


データベースシステム 結合アルゴリズム



結合演算とコストを考えるための例題

```
SELECT S.sname  
      FROM Reserves R, Sailors S  
WHERE R.sid=S.sid
```

- ▶ 各テーブルのシステムカタログ情報
 - ▶ Reservesテーブル(以下R)
 - ▶ ページ数:M
 - ▶ Sailorsテーブル(以下S)
 - ▶ ページ数:N



結合アルゴリズムの種類

- ▶ 入れ子型ループ結合
 - ▶ 一番素直な手法(ナイーブな手法)
- ▶ ブロック入れ子ループ結合
 - ▶ バッファプールを利用したループ結合の改良法
- ▶ ソートマージ結合
 - ▶ 大幅にIOコストを抑えられる有名な結合法
- ▶ ハッシュ結合
 - ▶ ハッシュ索引を有効に利用した結合法



入れ子型ループ結合

▶ 一番素直な結合演算方法

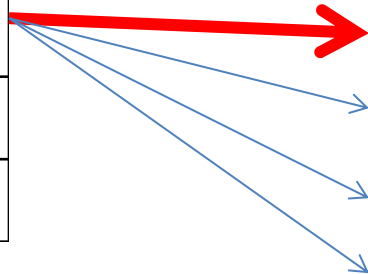
```
foreach tuple  $r \in R$  do  
  foreach tuple  $s \in S$  do  
    if  $r_i == b_j$  then add  $\langle r, s \rangle$  to result
```

