並列分散コンピューティング
(2)グループ操作

大瀧保広

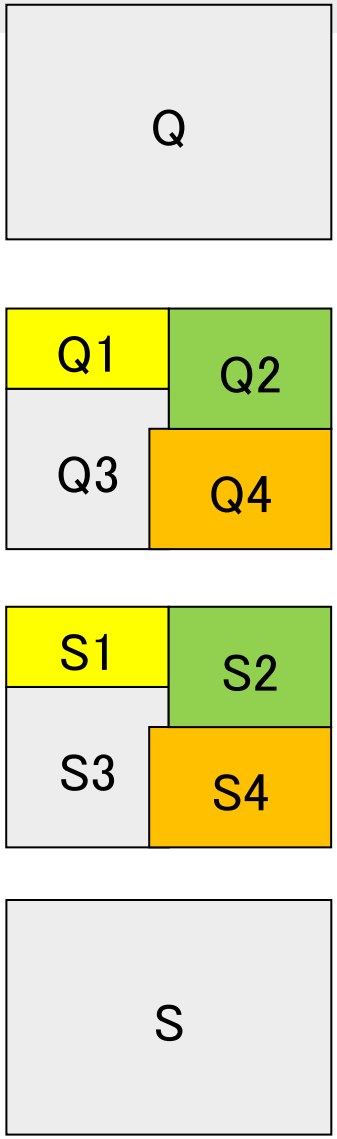
今日の内容

- 計算モデルとグループ操作

- 共有メモリモデルを用いたグループ操作
  - ブロードキャスト
  - リダクション
  - プレフィックス計算

- 相互結合ネットワーク上のグループ操作
  - 超立方体結合モデルを用いた基本的なグループ操作：
    1. ブロードキャスト
    2. リダクション
    3. プレフィックス計算
  - 2項木モデルを用いた基本的なグループ操作：
    1. ブロードキャスト
    2. リダクション
    3. プレフィックス計算

問題の並列解法



1. （前処理）
問題を部分問題に分割し、
各部分問題を各プロセッサに割り当てる。

2. （最大でプロセッサ数分の並列処理）
各プロセッサが部分問題を並列に解く。
部分解Siが求まる。

3. （後処理）
部分解を集めて統合し、
最終解 S を導く。

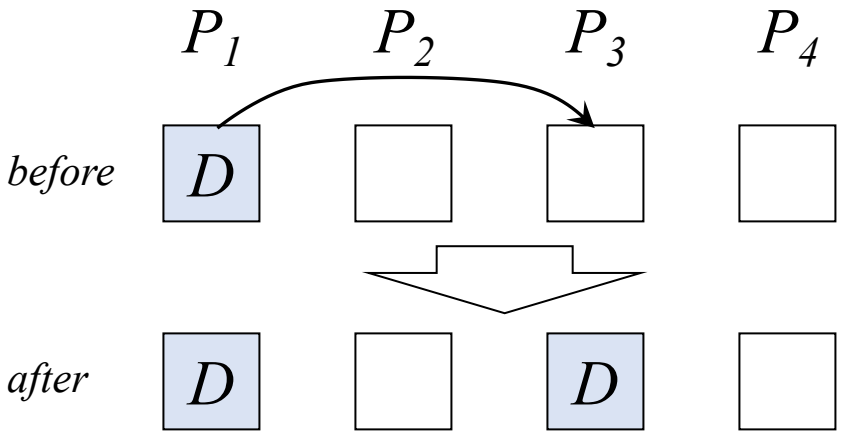
グループ操作とは

■並列アルゴリズムを
構築する際に必要となる、
複数のプロセッサが
関与する基本操作：
　　グループ処理、
　　集合操作、
　　集合通信、
　　集団通信
など。
書籍により呼び方は様々
(Group communication)

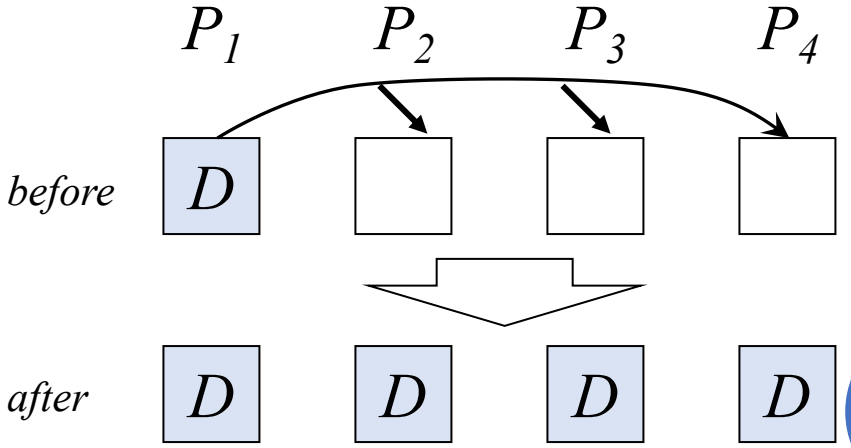
■ユニキャスト
■ブロードキャスト
■マルチキャスト
■分配
■収集
■全収集
■完全交換
■リダクション
■プレフィックス計算

