www データベース技術　森下真一

参考教科書

H. Garcia-Morina, J. D. Ullman, and J. Widom.
Database System Implementation.
Prentice Hall, 2000, ISBN 0-13-040264-8
Chapter 6, 7

Jeffrey D. Ullman: Principles of Database and Knowledge-base Systems
Volume I, Computer Science Press, 1988. ISBN 0-7167-8158-1
Chapter 9　Transaction Management

講義　URL

http://mlab.cb.k.u-tokyo.ac.jp/~moris/lecture/index.htm

予定

- 11/5, 12, 19, 26
  毎回レポート課題を出します

- レポート〆切日　2007年12月14日（厳守）
  提出先　斉藤助教
  leo@cb.k.u-tokyo.ac.jp

問合せの代数

関係

属性　　例　　name, department,　weight

レコード　　属性の値の線形リスト
(Tom, IS)　(Paul, Physics)

関係　　レコードの集合
R(name, department)

R(name, department)={ (Tom,IS), (John,IS),(Paul,Physics) }

name	department
Tom	IS
John	IS
Paul	Physics

レコード t=(Tom, IS)

t[name] = Tom
t[name,deparment] = (Tom,IS)

Bags と 集合 意味論

Bags 意味論

name	score
Tom	32
Tom	29
Tom	32
John	45
John	48

集合意味論

name	score
Tom	32
Tom	29
John	45
John	48

{ (Tom,32), (Tom,29), (Tom,32), (John,45), (John48) }

問合せの代数

- 和　交わり　差
- 重複削除
- 選択
- 射影
- 積
- ジョイン (Natural, Equi-, Theta-)
- dangling レコード
- セミジョイン　アウタージョイン
- グループ分け

和　交わり　差

おなじ属性をもつ関係 R と S

Bags意味論での定義

和 (Union)	R ∪ S	各レコード r は R と S に出現する 回数の和だけ、r は R ∪ S に出現
交わり (Intersection)	R ∩ S	各レコード r は R と S に出現する回数 で少ない方の回数だけR ∩ S に出現
差 (Difference)	R ― S	各レコード r は R と S に出現する 回数の差だけ R ― S に出現

R={0,1,1,1,3,3}　　S={1,2,2,3,3,3}

R ∪ S = {0,1,1,1,1,2,2,3,3,3,3,3}

R ∩ S = {1,3,3}

R ― S = {0,1,1}

和　交わり　差

Bags 意味論

R={0,1,1,1,3,3}　S={1,2,2,3,3,3}

R ∪ S = {0,1,1,1,1,2,2,3,3,3,3,3}
R ∩ S = {1,3,3}
R ― S = {0,1,1}

集合意味論

R={0,1,3}　S={1,2,3}

R ∪ S = {0,1,2,3}
R ∩ S = {1,3}
R ― S = {0}

重複削除

関係 R 中の重複したレコードを除く演算 $\delta(R)$

R

a	b
1	2
2	3
2	3

$\delta(R)$

a	b
1	2
2	3

SQL では DISTINCT が重複を除去

SQL での UNION, INTERSECT, EXCEPT は集合意味論
R UNION S は $\delta(R \cup S)$ に対応

以降は bags 意味論。 必要におうじて重複除去して集合意味論。

問題

R = { (1,1), (1,1), (1,2), (2,1), (2,1),
(2,2),(2,2),(2,2) }

S = { (1,1), (2,1), (2,1), (2,1), (2,2), (2,2) }

R と S の和、差、交わりを求めよ

得られた和、差、交わりの重複を除去せよ

選択 (Selection)

関係 R から条件 C をみたすレコードを選択　$\sigma_C R$

条件 C の中では、たとえば
(1)　数式演算 (四則など)や文字列処理演算
(2)　比較演算(<,>,等)
(3)　命題論理の結合子(AND, OR, NOT)

a	b
0	1
2	1
2	1
3	1
3	5

$\sigma_{a>1} R(a,b)$

a	b
2	1
2	1
3	1
3	5

$\sigma_{a>1 \text{ AND } \text{NOT}(a+b>5)} R(a,b)$

a	b
2	1
2	1
3	1

射影（Projection）

関係 R のレコードを属性リスト L へ射影　$\pi_L R$

L の属性としては
(1)　関係 R の属性そのもの
(2)　R　の属性から、a+b → x のように属性をつくってもよい

a	b	c
1	2	3
2	1	2
2	1	2
3	2	5
5	0	3

$\pi_{a+b \to x,\ a+c \to y} R(a,b,c)$

x	y
3	4
3	4
5	8
5	8

Bags
意味論

問題

R(a,b,c)

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

以下の演算結果を示せ

$\Sigma_{(a=0 \text{ OR } b=0) \text{ AND } c=0} R(a,b,c)$

$\sigma_{a*b*c=0} R(a,b,c)$

$\sigma_{a+b=1 \text{ OR } b+c=0} R(a,b,c)$

$\sigma_{a+b=2 \text{ OR } b-c=1} R(a,b,c)$

$\sigma_{a*b=0 \text{ OR } b*c=1} R(a,b,c)$

積(Product)

関係 R、S からレコードを選択し連結したレコードの全体　R × S

同一名の属性 a が R と S に現れる場合　R.a, S.a と改名

R

a	b
1	2
2	3
2	3

S

a	c
4	5
5	6

Bags 意味論

R × S

R.a	b	S.a	c
1	2	4	5
1	2	5	6
2	3	4	5
2	3	5	6
2	3	4	5
2	3	5	6

ジョイン(Join)

複数の関係に、積・選択・射影を適用してつくる関係

Natural (Equi-)Join $R \bowtie S = \pi_L(\sigma_C(R \times S))$

条件 C は R と S に共通する属性 a,b,... を等しいとみなす
R.a=S.a AND R.b=S.b AND ...
属性リスト L は a,b,...
属性の重複、たとえば R.a と S.a は一つにまとめる

R

a	b
1	2
2	3
2	4

S

a	c
1	5
2	6

$R \bowtie S$

a	b	c
1	2	5
2	3	6
2	4	6

Natural Join の例

R

a	b
1	2
2	3
2	4

S

a	c
1	5
2	6

R × S

R.a	b	S.a	c
1	2	1	5
1	2	2	6
2	3	1	5
2	3	2	6
2	4	1	5
2	4	2	6

$R \bowtie S = \pi_{R.a \to a,b,c}(\sigma_{R.a=S.a}(R \times S))$

a	b	c
1	2	5
2	3	6
2	4	6

$\sigma_{R.a=S.a}(R \times S)$

R.a	b	S.a	c
1	2	1	5
2	3	2	6
2	4	2	6

問題

R(a,b,c)

a	b	c
1	1	1
1	2	1
2	1	2
2	2	2

R(c,d,f)

c	d	f
1	1	1
1	2	1
2	2	2
2	2	2

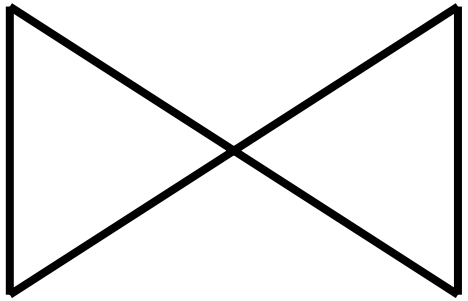
R(f,g,a)

f	g	a
1	2	2
1	2	2
2	1	1
2	2	1

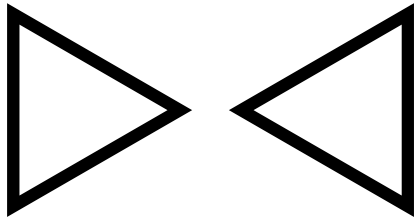
以下の演算結果をしめせ

R(a,b,c) ▷◁ R(c,d,f)
R(c,d,f) ▷◁ R(f,g,a)
(R(a,b,c) ▷◁ R(c,d,f)) ▷◁ R(f,g,a)

ジョイン記号の代用



正しい ジョイン記号



Windings 3 の wv で代用

Theta Join

一般的なジョイン

$$\text{Theta-Join}\quad R \ \triangleright\!\triangleleft_C \ S = \sigma_C(R \times S)$$

関係 R × S の属性にたいする条件 C をつくる

研究当初は x θ y（ただし θ = >, <, ...) の条件だけを

扱ったので Theta-（θ）とよばれる

Theta Join の例

R

a	b
1	2
2	3
2	4

S

a	c
1	5
2	6

R × S

R.a	b	S.a	c
1	2	1	5
1	2	2	6
2	3	1	5
2	3	2	6
2	4	1	5
2	4	2	6

$\sigma_{b+c>8}(R \times S)$

R.a	b	S.a	c
2	3	2	6
2	4	1	5
2	4	2	6

dangling レコードとジョイン

R

a	b
1	2
2	3
2	4

S

a	c
3	5
2	6

R ▷◁ S

a	b	c
2	3	6
2	4	6

R の (1,2) は S の中で属性 a が一致するレコードがない。
dangling レコードと呼ぶ。　他の例 (3,5) ∈ S。

R から dangling レコードを除く操作　　セミジョイン
R ▷◁ S に dangling レコードを残す操作　　アウタージョイン

dangling レコードの例

遺伝子観測量のデータ
微量に出現しているため観測がむずかしい遺伝子が多い

R

遺伝子ID	肝臓
1	0.5
3	0.2
4	0.3

S

遺伝子ID	肺
2	0.6
3	0.2
5	0.4

R ▷◁ S

遺伝子

ID	肝臓	肺
3	0.2	0.2

問題

R(a,b,c)

a	b	c
1	1	1
1	2	1
2	1	2
2	2	2

R(c,d,f)

c	d	f
1	1	1
1	2	1
2	1	2
2	2	2

R(f,g,a)

f	g	a
1	1	2
1	2	2
2	1	1
2	2	1

以下の2つの関係の間では、どれが dangling レコードであるか？

(R(a,b,c) ▷◁ R(c,d,f)) と R(f,g,a)

セミジョイン（semi-join）

R ▷◁ S で分かる R の dangling レコードを除く操作

$$R \ltimes S = \pi_L (R \triangleright\triangleleft S)$$

L は R の属性のリスト

R

a	b
1	2
2	3
2	4

S

a	c
3	5
2	6

R ▷◁ S

a	b	c
2	3	6
2	4	6

R ⋉ S

a	b
2	3
2	4

問題

R(a,b,c)

a	b	c
1	1	1
1	2	1
2	1	2
2	2	2

R(c,d,f)

c	d	f
1	1	1
1	2	1
2	1	2
2	2	2

R(f,g,a)

f	g	a
1	1	2
1	2	2
2	1	1
2	2	1

以下のセミジョインを計算せよ

$$(R(a,b,c) \triangleright\!\triangleleft R(c,d,f)) \ltimes R(f,g,a)$$

問題

R(a,b) ⋉ R(b,c)　がbags 意味論と集合意味論では異なる関係を生成するような関係R(a,b) とR(b,c) の例を述べよ。

アウタージョイン（outer-join）

R ▷◁ S に dangling レコードを追加した関係

$R \triangleright\!\triangleleft_{outer} S$

dangling レコードに null 記号（⊥）を補うことで情報の欠落をふせぐ

R

a	b
1	2
2	3
2	4

S

a	c
3	5
2	6

R ▷◁ S

a	b	c
2	3	6
2	4	6

$R \triangleright\!\triangleleft_{outer} S$

a	b	c
2	3	6
2	4	6
1	2	⊥
3	⊥	5

アウタージョインの例

R

遺伝子ID	肝臓
1	0.5
3	0.2
4	0.3

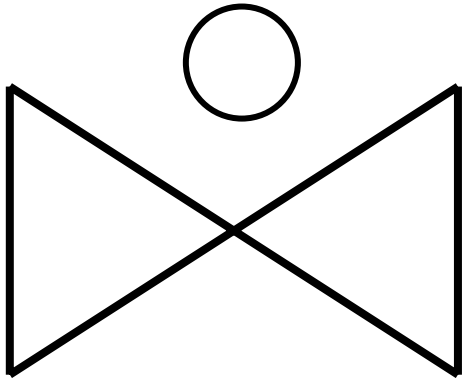
S

遺伝子ID	肺
2	0.6
3	0.2
5	0.4

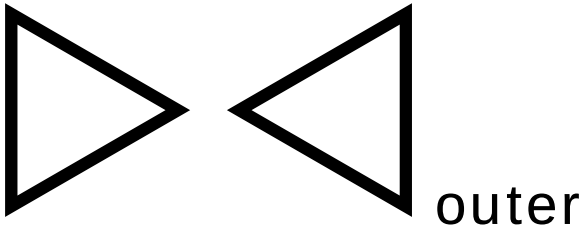
R $\triangleright\triangleleft_{outer}$ S

遺伝子ID	肝臓	肺
1	0.5	⊥
2	⊥	0.6
3	0.2	0.2
4	0.3	⊥
5	⊥	0.4

アウタージョイン記号は…



正しい
アウタージョイン記号



代用品

問題

R(a,c)

a	c
1	1
1	1
2	2

R(c,f)

c	f
1	1
1	1
2	2
2	2

R(f,a)

f	a
1	2
2	1
2	1

上の3つの関係をアウタージョインした結果を求めよ

グループ分け（Grouping）と集約(Aggregation)

３件以上お店のある地域で、一番安いラーメンの価格は？

name	district	price
1	Tokyo	650
2	Tokyo	550
3	Tokyo	750
4	Tokyo	850
5	Yokohama	850
6	Yokohama	550
7	Yokohama	600
8	Chiba	600
9	Chiba	450

SQL文

```
SELECT district, MIN(price) AS min
FROM R
GROUP BY district
HAVING COUNT(name) ≥ 3
```

グループ分けと集約

3件以上お店のある地域で、一番安いラーメンの価格は？

name	district	price
1	Tokyo	650
2	Tokyo	550
3	Tokyo	750
4	Tokyo	850
5	Yokohama	850
6	Yokohama	550
7	Yokohama	600
8	Chiba	600
9	Chiba	450

R1 =

$\gamma_{\text{district,MIN(price)}\rightarrow\text{min,COUNT(name)}\rightarrow\text{count}}\ R$

district	min	count
Tokyo	550	4
Yokohama	550	3
Chiba	450	2

$\pi_{\text{district, min}}(\sigma_{\text{count}\geq 3}\ R1)$

district	min
Tokyo	550
Yokohama	550

グループ分けと集約

$\gamma_L R = \gamma_{\text{district,MIN(price)→min,COUNT(name)→count}} R$

L の元は

- グループ分けの対象となる属性（複数可）

  district と soup（スープの種類）でグループ分けなど

- 属性を引数とする集約演算子

  MIN(price), MAX(price), AVG(price),
  SUM(price), count(name) など

問題

以下の問合せを演算をもちいて表現せよ

3件以下お店のある地域で、一番安いラーメンの価格は？

各地域におけるラーメンの平均価格は？

問合せの代数： ソート

$\tau_{a1,a2,a3,\ldots,an}R$

R の元を属性 a1 の値でソートする。
a1 で同一のレコードは a2 でソートし、an まで使って辞書式順序でソート。

ソートは問合せの実行スピードを上げるのに利用

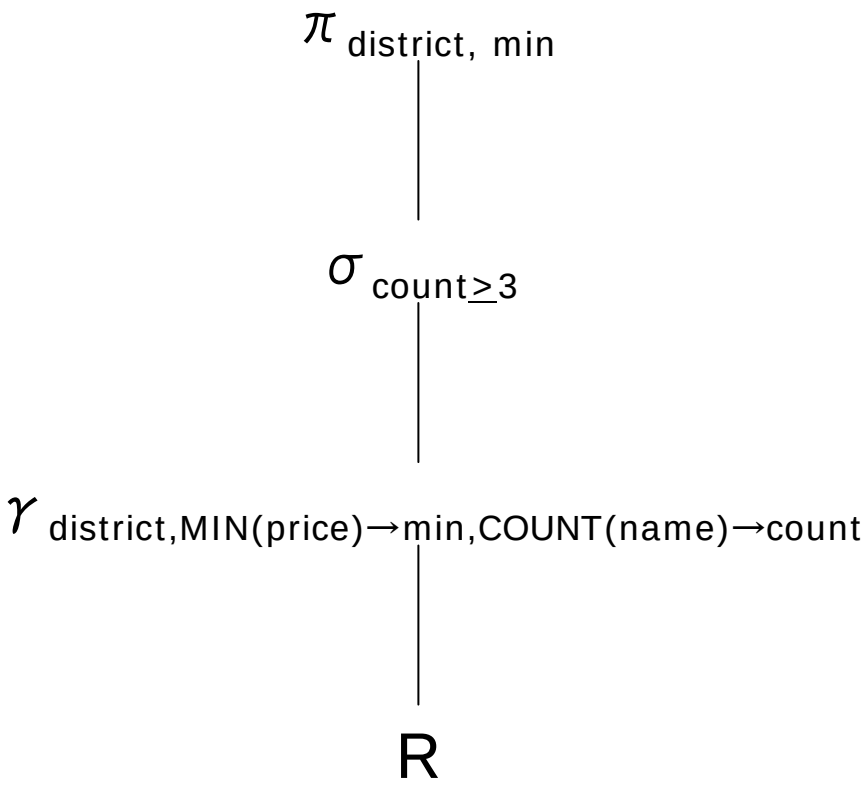
a	b	c
2	1	3
3	1	2
2	1	4
2	0	5

$\tau_{a,b,c}R$

a	b	c
2	0	5
2	1	3
2	1	4
3	1	2

問合せ木(Expression Tree)

$$\pi_{\text{district, min}}(\sigma_{\text{count}\geq 3}(\gamma_{\text{district,MIN(price)}\rightarrow\text{min,COUNT(name)}\rightarrow\text{count}} R))$$



問合せ木

R

a	b
1	2
2	3
2	4

S

a	c
1	5
2	6

$\sigma_{a>1 \text{ AND } c>5} R \triangleright\triangleleft S$

a	b	c
2	3	6
2	4	6

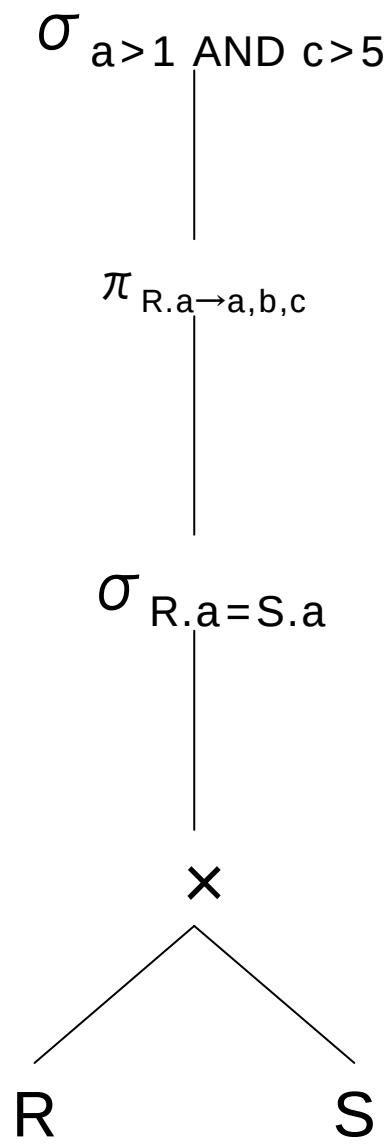
問合せ木

a	b	c
2	3	6
2	4	6

a	b	c
1	2	5
2	3	6
2	4	6

R.a	b	S.a	c
1	2	1	5
1	2	2	6
2	3	1	5
2	3	2	6
2	4	1	5
2	4	2	6

R.a	b	S.a	c
1	2	1	5
2	3	2	6
2	4	2	6



問合せ木の改良

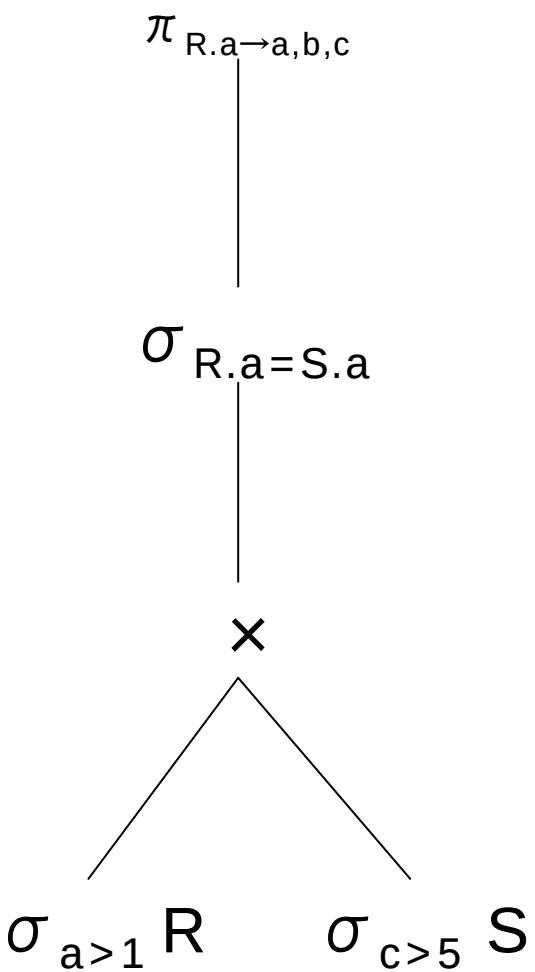
a	b	c
2	3	6
2	4	6

R.a	b	S.a	c
2	3	2	6
2	4	2	6

R.a	b	S.a	c
2	3	2	6
2	4	2	6

a	b
2	3
2	4

a	c
2	6



選択を最初に実行

問合せの代数の実装

講義順序

まず基本演算の実装

$\cup \quad \cap \quad - \quad \delta \quad \sigma \quad \pi \quad \triangleright\triangleleft \quad \gamma$

つづいて演算を複数組み合わせた問合せの実装

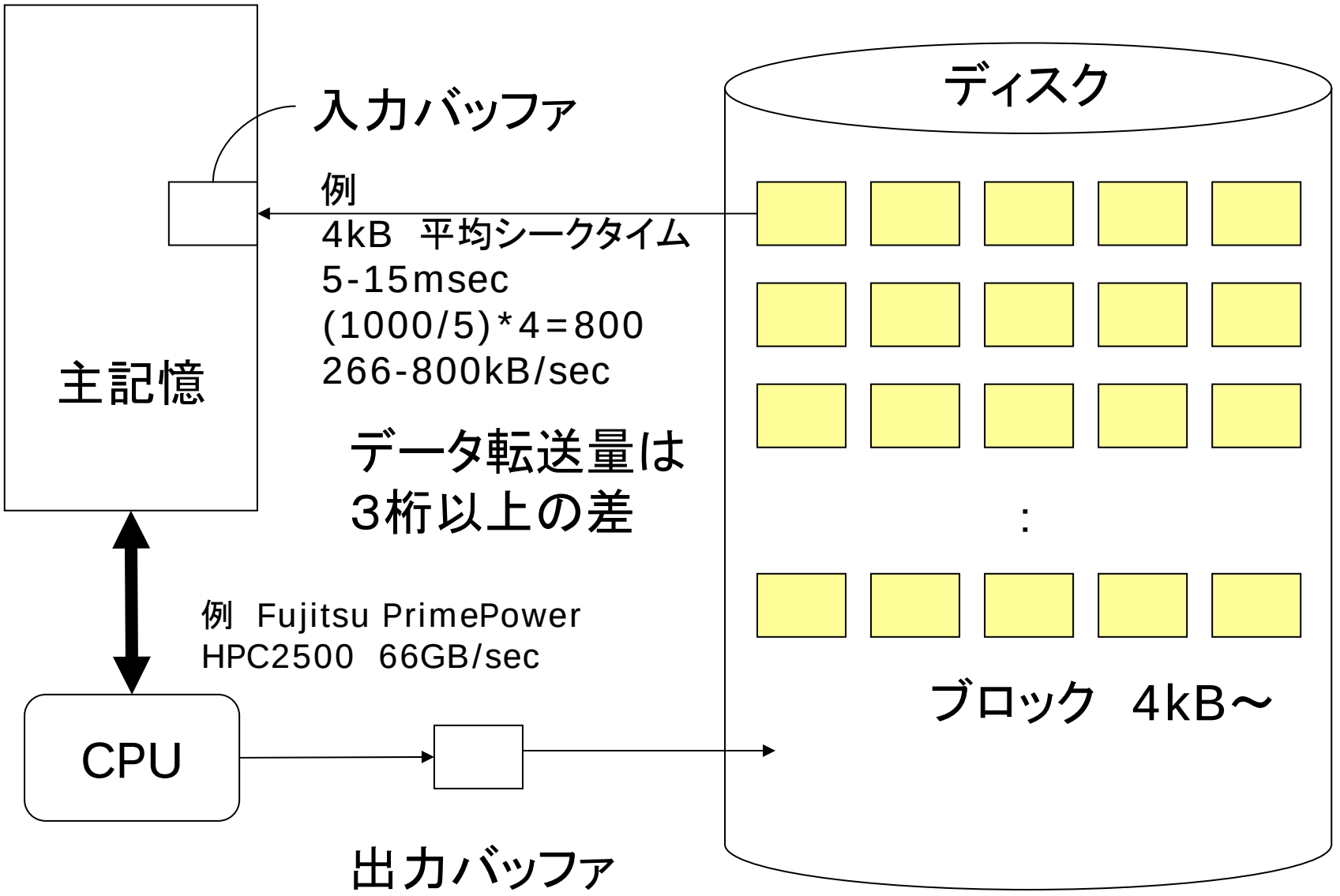
$\sigma_{a>1 \text{ AND } c>5} R \triangleright\triangleleft S$

$R(a,b,c) \triangleright\triangleleft R(b,c,d) \triangleright\triangleleft R(b,d,e) \triangleright\triangleleft R(c,d,f)$

$R(a,b) \triangleright\triangleleft R(b,c) \triangleright\triangleleft R(c,d) \triangleright\triangleleft R(d,a)$

$\pi_{a,e,f}(R(a,b,c) \triangleright\triangleleft R(b,c,d) \triangleright\triangleleft R(b,d,e) \triangleright\triangleleft R(c,d,f))$

