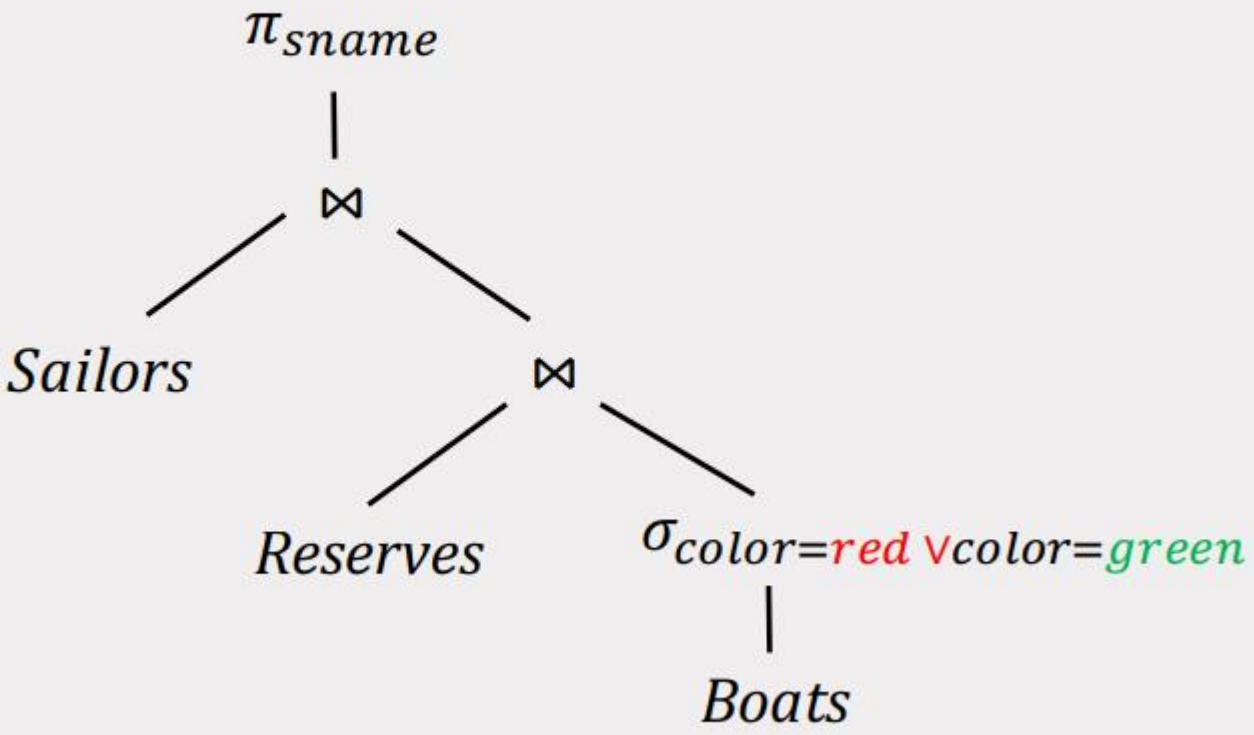
Piano logico

Q2: *What are the names of the sailors who have reserved a red boat ?*

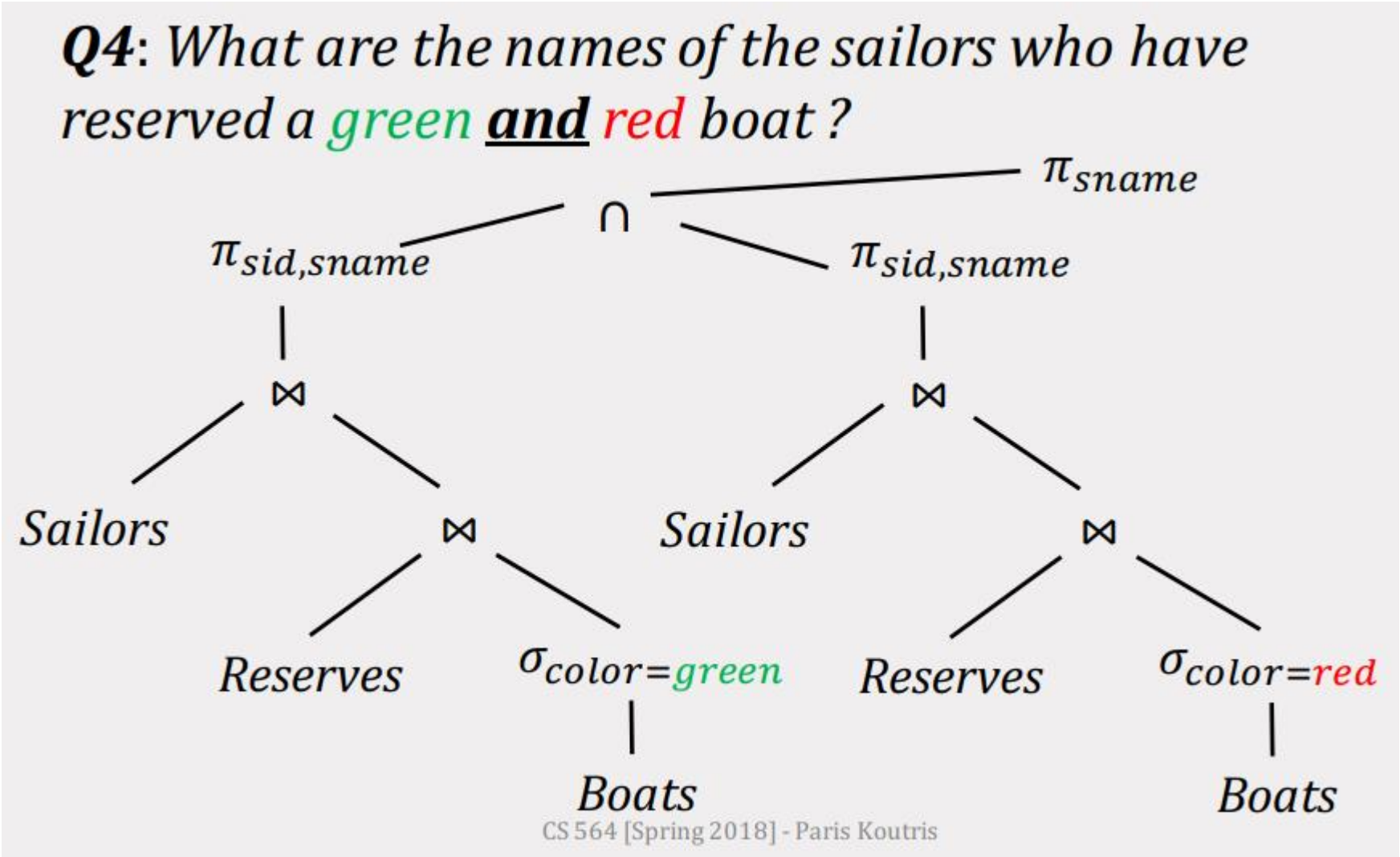


Piano logico

Q3: *What are the names of the sailors who have reserved a green* ***or*** *red boat ?*



Piano logico



Calcolo del costo di esecuzione

Sailors (sid, sname, srating, age)
Boats(bid, bname, color)
Reserves(sid, bid, date, rname)

Query:

```
SELECT S.sid, R.rname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid
AND B.bid = R.bid
AND B.color = red
```

Assunzioni di partenza

Sailors (<u>sid</u>, sname, srating, age)
Boats(<u>bid</u>, bname, color)
Reserves(<u>sid, bid, date</u>, rname)