基本的なグループ操作

■**ユニキャスト（Unicast）**
2プロセッサ間での
データ送受信

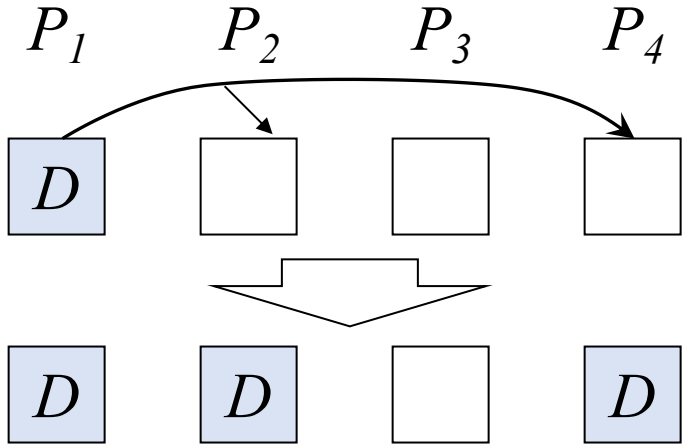


■**ブロードキャスト（Broadcast）**
すべてのプロセッサに
データ送信する

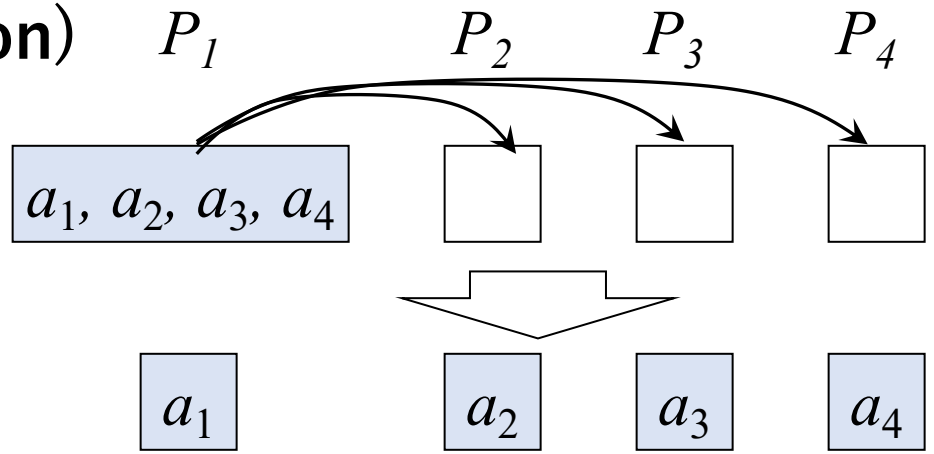


基本的なグループ操作（つづき）

■**マルチキャスト（Multicast）**
複数のプロセッサに
データを送信する。
ブロードキャストの部分操作。

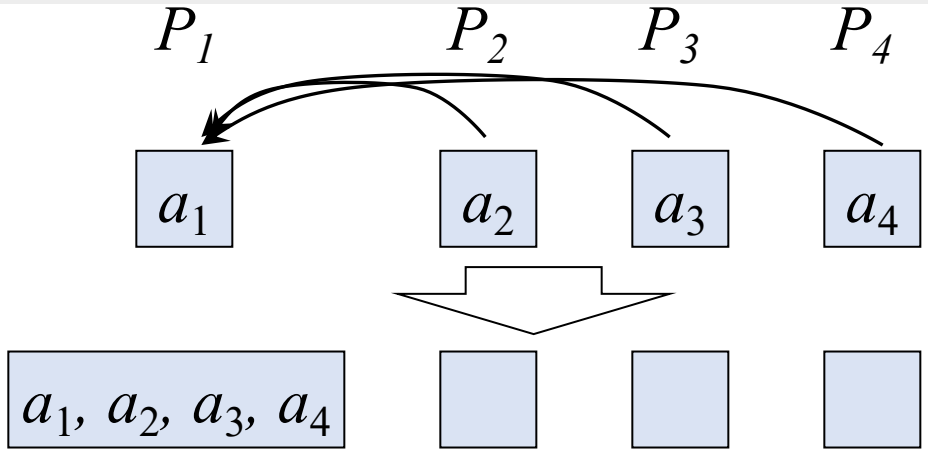


■**分配（Scatter; Distribution）**
各プロセッサにそれぞれ
異なるデータを送信する。

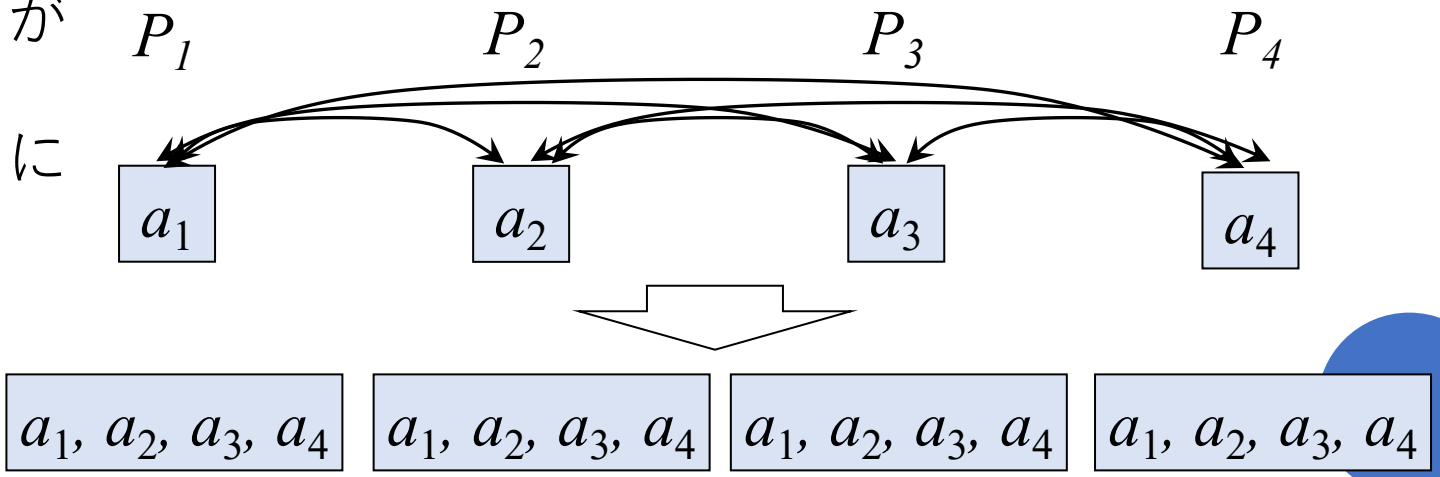


基本的なグループ操作（つづき）

■**収集 (Gather)**
あるプロセッサに
各プロセッサから
データを集める。
Scatterの逆。



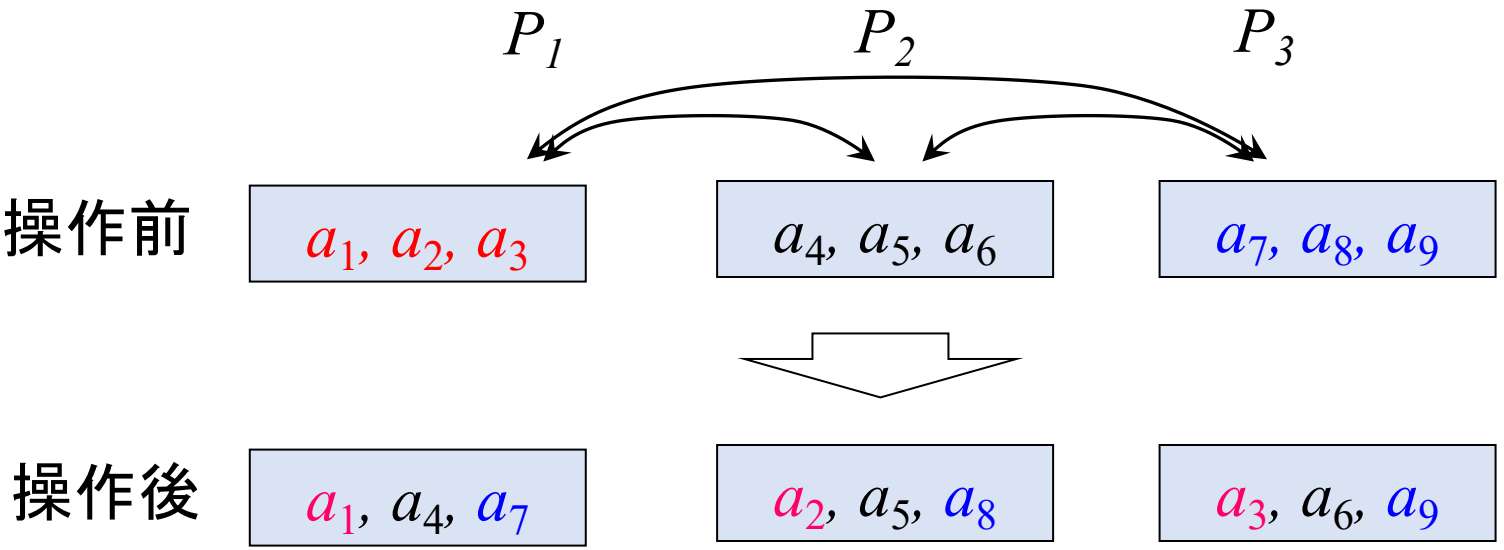