R

sid	sname
22	dustin
28	yuppy
31	lubber

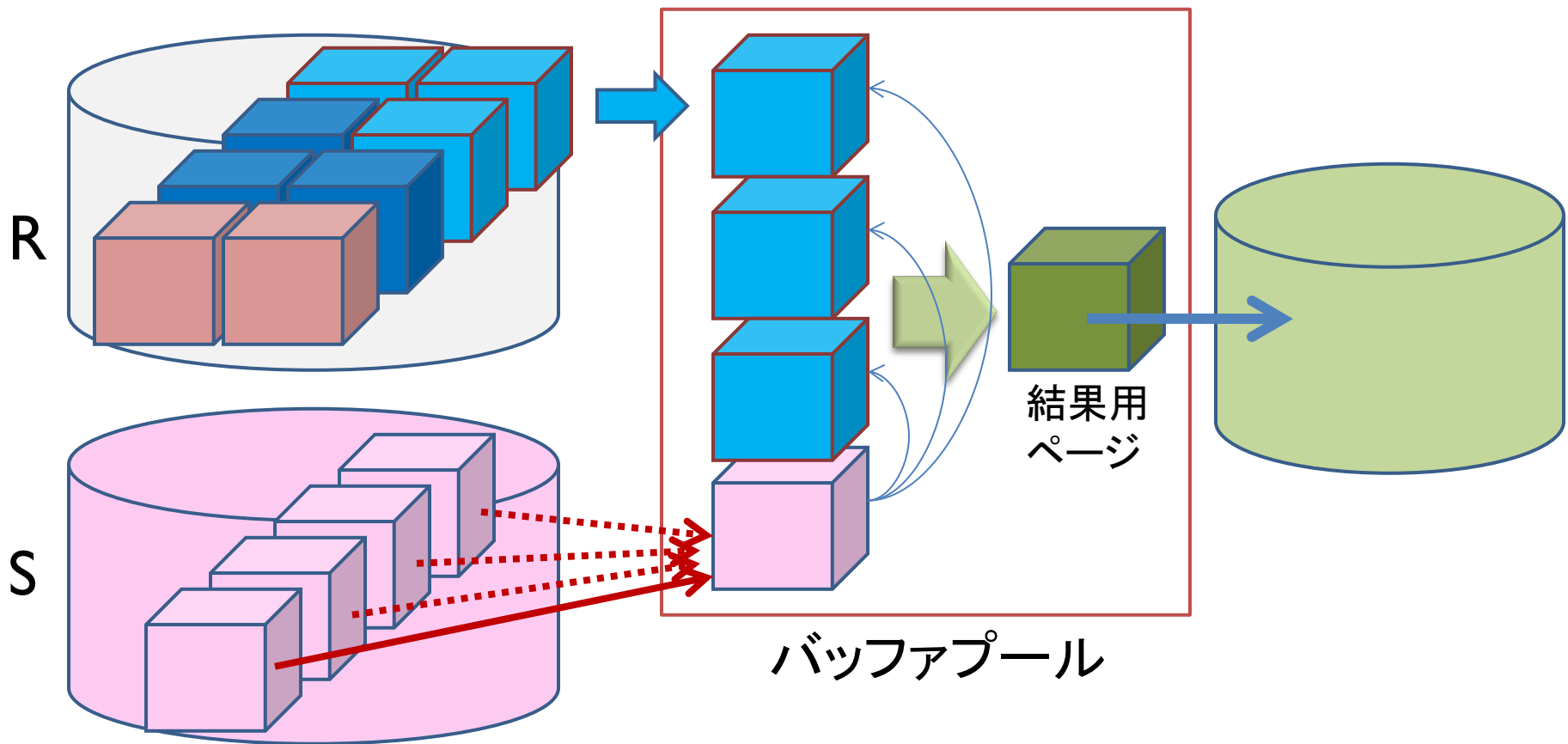
S

sid	bid
22	103
28	101
28	102
31	102



ブロック入れ子ループ結合

- ▶ 共有メモリ内のバッファプールを利用する
 - ▶ Rのページをブロックに分け、ブロック毎バッファプールへ



ブロック入れ子ループ結合

▶ アルゴリズムで書くとこんな感じ

```
Rbuf = load(R, B)
```

```
foreach tuple r ∈ Rbuf do
```

```
  Sbuf = load(S, 1)
```

```
  foreach tuple s ∈ Sbuf do
```

```
    if ri==bj then add <r,s> to result
```

load(R,B)