ディスク　ブロック　バッファ　データ転送ボトルネック



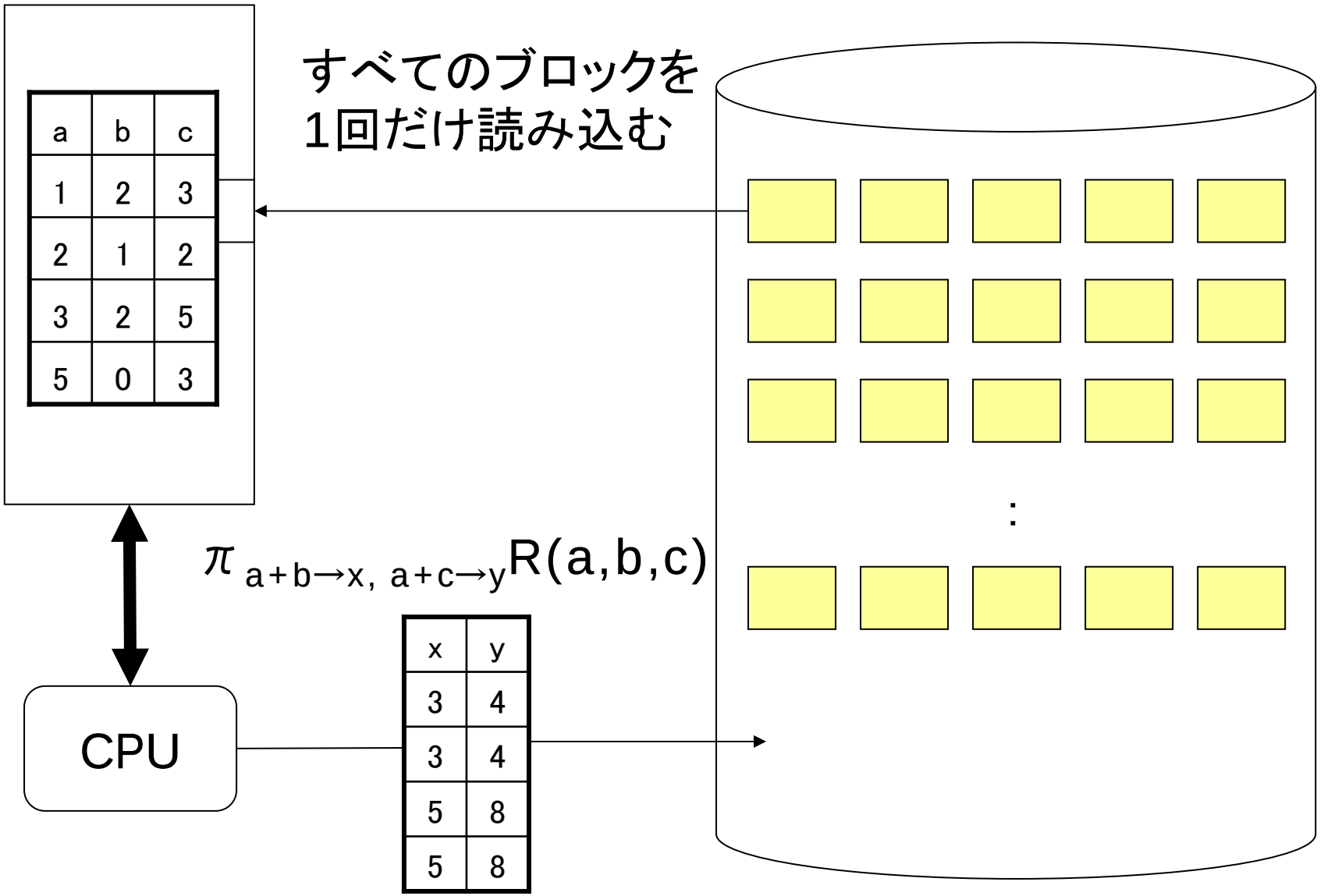
実装の方針

データベース(DB)読み込み回数をへらしたい

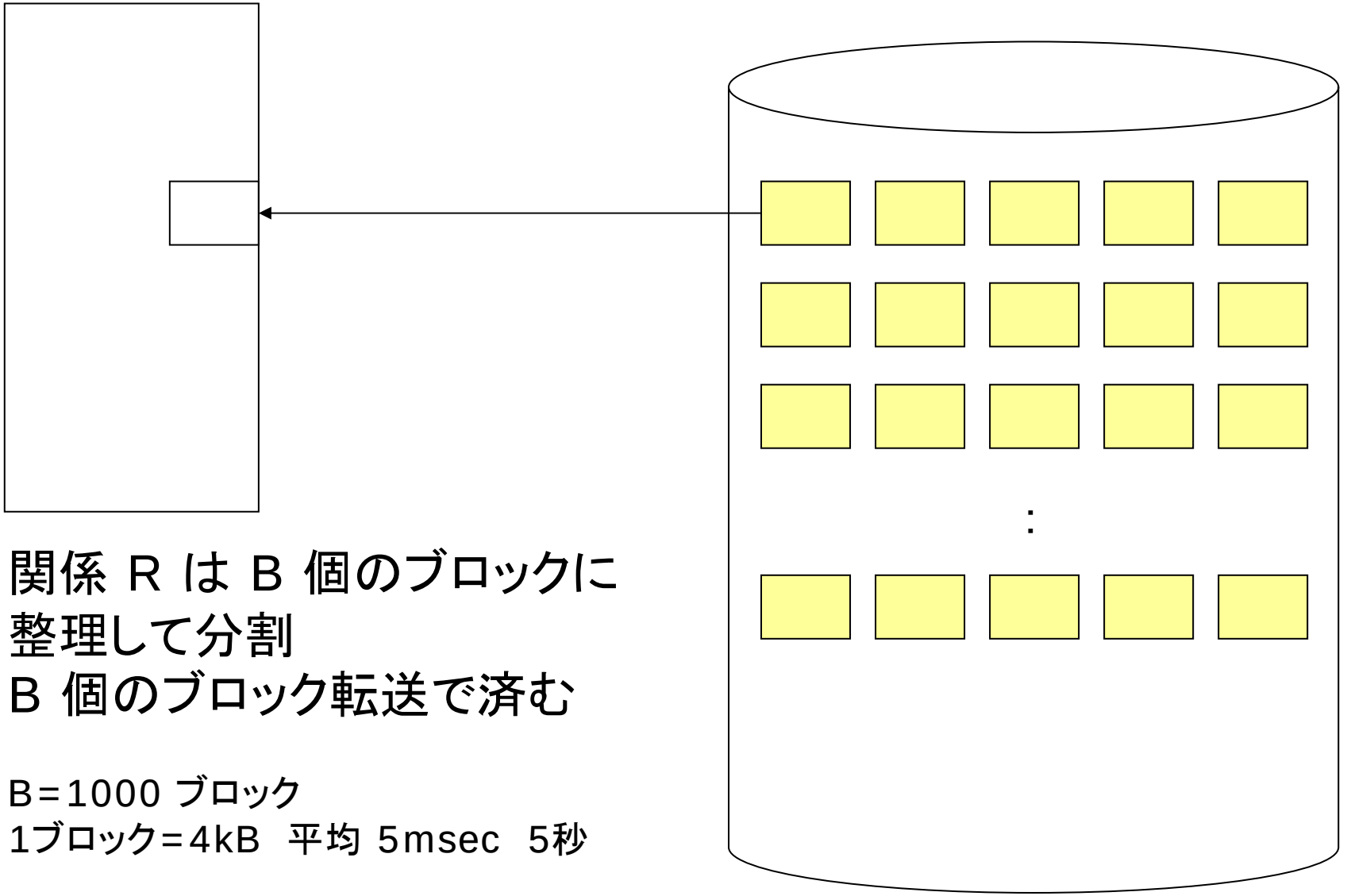
- 1回だけDBを走査すればよい容易な場合あり
- 1回目でデータを高速に処理できるように前処理
  - データベースをソート
  - ハッシュ表を生成
  - インデックスを生成

  2回目の読み込みで結果を書き出す

DBは1回だけ走査　選択σ　射影π



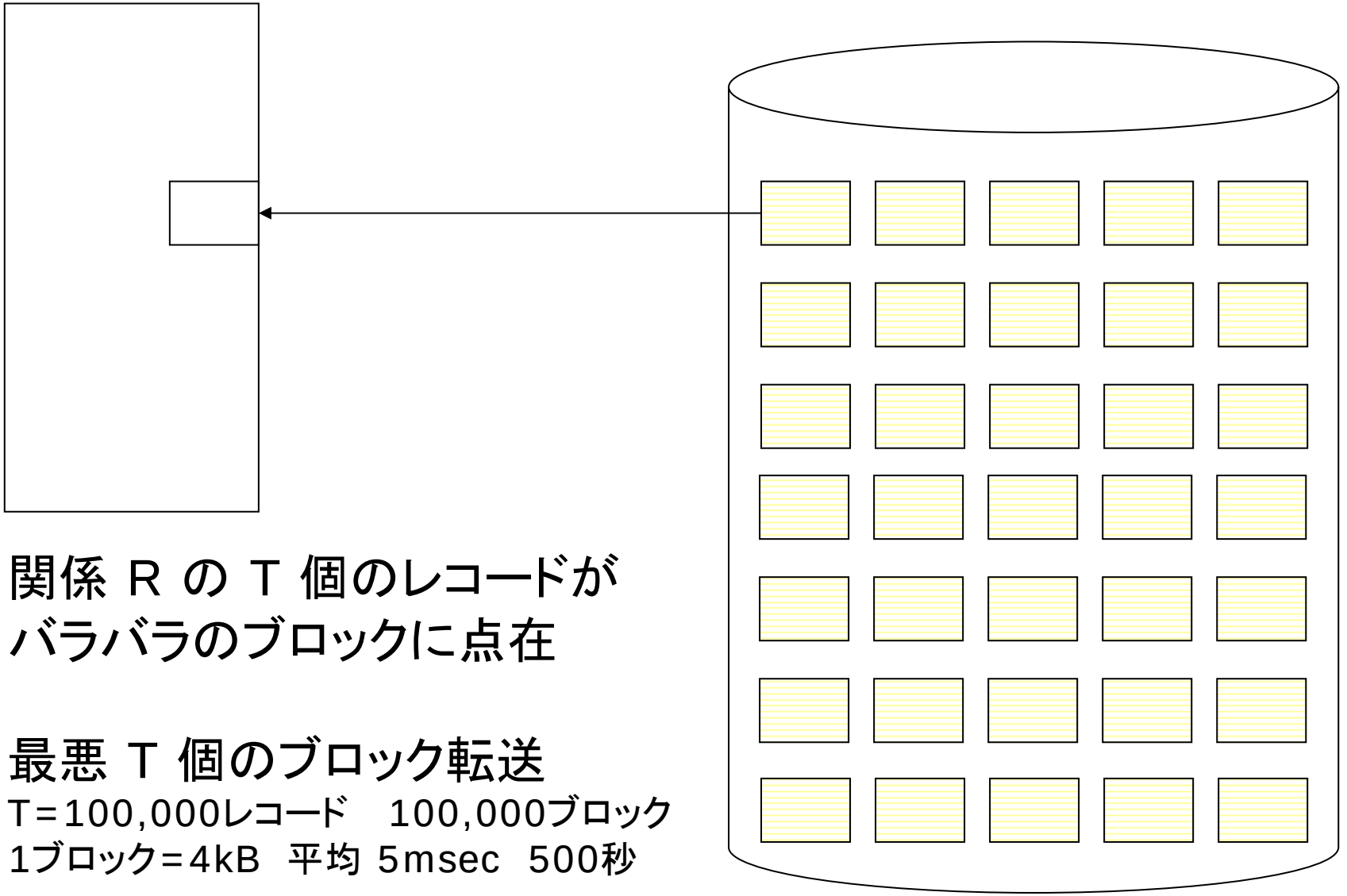
DBは1回だけ走査　選択σ　射影π



関係 R は B 個のブロックに
整理して分割
B 個のブロック転送で済む

B＝1000 ブロック
1ブロック＝4kB　平均 5msec　5秒

DBは1回だけ走査　選択σ　射影π



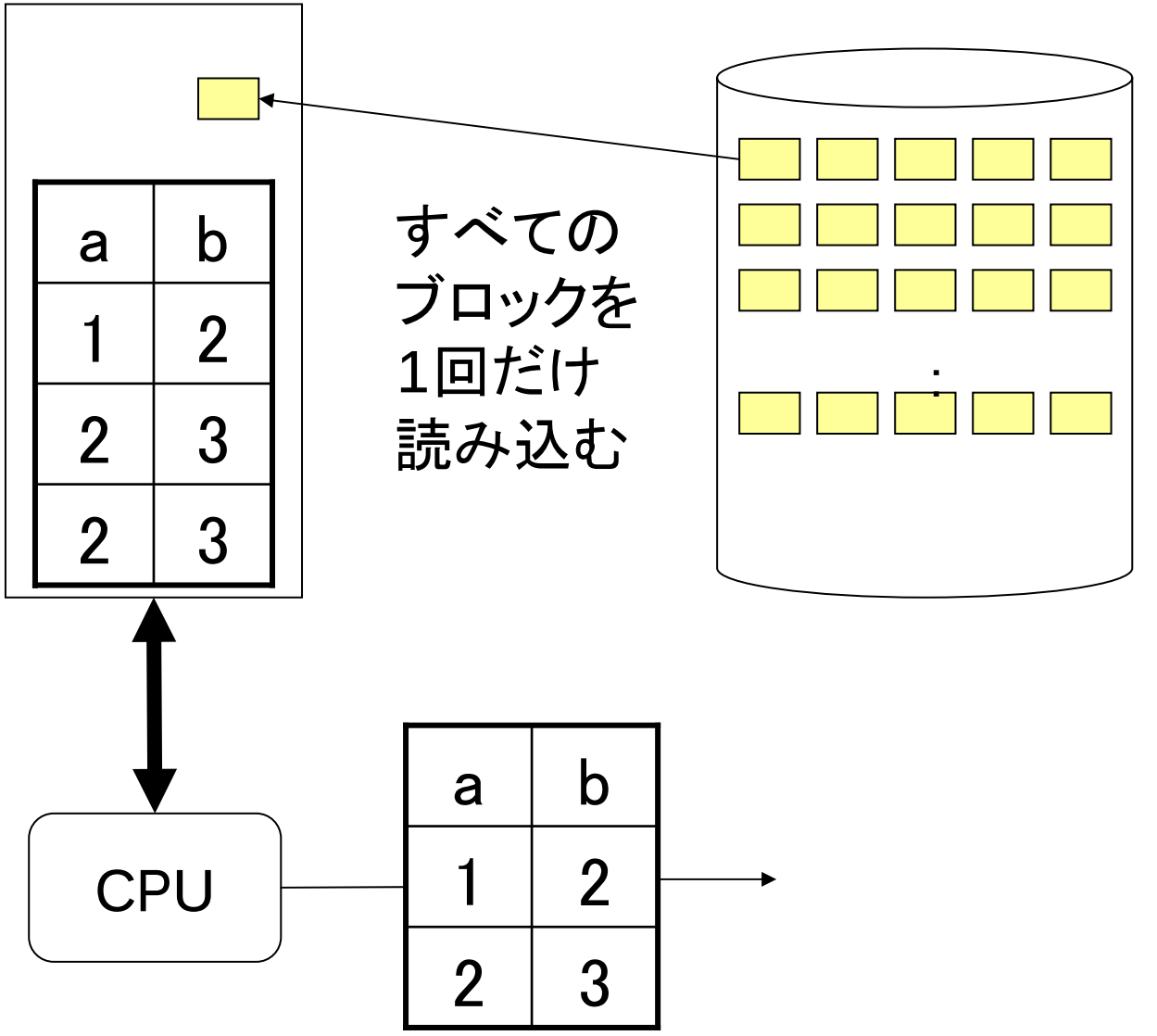
関係 R の T 個のレコードが
バラバラのブロックに点在

最悪 T 個のブロック転送
T＝100,000レコード　100,000ブロック
1ブロック＝4kB　平均 5msec　500秒

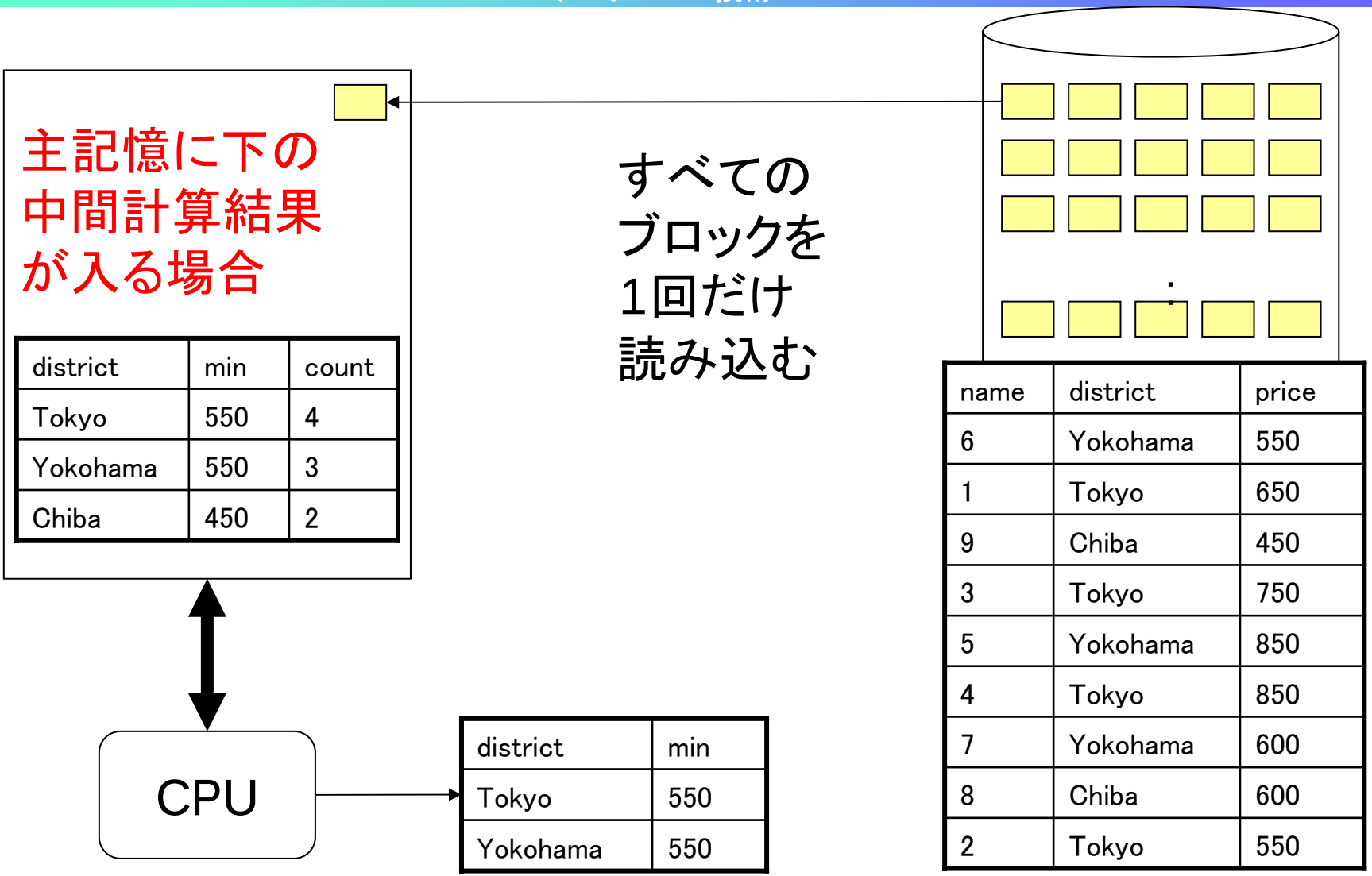
DBは1回だけ走査　重複削除　δ

十分な主記憶があり
既に読んだレコードか
否かの判定を高速に
実行できる場合

ハッシュ表や
平衡2分木で
管理できる



DBは1回だけ走査　集約δ



district	min	count
Tokyo	550	4
Yokohama	550	3
Chiba	450	2

district	min
Tokyo	550
Yokohama	550

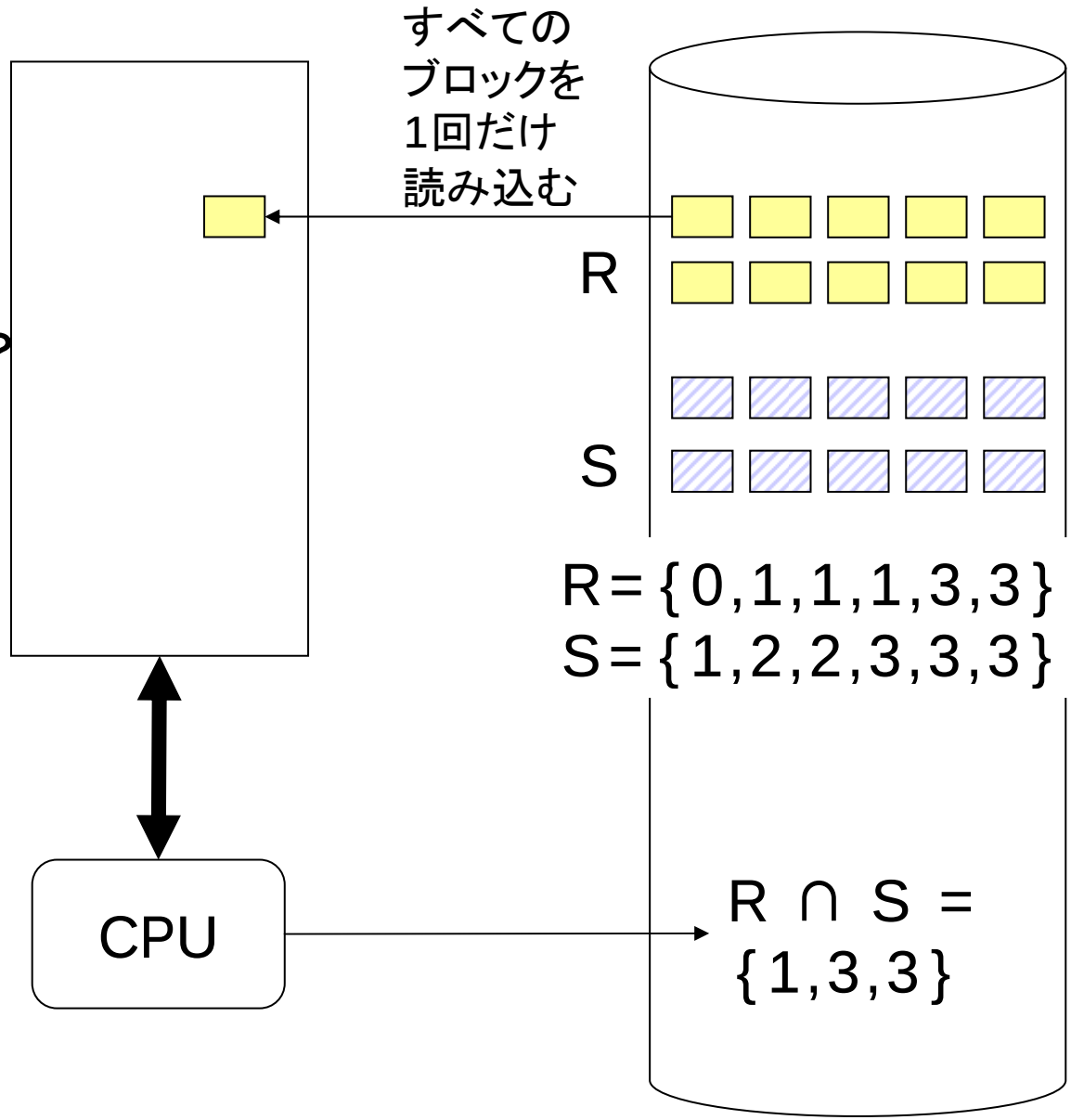
name	district	price
6	Yokohama	550
1	Tokyo	650
9	Chiba	450
3	Tokyo	750
5	Yokohama	850
4	Tokyo	850
7	Yokohama	600
8	Chiba	600
2	Tokyo	550

$$\pi_{\text{district, min}}(\sigma_{\text{count}>2}(\gamma_{\text{district,MIN(price)}\rightarrow\text{min,COUNT(name)}\rightarrow\text{count}} R))$$