■**全収集**＝情報の共有
各プロセッサが
もつデータを
各プロセッサに
送受信する。



基本的なグループ操作（つづき）

■**完全交換(Complete exchange)**

各プロセッサのもつそれぞれのデータを収集し、分配する。
（行列転置のようなイメージ）

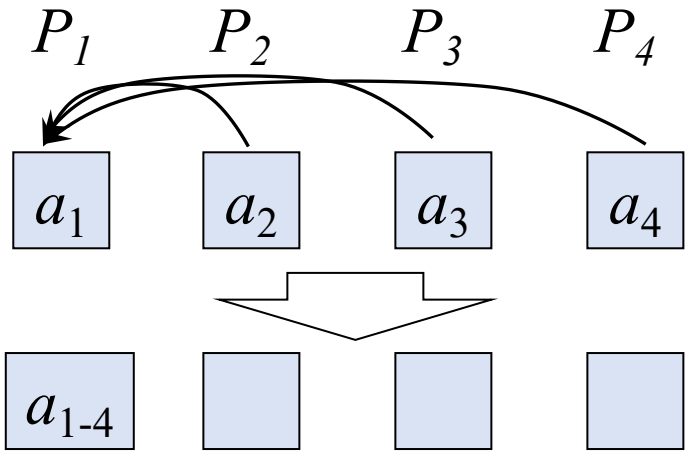


基本的なグループ操作（つづき）

■**リダクション (Reduction)**

各プロセッサのもつデータを2項演算し、
その結果をあるプロセッサで求める。

■2項演算：
算術演算（加算、乗算）
論理演算
最大値,最小値など



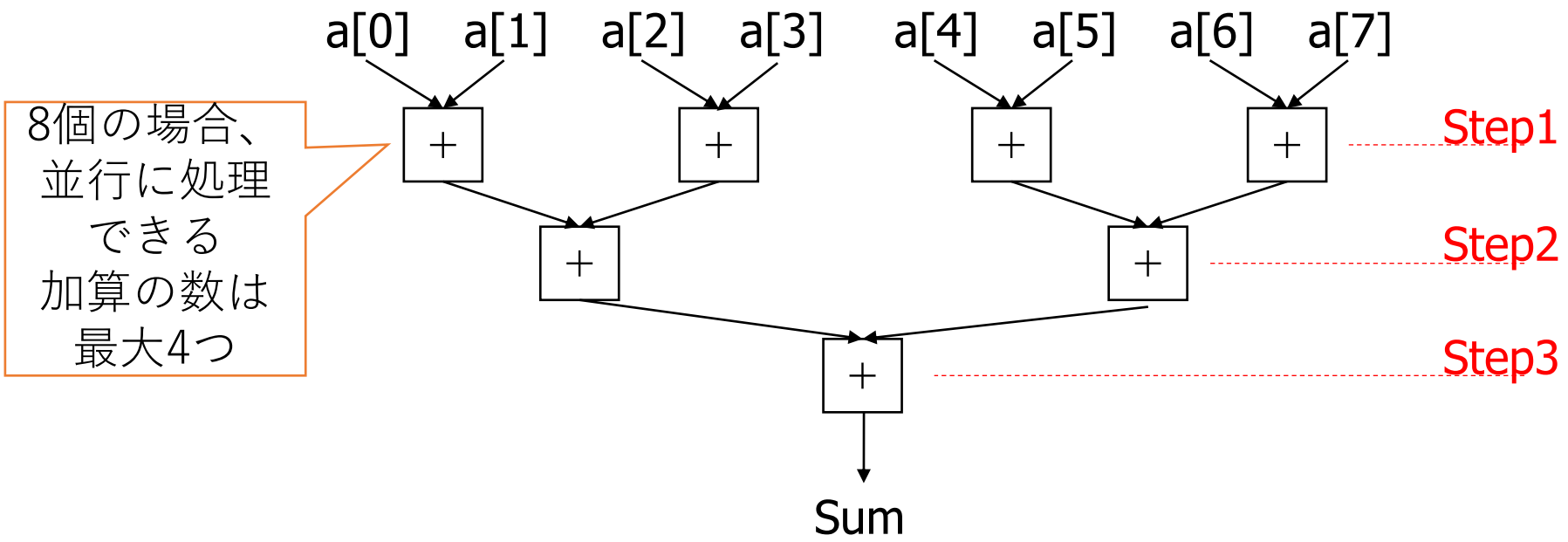
■2項演算を記号 ⊕で表すと、リダクションの結果は

$a_{1\text{-}N} = a_1 \oplus a_2 \oplus \ldots \oplus a_N$

簡単に言えばこれをやりたい

例：総和

配列に格納されている8個の数値の合計を求めたい。
sum=a[0]+a[1]+ … + a[7]