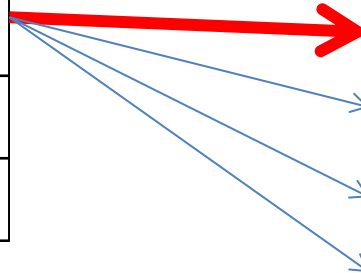
RからBページ分ストレージ
から取得しバッファプールに置く

R

sid	sname
22	dustin
28	yuppy
31	lubber

S

sid	bid
22	103
28	101
28	102
31	102



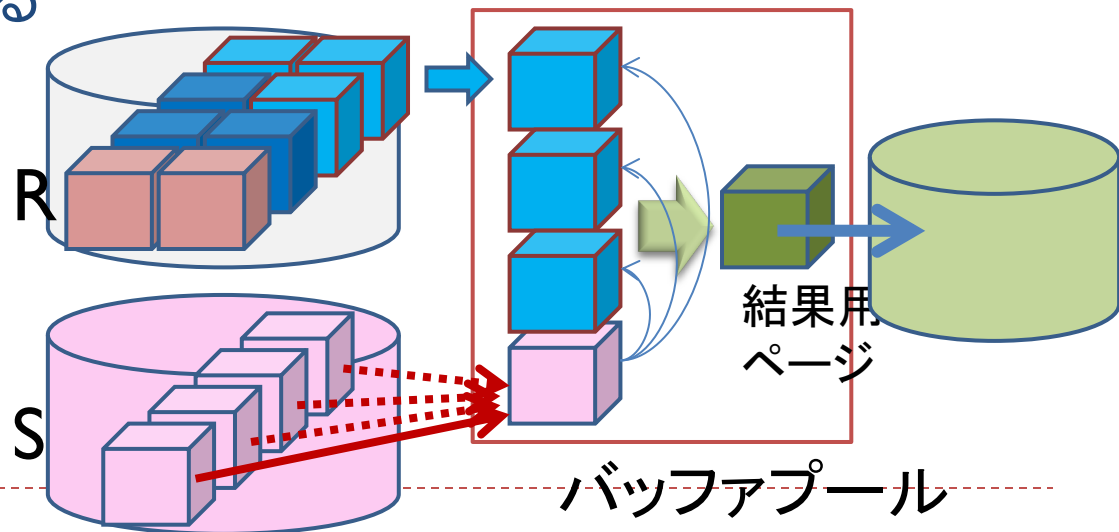
IOコストを考えよう

▶ ブロック型入れ子ループ結合のIOコスト

$$M + N \lceil M/B \rceil$$

▶ 理由

- ▶ Rの1ブロック分を結合するためにSの全ページ(Nページ)を読みだす
- ▶ Rは $\lceil M/B \rceil$ ブロックある
- ▶ さらにRのMページが全部読み出されている



課題 1：ブロック型入れ子結合のコスト

- ▶ テーブルPが 1000 ページ, テーブルQが 500ページ
あるとします。バッファプールは22ページ確保できます。
これらのテーブルを以下の条件でブロック型入れ子結合
処理したときのIOコストを求めましょう
- ▶ (1) テーブルPをループの外側 (outer loop)
 テーブルQをループの内側 (inner loop) にした時
 - ▶ $1000 + 1000 * 500 / 20 = 26000$
- ▶ (2) テーブルPをループの内側, Qを外側にした時
 - ▶ $500 + 500 * 1000 / 20 = 25500$
- ▶ (3) 大きさが異なるテーブルを入れ子結合した時に
 どちらをループの外側にすると良いでしょうか？