DBは1回だけ走査　和　交わり　差

R ∪ S はブロックを
コピーすればよい

R ∩ S、 R − S は？



Rのレコードの頻度表
が入る場合

レコード	R での頻度	S での頻度
0	1	0
1	3	1
3	2	3

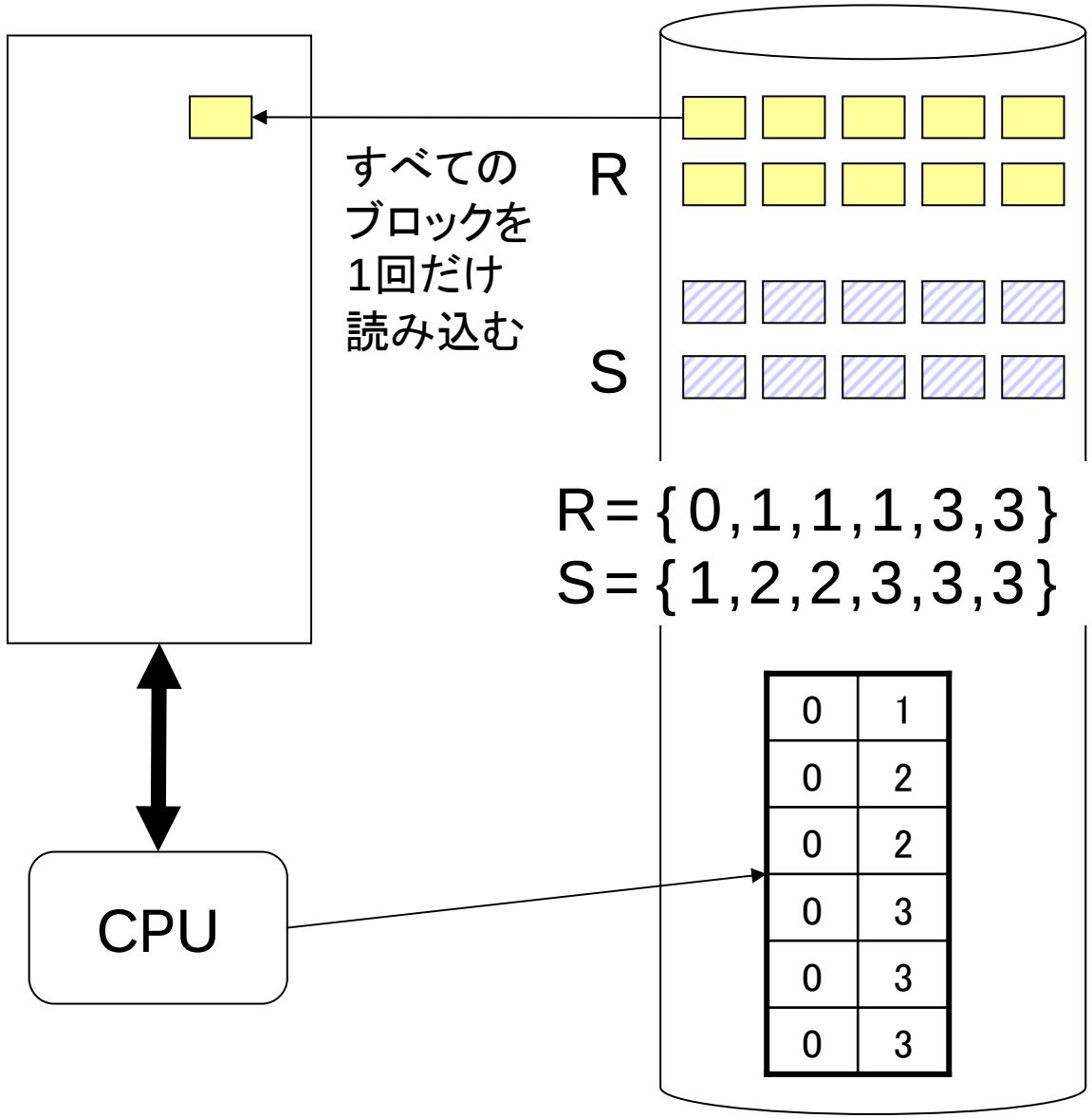
DBは1回だけ走査　積

R もしくは S が
主記憶に入る場合

R が入る場合、まず
R を主記憶に格納

R={0,1,1,1,3,3}

Sの各レコードに対し
Rの各レコードを連結
すればよい

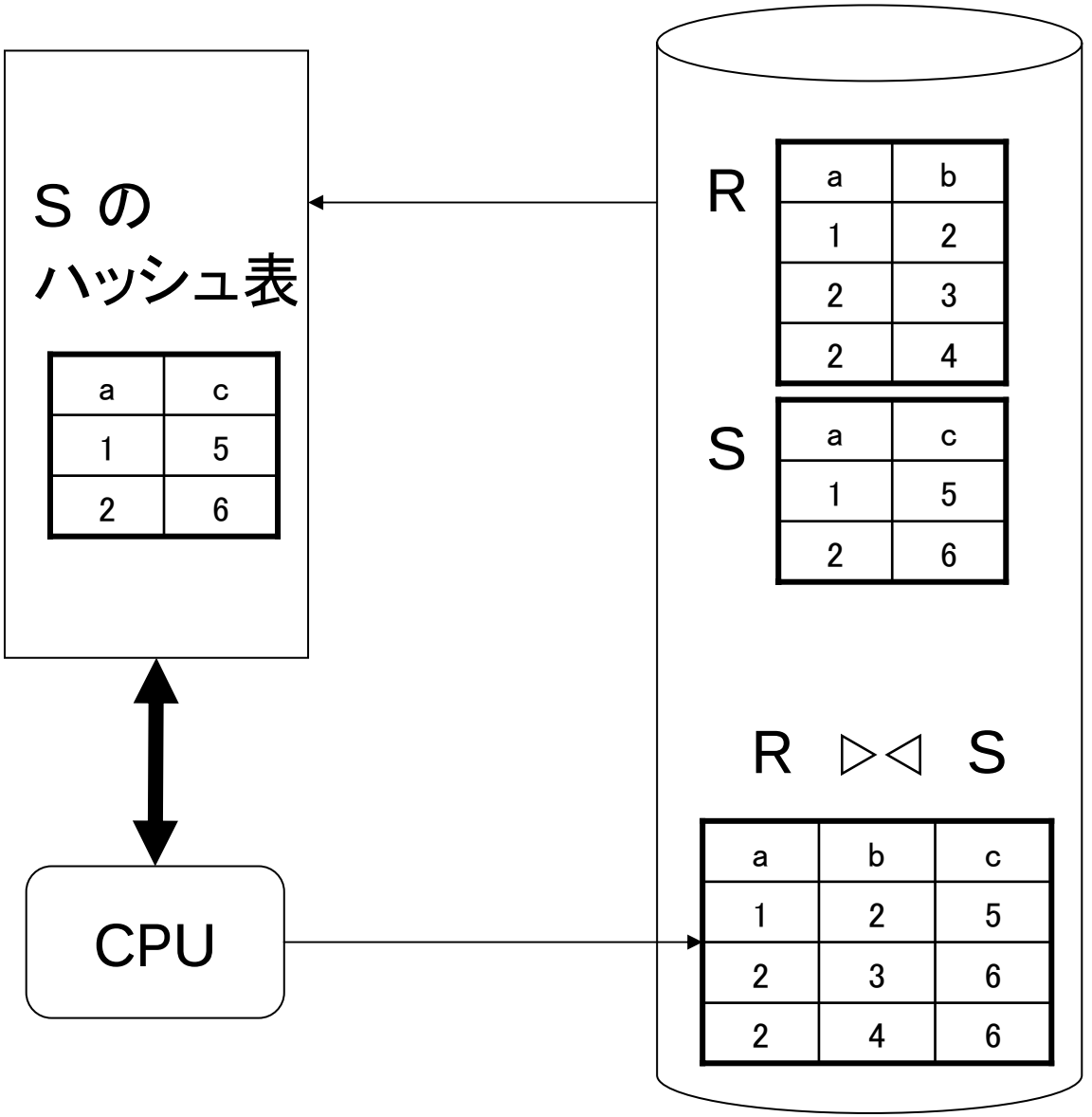


DBは1回だけ走査　Natural Join

R か S をすべて
主記憶に格納でき、
属性 a の値をもつ
レコードを高速に
検索できる場合

ハッシュ表や平衡2分木

R の各レコードを読み、
ハッシュ表を参照

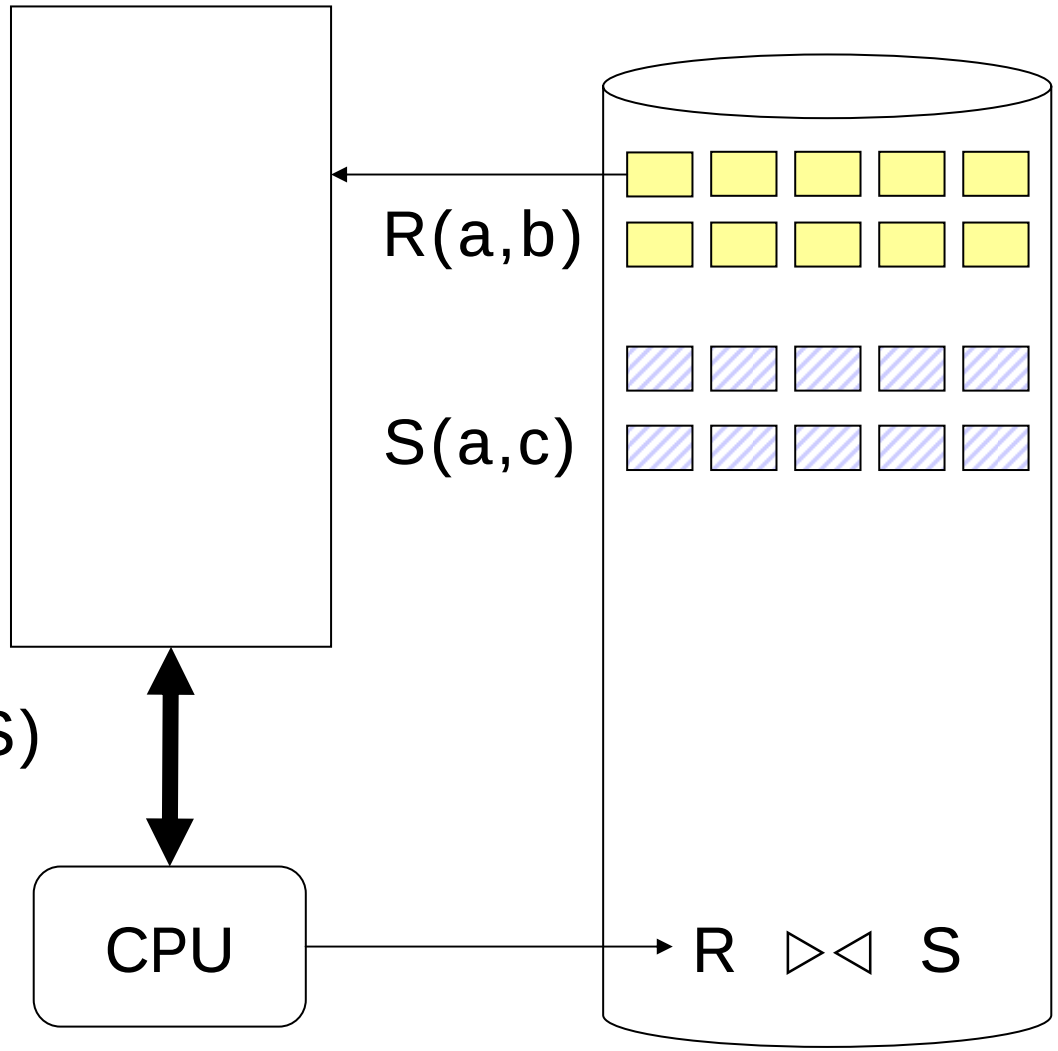


Nested-Loop Join

R、S どちらも主記憶に
格納できない場合

```
for s ∈ S do
  for r ∈ R do
    r と s が属性 a で一致
    するならば、r と s を
    融合したレコードを出力
```

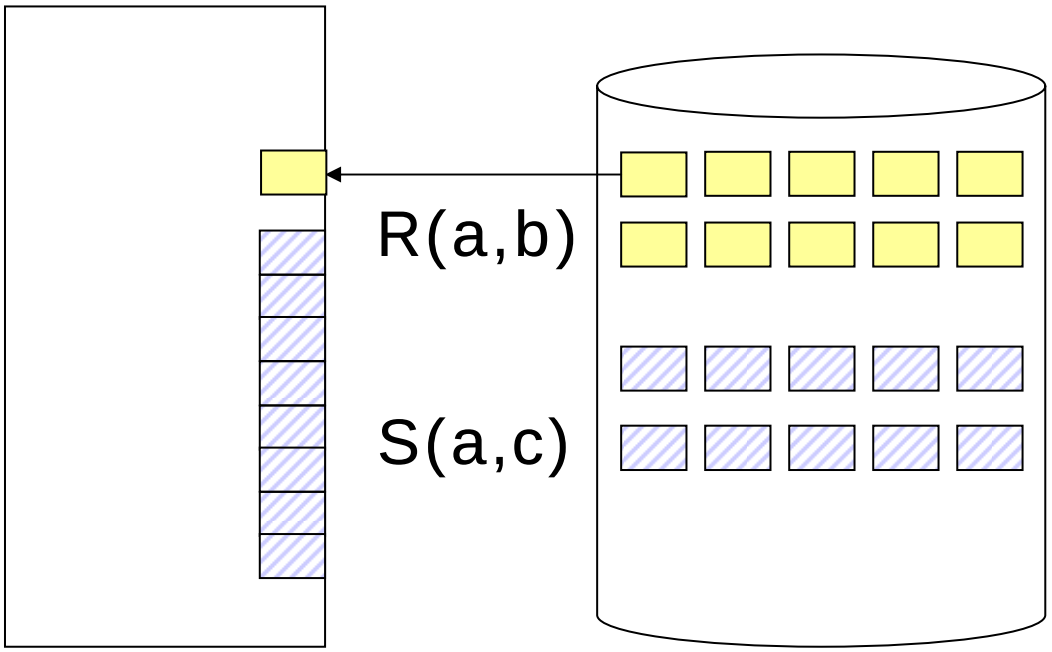
RとSのブロック数 B(R),B(S)
B(R)×B(S)個のブロックが
読み出される



ブロックを考慮した Nested-Loop Join の工夫

外のループ
S はブロック単位に
主記憶にできるだけ、
例えば M-1　ブロック読む。
属性 a で高速検索。

内のループ
R もブロック単位に読み、
属性 a が一致する S の
レコードを高速検索



ブロック数 B(R),B(S)

ブロック読み出し総回数
B(S)/(M-1) × (M-1 +B(R))
= B(S)+B(S)B(R)/M-1 ≒ B(R)B(S)/M

問合せの代数の実装

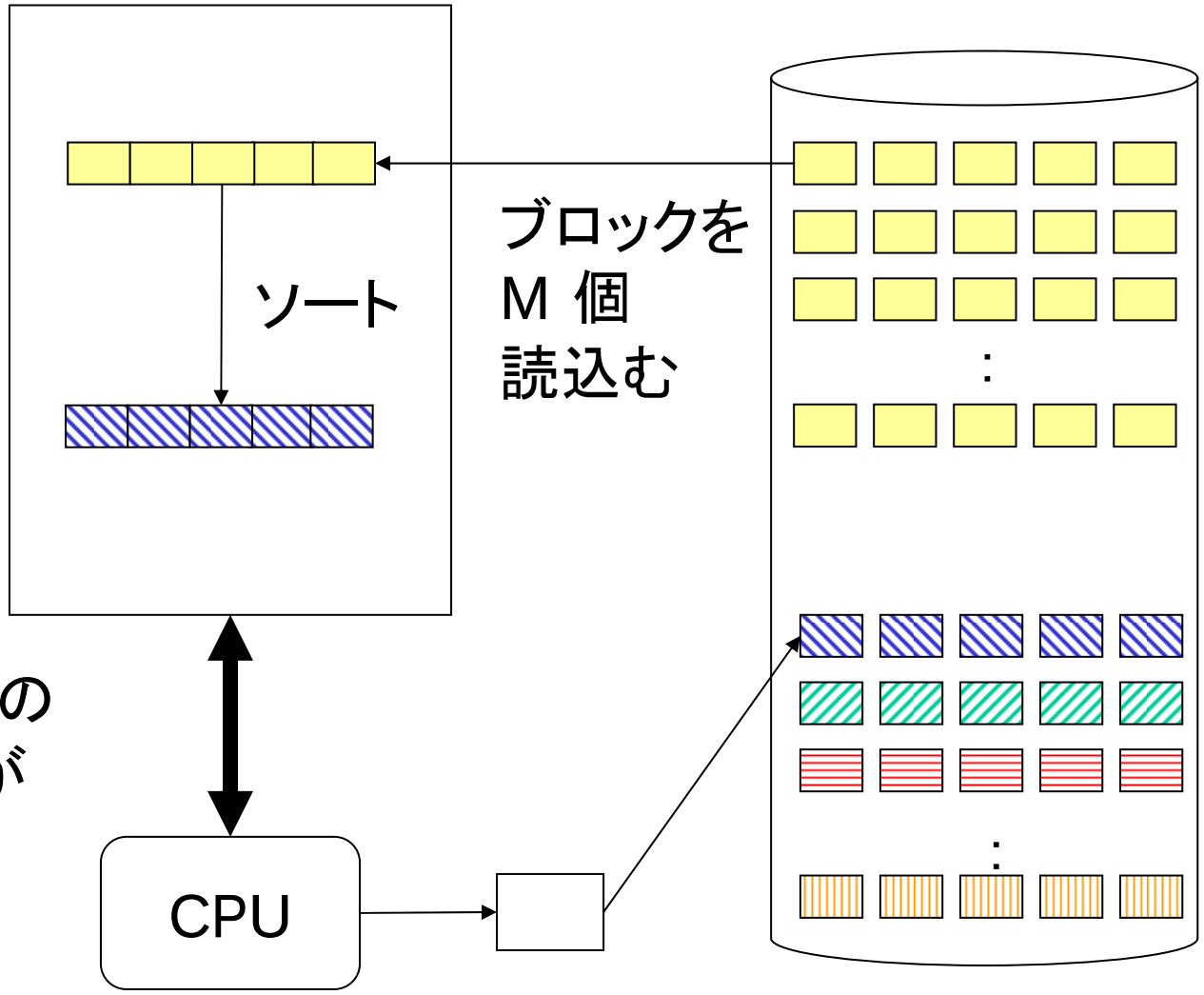
DBを2回走査することで
より大きな DB を処理できる方法

ソートをつかう場合

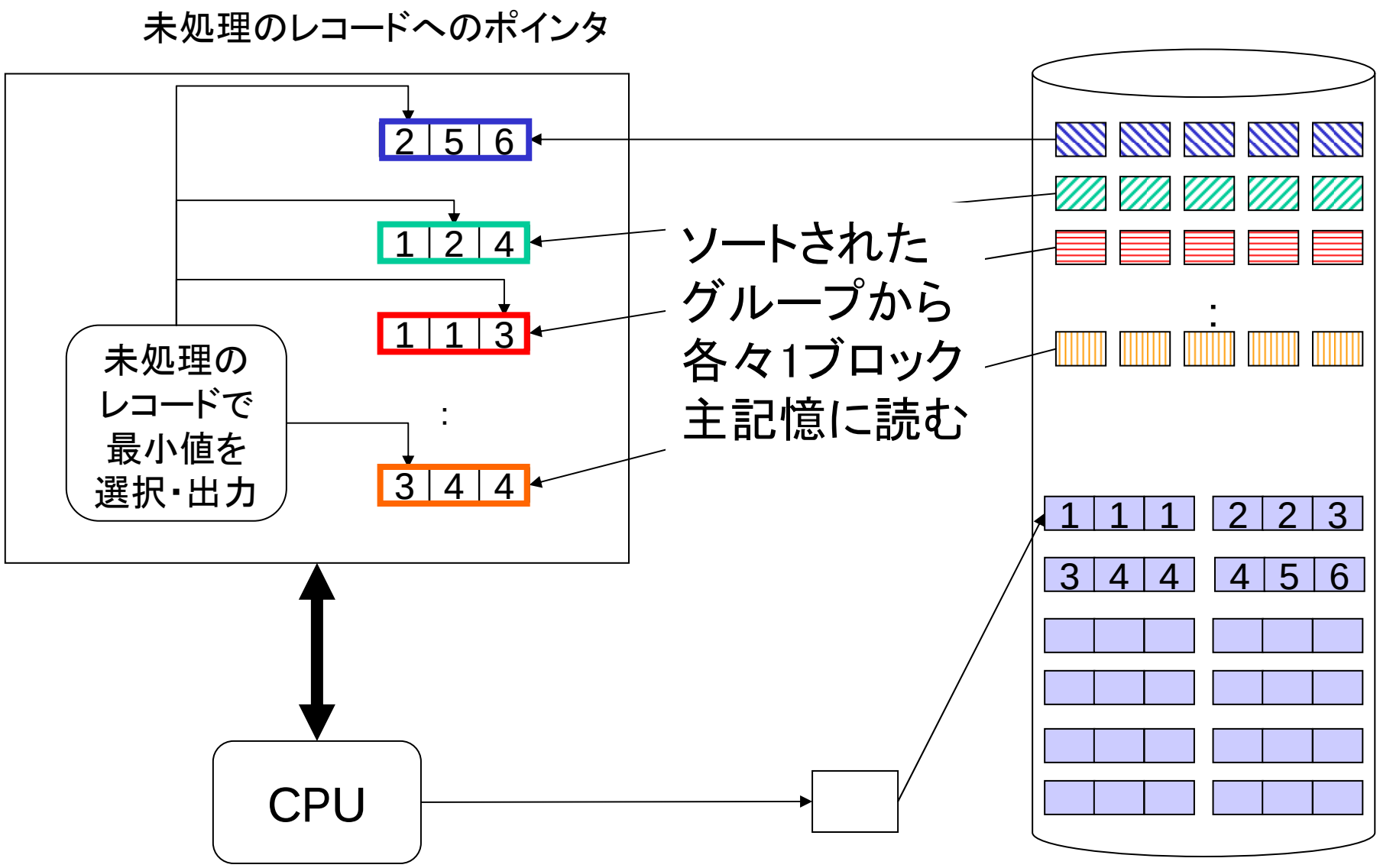
DB を2回走査する方法　ソート

主記憶に入らないDB のソート
Two-Phase Multiway Merge Sort

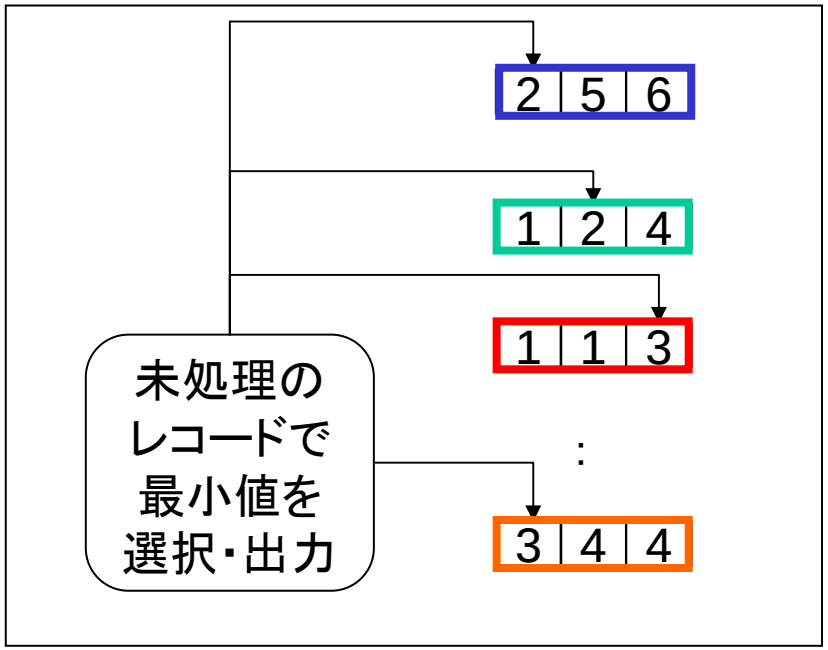
- ブロックを M 個 読込む
- 主記憶内でソート ブロックを読込む 時間内に通常は ソートできる
- ソートされた M 個の ブロックグループが 複数できる

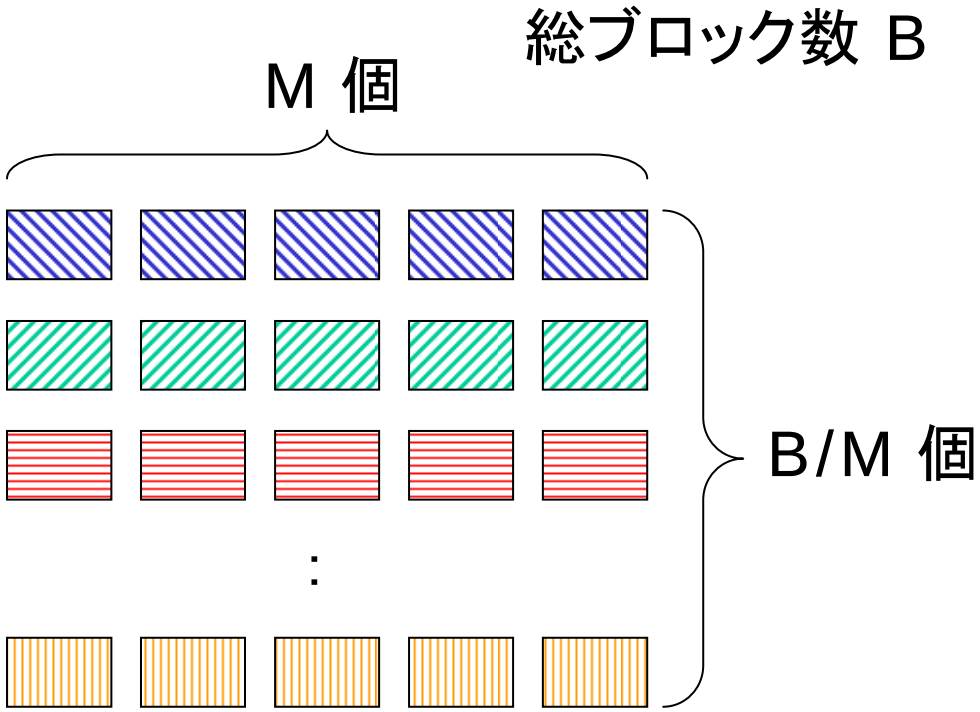


Two-Phase Multiway Merge Sort



Two-Phase Multiway Merge Sort のデータ量の限界



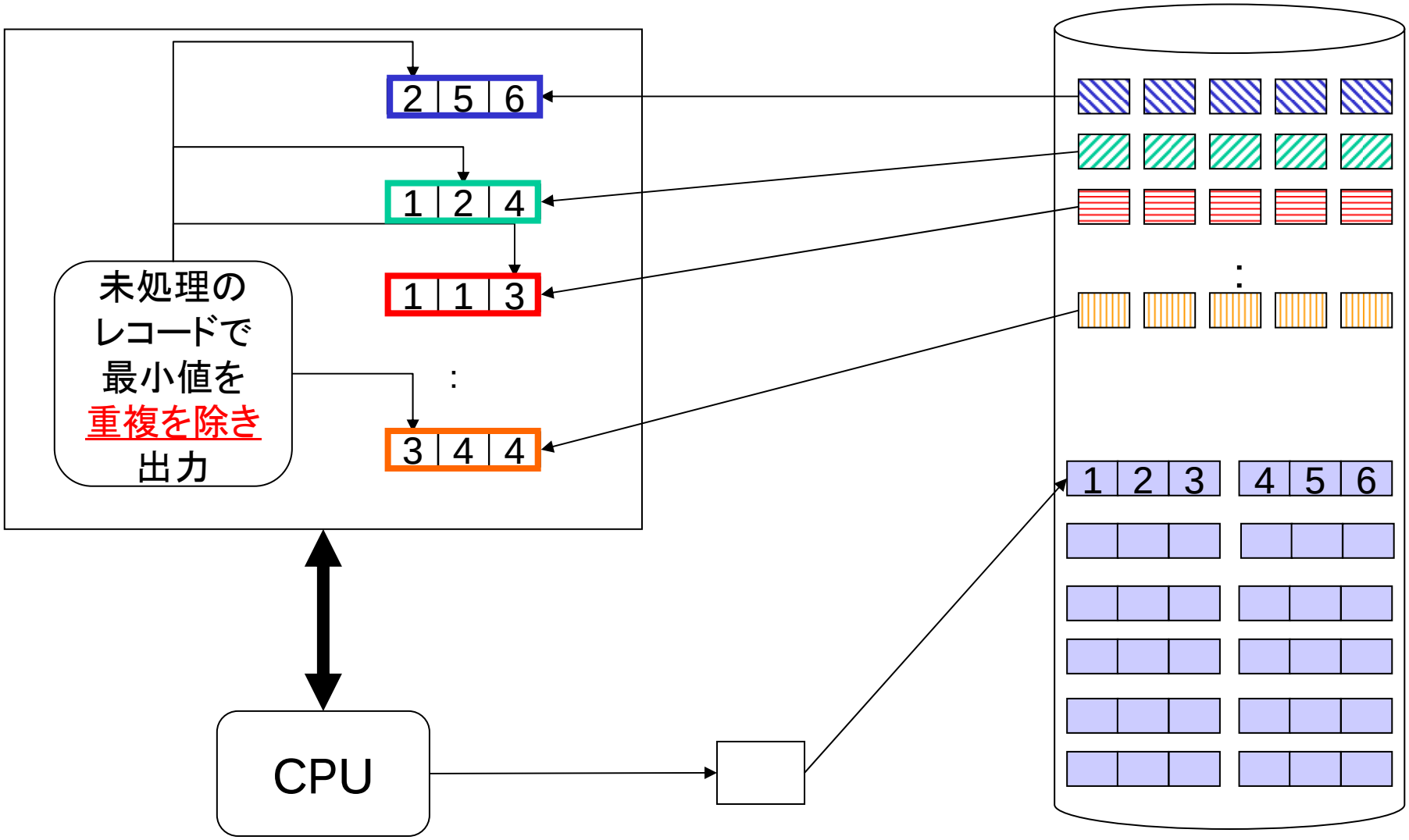


主記憶にブロックを M 個保持できる場合、$B/M \leqq M$
$B \leqq M^2$ のデータまでソートできる。　入出力は 4B ブロック。

例　4kBのブロックを, 15msec で転送できる場合の合計時間
$M = 10^3$, $B = 10^6$　主記憶4MBで 4GB　を　60,000秒
$M = 10^4$, $B = 10^8$　主記憶40MBで 400GB　を　6,000,000秒（約70日）