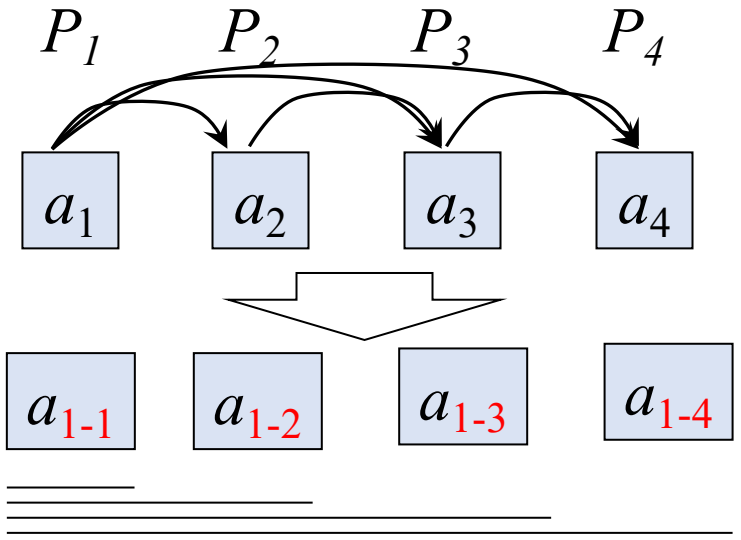


この図はデータフローとして描かれている。
これを計算モデル上でどう実現するか、ということ。

基本的なグループ操作（つづき）

■**プレフィックス操作 (Prefix computation)**

■最初のプロセッサから各プロセッサまでの
データを2項演算し、
その結果を
それぞれのプロセッサに求める。



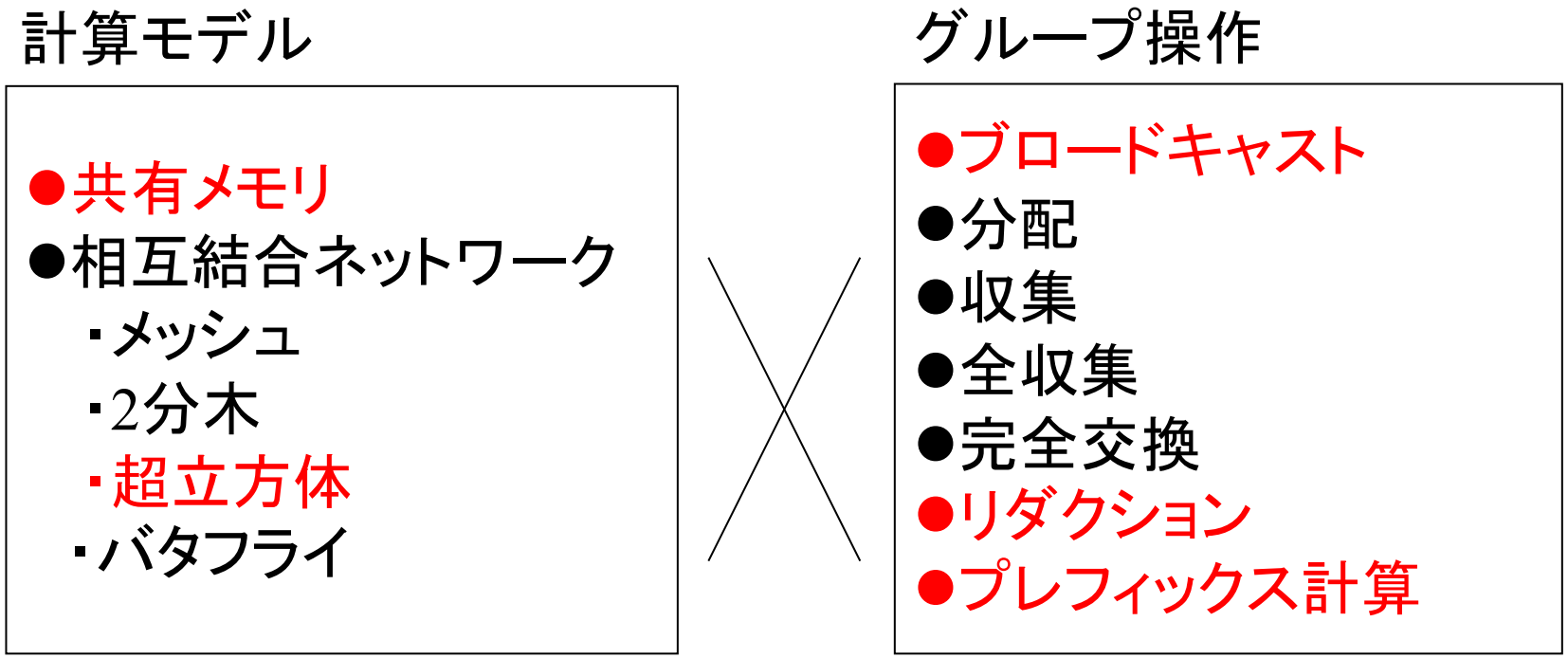
■最初のプロセッサから
そのプロセッサまでの
データ数の合計など

■2項演算を記号⊕で表すと、プレフィックス計算は

$a_{1\text{-}i} = a_1 \oplus a_2 \oplus \ldots \oplus a_i \quad (1 \leqq i \leqq N)$