ソートマージ結合

- ▶ RとSを値の小さい順にソートする
- ▶ TrとGsが等しくなるまで, TrとGsで値の小さい方のポインタを下に移動する

R

Tr

	sid	sname
r1	22	dustin
r2	28	yuppy
r3	28	Lubber
r4	32	Adam

S

	sid	bid
s1	22	103
s2	28	101
s3	28	102
s4	31	102

Gs

Ts

結合をチェックするレコードのペア

(r1, s1) → 結合

Tr=Gsなら繰り返し

ソートマージ結合

- ▶ TsをGsの位置からTrと同じ値がある分だけ下に動かしながら結合する
- ▶ $Tr < Ts$ になったら, Trを一つ下にずらす

R

	sid	sname
r1	22	dustin
r2	28	yuppy
r3	28	Lubber
r4	32	Adam

Tr

S

	sid	bid
s1	22	103
s2	28	101
s3	28	102
s4	31	102

Tr>Gsなら前頁へ

Gs

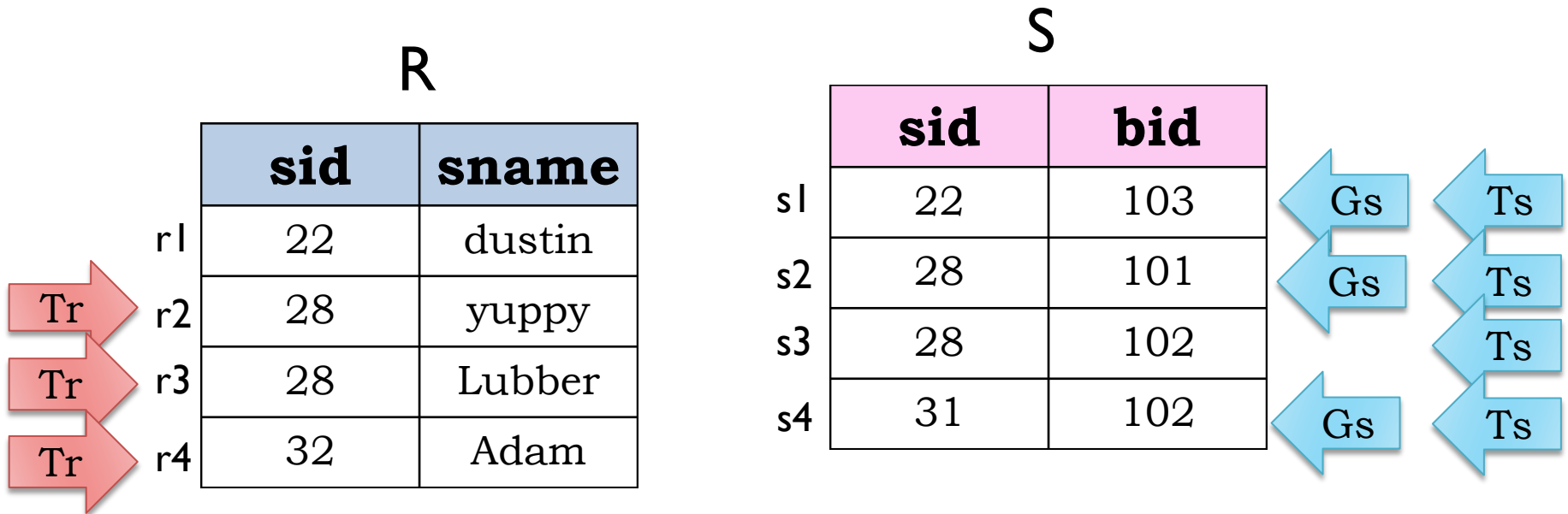
Ts

結合をチェックするレコードのペア

$(r1, s1) \rightarrow \text{結合}, (r1, s2) \rightarrow \times,$

ソートマージ結合

- ▶ 続き
- ▶ どちらかが最後に行きつくまでやる



結合をチェックするレコードのペア

$(r1, s1) \rightarrow \text{結合}$, $(r1, s2) \rightarrow \times$, $(r2, s1) \rightarrow \times$, $(r2, s2) \rightarrow \bigcirc$, $(r2, s3) \rightarrow \bigcirc$
 , $(r2, s4) \rightarrow \times$, $(r3, s2) \rightarrow \bigcirc$, $(r3, s3) \rightarrow \bigcirc$, $(r3, s4) \rightarrow \times$

課題 2

- ▶ Sort Merge Joinのアルゴリズムの下にある二つのテーブル (Figure 14.9, 14.10) を属性sidでソートマージ結合した時に、結合をチェックするレコードのペアを列挙しましょう。
 - ▶ 左側のテーブルのレコードに r_1, \dots, r_6 とレコードIDを振ってください
 - ▶ 右側のテーブルのレコードに s_1, \dots, s_6 とレコードIDを振ってください



ソートマージ結合のコスト

- ▶ 二つのテーブルのソート
 - ▶ $2M\log M + 2N\log N$
- ▶ 二つのテーブルのマージ
 - ▶ $M + N$
 - ▶ 途中ちょっと戻るところはメモリ上の操作なのでIOコストには影響がない
 - ▶ 結合結果の書き込みはRとSの内容によるので計算結果には入れない
- ▶ $M=1000, N=500, \log M, \log N \doteq 2$ のとき
$$2*1000*2 + 2*500*2 + 1000 + 500 = 7500$$
- ▶
$$2*1000*3 + 2*500*3 + 1000 + 500 = 10500$$



ハッシュ結合

基本的なハッシュ結合

結果

22	103	dustin
----	-----	--------

R

sid	sname
22	dustin
28	yuppy
31	lubber

h(sid)

S

sid	bid
22	103
28	101
28	102
31	102

h(sid)