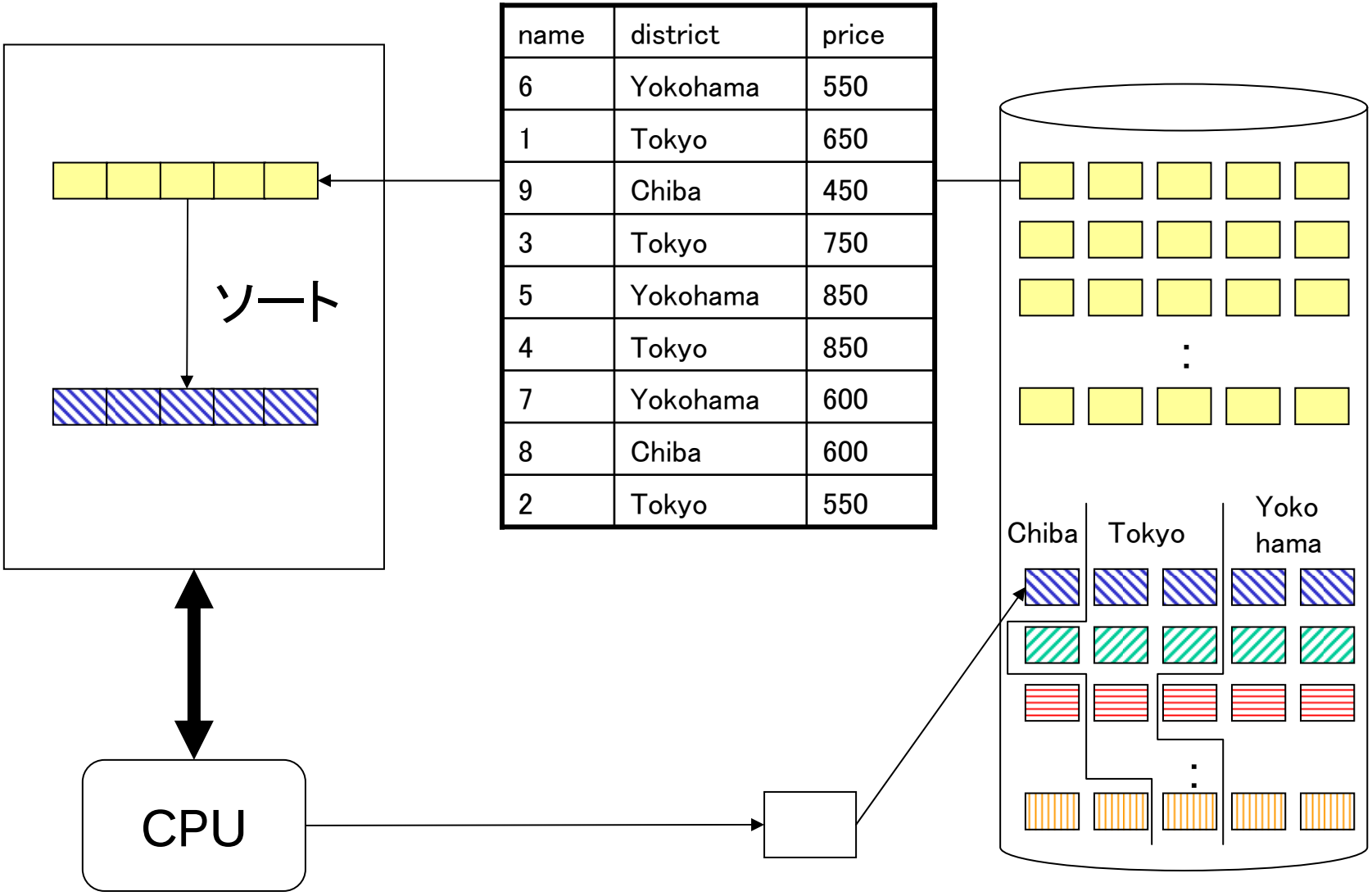
ソートをつかい DB を2回走査して重複除去

M 個のブロック毎にソートしておく



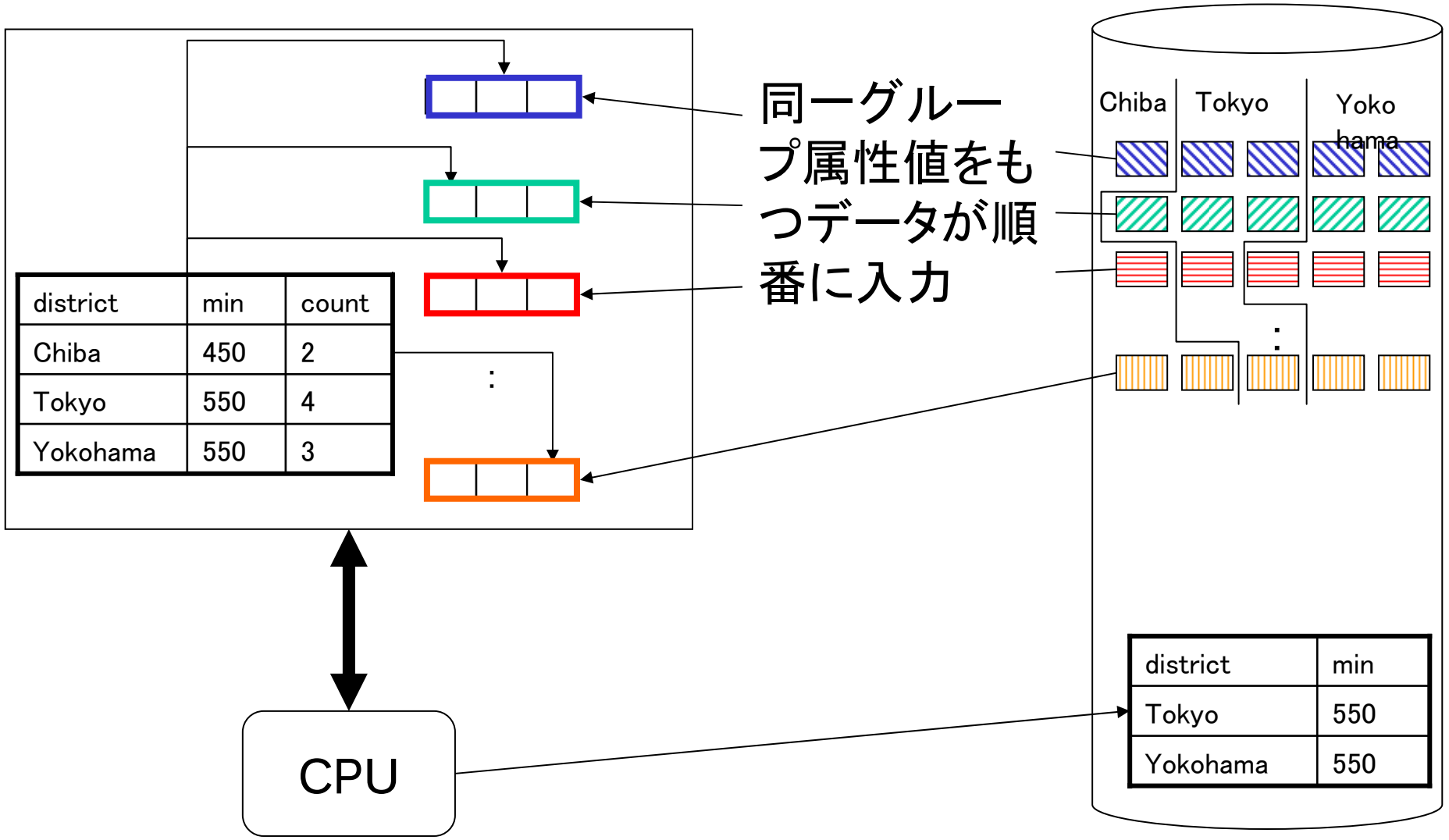
ソートを使ったグループ分けと集約

M 個のブロック毎にグループの対象属性でソートしておく



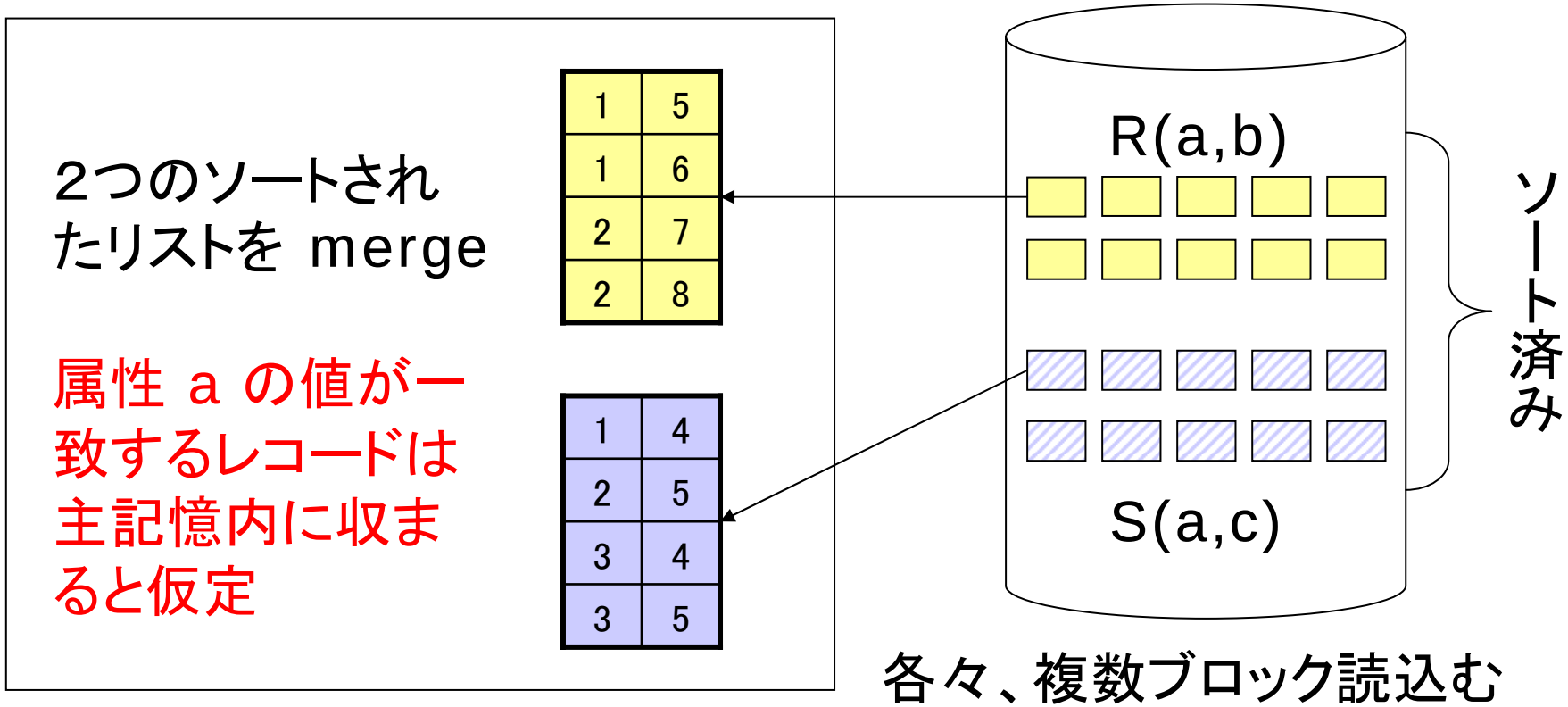
name	district	price
6	Yokohama	550
1	Tokyo	650
9	Chiba	450
3	Tokyo	750
5	Yokohama	850
4	Tokyo	850
7	Yokohama	600
8	Chiba	600
2	Tokyo	550

ソートを使ったグループ分けと集約



ソートを使ったジョインの実装

R(a,b) ▷◁ S(a,c)
まず R(a,b) および S(a,c) を属性 a で一気にソートしてしまう
4B(R)+4B(S) 個のブロック転送



各々、複数ブロック読込む
B(R)+B(S) 個ブロック転送

ソートを使ったジョインの実装

効き目がない最悪のデータ
属性 a の値がすべて同じ

Nested-Loop Join で対応

R(a,b)

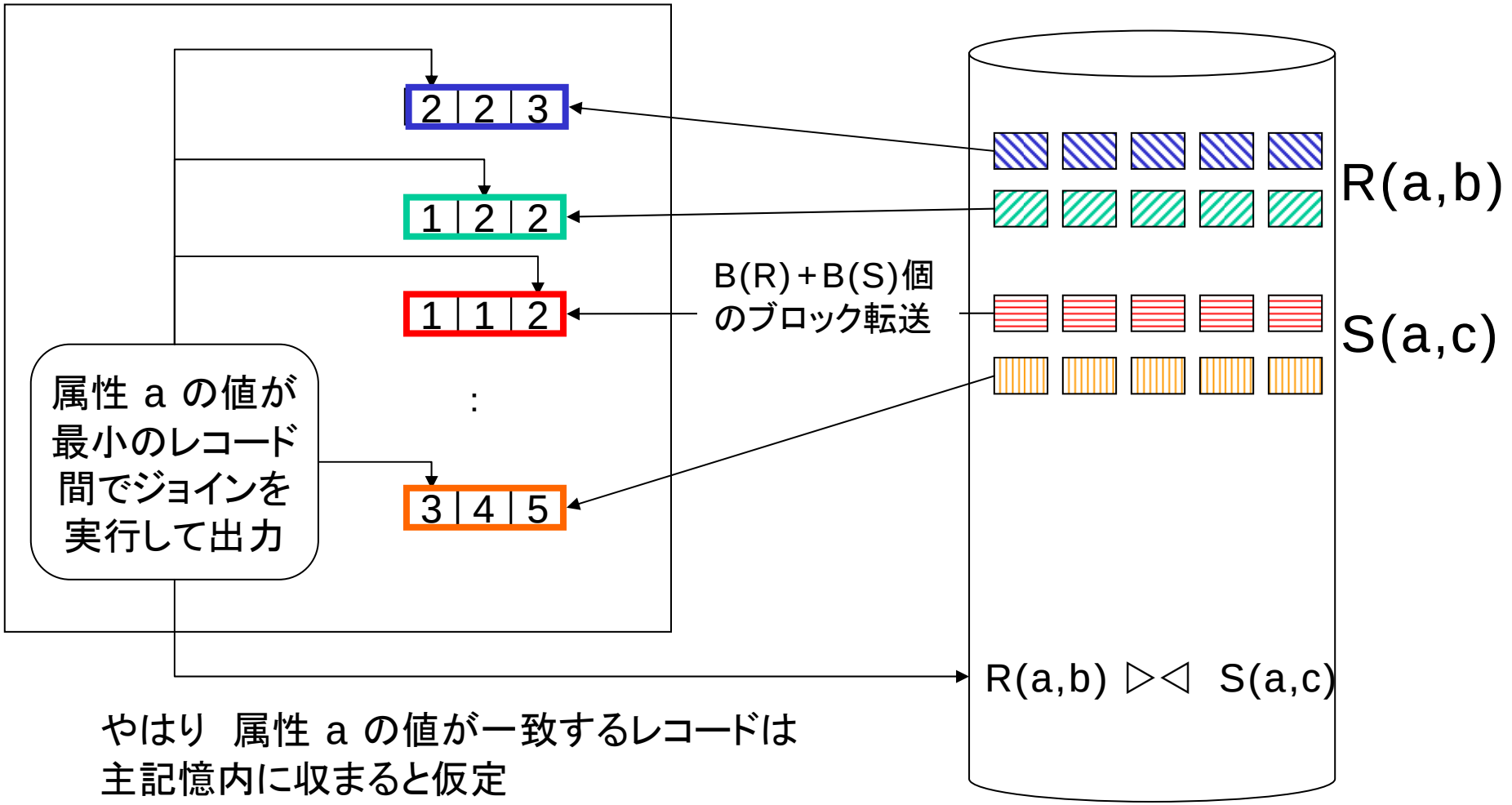
1	1
1	2
1	3
1	4
1	5
1	6

S(a,c)

1	1
1	2
1	3
1	4
1	5
1	6

ソートを使ったジョインの実装（改良版）

M 個のブロック毎に属性 a でソート (全体をソートしない)
2B(R)＋2B(S) 個のブロック転送で済ましておく



やはり 属性 a の値が一致するレコードは
主記憶内に収まると仮定

問題

∪、∩、－　を実装する際に、各データの頻度表を主記憶内に格納できれば、二次記憶中のデータを1回走査するだけで済む方法を述べた。

頻度表が主記憶に入らないほど大きなデータを処理する場合に、ジョインを実装した方式のようにソートを使って ∪、∩、－　を実装する方式を述べよ。

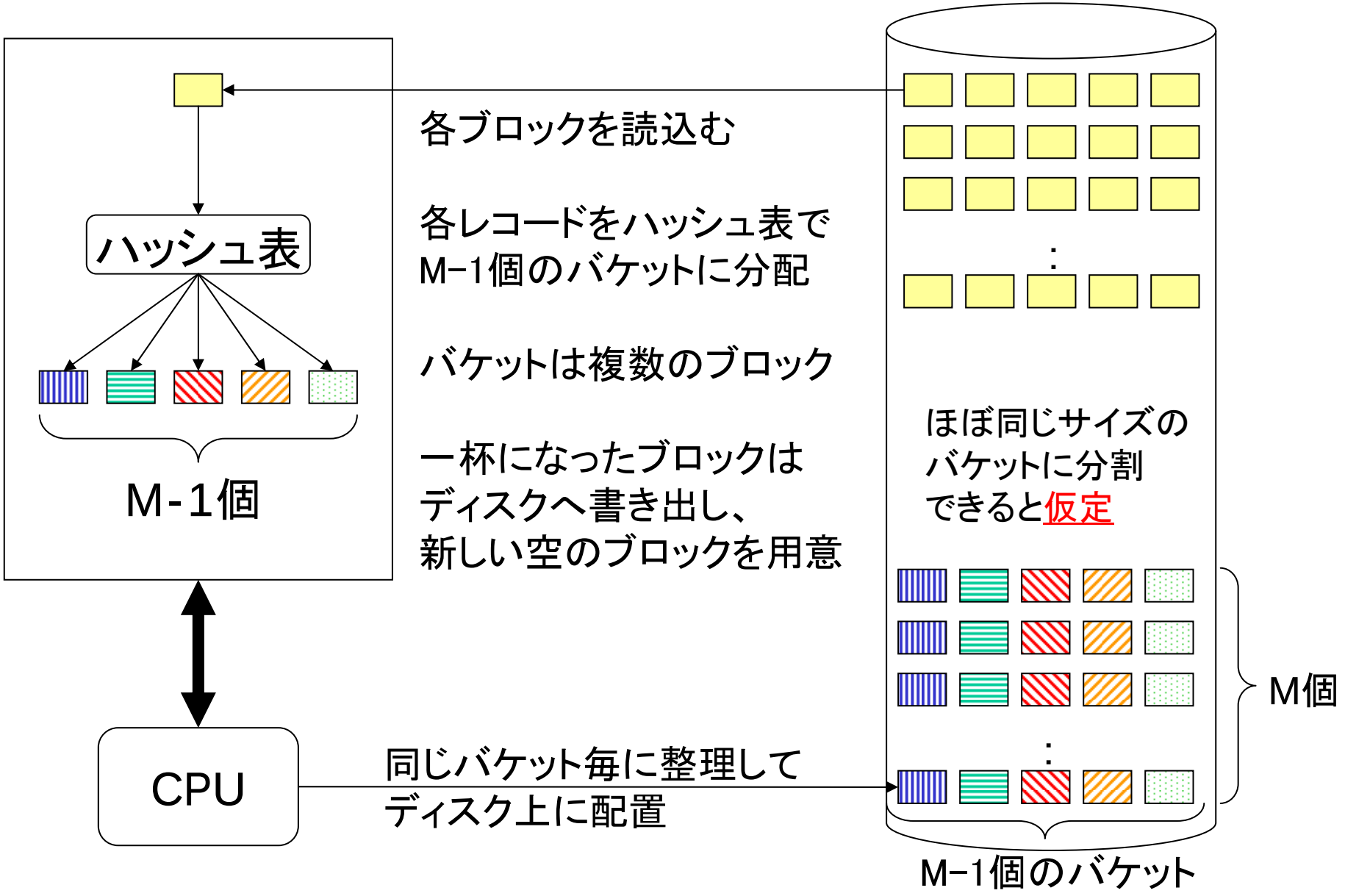
問合せの代数の実装

DBを2回走査することで
より大きな DB を処理できる方法

ハッシュ関数をつかう場合

問合せの並列化

ハッシュ関数をつかって関係を分割

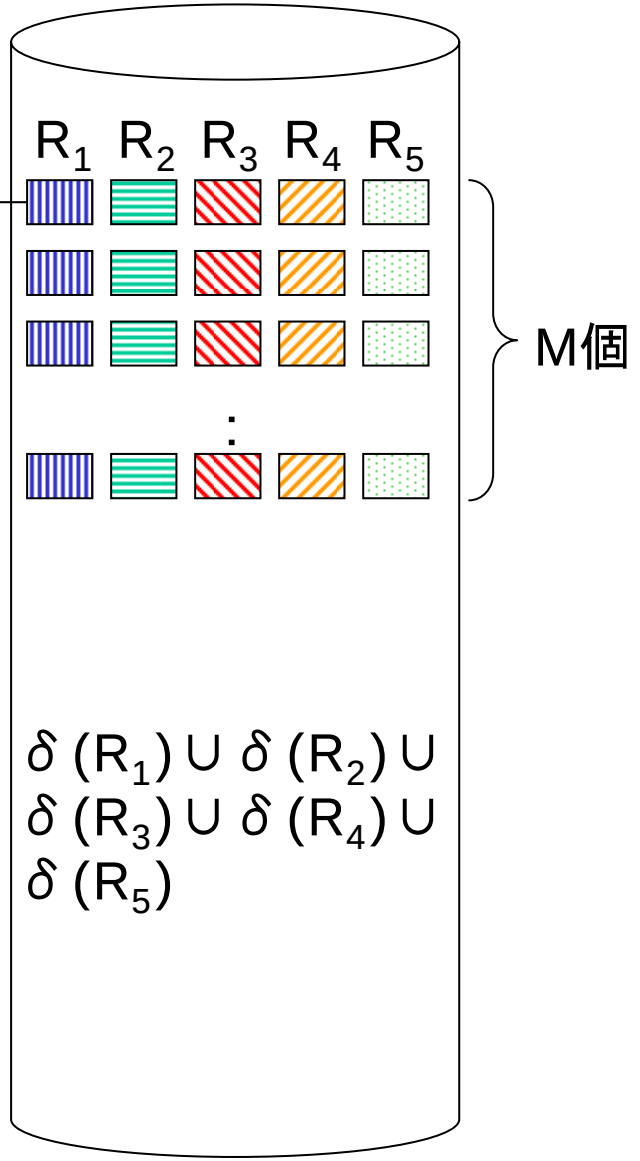


ハッシュをつかった重複除去

同じ値をもつレコードは必ず
同じバケットにあると仮定

（むろん同じ値をもつレコードがバケットに入らないほどあれば、この仮定は崩れる。ソートをつかった方法を利用すべき）

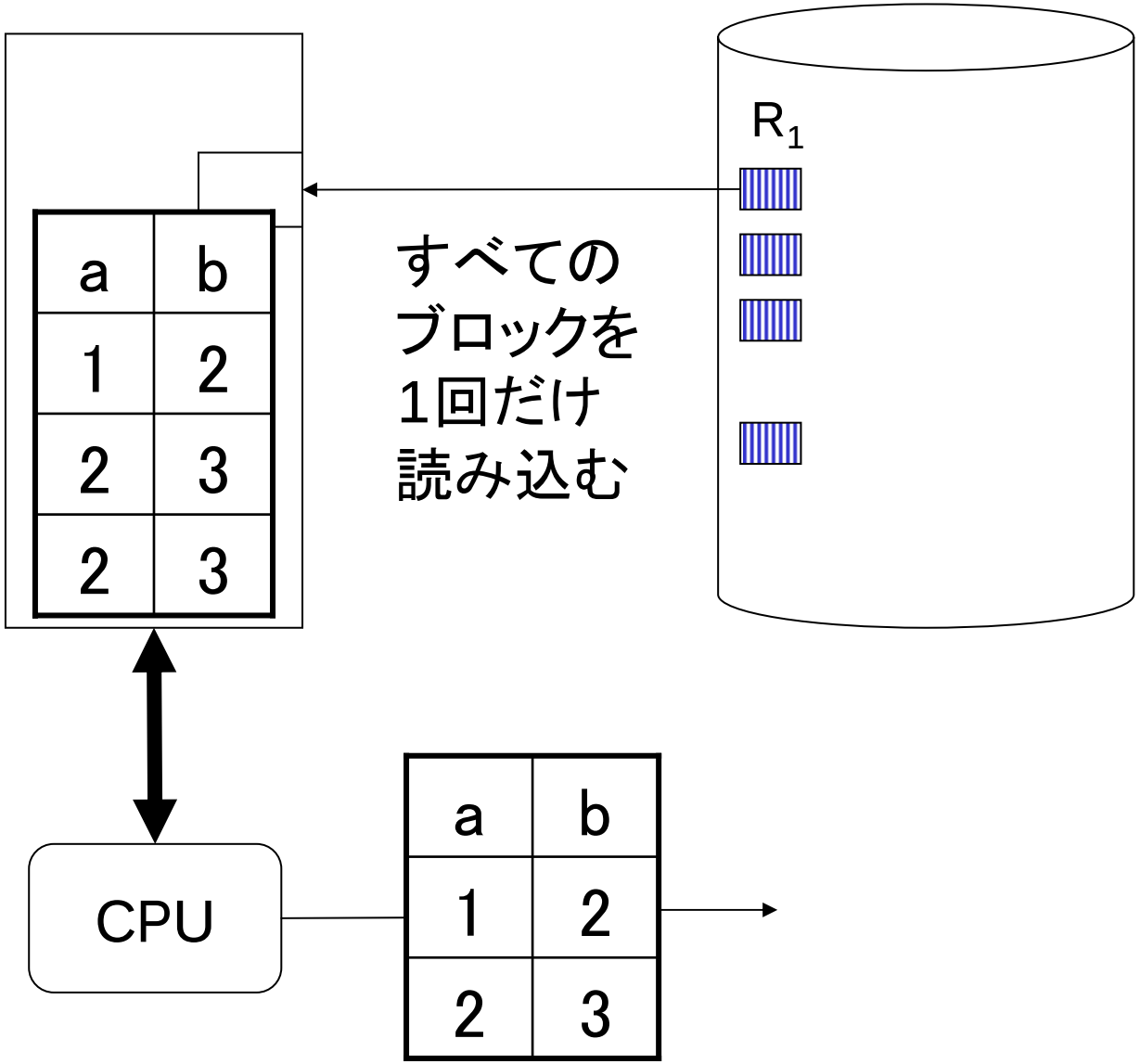
バケット毎に独立して
重複除去を実行できる
DBを一回だけ走査する
重複除去を各バケットに
対して実行