計算モデル上でのグループ操作

■計算モデルの上で、グループ操作をどのように実現するか



共有メモリでのグループ操作

共有メモリ

- 共有メモリは複数のプロセッサからアクセスされる。
- 同じアドレスに対するアクセスを許すかどうかで、さらに場合分けされる。

		read	
		exclusive	concurrent
write	exclusive	EREW	CREW
	concurrent	ERCW	CRCW

- 問題なのは同時に書き込もうとする場合である

脱線： もしも同時書き込みを許すとしたら？

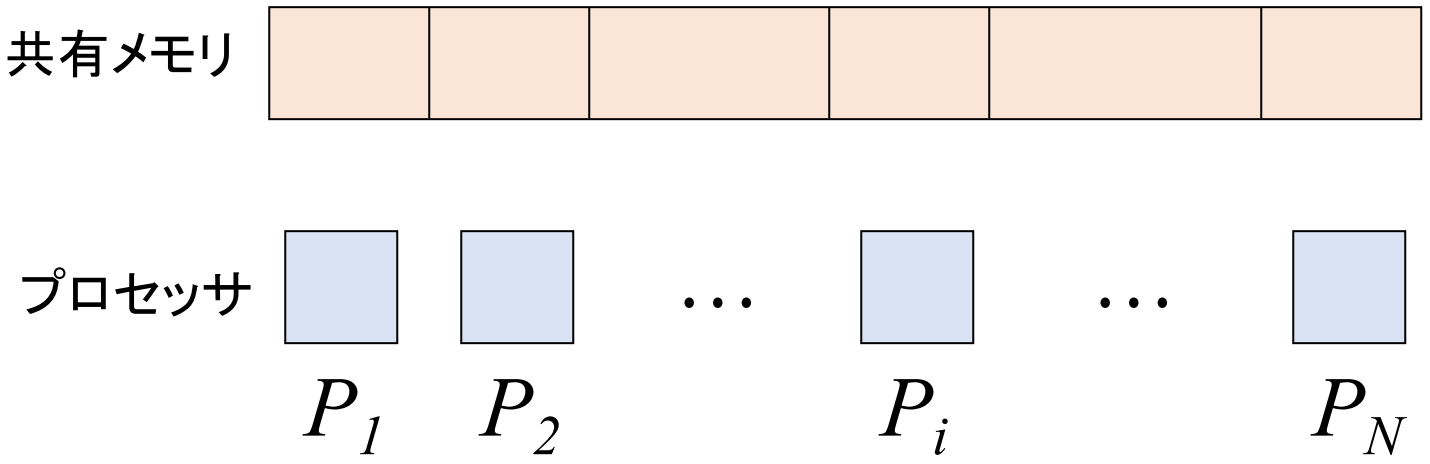
複数のプロセッサが同じ番地に書き込もうとすることは
実際に起こりうる。この時、同時アクセスを認めるとしたら、
どのような動きが考えられるだろうか。

例：

■任意の一つのプロセッサの値が書き込まれる。
■プロセッサに番号(ID)を振っておき、
書き込もうとしたプロセッサの中で(例えば)最も番号の小さい
プロセッサの値が書き込まれる。
■書き込もうとしたプロセッサの値の総和が書き込まれる。
■書き込もうとした全てのプロセッサの値が等しい場合のみ書き込みを
認める。それ以外のときには書き込まれない（値の更新が起きない）。