ハッシュテーブル

01	22	dustin	
02			
03	31	lubber	
04	28	yuppy	

ハッシュ結合

▶ 基本的なハッシュ結合のコスト

$$M+N$$

▶ 問題点

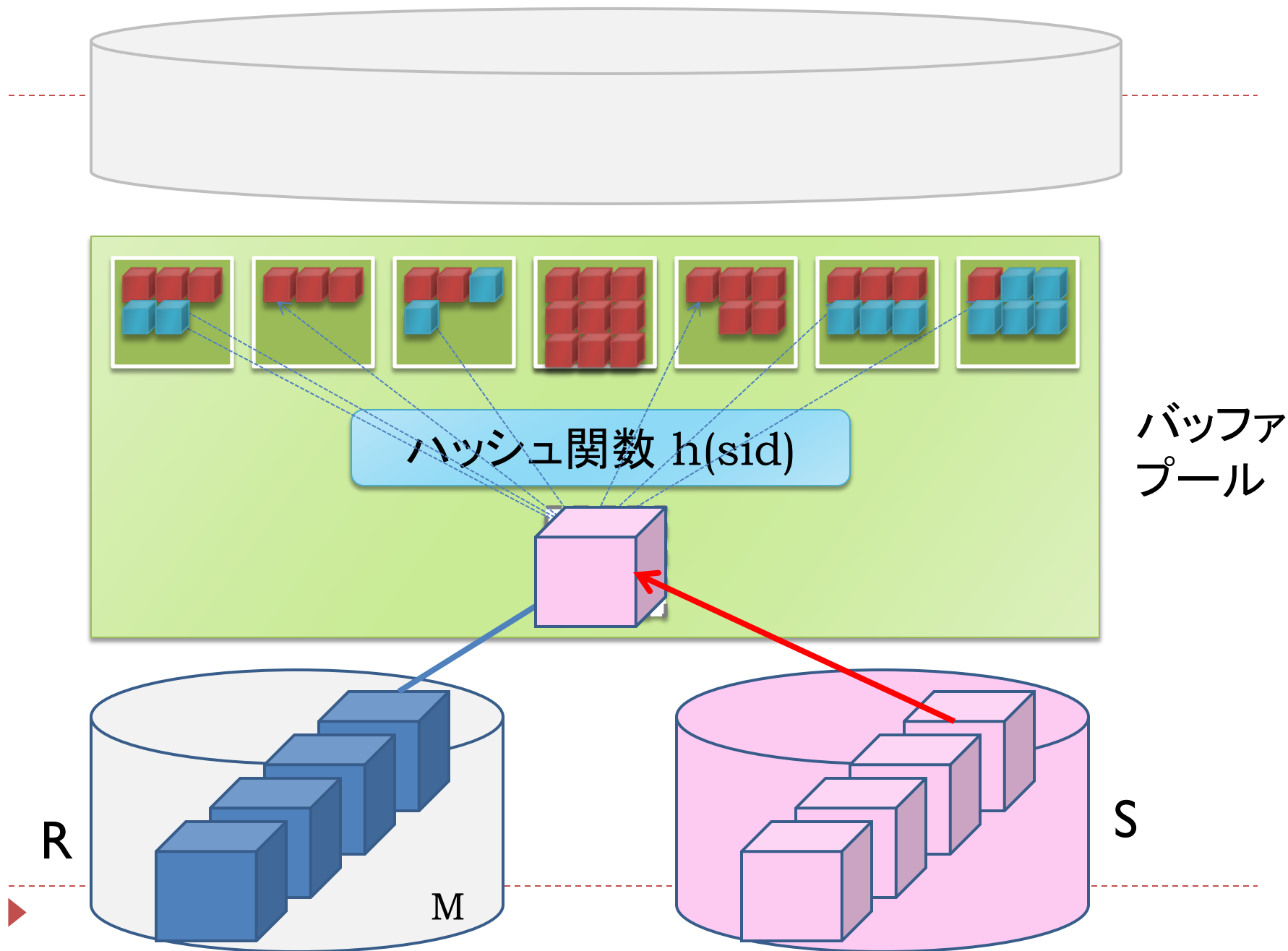
- ▶ テーブルRのハッシュテーブルがメモリに乗りきらないくらい大きかったら対処できない