DBは1回だけ走査する重複削除を各バケットに適用

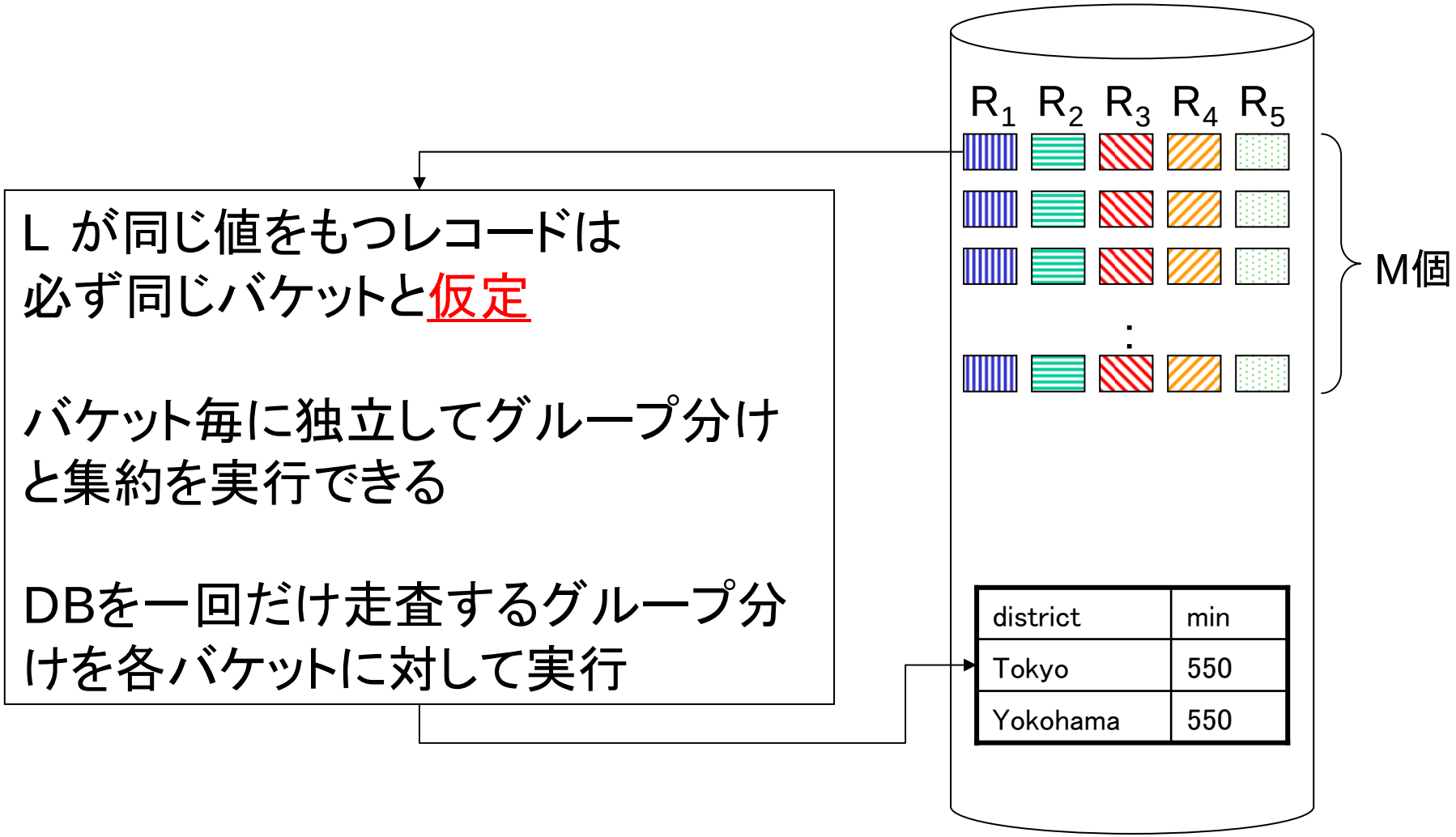
十分な主記憶があり
既に読んだレコードか
否かの判定を高速に
実行できる場合

ハッシュ表や
平衡2分木で
管理できる



ハッシュをつかったグループ分けと集約

$\gamma_L(R)$　　グループ分けの対象となる属性 L の値をつかいハッシュ関数で分割しておく

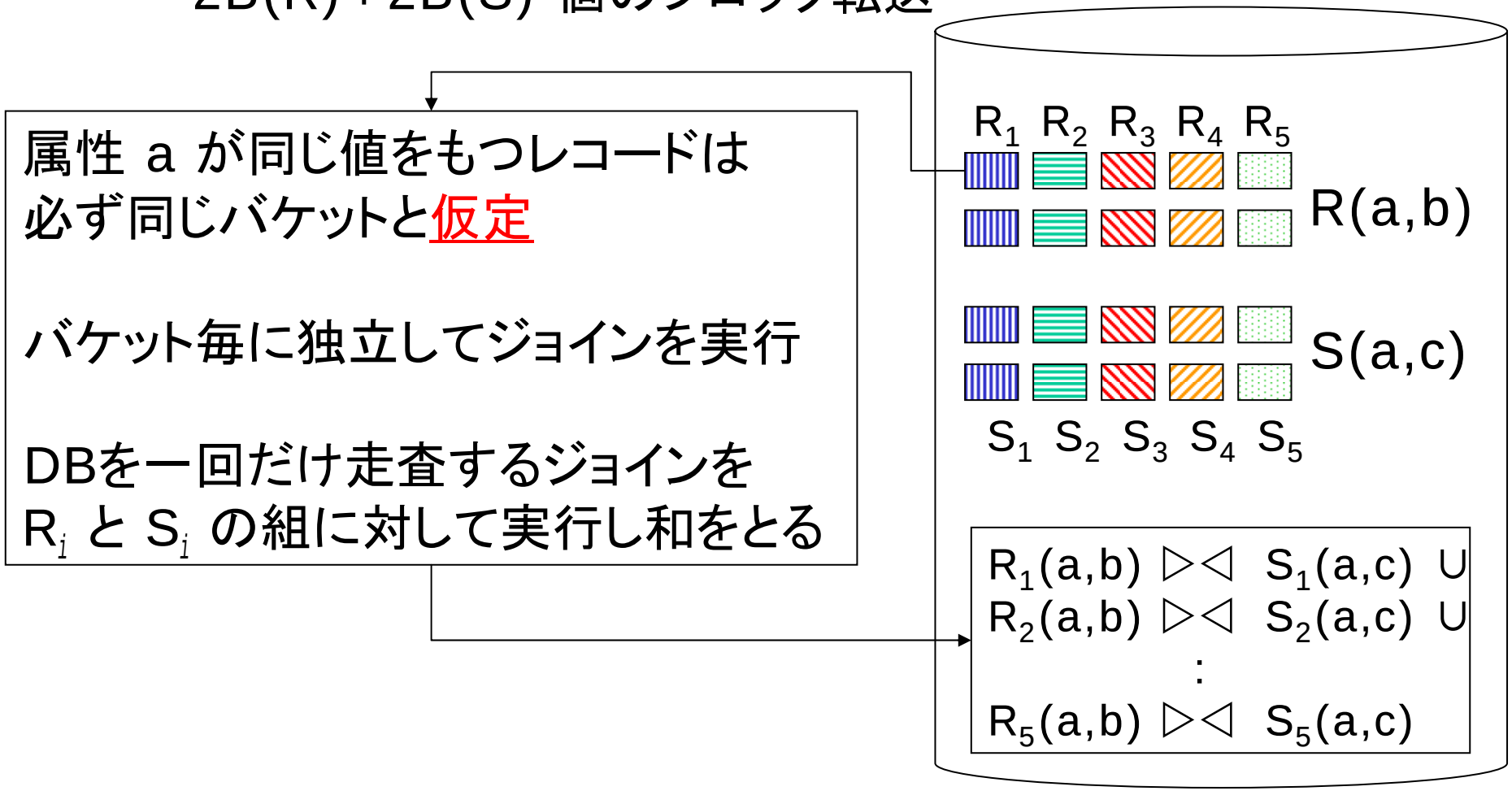


ハッシュ・ジョイン

$R(a,b) \triangleright\triangleleft S(a,c)$

まず　R(a,b) および S(a,c) を属性 a の値によりハッシュで分割

2B(R)＋2B(S) 個のブロック転送

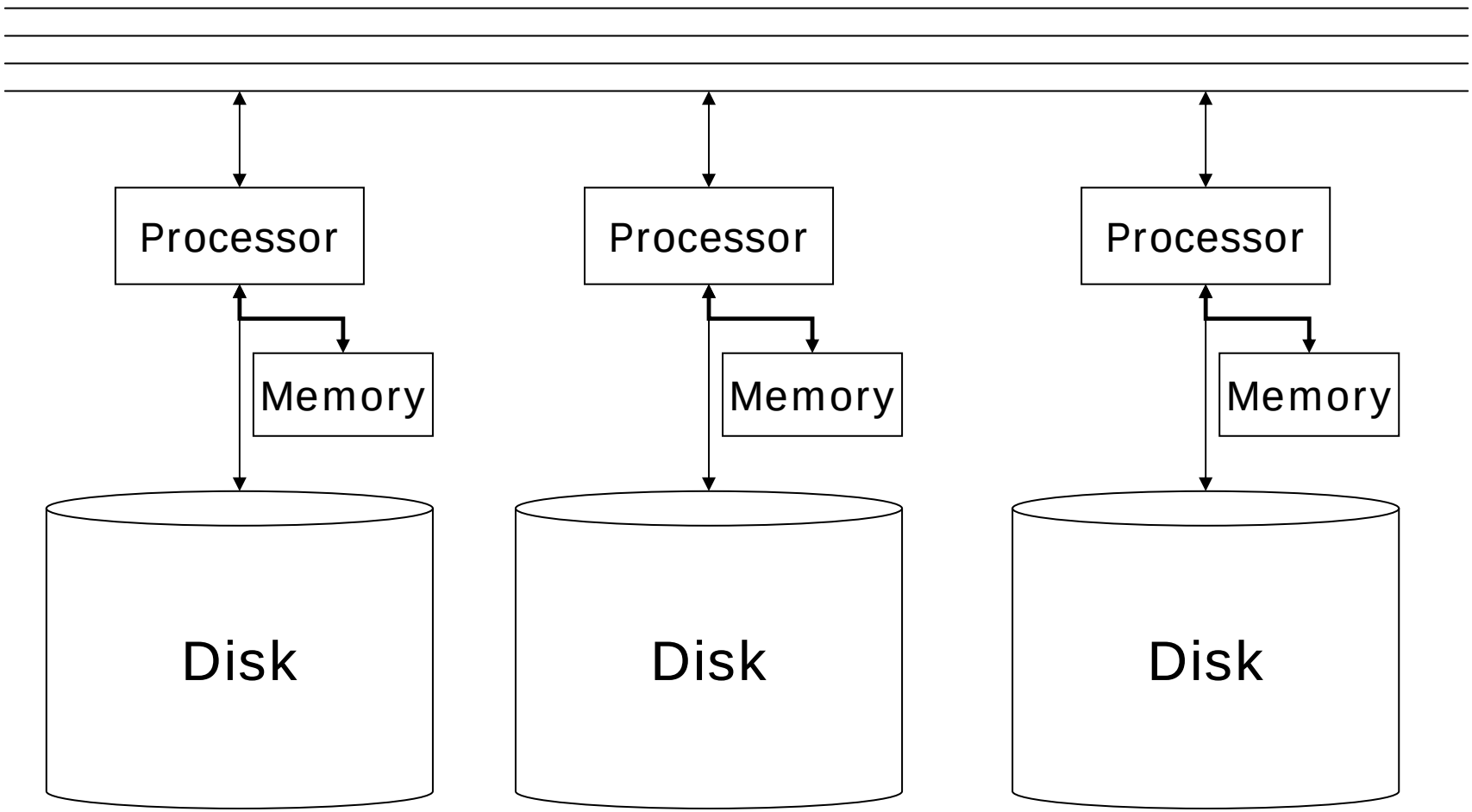


問題

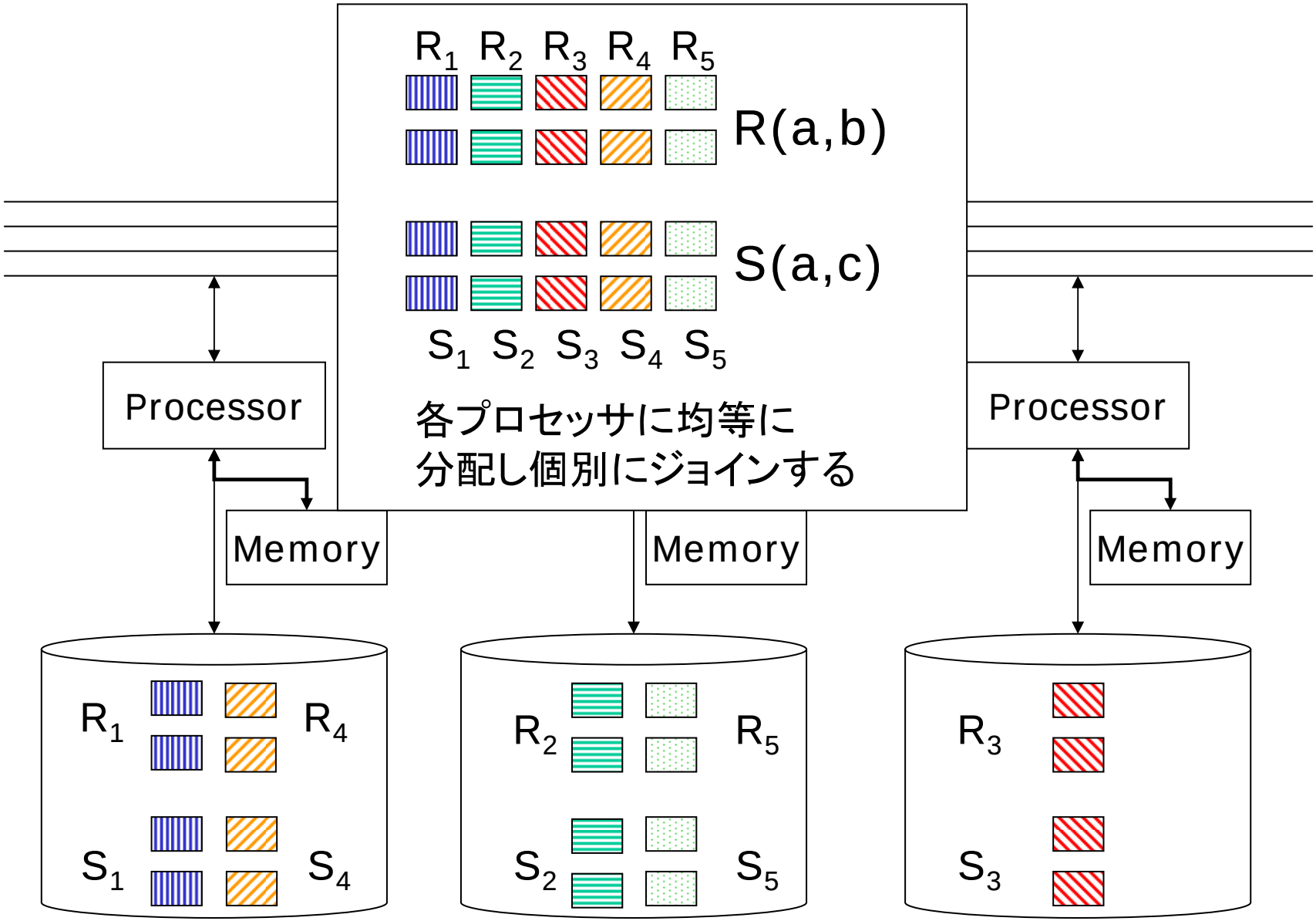
頻度表が主記憶に入らないほど大きなデータを処理する場合に、ハッシュ表を使って ∪、∩、－　を実装する方式を述べよ。

Shared-Nothing 型並列計算機でのハッシュ・ジョイン

DB 問合せは、Shared-Nothing 型並列計算機（たとえば Blade クラスター）
での実行になじむため Shared-Nothing 型はデータベース用計算機として標準的



Shared-Nothing 型並列計算機でのハッシュ・ジョイン

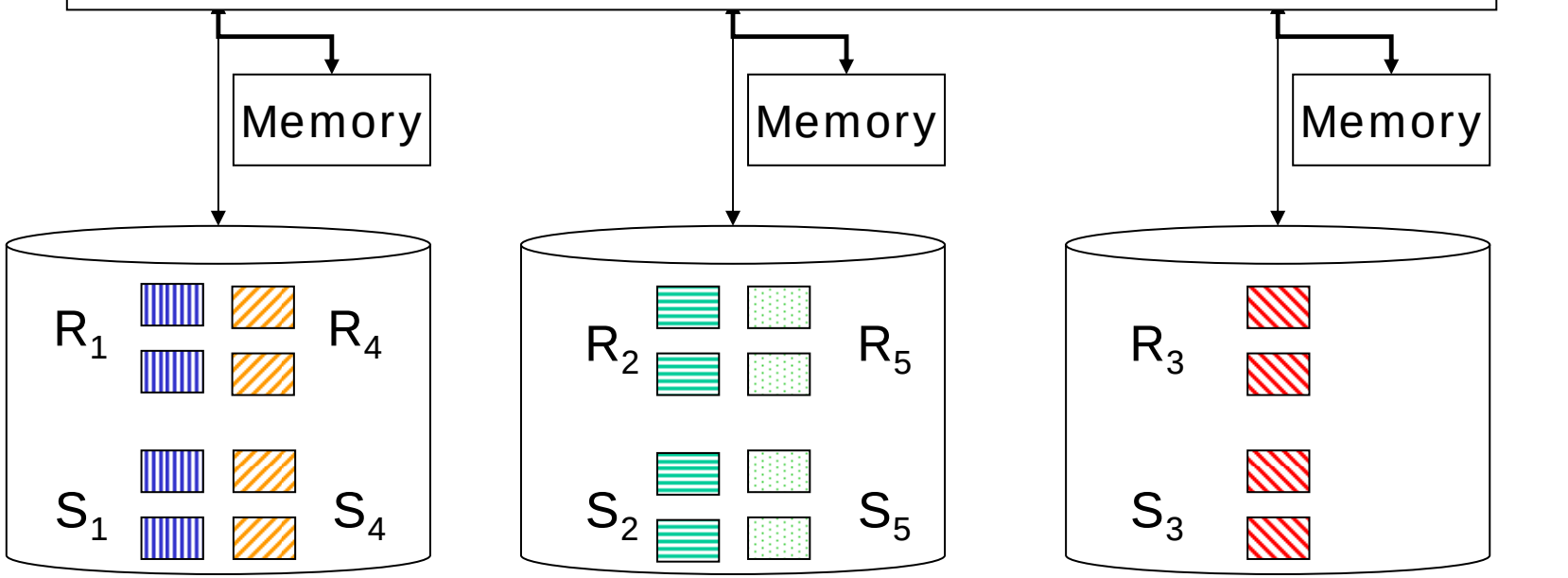


Shared-Nothing 型並列計算機でのハッシュ・ジョイン

p 個のディスクへ分配し、入出力を並列化することで性能を向上
均等に分配された後は、各プロセッサは1回の入力で
(B(R)+B(S))/p ブロックだけ転送すればよい

例

B(R)=1000, B(S)=500, プロセッサ数=10
各プロセッサに 150ブロック



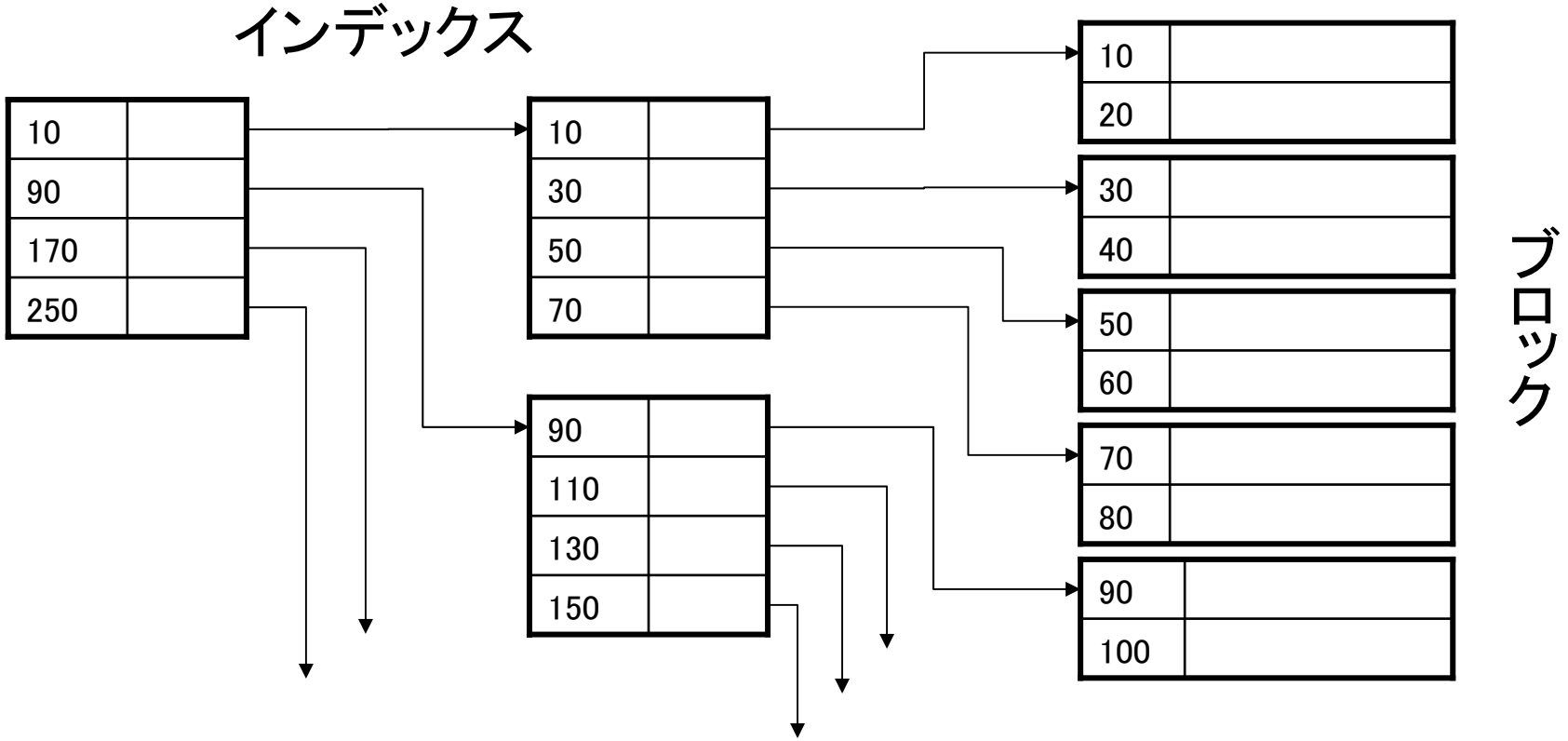
問合せの代数の実装

インデックスが利用できる場合

インデックス

インデックス：　属性 a の値が v となるレコードが
どのブロックにあるか、おおよその位置を保持した表

例えば、識別子(ID) の順番にレコードをディスク上に格納
インデックスは各 ID が格納されたブロックのおおよその位置を保持



その他のインデックス

- ２次的インデックス
  複数の属性にインデックス （例　名前と電話番号）
- B-木 (B-trees)
- ハッシュ表

技術的工夫

1. 更新への対応
2. 大量のデータにもアクセス速度を劣化させない

詳しくは　参考教科書の ４ 章を参照

H. Garcia-Morina, J. D. Ullman, and J. Widom. Database System Implementation. Prentice Hall, 2000, ISBN 0-13-040264-8

インデックスを利用した問合せの高速化

$\sigma_{a=v}(R)$

属性 a にインデックスがある場合
⇒ インデックスをつかった高速なアクセス

$R(a,b) \triangleright\triangleleft S(a,c)$

S(a,c) の属性 a にインデックスがある場合
⇒ R(a,b) の各レコード毎に
インデックスをつかった問合せを実行

問合せ速度を改善するための代数的操作

交換律と結合律

$R \times S = S \times R$ $\qquad$ $(R \times S) \times T = R \times (S \times T)$

Natural Join

$R \bowtie S = S \bowtie R$ $\qquad$ $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$R \cup S = S \cup R$ $\qquad$ $(R \cup S) \cup T = R \cup (S \cup T)$

$R \cap S = S \cap R$ $\qquad$ $(R \cap S) \cap T = R \cap (S \cap T)$

∪ と ∩ に関する交換律と結合律は、bags 意味論と集合意味論のどちらでも成立

代数的操作がうまくゆかない例

集合意味論では成り立つ分配律が bags 意味論だと不成立

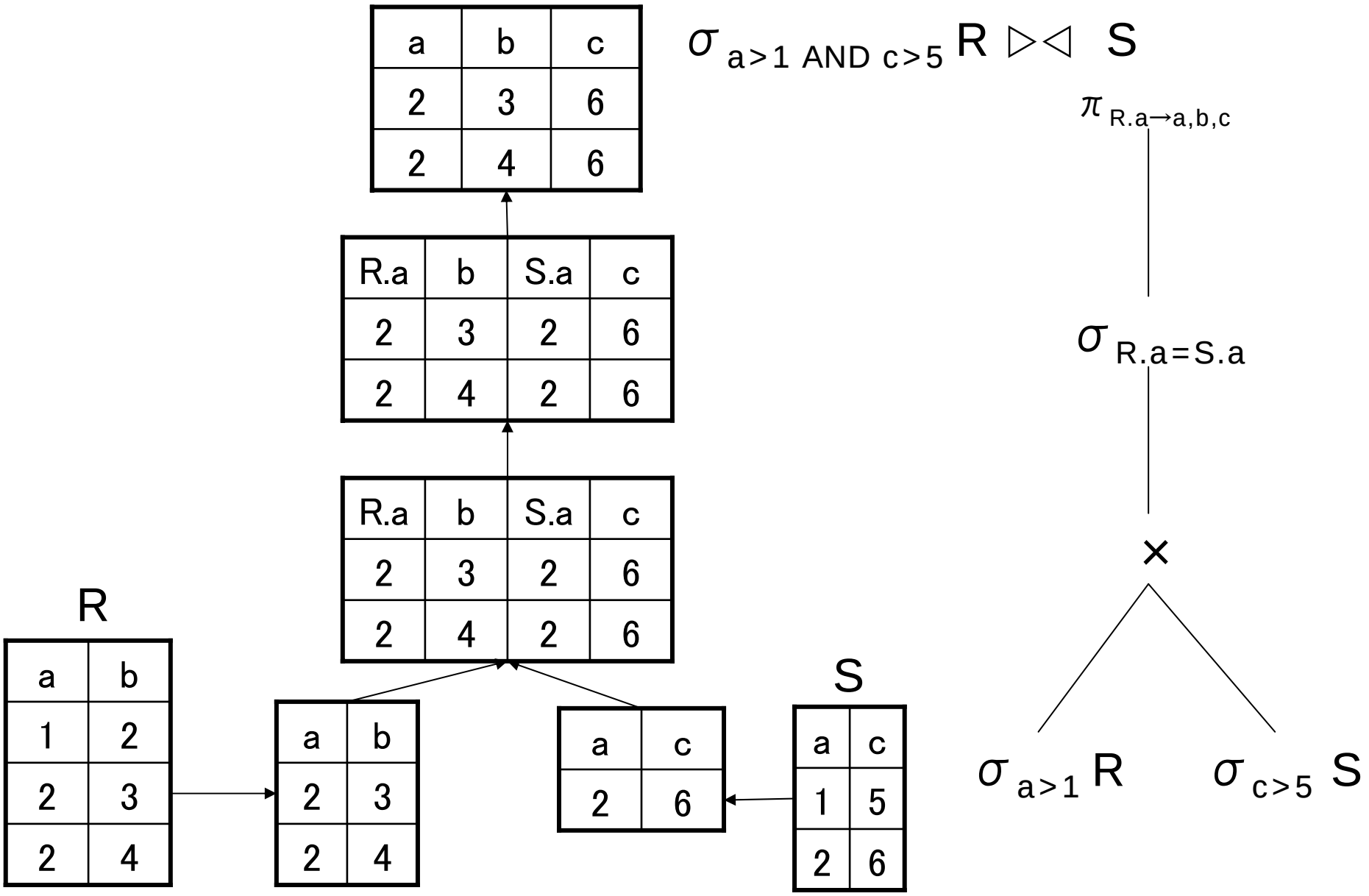
- $R = S = T = \{a\}$
- $R \cap (S \cup T) = \{a\} \cap \{a,a\} = \{a\}$
  $(R \cap S) \cup (R \cap T) = \{a\} \cup \{a\} = \{a,a\}$

Theta-Join では 結合律が意味をもたな場合がある

R(a,b), S(b,c), T(c,d)
$(R \triangleright\!\triangleleft_{R.b > S.b} S) \triangleright\!\triangleleft_{a < d} T$

$R \triangleright\!\triangleleft_{R.b > S.b} (S \triangleright\!\triangleleft_{a < d} T)$　　a は S と T の属性でない

選択を先行実行して効果がある例



選択を先行実行するための操作　AND, OR

$\sigma_{C1\ AND\ C2}(R) = \sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$