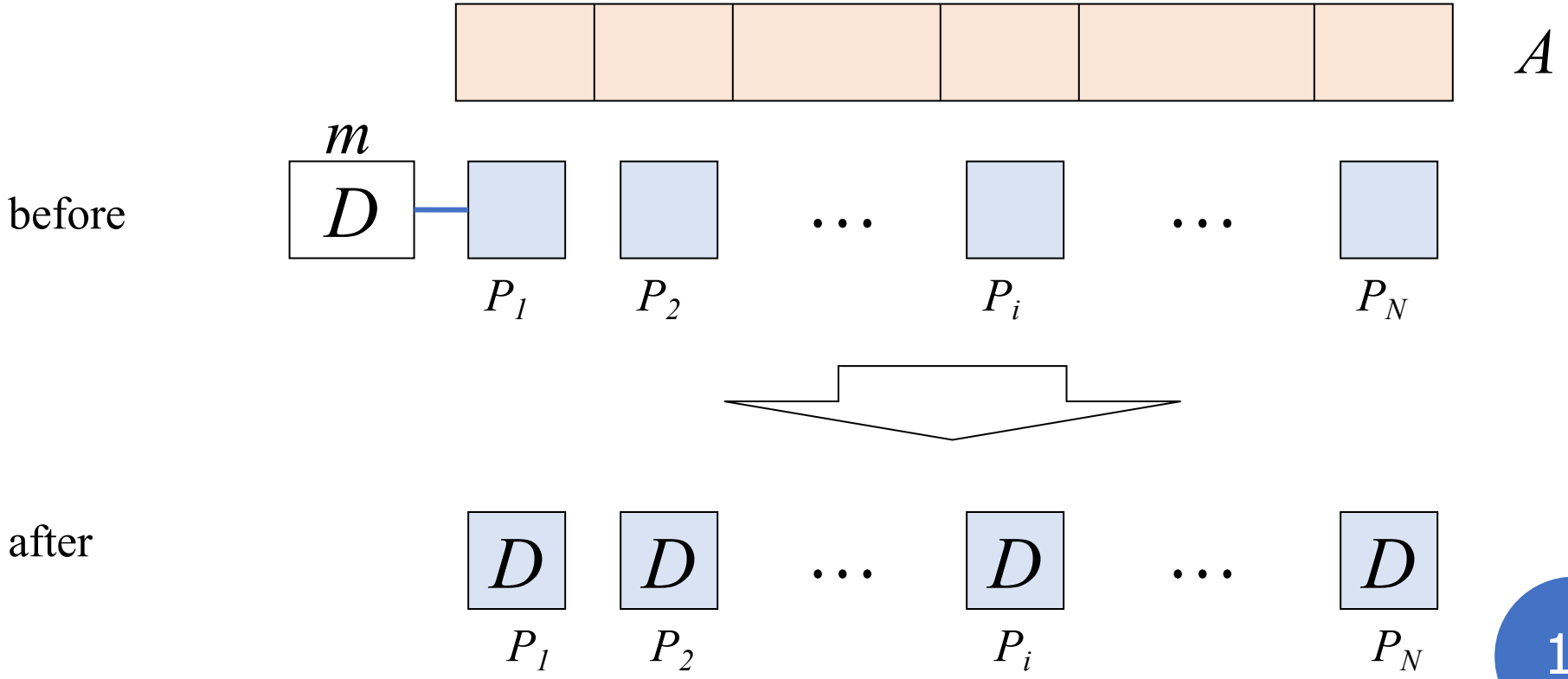
ここでの共有メモリモデルの仮定はEREW型

- どのプロセッサも共有メモリのすべてのアドレスにアクセスできる。
- 一つのプロセッサが同時にアクセスできるアドレスは一つ。
- あるアドレスに同時に読み書きできるプロセッサ数は一つ。
- 異なるアドレスならば並行に読み書きが実行できる。



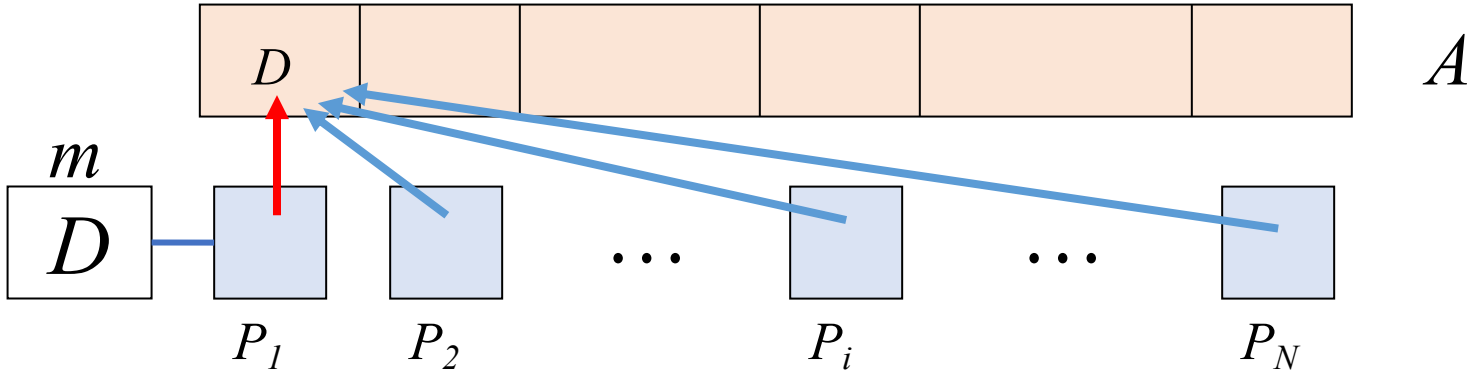
「ブロードキャスト」操作の実現

■長さNの共有メモリAを用いて、メモリ位置mにある
データDを、N個のプロセッサにブロードキャストしたい。



例えば、こんな方法でできる

■長さNの共有メモリAを用いて、メモリ位置mにある
データDを、N個のプロセッサにブロードキャストしたい。



Step 1. P_1がmからデータDを読み出し、それを共有メモリに書き込む。
Step 2. P_2 が共有メモリからデータDを読み出す。
Step 3. P_3 が共有メモリからデータDを読み出す。
…
Step N. P_N が共有メモリからデータDを読み出す。完了