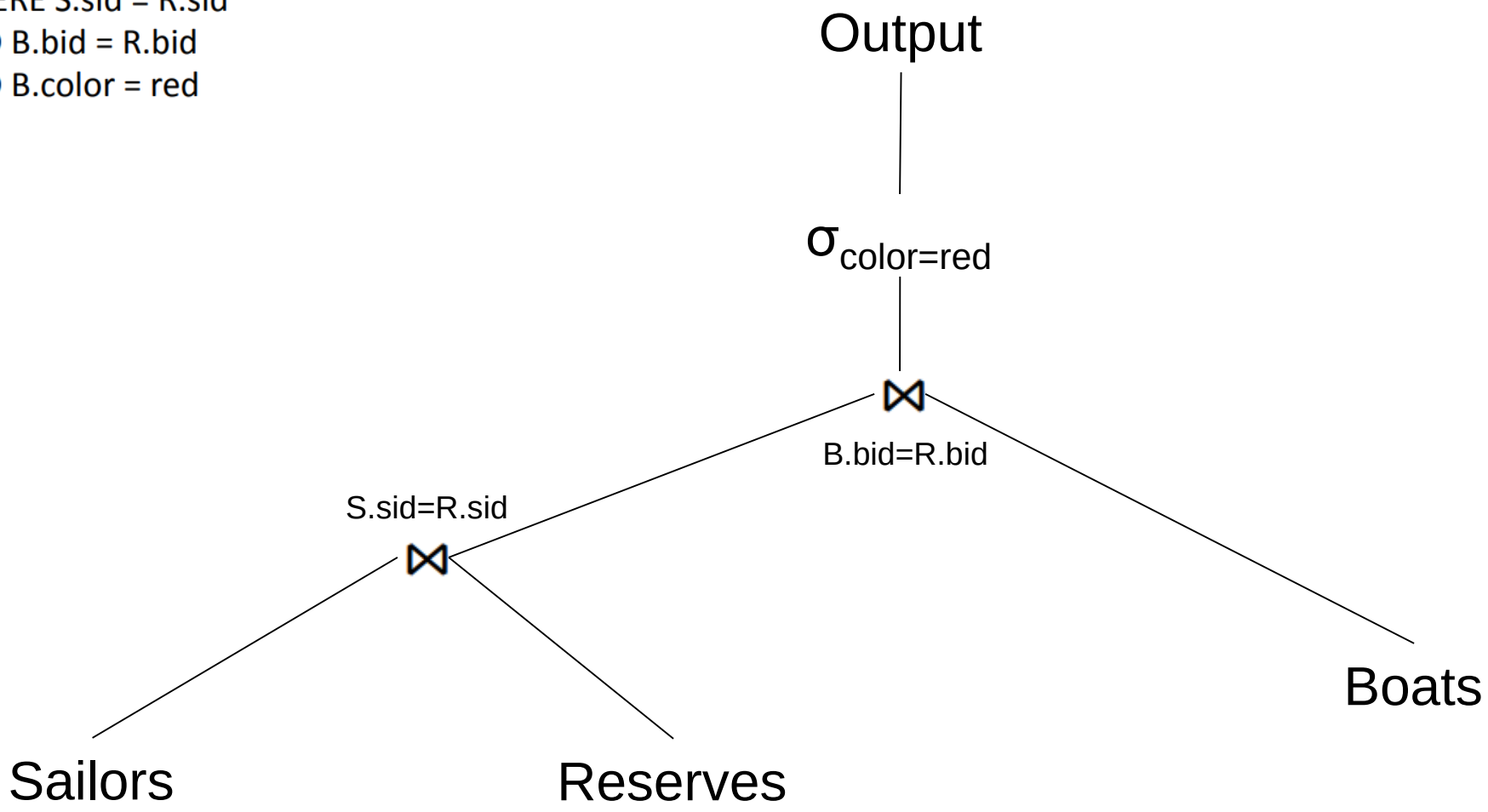
- $|S| = 12$
- $|B| = 4$
- $|R| = 10$
- f(color) = 1/3
- f(B.bid = R.bid) = 1/4
- f(S.sid = R.sid) = 1/12

- $t_S = t_B = t_R = 25$ byte
- P = 50 byte
- $P / t_x = 2$
- $P_S = 6$
- $P_B = 2$
- $P_R = 5$