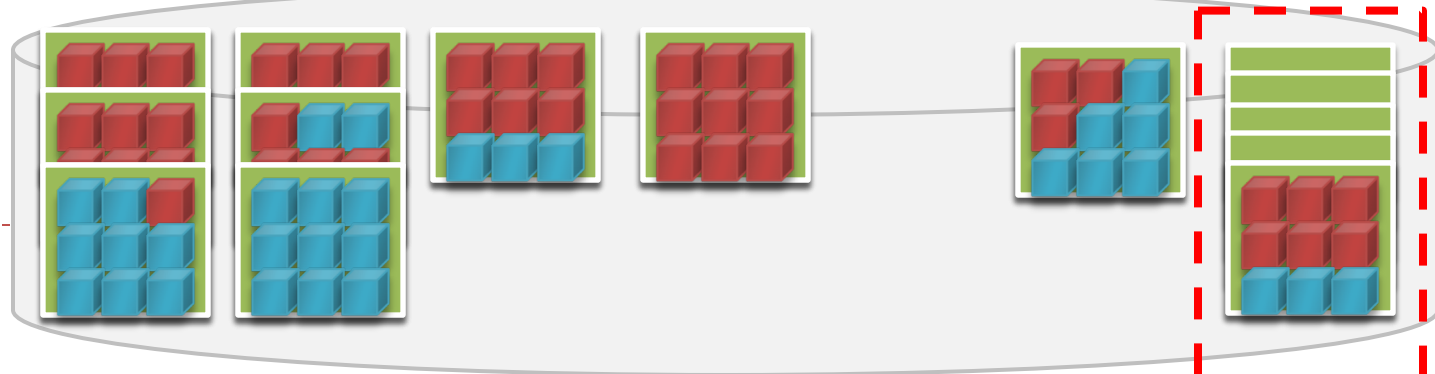


さまざまな手法(ハイブリッドハッシュ結合など)が提案されている
→ 代表的なアルゴリズムとしてグレースハッシュ結合を紹介