。
→ステップ数N

ブロードキャスト：方針と手順

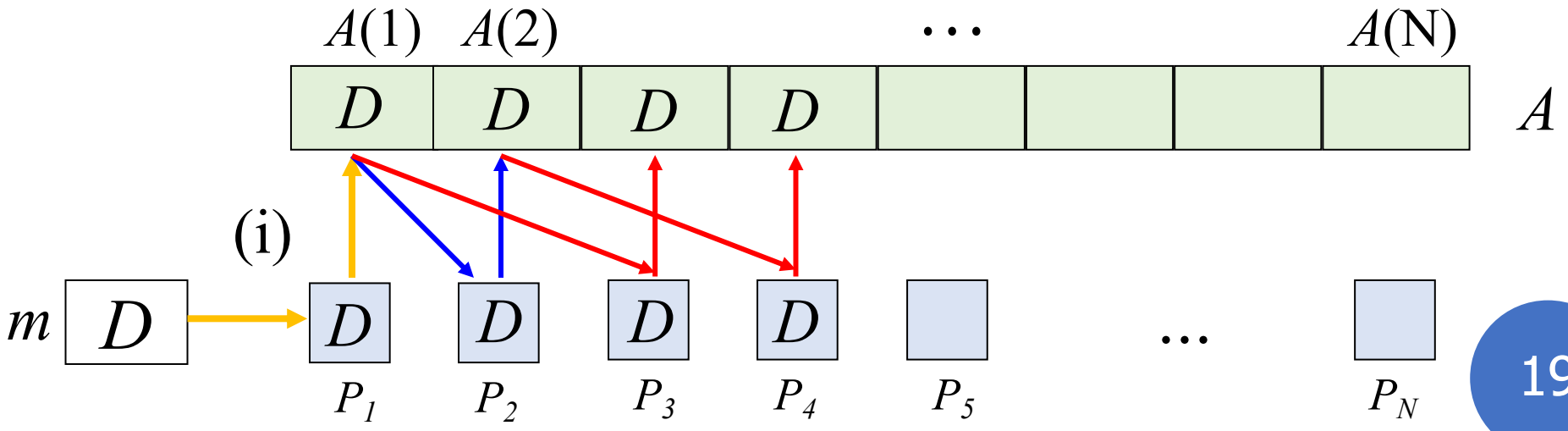
■方針：データDをもつプロセッサの数を、ステップごとに倍にする。

■手順

i. プロセッサ P_1 がメモリ位置 m からデータ D を読み込み、
共有メモリ位置 $A(1)$ に書き込む。

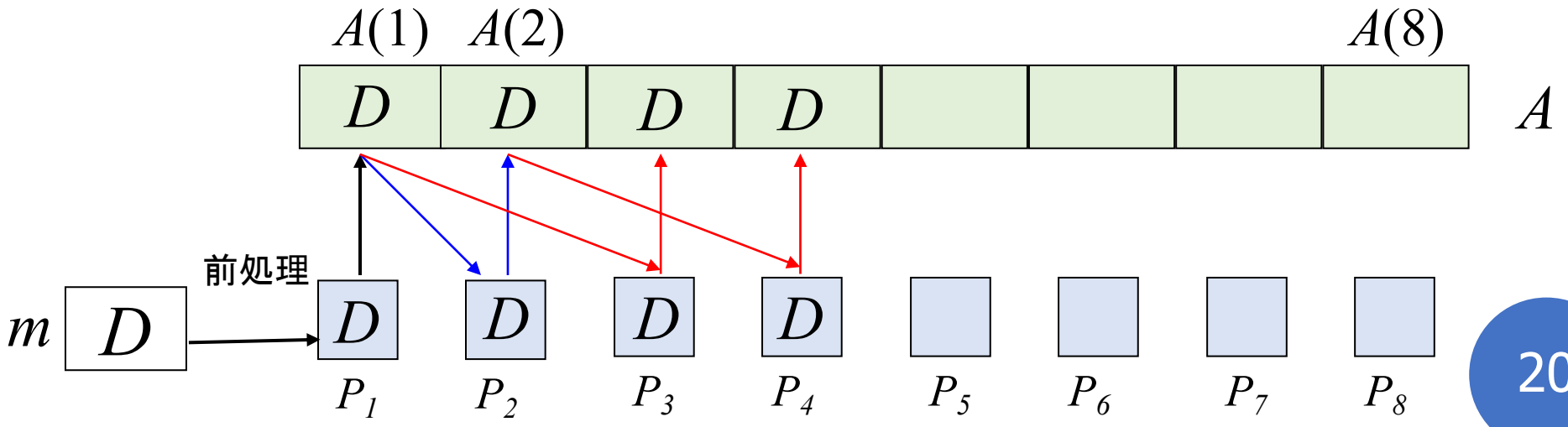
ii. 各プロセッサが共有メモリAからデータDを排他的に読み込み、
対応する共有メモリに排他的に書き込む。

iii. ステップ ii. を必要なだけ繰り返す。



ブロードキャスト：N=8の例

step 0　　距離 1 ($=2^0$) ずれた位置のメモリから値を取得し、書き込む。
step 1　　距離 2 ($=2^1$) ずれた位置のメモリから値を取得し、書き込む。
:
step j　　距離　　2^j の位置のメモリから値を取得。



ループごとの処理の様子

ループ j	i	プロセッサ P_i	読む メモリ位置	書く メモリ位置
前処理	1	P_1	m	A(1)
0	2	P_2	A(1)	A(2)
1	3 4	P_3 P_4	A(1) A(2)	A(3) A(4)
2	5 6 7 8	P_5 P_6 P_7 P_8	A(1) A(2) A(3) A(4)	A(5) A(6) A(7) A(8)
3	9 10 11	P_9 P_{10} P_{11}	A(1) A(2) A(3)	A(9) A(10) A(11)

操作の一般化 (j 回目のループ での処理)

ループ j	i	プロセッサ P_i	読む メモリ位置	書く メモリ位置
j	2^j+1 : $i = 2^j+i-2^j$: 2^j+2^j	P_{2^j+1} : P_i : $P_{2^j+2^j}$	$A(1)$: $A(i-2^j)$: $A(2^j)$	$A(2^j+1)$: $A(i)$: $A(2^{j+1})$

何回繰り返せばいいのか

ループ j	i	プロセッサ P_i	読む メモリ位置	書く メモリ位置
前処理	1	P_1	m	A(1)
0	2	P_2	A(1)	A(2)
1	3 4	P_3 P_4	A(1) A(2)	A(3) A(4)
:				
j				
:				
$(\log_2 N)$-1				

ブロードキャスト：手続き化

Procedure Broadcast(m, N, A)

Stage 1:

プロセッサ$P1$ は

メモリ位置 m から値を読む；

メモリ位置 $A(1)$ に値を書く；

Stage 2:

for j=0 to ($\log_2 N$)-1 do

for i=2^j +1 to 2^{j+1} do in parallel /* 各プロセッサ並列 */

プロセッサP_i は

メモリ位置 $A(i-2^j)$ から値を読む；

メモリ位置 $A(i)$ に値を書く；

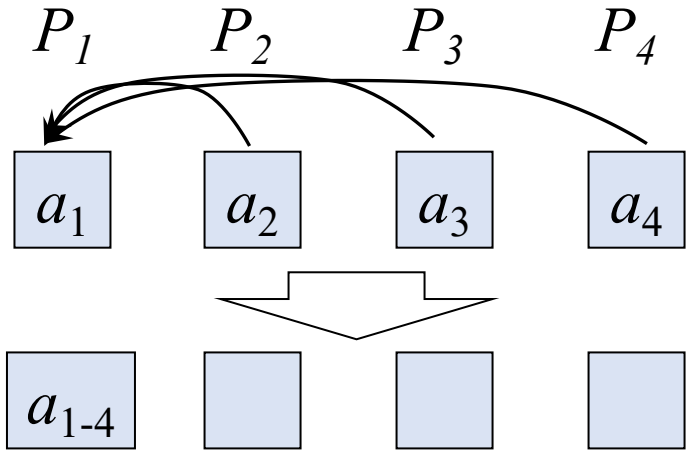
end for

end for.

このアルゴリズムの実行ステップ数は $\log_2 N$ に比例する。
したがって実行時間 T=O(log N).