C1 と C2 どちらを先に実行してもよい

R に重複したレコードが存在しない場合

$\sigma_{C1\ OR\ C2}(R) = \sigma_{C1}(R) \cup_{set} \sigma_{C2}(R)$

ただし　$\cup_{set}$ は集合和

R が重複したレコードが存在する場合

$\sigma_{C1\ OR\ C2}(R) = \sigma_{C1}(R) \cup \sigma_{C2}(R)$

は必ずしも成り立たない

$\cup$ は bag 意味論であることに注意

問題

Bags 意味論において、以下の関係が成立しない反例と、成立する場合を示せ。

$$\sigma_{C1\ OR\ C2}(R) = (\sigma_{C1}(R)) \cup (\sigma_{C2}(R))$$

選択を先行実行するための操作　∪、－、×、▷◁、∩

$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$

$\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S) = \sigma_C(R) - S$

以下、条件 C に現れる属性は、すべて R の属性と仮定

$\sigma_C(R \times S) = \sigma_C(R) \times S$

積 × の場合、R と S は共通の属性を含まない

$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$

C が R と S に共通する属性を含む場合でも成立

$\sigma_C(R \cap S) = \sigma_C(R) \cap S$

選択を先行実行するための操作　∪、－、×、▷◁、∩

例　　選択の先行実行

$\sigma_{(a=1\ OR\ a=3)\ AND\ b<c}(R(a,b) \triangleright\triangleleft S(b,c))$

$\sigma_C(R \triangleright\triangleleft S) = \sigma_C(R) \triangleright\triangleleft S$　は使えない
条件が R(a,b) の属性以外に属性 c にも作用するから

$= \sigma_{a=1\ OR\ a=3}(\sigma_{b<c}(R(a,b) \triangleright\triangleleft S(b,c)))$
b<c を push

$= \sigma_{a=1\ OR\ a=3}(R(a,b) \triangleright\triangleleft \sigma_{b<c} S(b,c))$
b<c を さらに push

$= (\sigma_{a=1\ OR\ a=3} R(a,b)) \triangleright\triangleleft (\sigma_{b<c} S(b,c))$
a=1 OR a=3 は R(a,b) のみに作用

問題

以下の2つの性質が成立することを説明せよ（できれば証明せよ）

$\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S) = \sigma_C(R) - S$

$\sigma_C(R \triangleright\triangleleft S) = \sigma_C(R) \triangleright\triangleleft S$

条件 C に現れる属性は、すべて R の属性と仮定

射影の先行実行

- 選択の先行実行
  計算途中で生成されるレコードの数を減らす効果

- 射影の先行実行は、効果的か？
  表の幅を狭くするが、レコードの数を減らさない
  （bags 意味論の場合）
  選択の先行実行ほどの効果はない

- 集合意味論の場合
  射影すると重複レコードを除ける場合がある

射影を先行実行するための操作

ジョイン等を実行するのに必要な属性と被射影属性を残す

- $\pi_L(R \triangleright\triangleleft S) = \pi_L(\pi_M R \triangleright\triangleleft \pi_N S)$
  M: S もしくは L に現れる R の属性全体
  N: R もしくは L に現れる S の属性全体

$\pi_{a+e\rightarrow x,b\rightarrow y}(R(a,b,c,d) \triangleright\triangleleft S(c,e,f))$
$= \pi_{a+e\rightarrow x,b\rightarrow y}(\pi_{a,b,c}R(a,b,c,d) \triangleright\triangleleft \pi_{c,e}S(c,e,f))$

- $\pi_L(R \times S) = \pi_L(\pi_M R \times \pi_N S)$
  M = L に現れる R の属性全体
  N = L に現れる S の属性全体

射影を先行実行するための操作

- $\pi_L(R \cup S) = \pi_L(R) \cup \pi_L(S)$
- $\pi_L(R \cap S) = \pi_L(R) \cap \pi_L(S)$ は必ずしも成り立たない
  （$\cap$ は bags 意味論）
- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$

  M は L もしくは C に現れる R の属性

問題

- $\pi_{a,f}(R(a,b,c,d) \triangleright\triangleleft S(c,d,e,f))$ において
射影を先行実行する式を示せ

- $\pi_L(R \cap S) = \pi_L(R) \cap \pi_L(S)$ が成立しない例を示せ
（$\cap$ は bags 意味論）

重複除去を先行実行するための操作

重複除去によりレコード数が減ることを期待して

重複除去を先行して実行する

$$\delta(R \times S) = \delta(R) \times \delta(S)$$

$$\delta(R \rhd\lhd S) = \delta(R) \rhd\lhd \delta(S)$$

$$\delta(R \rhd\lhd_C S) = \delta(R) \rhd\lhd_C \delta(S)$$

$$\delta(\sigma_C(R)) = \sigma_C(\delta(R))$$

$$\delta(R \cap S) = \delta(R) \cap \delta(S)$$

問題

成立しない性質　反例を示せ

$\delta(R \cup S) = \delta(R) \cup \delta(S)$

$\delta(R - S) = \delta(R) - \delta(S)$

$\delta(\pi_a(R(a,b))) = \pi_a(\delta(R(a,b)))$

尚、集合意味論ではこれらの性質は成立

なぜならレコードの重複を許さない意味論だから

SQL 文の構文解析木から効率的な問合せを合成

簡略化した SQL

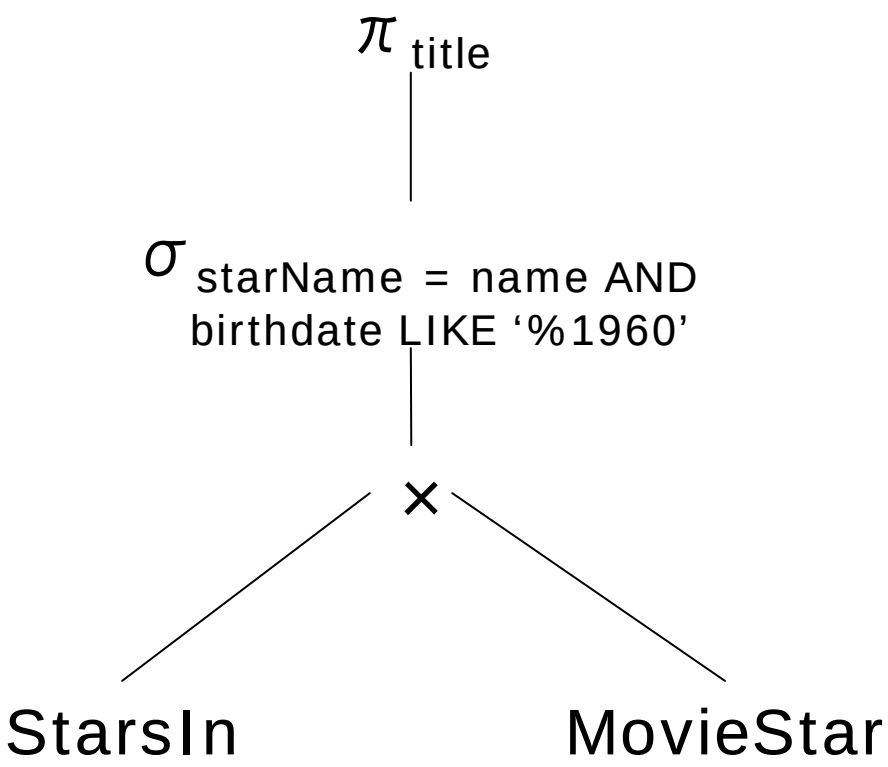

```
<Query> ::= <SFW>
<Query> ::= ( <Query> )

<SFW> ::= SELECT <SelList>
          FROM <FromList>
          WHERE <Condition>
<SelList> ::= <Attribute>, <SelList>
<SelList> ::= <Attribute>
<FromList> ::= <Relation>, <FromList>
<FromList> ::= <Relation>
<Condition> ::=
      <Condition> AND <Condition>
<Condition> ::=
      <Tuple> IN <Query>
<Condition> ::=
      <Attribute> = <Attribute>
<Condition> ::=
      <Attribute> LIKE <Pattern>

<Tuple> ::= <Attribute>
```

```
StarsIn(title, year, starName)
MovieStar(name, address,
                  gender, birthdate)

SELECT title
FROM StarsIn
WHERE starName IN (
  SELECT name
  FROM MovieStar
  WHERE birthdate LIKE '%1960'
);

SELECT title
FROM StarsIn, MovieStar
WHERE starName = name AND
      birthdate LIKE  '%1960'
```

SQL文から問合せ木の生成

StarsIn(title, year, starName)
MovieStar(name, address,
gender, birthdate)

```
SELECT title
FROM StarsIn, MovieStar
WHERE starName = name AND
      birthdate LIKE '%1960'
```



SQL文の問合せ木への変換
（木の下から順番に）

<FromList> の関係の積

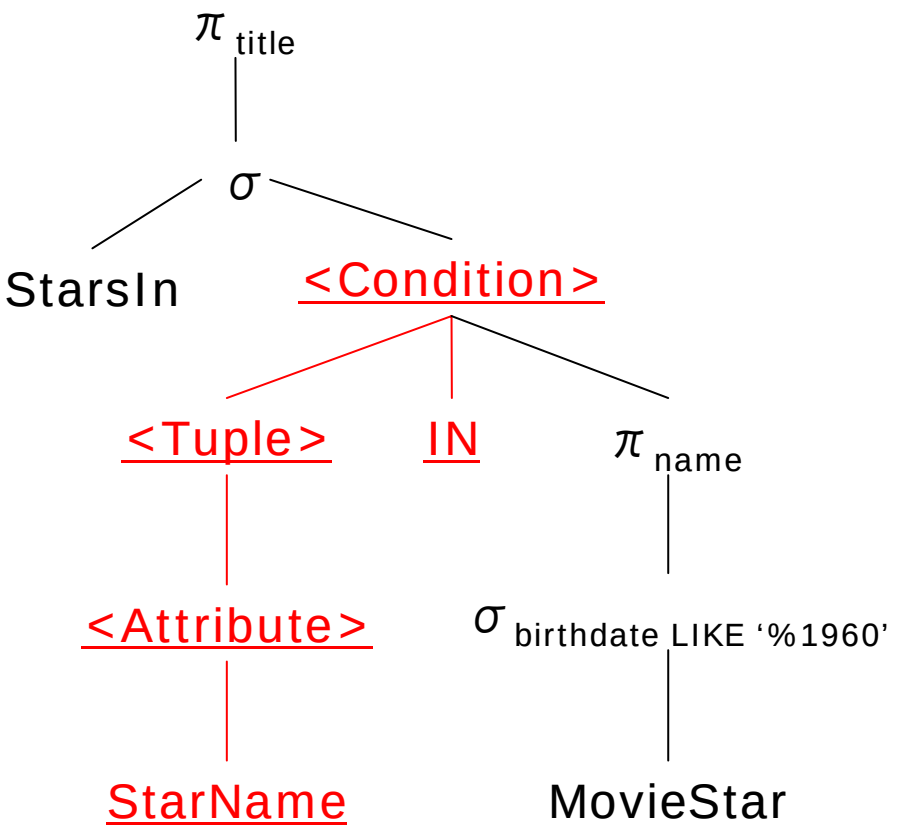
<Condition> を C として
選択 σ_C を適用

<SelList> の属性リスト L で
射影 π_L を適用

問合せ木

StarsIn(title, year, starName)
MovieStar(name, address,
gender, birthdate)

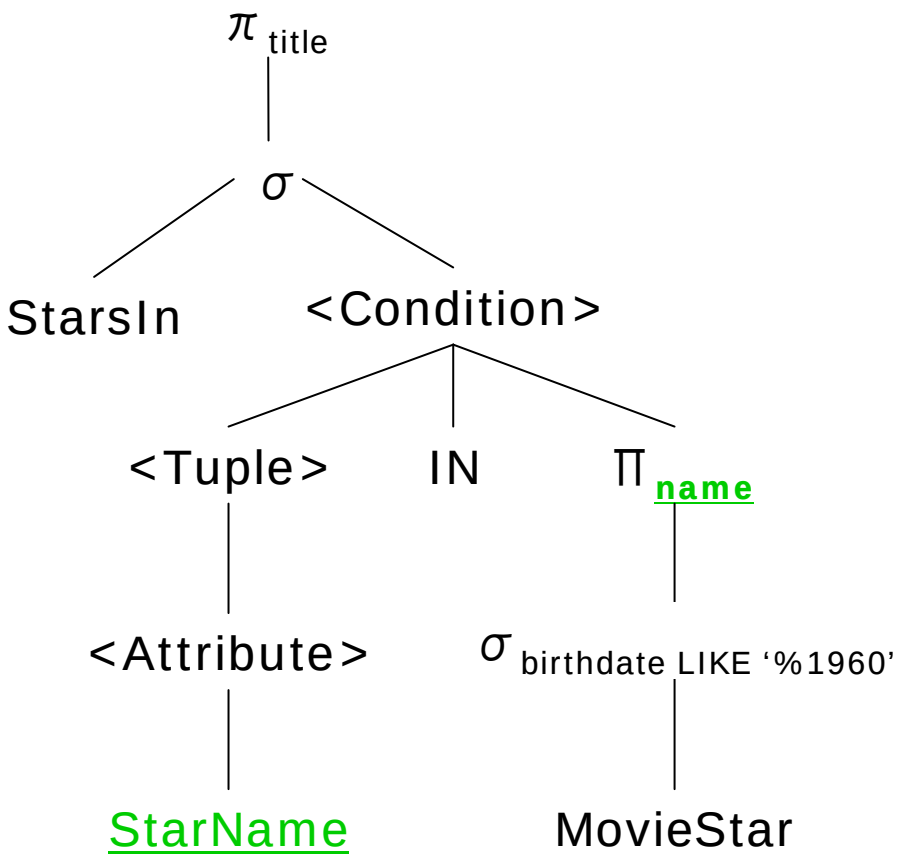
```
SELECT title
FROM StarsIn
WHERE starName IN (
   SELECT name
   FROM MovieStar
   WHERE birthdate LIKE '%1960'
);
```

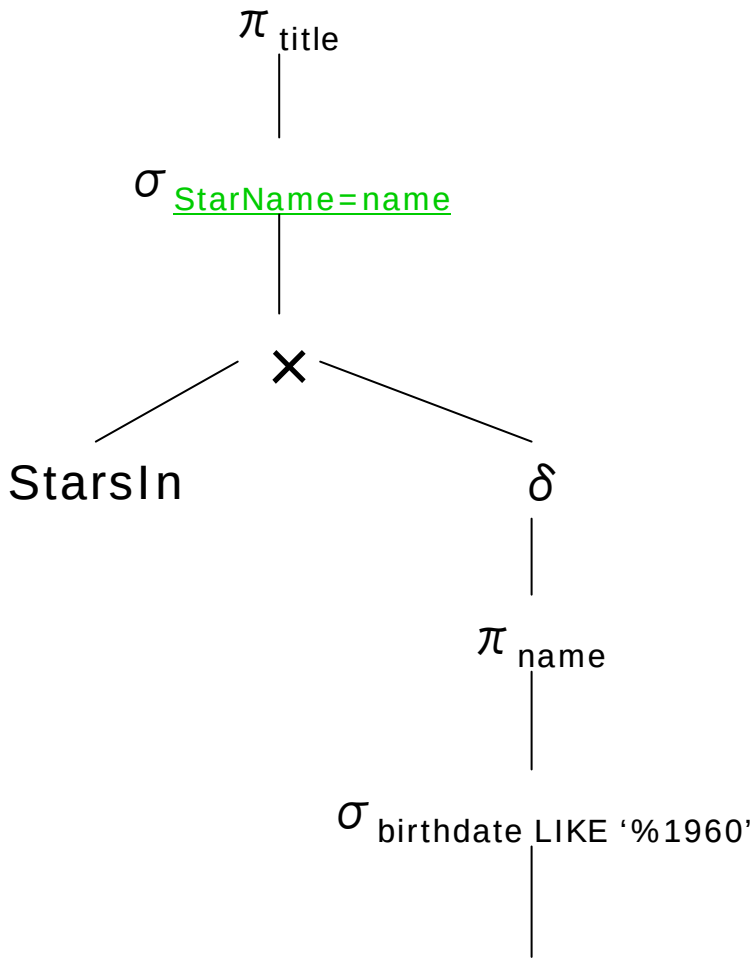


部分的問合せを
そのままの構造で表示

starName 各々について
MovieStar を検索する
Nested loop 型問合せ

問合せ木の変換





註: '<Tuple> IN 関係 R' で抽出されるレコードは重複しない。
結果を一致させるため、右の問合せ木で π_{name} の後に重複削除する δ をほどこす。

Joinを使った効率化

問合せコストの近似

問合せコストは中間計算結果の大きさに依存

問合せの実行中に生成される中間計算結果

- ブロック転送回数を低減するためにも
  ブロックにまとめて保存

- 中間計算結果の大きさ
  （レコード数　×　レコード長）を近似したい

- したがってレコード数を近似することが重要

中間計算結果のレコード数を近似

近似の方法

- 論理的に矛盾がないこと
  たとえば (R▷◁T)▷◁S と R▷◁(T▷◁S) の 近似値は同じであるべき

- 簡単に計算ができること
  近似が実際に計算するのと同程度に重たいと無意味

- 計算が容易なパラメータ （近似は粗くても妥協する）

  T(R): 関係 R のレコード数
  V(R,a): 属性 a における、異なる値の数

選択結果のレコード数を近似

$S = \sigma_{a=1}(R)$

a の値が異なるレコード数が同数ならば
（むろん現実には偏りがあるが）

$$T(S) = T(R) / V(R,a)$$

T(S), T(R) は S と R のレコード数

$S = \sigma_{a \neq 1}(R)$ の場合

$$T(S) = T(R) (V(R,a)-1) / V(R,a)$$

選択結果のレコード数を近似

$S = \sigma_{C1\ OR\ C2}(R)$

C1 をみたすレコードの割合 $m1/T(R)$
C1 をみたさないレコードの割合 $1 - m1/T(R)$

$$T(S) = T(R) \times (1 - \underbrace{(1 - m1/T(R)) \times (1 - m2/T(R))}_{\text{C1もC2もみたさないレコードの割合}})$$

ジョイン結果のレコード数を近似するためのヒューリスティクス

R(a,b)	S(b,c)
T(R)＝1000	T(S)＝2000
V(R,b)＝20	V(S,b)＝50

仮定 1（属性値の包含）

属性 b の異なる値の数は、R が 20 個、S が 50 個

一般に V(R,b) ≦ V(S,b) ならば

R の属性 b の値は、すべて S にも現れると仮定

仮定 2（属性値の均等分布）

R のレコードは S のレコードと 1/V(S,b) の確率でジョインする

R(a,b) ▷◁ S(b,c) のレコード数:

$$T(R) \times \underbrace{(1 / V(S,b)) \times T(S)}_{\text{属性値の包含性}} = T(R)T(S) / V(S,b)$$

一般に $T(R)T(S) / \max\{V(R,b), V(S,b)\}$

ジョイン結果のレコード数を近似

R(a,b)	S(b,c)	U(c,d)
T(R)=1,000	T(S)=2,000	T(U)=5,000
V(R,b)=20	V(S,b)=50	
	V(S,c)=100	V(U,c)=500

R(a,b) ▷◁ S(b,c)
T(R)T(S) / max{V(R,b),V(S,b)} = 40,000

(R(a,b) ▷◁ S(b,c)) ▷◁ U(c,d)
T(R▷◁S)T(U) / max{V(R▷◁S,c),V(U,c)}

仮定 3（属性値の保存）
R▷◁S で属性 c の値はジョイン後もすべて残ると仮定
つまり V(R▷◁S,c) = V(S,c)

= 40,000×5,000 / max{100,500} = 400,000

近似値がジョイン順序に依存しないこと

R(a,b)	S(b,c)	U(c,d)
T(R)=1,000	T(S)=2,000	T(U)=5,000
V(R,b)=20	V(S,b)=50	
	V(S,c)=100	V(U,c)=500

S(b,c) ▷◁ U(c,d)
T(S)T(U) / max{V(S,c),V(U,c)} = 20,000

R(a,b) ▷◁ (S(b,c) ▷◁ U(c,d))
T(R)T(S▷◁U) / max{V(R,b),V(S▷◁U,b)}

属性値は保存されると仮定　V(S▷◁U,b)=V(S,b)

＝ 1,000×20,000 / max{20,50} = 400,000

複数属性上の Natural Join のレコード数を近似

$R(a,b1,b2) \triangleright\triangleleft S(b1,b2,c)$ のレコード数

$$\frac{T(R)T(S)}{\max\{V(R,b1), V(S,b1)\}\ \max\{V(R,b2), V(S,b2)\}}$$

R の属性 b1 の値が、S のレコードに現れる確率:
属性値の包含と、均等分布を仮定すると

- $V(R,b1) \geqq V(S,b1)$ の場合
  $(V(S,b1) / V(R,b1)) \times (1 / V(S,b1)) = 1 / V(R,b1)$

- $V(R,b1) < V(S,b1)$ の場合: $1 / V(S,b1)$
- つまり $1 / \max\{V(R,b1), V(S,b1)\}$

b2 の場合も同様に、 確率は $1 / \max\{V(R,b2), V(S,b2)\}$

最後に b1 と b2 の値は独立に分布すると仮定

複数属性上の Natural Join のレコード数を近似

例　　R(a,b1,b2)　　S(b1,b2,c)
T(R)＝1000　　T(S)＝2000
V(R,b1)＝20　　V(S,b1)＝50
V(R,b2)＝100　　V(S,b2)＝50

R(a,b1,b2) ▷◁ S(b1,b2,c) のレコード数

$$\frac{T(R)T(S)}{\max\{V(R,b1),\ V(S,b1)\}\ \max\{V(R,b2),\ V(S,b2)\}}$$

＝ 1000×2000 / 50×100
＝ 400

近似値がジョイン順序に依存しないこと

R(a,b)	S(b,c)	U(c,d)
T(R)=1,000	T(S)=2,000	T(U)=5,000
V(R,b)=20	V(S,b)=50	
	V(S,c)=100	V(U,c)=500

$R(a,b) \triangleright\triangleleft U(c,d)$ $T(R)T(U) = 5{,}000{,}000$

$(R(a,b) \triangleright\triangleleft U(c,d)) \triangleright\triangleleft S(b,c)$

$$\frac{T(R \triangleright\triangleleft U)\ T(S)}{\max\{V(R \triangleright\triangleleft U,b), V(S,b)\}\ \max\{V(R \triangleright\triangleleft U,c), V(S,c)\}}$$

$V(R \triangleright\triangleleft U,b)=V(R,b)$, $V(R \triangleright\triangleleft U,c)=V(U,c)$ 属性値の保存より

$= (5{,}000{,}000 \times 2{,}000) / (\max\{20,50\}\ \max\{500,100\})$
$= 400{,}000$

n個の関係をジョインした結果のレコード数を近似

$$S = R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n$$

属性 a が k 個の関係 $R_1, R_2, R_3, \ldots, R_k$ に現れ、しかも

$$V(R_1,a) \leqq V(R_2,a) \leqq V(R_3,a) \leqq \ldots \leqq V(R_k,a)$$

であると、ジョインの交換律より、一般性を失わず仮定

$R_1, R_2, R_3, \ldots, R_k$ から各々レコードを選択したとき、
属性 a の値が一致する確率 P(a) は

$$\frac{1}{V(R_2,a)\ V(R_3,a)\ \ldots\ V(R_k,a)}$$

ただし　属性値の包含と、均等分布を仮定

S の大きさは

$T(R_1)T(R_2)\ldots\ T(R_n) \times \prod \{P(a_j) \mid a_j$ は複数回出現する属性$\}$

この近似値はジョインの順序に依存しない

n個のジョイン結果のレコード数を近似

R(a,b,c)
T(R)＝1000
V(R,a)＝100
V(R,b)＝20
V(R,c)＝200

S(b,c,d)
T(S)＝2000
V(S,b)＝50
V(S,c)＝100
V(S,d)＝400

U(b,e)
T(U)＝5000
V(U,b)＝200
V(U,e)＝500

$$1000 \times 2000 \times 5000 \times \frac{1}{50 \times 200} \times \frac{1}{200}$$

属性b　　属性c

＝5000

問合せコスト近似用のパラメータ

- 計算が容易なパラメータ

  T(R): 関係 R のレコード数
  V(R,a): 属性 a における、異なる値の数

- V(R,a) が小さい場合、
  値ごとに頻度分布をとることで、より正確に近似可能

頻度分布を利用してジョインのレコード数を近似

R(a,b)

b の値	頻度
0	150
1	200
5	100
その他	550
合計	1,000

S(b,c)

b の値	頻度
0	100
1	80
2	70
その他	250
合計	500

R(a,b) ▷◁ S(b,c)

b の値	頻度
0	150×100 = 15,000
1	200×80 = 16,000
2	50×70 = 3,500
5	100×25 = 2,500
その他	50×25×9 = 11,250
合計	48,250

V(R,b) = 14
他の値の数は11個

V(S,b) = 13
他の値の数は10個

b の値	頻度
2	550/11=50

b の値	頻度
5	250/10=25

0, 1, 2, 5 以外に現れる異なる b の値の数は、
R が10個、Sが9個。 両方に現れるのは9個。

頻度分布が均一と
考える従来の近似では

1,000×500/14=35,714

ヒストグラムを利用した近似

過去10年間で、最高気温が同一の3月と7月の日を列挙せよ

March(day, temp)
July(day, temp)

```
SELECT March.day, July.day
FROM March, July
WHERE March.temp = July.temp
```

15-19 度　20×5/5　= 20 レコード
20-24 度　10×20/5 = 40 レコード
合計　60 レコード

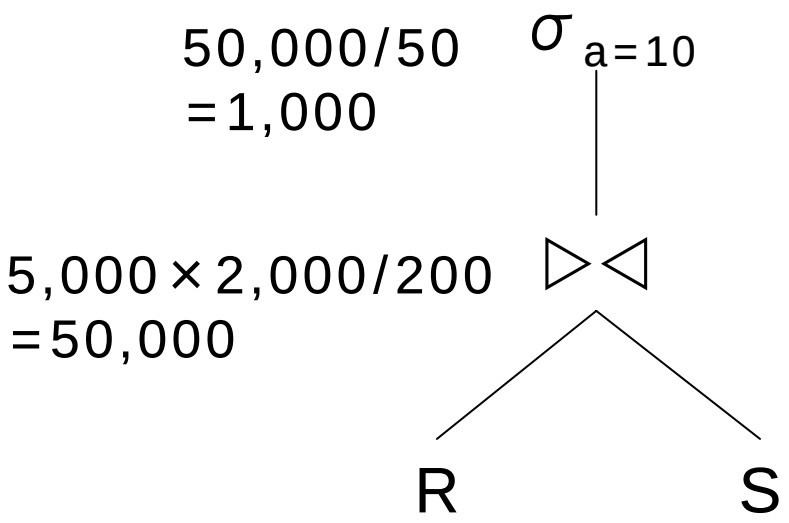
均等な頻度分布を仮定して近似すると
310×310 / 40 = 2402.5

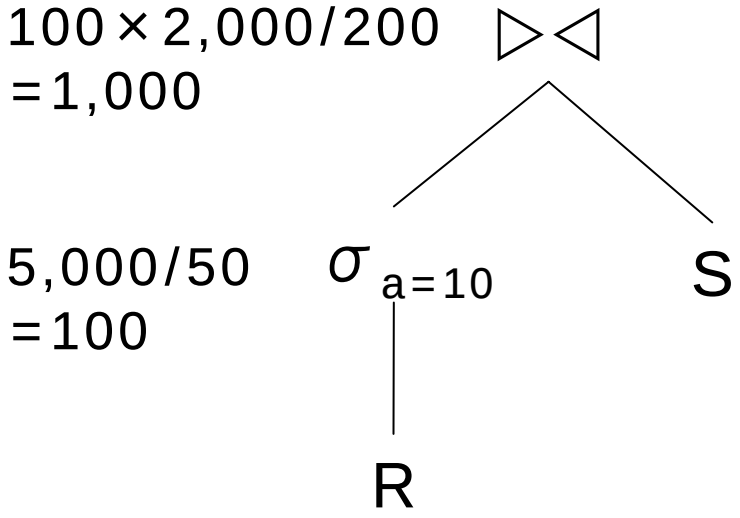
最高気温（摂氏）	3月の日数	7月の日数
0–4	50	0
5–9	130	0
10–14	100	0
15–19	20	5
20–24	10	20
25–29	0	110
30–34	0	160
35–39	0	15
合計	310	310