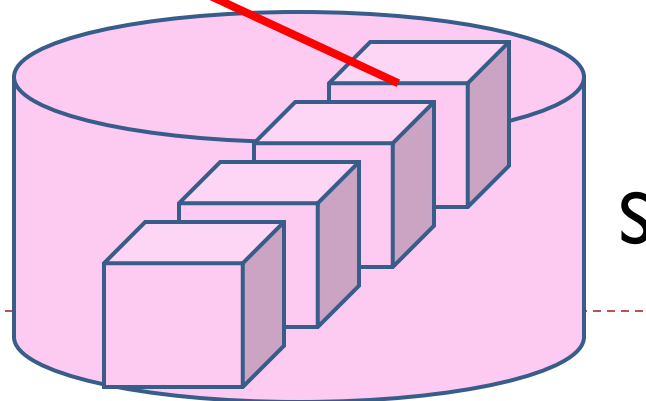
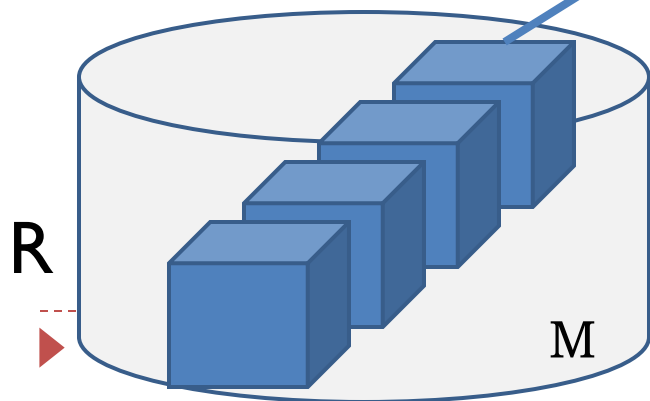
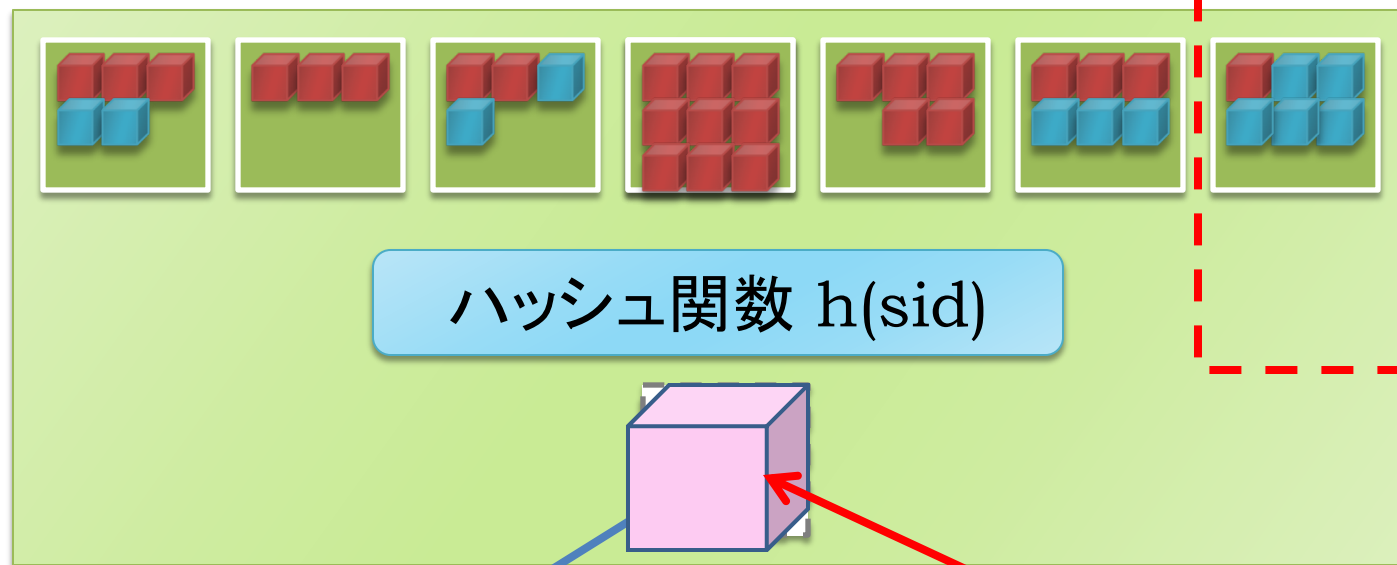