Esempio

▪ |S| = 12	▪ $t_S = t_B = t_R$ = 25 byte
▪ |B| = 4	▪ P = 50 byte
▪ |R| = 10	▪ $P / t_x = 2$
▪ f(color) = 1/3	▪ $P_S = 6$
▪ f(B.bid = R.bid) = 1/4	▪ $P_B = 2$
▪ f(S.sid = R.sid) = 1/12	▪ $P_R = 5$

```
SELECT S.sid, R.rname
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid
AND B.bid = R.bid
AND B.color = red
```



Calcolo della cardinalità dell'output

▪ |S| = 12	▪ $t_S = t_B = t_R$ = 25 byte
▪ |B| = 4	▪ P = 50 byte
▪ |R| = 10	▪ $P / t_x = 2$
▪ f(color) = 1/3	▪ $P_S = 6$
▪ f(B.bid = R.bid) = 1/4	▪ $P_B = 2$
▪ f(S.sid = R.sid) = 1/12	▪ $P_R = 5$

- La cardinalità delle varie operazioni viene approssimata bottom-up nell'albero di esecuzione
- Osservazione: la cardinalità dell'output sarà:

  $|Res| <= |R| \times |S| \times |B| <= 10 \times 12 \times 4 <= 480$

- Questo valore viene ridotto da ognuna delle condizioni usando il concetto di **selectivity factor** *f* (vedi lezione sui costi delle query)
- Come sappiamo, *f* può solo essere stimato

Esempio: cardinalità

▪ |S| = 12	▪ $t_S = t_B = t_R$ = 25 byte
▪ |B| = 4	▪ P = 50 byte
▪ |R| = 10	▪ $P / t_x = 2$
▪ f(color) = 1/3	▪ $P_S = 6$
▪ f(B.bid = R.bid) = 1/4	▪ $P_B = 2$
▪ f(S.sid = R.sid) = 1/12	▪ $P_R = 5$

- Stima della cardinalità dell'output:

  12 * 10 * 4 * 1/12 * 1/4 * 1/3 = 3,4

  - S.sid=R.sid → 12 * 10 * 1/12 = 10
  - B.bid=R.bid → 4 * 10 * 1/4 = 10
  - color = 'red' → 10 * 1/3 = 3,4

Esempio: costo (I/O)

▪ |S| = 12 ▪ |B| = 4 ▪ |R| = 10 ▪ f(color) = 1/3 ▪ f(B.bid = R.bid) = 1/4 ▪ f(S.sid = R.sid) = 1/12	▪ $t_S = t_B = t_R$ = 25 byte ▪ P = 50 byte ▪ $P / t_x = 2$ ▪ $P_S = 6$ ▪ $P_B = 2$ ▪ $P_R = 5$

- Costo del primo JOIN: $P_R + P_S * P_R$
  - 5 + 6 * 5 = 35
- Costo del secondo JOIN: P_B
  - 2 (leggo le due pagine che contengono le tuple di B)
  - NB: l'output del primo JOIN è già in memoria, quindi basta caricare Boats e verificare (al volo) quali tuple soddisfano la condizione del JOIN
- Il costo della selezione '*color* = *red*' è nullo (nessuna operazione di I/O)
  - Viene eseguita in memoria sull'output delle operazioni precedenti
- Costo totale: 37

Indici

▪ |S| = 12	▪ $t_S = t_B = t_R$ = 25 byte
▪ |B| = 4	▪ P = 50 byte
▪ |R| = 10	▪ $P / t_x = 2$
▪ f(color) = 1/3	▪ $P_S = 6$
▪ f(B.bid = R.bid) = 1/4	▪ $P_B = 2$
▪ f(S.sid = R.sid) = 1/12	▪ $P_R = 5$

- Cosa cambierebbe se avessimo a disposizione degli indici?
- Per esempio, supponiamo di avere un *hash index* sull'attributo `Sailors.sid`
- Il costo del primo JOIN sarebbe:

$$P_R + |R|*(L_h + |S_{s.sid = r.sid}|)$$

ovvero:

$$5 + 10*(1 + 1) = 25$$