問合せ木の代数的変換と最終結果の近似

R(a,b)	S(b,c)
T(R)=5,000	T(S)=2,000
V(R,a)=50	
V(R,b)=100	V(S,b)=200
	V(S,c)=100

$\sigma_{a=10}(R(a,b) \triangleright\triangleleft S(b,c))$　　$(\sigma_{a=10}R(a,b)) \triangleright\triangleleft S(b,c)$





トランザクション管理

参考文献

Jeffrey D. Ullman: *Principles of Database and Knowledge-base Systems*
Volume I, Computer Science Press, 1988. ISBN 0-7167-8158-1

Chapter 9　Transaction Management

トランザクション

データの読み出しや書き込みからなる処理の単位
並列に実行される

トランザクション管理の重要性

更新が必要なデータベース
- 座席予約システム
- ダブルブッキングの回避
- 高速な処理時間よりもデータ更新の安全性を確保したい

読み出すことが中心のデータベース
- 統計データベース
- WWWデータベース
- 並列な読み出し要求を数多く処理したい

例題　普通預金口座　A

T1: 1000円を振り込むトランザクション

```
READ A;
A := A + 1000;
WRITE A;
```

T2: 1000円を引き出すトランザクション

```
READ A;
A := A − 1000;
WRITE A;
```

スケジュール：　時間軸に沿ってステートメントを並べた列

	A
	4000
T1: READ A	
T1: A := A+1000	
T1: WRITE A	5000
T2: READ A	
T2: A := A−1000	
T2: WRITE A	4000

順番に実行するスケジュール

	A
	4000
T1: READ A	
T1: A := A+1000	
T2: READ A	
T2: A := A−1000	
T1: WRITE A	5000
T2: WRITE A	3000

おかしな結果！

例題
普通預金口座　A, B

T1:　AからBへ
2000円送金

```
READ A;
IF A >= 2000
    A := A - 2000;
WRITE A;

READ B;
B := B + 2000;
WRITE B;
```

T2:　BからAへ
5000円送金

```
READ A;
A := A + 5000;
WRITE A;

READ B;
IF B >= 5000
    B := B - 5000;
WRITE B;
```

T1, T2 を実行する過程で A と B の値が変化する様子

	A	B
	4000	4000
T1	2000	
T1		6000
T2	7000	
T2		1000

	A	B
	4000	4000
T2	9000	
T1	7000	
T1		6000
T2		1000

	A	B
	4000	4000
T2	9000	
T1	7000	
T2		abort

abort: 実行不能でトランザクションを消滅させる操作
T2の影響を元に戻す必要あり

トランザクションの原子性（Atomicity）の保証

整列スケジュールでは各トランザクションを
原子のような塊として実行する。

原子性を保証するとは、
各トランザクションを並列に実行するとき、最終結果が、
ある整列スケジュールの
結果と一致することを保証すること。

例

	A	B
	4000	4000
T2	9000	
T1	7000	
T2		abort
T2復帰	7000−5000	
	=2000	
T1		6000

	A	B
	4000	4000
T1	2000	
T1		6000

整列スケジュール

問題

T1: 1000円を振り込むトランザクション

```
READ A;
A := A + 1000;
WRITE A;
```

T2: 1000円を引き出すトランザクション

```
READ A;
A := A − 1000;
WRITE A;
```

A の値が変化する様子を示せ

	A
	4000
T2: READ A	
T2: A := A−1000	
T1: READ A	
T1: A := A+1000	
T2: WRITE A	
T1: WRITE A	

	A
	4000
T2: READ A	
T2: A := A−1000	
T1: READ A	
T1: A := A+1000	
T1: WRITE A	
T2: WRITE A	

T1: AからBへ
2000円送金

```
READ A;
IF A >= 2000
    A := A - 2000;
WRITE A;

READ B;
B := B + 2000;
WRITE B;
```

T2: BからAへ
5000円送金

```
READ A;
A := A + 5000;
WRITE A;

READ B;
IF B >= 5000
    B := B - 5000;
WRITE B;
```

問題　A, B の値が変化する様子を示せ

	A	B
	4000	4000

```
T1: READ A;
T1: IF A >= 2000
T1:     A := A - 2000;
T2: READ A;
T2: A := A + 5000;
T2: WRITE A;
T1: WRITE A;

T1: READ B;
T1: B := B + 2000;
READ B;
T2: IF B >= 5000
T2:     B := B - 5000;
T2: WRITE B;
T1: WRITE B;
```

Lock

データベースの更新単位をアイテム、
アイテムの大きさを粒度（granularity）と呼ぶ

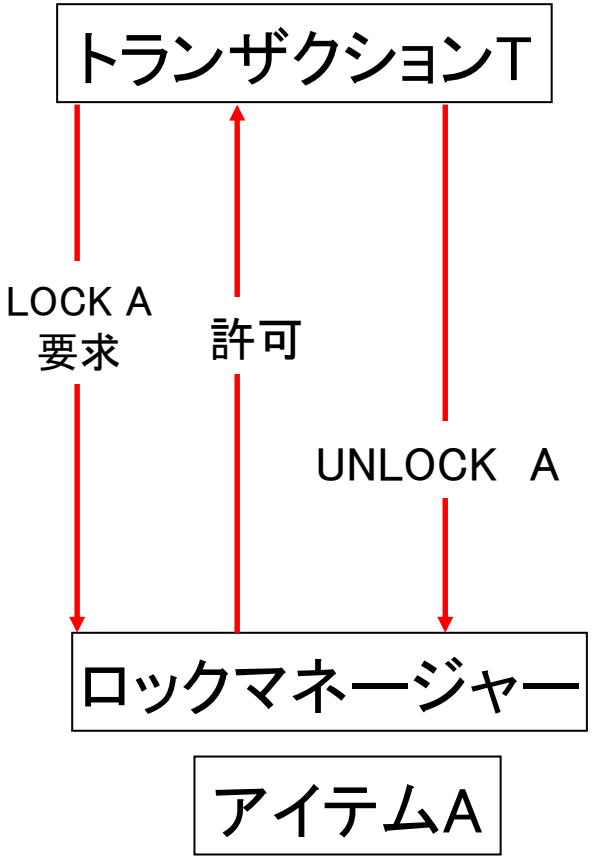
アイテムへのアクセスはロックにより管理
排他的なロックのかかっているアイテムには
他のトランザクションのアクセスを禁止

アイテムの粒度が大きい場合、
ロック管理システムの負担は軽くなるが、
同時に実行できるトランザクション数は減る

粒度の選択は典型的トランザクションを考えてきめる
預金口座のように小さい単位でトランザクションが
実行される場合は粒度は小さくする

ロックは様々な種類があるが当面は一種類とし、
アイテムAに対するロックを　LOCK　A　と記述

- トランザクション T はアイテム A に READ/WRITEする前に　LOCK　A　をロックマネージャーに要求
- ロックマネージャーはアイテム A にロックがかかっていない場合に T が LOCK Aすることを許可
- Aがロックされている間は他のトランザクションはAにアクセスできない
- TはAに対する処理を終了したら、ロックマネージャーに　UNLOCK　A　を要求し、ロックマネージャーはアイテムAに対するロックを解除



例題　普通預金口座　A

T1: 1000円を振り込むトランザクション

```
LOCK A;
READ A;
A := A + 1000;
WRITE A;
UNLOCK A;
```

T2: 1000円を引き出すトランザクション

```
LOCK A;
READ A;
A := A − 1000;
WRITE A;
UNLOCK A;
```

	A
	4000
T1: LOCK A	
T1: READ A	
T1: WRITE A	5000
T1: UNLOCK A	
T2: LOCK A	
T2: READ A	
T2: WRITE A	4000
T2: UNLOCK A	

	A
	4000
T2: LOCK A	
T2: READ A	
T2: WRITE A	3000
T2: UNLOCK A	
T1: LOCK A	
T1: READ A	
T1: WRITE A	4000
T1: UNLOCK A	

	A
	4000
T1: LOCK A	
T1: READ A	
~~T2: LOCK A~~	
~~T2: READ A~~	

問題

T1: 1000円を振り込むトランザクション

```
LOCK A;
READ A;
A := A + 1000;
WRITE A;
UNLOCK A;
```

T2: 1000円を引き出すトランザクション

```
LOCK A;
READ A;
A := A − 1000;
WRITE A;
UNLOCK A;
```

LOCK の利用により下のスケジュールが生み出す問題点が回避されることを示せ

	A
	4000
T2: READ A	
T2: A := A−1000	
T1: READ A	
T1: A := A+1000	
T2: WRITE A	
T1: WRITE A	

各トランザクションは
一つのアイテムに高々一回だけLOCKをかける

```
T
LOCK A;
A := A+1;
UNLOCK A;
LOCK B;
LOCK A;
B := B+A;
UNLOCK A;
UNLOCK B;
```

```
T
LOCK A;
A := A+1;
LOCK B;
B := B+A;
UNLOCK A;
UNLOCK B;
```

Livelock

トランザクション T_i

```
LOCK A;
READ A;
A := A + i;
WRITE A;
UNLOCK A;
```

T1: LOCK A;	T1が最初にAをロック
T1: READ A;	
T1: A := A+1;	
T1: WRITE A;	T2, T3 が　LOCK A　を要求
T1: UNLOCK A;	
T3: LOCK A;	ロックマネージャーがT3にロックを許可
T3: READ A;	
T3: A := A+1;	T4 が　LOCK A　を要求 T2 は待ち続けている
T3: WRITE A;	
T3: UNLOCK A;	
T4: LOCK A;	ロックマネージャーが T4 にロックを許可
T4: READ A;	
T4: A := A+1;	
T4: WRITE A;	
T4: UNLOCK A;	

簡単な回避方法（First-come-first-served strategy）
LOCK　A　を要求するトランザクションに要求順にロックを許可

Deadlock

```
T1:  AからBへ
2000円送金

LOCK A;
LOCK B;

READ A;
IF A >= 2000
    A := A - 2000;
WRITE A;
READ B;
B := B + 2000;
WRITE B;

UNLOCK A;
UNLOCK B;
```

```
T2:  BからAへ
5000円送金

LOCK B;
LOCK A;

READ A;
A := A + 5000;
WRITE B;
READ B;
IF B >= 5000
    B := B - 5000;
WRITE B;

UNLOCK B;
UNLOCK A;
```

T1: LOCK A;　許可
T2: LOCK B;　許可

T1: LOCK B;　不許可
T2: LOCK A;　不許可

T1とT2は永遠に
待ちつづける

プロトコルを工夫してDeadlock を起こさない手法1

各トランザクションは同時にすべてのロック要求を出す
ロックマネージャーは、この全要求を許可するか、全てを不許可

```
T1:  AからBへ
2000円送金

LOCK A;
LOCK B;
READ A;
IF A >= 2000
     A := A - 2000;
WRITE A;
READ B;
B := B + 2000;
WRITE B;
UNLOCK A;
UNLOCK B;
```

```
T2:  BからAへ
5000円送金

LOCK B;
LOCK A;
READ A;
A := A + 5000;
WRITE B;
READ B;
IF B >= 5000
     B := B - 5000;
WRITE B;
UNLOCK B;
UNLOCK A;
```

T1: LOCK A　許可
T1: LOCK B　許可

T1: UNLOCK A
T1: UNLOCK B

T2: LOCK B　許可
T2: LOCK A　許可

プロトコルを工夫してDeadlock を起こさない手法2

- アイテムに順序を導入

A ＞ B

- 各トランザクションは
  順序の大きい順に
  アイテムをロック

T1: AからBへ
2000円送金

LOCK A;
LOCK B;
:

T2: BからAへ
5000円送金

LOCK B;
LOCK A;
:

T2: BからAへ
5000円送金

LOCK A;
LOCK B;
:

T1の実行後にT2が処理される

アイテムに順序を入れる方法がDeadlock を起こさない理由

背理法： Deadlock 状態のトランザクション集合があると仮定

前処理： アイテムをロックしていないトランザクションを
除いても、集合は依然として Deadlock状態

観察： 各トランザクション T_i は他のトランザクション T_{i+1} が
ロックしているアイテム A_i の開放を待っている

$T_i \xrightarrow[A_i]{} T_{i+1}$

注意
T_i は A_i より順序の
大きいアイテムをロック

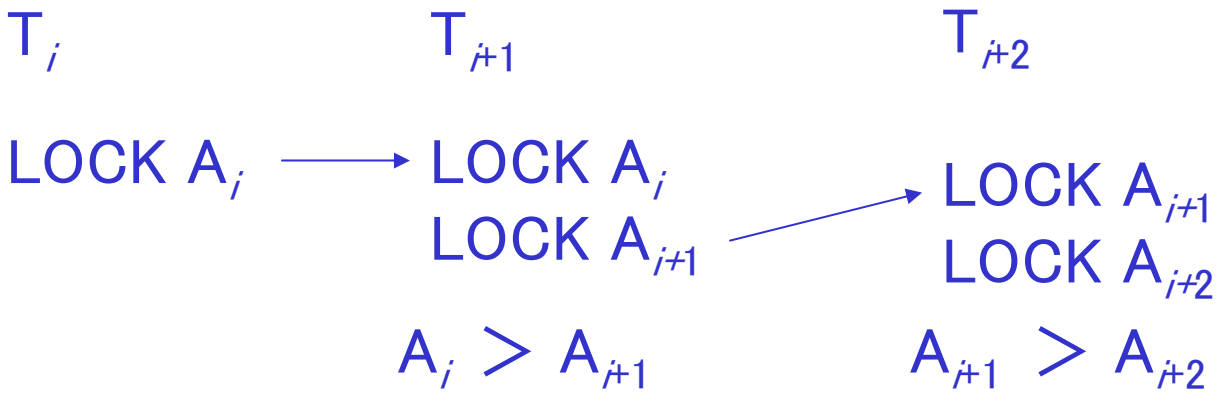
T_i	T_{i+1}	T_{i+2}
LOCK A_i →	LOCK A_i	
	LOCK A_{i+1} →	LOCK A_{i+1}
		LOCK A_{i+2}
	$A_i > A_{i+1}$	$A_{i+1} > A_{i+2}$

ある T_1 を選択して、鎖を構成

$A_1 > A_2 > \cdots\cdots > A_{n-1}$

ループしない有限の下降鎖

T_n は,いつか A_{n-1} を UNLOCK.
アイテムをロック
していない T_n が必ず
存在し、矛盾



スケジューラーがDeadlock を検知して abort 等で対処する方法

Waits-for Graph

トランザクションがノード

T1がロックしているアイテムA
のUNLOCKをT2が待っている
とき、「T1の次はT2」を表現する
ためT1からT2へ有向辺を引き
Aをラベルとしてつける

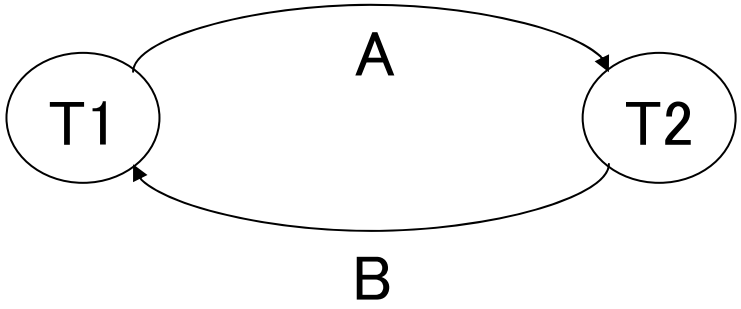
閉路の存在は Deadlockを意味

たとえば　T2 を abort して B への
ロックを開放し、T1 へ渡す

```
T1:  AからBへ
2000円送金

LOCK A;
LOCK B;
:
UNLOCK A;
UNLOCK B;
```

```
T2:  BからAへ
5000円送金

LOCK B;
LOCK A;
:
UNLOCK B;
UNLOCK A;
```

T1: LOCK A;　許可
T2: LOCK B;　許可



問題

T1	T2	T3
LOCK A	LOCK B	LOCK C
LOCK B	LOCK C	LOCK A
A := A+B;	B := B+C;	C := C+A;
UNLOCK A	UNLOCK B	UNLOCK C
UNLOCK B	UNLOCK C	UNLOCK A

1. LOCK文をアイテムのアルファベッド順で並べる方式により、deadlock を回避することを考える。T1, T2, T3 から同時にロック要求があり、最初に T3 が実行が認められたとする。2番目、3番目に実行されるトランザクションとスケジュールを述べよ。

2. Waits-for graph により deadlock を解消することを考える。T1, T2, T3 から同時にロック要求があり、wait-for graph に閉路ができたとき、T3 を abort したとする。1番目、2番目に実行されるトランザクションを述べよ。

整列化可能性
(Serializability)

トランザクションの集合をある順序で整列化した列を

$T_1\ T_2\ \ldots\ldots\ T_k$

i=1,2,…,k の順番で T_i のステートメントを実行する
スケジュールを整列スケジュール (serial schedule) と呼ぶ

トランザクションをプログラムしたときのユーザの心境は
各トランザクションの実行を他のトランザクションが
邪魔しないことを仮定している

つまりトランザクション全体がある整列スケジュールとして
実行されることを思ってプログラムする

しかしシステムは効率を重視してトランザクションの
順番を入れ替えて実行するかもしれない

入れ替えをしても、その実行結果は
ある整列スケジュールの実行結果と一致してほしい

スケジュールが整列化可能とは、実行結果が
ある整列スケジュールの実行結果と一致することと定義する

この概念をどのように定式化するか？

T1: Aのx倍をBへ振り込む

```
LOCK A;
LOCK B;
READ A;
READ B;
B := B+A * x;
WRITE B;
UNLOCK A;
UNLOCK B;
```

T2: Bのx倍をAへ振り込む

```
LOCK B;
READ B;
UNLOCK B;
LOCK A;
READ A;
A := A+B * x;
WRITE A;
UNLOCK A;
```

T2: LOCK B;
T2: READ B;
T2: UNLOCK B;

T1: LOCK A; LOCK B;
T1: READ A; READ B;
T1: B := B+A * x; WRITE B;
T1: UNLOCK A; UNLOCK B;

T2: LOCK A;
T2: READ A;
T2: A := A+B * x;
T2: WRITE A;
T2: UNLOCK A;

整列スケジュール

T1 B:=B+A*x
T2 A:=A+B*x

T2 A:=A+B*x
T1 B:=B+A*x

A=B=1000
x=2

T1: B=3000
T2: A=7000

T2: A=3000
T1: B=7000

T2: B=1000
T1: B=3000
T2: A=3000

整列化不可能

A=0
B=1000
x=2

T1: B=1000
T2: A=2000

T2: A=2000
T1: B=5000

T2: B=1000
T1: B=1000
T2: A=2000

整列化可能？

スケジュール S が整列化可能を、「どのような初期状態に対しても、S と実行結果が一致する整列スケジュールが存在すること」と定義すると困る…

初期状態の候補は無限にあり、有限ステップでチェックできるか？
帰納的関数がある場合、2つの帰納的関数の等価性は一般に決定不能

スケジュール S が整列化可能とは、
どのような初期状態に対しても、どのような実行文に対しても、S と実行結果が一致する整列スケジュールは存在すること。

どのような実行文に対しても等価となる整列スケジュールは存在するか？

T1: Aのx倍をBへ振り込む

LOCK A;
LOCK B;

UNLOCK A;
UNLOCK B;

T2: Bのx倍をAへ振り込む

LOCK B;

UNLOCK B;
LOCK A;

UNLOCK A;

整列スケジュール

T2: LOCK B;
T2: UNLOCK B;

T1: LOCK A;
T1: LOCK B;
T1: UNLOCK A;
T1: UNLOCK B;

T2: LOCK A;
T2: UNLOCK A;

T1: LOCK A;
T1: LOCK B;
T1: UNLOCK A;
T1: UNLOCK B;

T2: LOCK B;
T2: UNLOCK B;
T2: LOCK A;
T2: UNLOCK A;

T2: LOCK B;
T2: UNLOCK B;
T2: LOCK A;
T2: UNLOCK A;

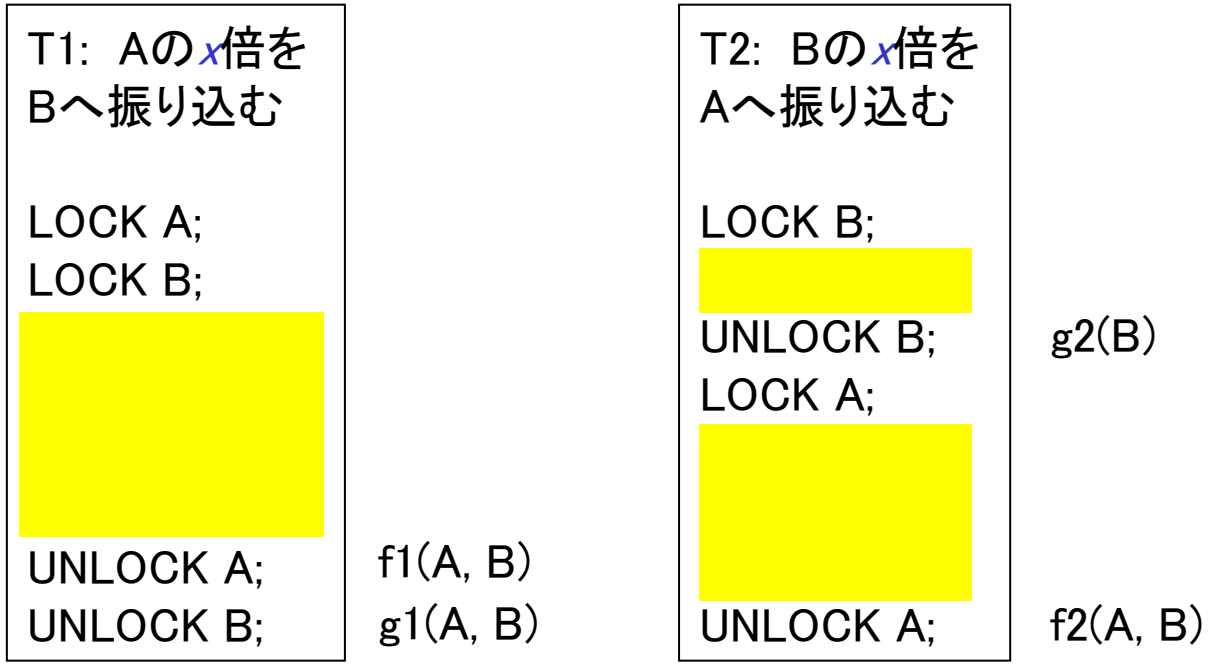
T1: LOCK A;
T1: LOCK B;
T1: UNLOCK A;
T1: UNLOCK B;

LOCK A と UNLOCK A
のペアに対して
新しい関数 f を割当てる

f の引数は UNLOCK A
の前にトランザクション中で
ロックされる全ての
アイテムの値

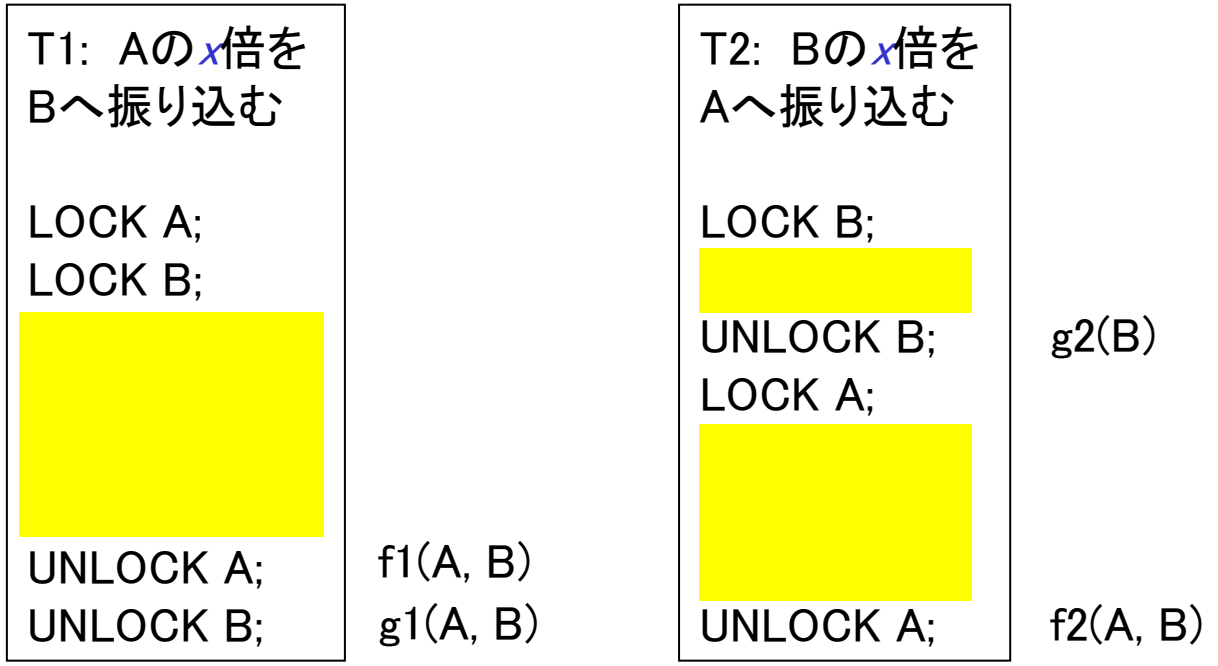
T1: Aの*x*倍を Bへ振り込む		T2: Bの*x*倍を Aへ振り込む	
LOCK A;		LOCK B;	
LOCK B;			
		UNLOCK B;	g2(B)
		LOCK A;	
UNLOCK A;	f1(A, B)		
UNLOCK B;	g1(A, B)	UNLOCK A;	f2(A, B)

		A	B
T2: LOCK B;		a	b
T2: UNLOCK B;	g2(B)		g2(b)
T1: LOCK A;			
T1: LOCK B;			
T1: UNLOCK A;	f1(A,B)	f1(a, g2(b))	
T1: UNLOCK B;	g1(A,B)		g1(f1(a, g2(b)), g2(b))
T2: LOCK A;		f2(f1(a,g2(b)), g2(b))	
T2: UNLOCK A;	f2(A,B)	~~f2(f1(a,g2(b)),~~	
		~~g1(f1(a, g2(b)), g2(b)))~~	



整列スケジュール1

	A	B
T1: LOCK A;	a	b
T1: LOCK B;		
T1: UNLOCK A;	f1(a,b)	
T1: UNLOCK B;		g1(f1(a,b), b)
T2: LOCK B;		
T2: UNLOCK B;		g2(g1(f1(a,b), b))
T2: LOCK A;		
T2: UNLOCK A;	f2(f1(a,b), g2(g1(f1(a,b), b)))	



整列スケジュール2

	A	B
T2: LOCK B;	a	b
T2: UNLOCK B;		g2(b)
T2: LOCK A;		
T2: UNLOCK A;	f2(a, g2(b))	
T1: LOCK A;		
T1: LOCK B;		
T1: UNLOCK A;	f1(f2(a,g2(b)), g2(b))	
T1: UNLOCK B;		g1(f1(f2(a,g2(b))), g2(b))