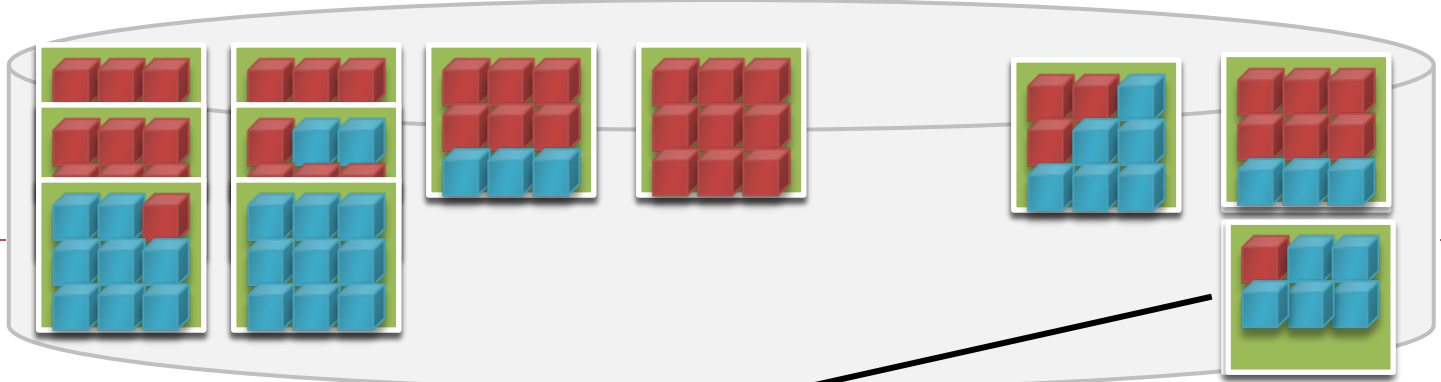
グレースハッシュ結合



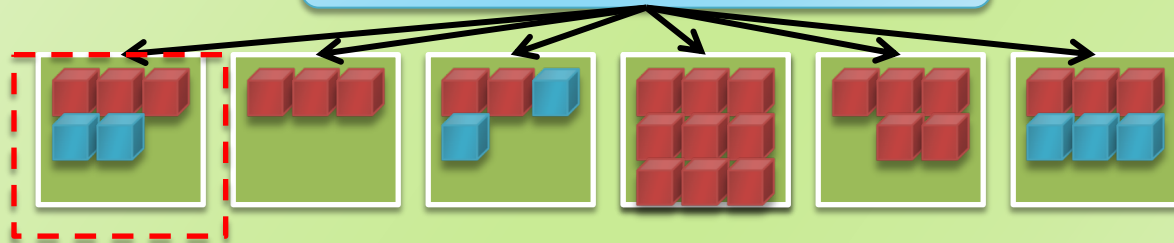


同じブロック
に所属する
レコードが
結合される
ことになる！





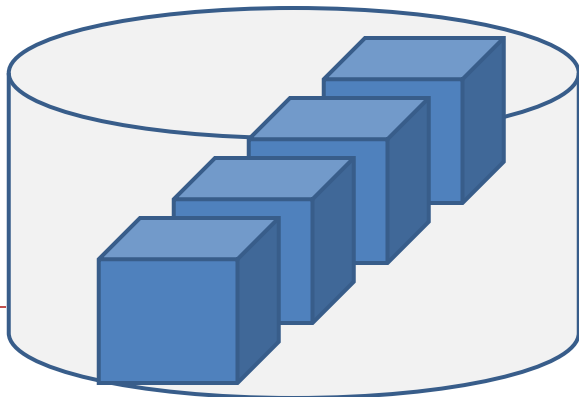
ハッシュ関数 $h2(sid)$



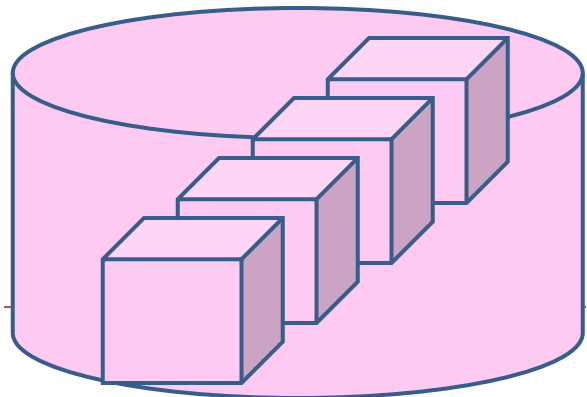
同じページ内に
あるレコードの組合せを
チェックして結合する



R



S



```

// Partition  $R$  into  $k$  partitions
foreach tuple  $r \in R$  do
    read  $r$  and add it to buffer page  $h(r_i)$ ;           // flushed as page fill-

// Partition  $S$  into  $k$  partitions
foreach tuple  $s \in S$  do
    read  $s$  and add it to buffer page  $h(s_j)$ ;           // flushed as page fill-

// Probing phase
for  $l = 1, \dots, k$  do {

    // Build in-memory hash table for  $R_l$ , using  $h_2$ 
    foreach tuple  $r \in$  partition  $R_l$  do
        read  $r$  and insert into hash table using  $h_2(r_i)$  ;

    // Scan  $S_l$  and probe for matching  $R_l$  tuples
    foreach tuple  $s \in$  partition  $S_l$  do {
        read  $s$  and probe table using  $h_2(s_j)$ ;
        for matching  $R$  tuples  $r$ , output  $\langle r, s \rangle$  };

    clear hash table to prepare for next partition;
}

```