3つのスケジュールの最終結果

	A	B
	~~f2(f1(a,g2(b)),~~ ~~g1(f1(a, g2(b)), g2(b)))~~	
	f2(f1(a,g2(b)), g2(b))	g1(f1(a, g2(b)), g2(b))
整列スケジュール1	f2(f1(a,b), g2(g1(f1(a,b), b)))	g2(g1(f1(a,b), b))
整列スケジュール2	f1(f2(a,g2(b)), g2(b))	g1(f1(f2(a,g2(b))), g2(b))

記号列として異なる

記号列はUNLOCKの実行履歴を表現している

- 与えられたスケジュールと等価な
  整列スケジュールが見つかれば整列可能

- 全ての整列スケジュールを生成して検査する
  のは計算コストが大きい

  n個のトランザクションからは整列スケジュールが n! 個

問題

T1: LOCK B
T1: UNLOCK B

T2: LOCK A
T2: UNLOCK A
T2: LOCK B
T2: UNLOCK B

T3: LOCK A
T3: UNLOCK A
T3: LOCK B
T3: UNLOCK B

上の3つのトランザクションからなる以下の2つのスケジュールが整列可能か否か、UNLOCK に関数を割り当てる手法により調べよ

T2: LOCK A
T2: UNLOCK A
T3: LOCK A
T3: UNLOCK A
T1: LOCK B
T1: UNLOCK B
T2: LOCK B
T2: UNLOCK B
T3: LOCK B
T3: UNLOCK B

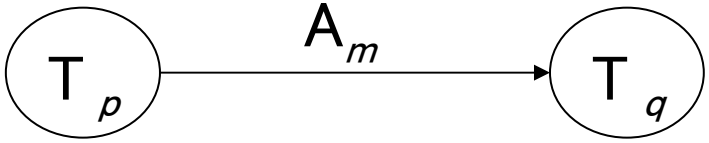
T1: LOCK B
T1: UNLOCK B
T2: LOCK A
T2: UNLOCK A
T2: LOCK B
T2: UNLOCK B
T3: LOCK A
T3: UNLOCK A
T3: LOCK B
T3: UNLOCK B

整列化可能性を判定するアルゴリズム

スケジュールの整列化可能性を判定するアルゴリズム

スケジュールは　T: LOCK A または T: UNLOCK A
の形をしたステートメントの列 $a_1, a_2, \ldots, a_n$ とする

整列化可能性判定グラフの構成
各 $a_i = T_p$: UNLOCK A_m について $a_j = T_q$: LOCK A_m （$p \neq q$）
となる $j > i$ が存在する場合、最小の j を選択し有向辺を張る



整列化可能な場合に対応する
整列スケジュール中で T_p は T_q の前にある

T_p を始点としA_mをラベルにもつ有向辺は高々1つ

定理　グラフに閉路がなければ整列化可能、あれば不可能

整列化可能性判定グラフ
閉路がない場合

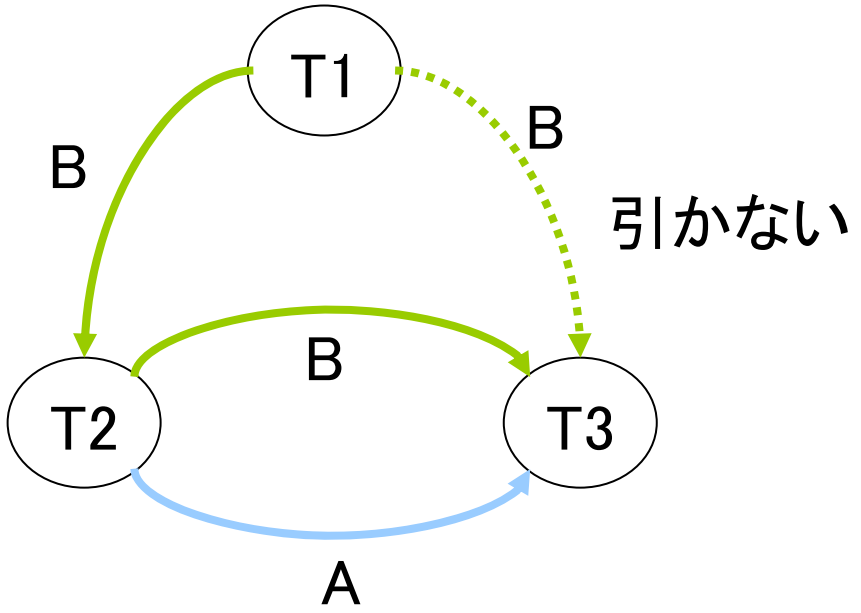
T2: LOCK A
T2: UNLOCK A

T3: LOCK A
T3: UNLOCK A

T1: LOCK B
T1: UNLOCK B

T2: LOCK B
T2: UNLOCK B

T3: LOCK B
T3: UNLOCK B



この関係は
考慮しない

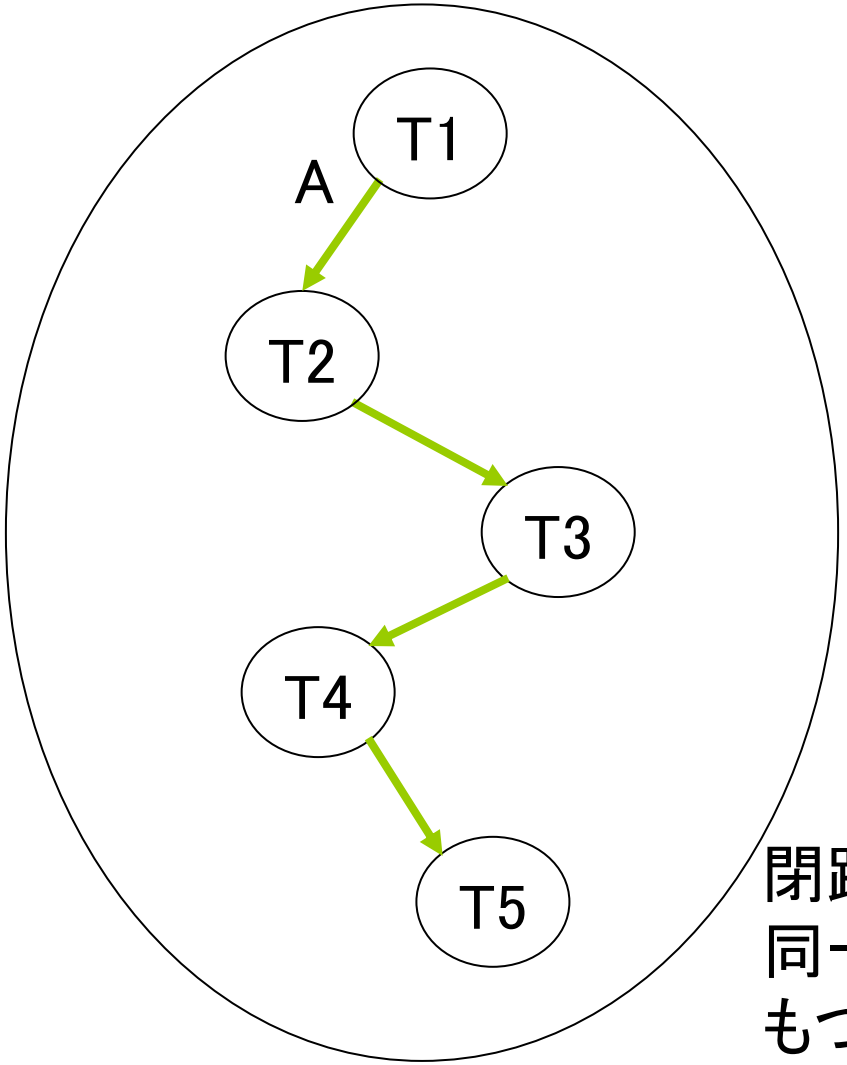
任意の始点から出る有向辺で
同一ラベルをもつ辺は高々一つ

各トランザクションは一つのアイテムに高々
一回だけLOCKをかける

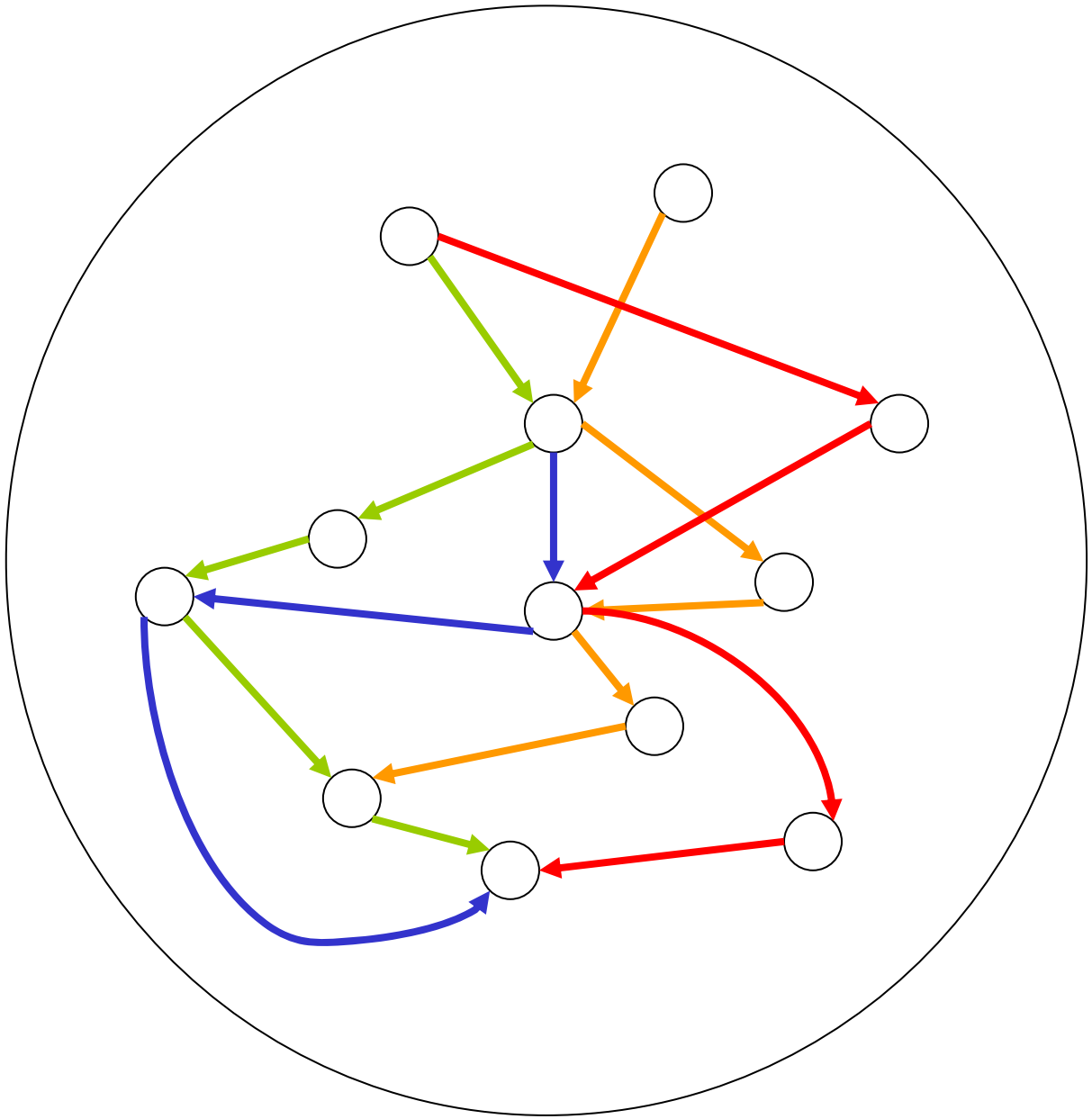
整列化可能性判定グラフ
閉路がない場合

与えられた
スケジュール

T1: LOCK A
T1: UNLOCK A
:
T2: LOCK A
T2: UNLOCK A
:
T3: LOCK A
T3: UNLOCK A
:
T4: LOCK A
T4: UNLOCK A
:
T5: LOCK A
T5: UNLOCK A



閉路のないグラフでは、
同一アイテムをラベルに
もつ有向辺は、連続した
列をつくる



一般には複数の
アイテムに対する
列が交差する

整列化可能性判定グラフ G の位相ソート

閉路がない場合には
Gにはいかなる有向辺の終点でないノード v が存在する

存在しないと仮定すれば、あるノードから有向辺を逆に辿る列をつくると、ノード数が有限なので、いつかは自分に戻る閉路ができる

v と v を始点とする有向辺を G から除く

このステップを繰返し、除かれたノード（トランザクション）列
T_1 T_2 …… T_k を位相ソート列と呼ぶ

位相ソート列を使って整列スケジュールを作る

与えられたスケジュール

T2: LOCK A
T2: UNLOCK A

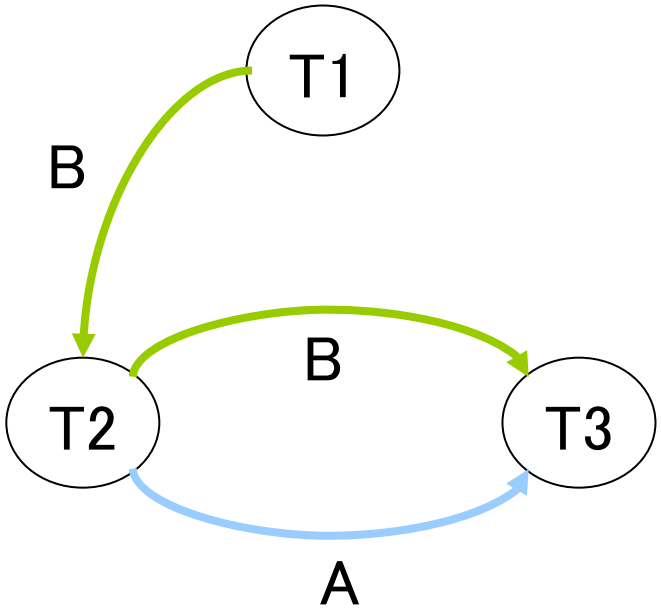
T3: LOCK A
T3: UNLOCK A

T1: LOCK B
T1: UNLOCK B

T2: LOCK B
T2: UNLOCK B

T3: LOCK B
T3: UNLOCK B

整列化可能性判定グラフ



位相ソート列　T1, T2, T3

整列スケジュール

T1: LOCK B
T1: UNLOCK B

T2: LOCK A
T2: UNLOCK A
T2: LOCK B
T2: UNLOCK B

T3: LOCK A
T3: UNLOCK A
T3: LOCK B
T3: UNLOCK B

問題

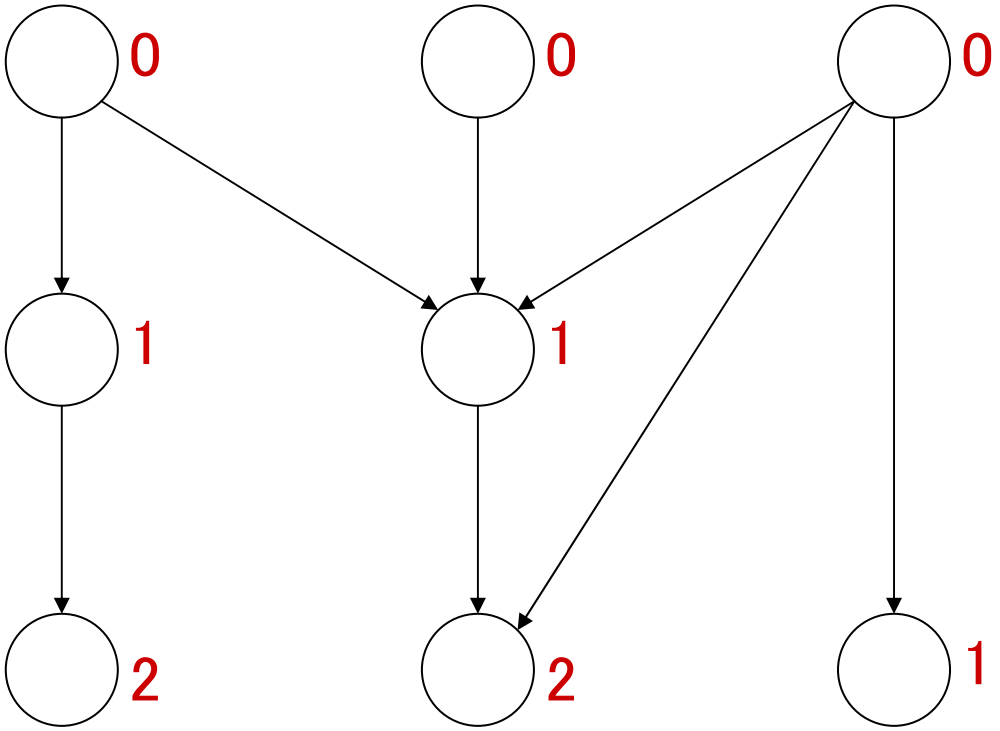
整列化可能性判定グラフを用いて以下の
スケジュールが整列化可能か否か求めよ

T2: LOCK B;
T2: UNLOCK B;
T1: LOCK A;
T1: LOCK B;
T1: UNLOCK A;
T1: UNLOCK B;
T3: LOCK A;
T3: UNLOCK A;
T2: LOCK A
T2: UNLOCK A

- 与えられたスケジュールの整列可能性を判定することは容易になった
- 現実にはトランザクションは動的に追加され、スケジュールは単調増加する
- 増加に応じて整列化可能性判定グラフを生成して閉路の存在検査をするべきか？　閉路が発見されたらそれまでの計算をやり直すのか？
- 整列化可能となるスケジュールを必ず生成できるトランザクションのプロトコル（生成規則）は考えられないか？

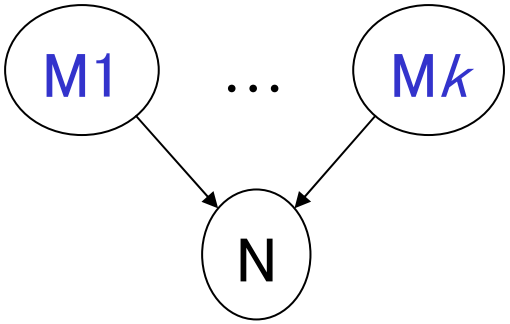
定理前半
スケジュール S の整列化可能性判定グラフ G が閉路を含まない場合、位相ソートでつくった整列スケジュール R は S と等価

準備



ノード N の深さ d(N)

N を終点とする有向辺の始点を N の隣接点

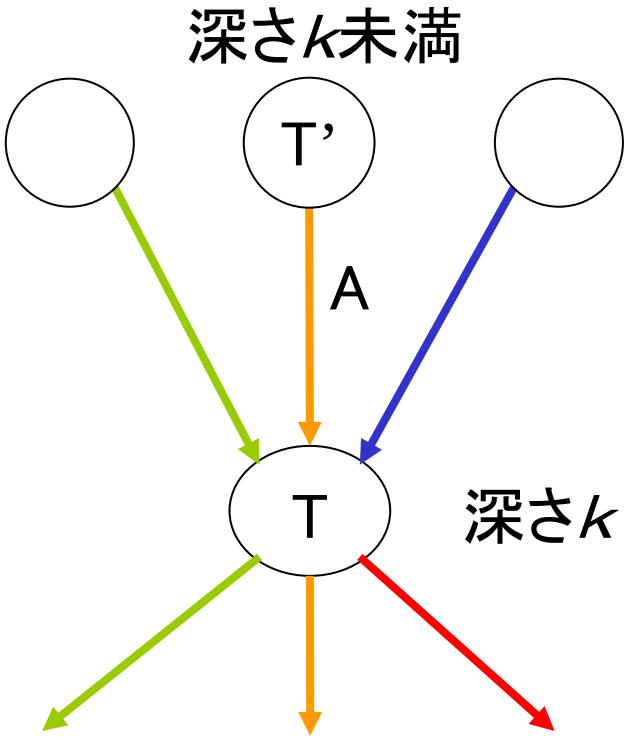


d(N) = 0　　Nに隣接点がない
= max{ d(M) | M は隣接点} + 1

スケジュール S と R で、同じトランザクションは
同一のアイテムに関して同じ値を読むことを証明

深さに関する帰納法

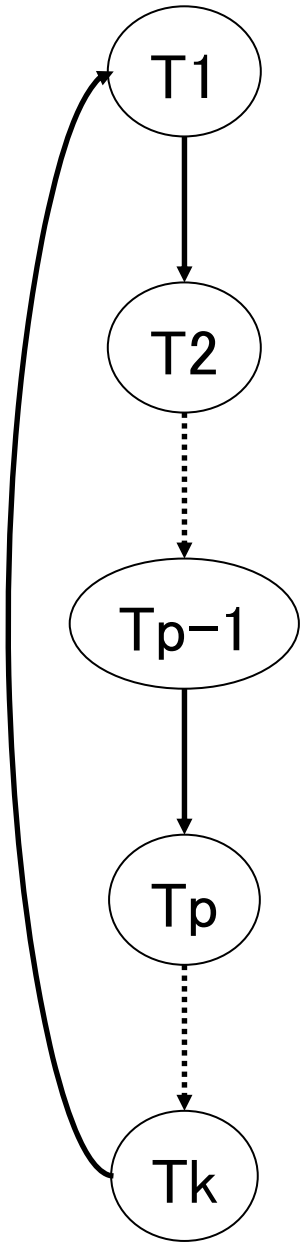
深さ0のとき、各アイテムの値は初期状態だから、OK



S, R で T がアイテム A を
LOCKする直前に A を UNLOCK
するトランザクションをT'とする

帰納法の仮定でS と R は
T' で A に同一の値を読み込む

S と R は T でも A に
同一の値を読み込む



定理後半　スケジュール S の整列化可能性判定
グラフが閉路を含む場合、Sは整列化不可能

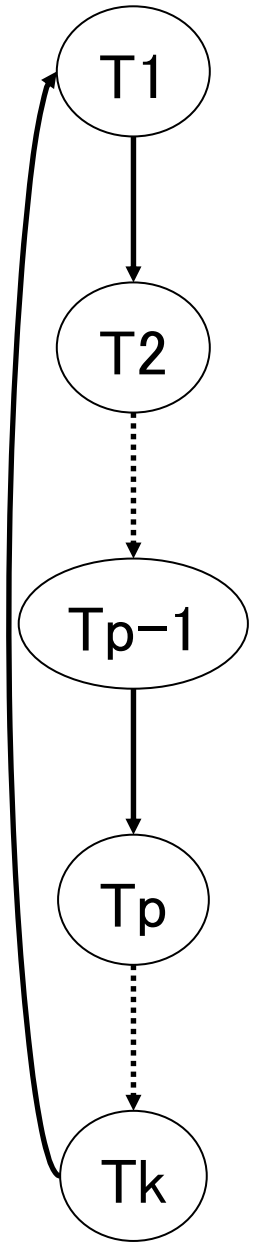
仮に等価な整列スケジュール R があるとする

Ti (i=1,…k)の中で Tp が R に最初に現れるとする

S	R
Tp−1: UNLOCK A　g(…)	
Tp: LOCK A	Tp: LOCK A
:	:
Tp: UNLOCK A　f(…)	Tp: UNLOCK A　f(…)

R では f(…) に g(…) は出現しない．Sでは出現.

R と S は等価でないので矛盾



もう一つ例

S		R	
Tp: LOCK A			
Tp−1: LOCK B			
Tp−1: UNLOCK B	g(B)		
		Tp: LOCK A	
Tp: LOCK B		Tp: LOCK B	
:		:	
Tp: UNLOCK A	f(A,B)	Tp: UNLOCK A	f(A,B)

R では f(…) に g(…) は出現しない．Sでは出現．

R と S は等価でないので矛盾

２相ロックプロトコル
(Two-phase Locking Protocol)

整列化可能となるスケジュール
を必ず生成するトランザクション
の生成規則（プロトコル）

２相ロックプロトコル

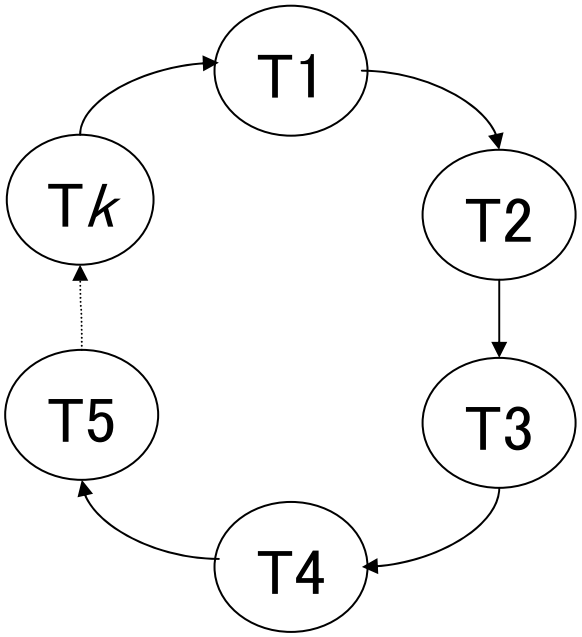
各トランザクションにおいて、すべてのLOCK文は、
すべてのUNLOCK文の前に実行する

定理
２相ロックプロトコルに従ってできた
スケジュール S は、必ず整列化可能

2相ロックの正当性の証明

背理法

整列化不可能な場合、整列化可能性判定グラフには閉路あり



T1 の UNLOCK 後に T2 の LOCK

T2 の UNLOCK 後に T3 の LOCK

:

T*k* の UNLOCK 後に T1 の LOCK

T1はUNLOCK後にLOCKしており
2相ロックプロトコルに従うことに矛盾

2相ロックプロトコルの最適性
条件は少しでも緩めると整列化可能性は失われる

T1 は2相ロックプロトコルに従わないとする

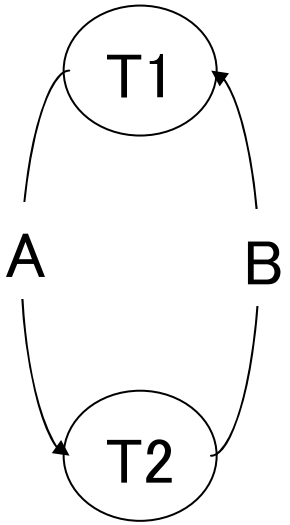
T1
LOCK A
:
UNLOCK A
:
LOCK B
:
UNLOCK B
:

T2
LOCK A
LOCK B
UNLOCK A
UNLOCK B

T1: LOCK A
:
T1: UNLOCK A

T2: LOCK A
T2: LOCK B
T2: UNLOCK A
T2: UNLOCK B

T1: LOCK B
:
T1: UNLOCK B



注意　2相ロックプロトコルで Deadlock を回避できるわけではない

```
T1:  AからBへ
2000円送金

LOCK A;
LOCK B;

READ A;
IF A >= 2000
     A := A − 2000;
WRITE A;
READ B;
B := B + 2000;
WRITE B;

UNLOCK A;
UNLOCK B;
```

```
T2:  BからAへ
5000円送金

LOCK B;
LOCK A;

READ A;
A := A + 5000;
WRITE B;
READ B;
IF B >= 5000
     B := B − 5000;
WRITE B;

UNLOCK B;
UNLOCK A;
```

T1: LOCK A;　許可
T2: LOCK B;　許可

T1: LOCK B;　不許可
T2: LOCK A;　不許可

T1とT2は永遠に
待ちつづける

まとめ

- 整列化可能性の概念を定義
- 整列化可能であることと
  整列化可能性判定グラフが閉路を含まないことは同値
- ２相ロックプロトコルは整列化可能の十分条件