リダクション

- リダクション (Reduction)
  各プロセッサのもつデータを2項演算し、
  その結果をプロセッサP_1で求めることにする。
  - 2項演算が加算の場合、
    リダクション操作は $a_{1\text{-}N} = a_1 + a_2 + \ .... + a_N$

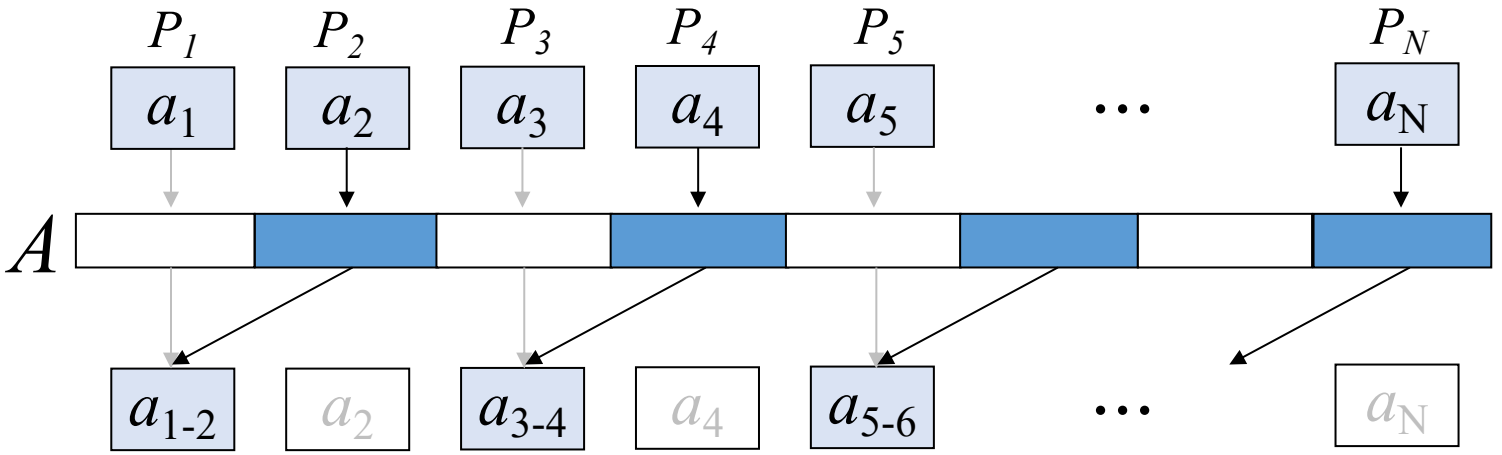


リダクション：方針と手順

■方針：求める部分和の範囲をステップごとに2倍にする。

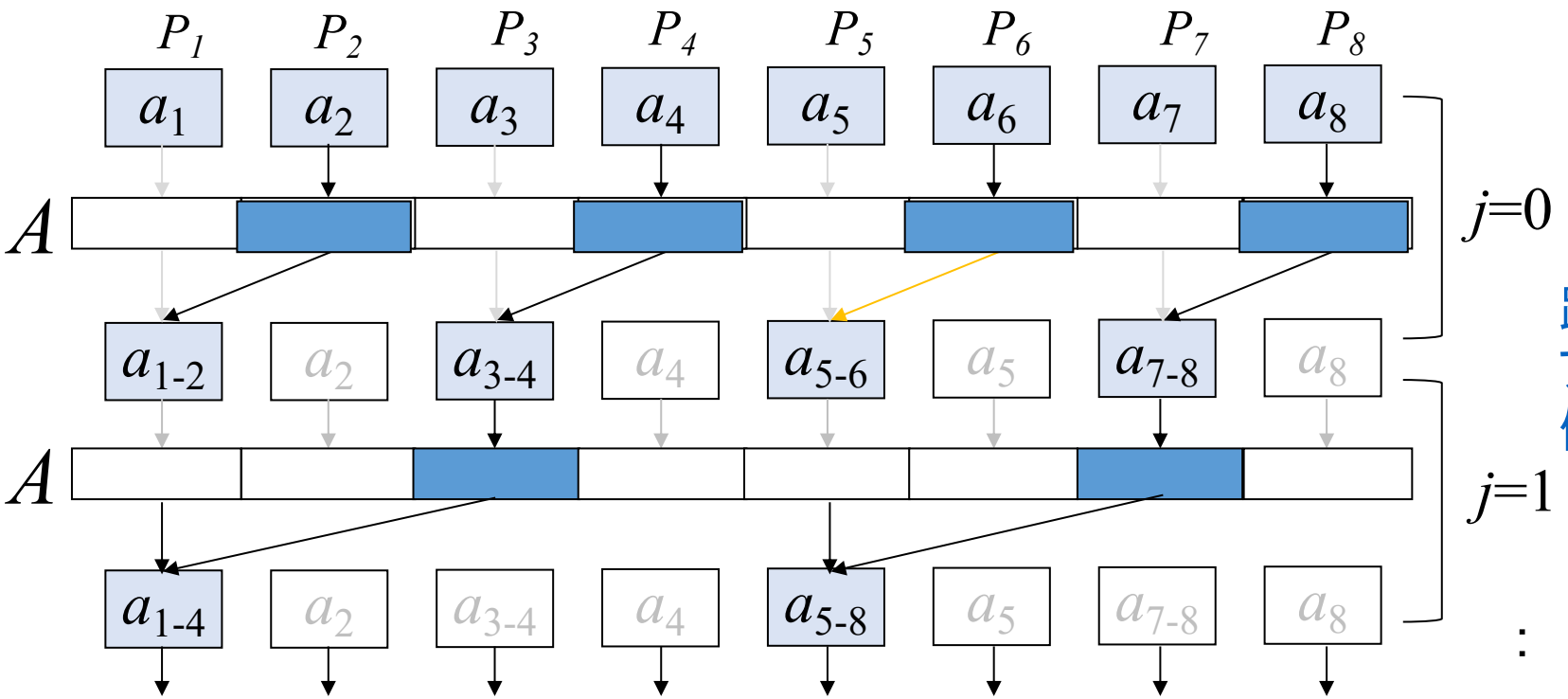
■手順

1. 各プロセッサは持っている値を共有メモリAに書く。
2. 各プロセッサは共有メモリAから他の値（部分和）を読む。
3. 読んだ値と持っている値を加算（して保持）する。
4. ステップ1～3を繰り返す。



リダクションの例

- 初期状態として各プロセッサ P_i が値 a_i をもつとき、共有メモリAを用いてリダクション操作を行う。N=8。



操作の一般化

■ステップ j でのプロセッサ i の操作を記述する。

j	i	プロセッサ P_i	書く メモリ位置	読む メモリ位置
0	2	P_2	$A(2)$	
	4	P_4	$A(4)$	
	6	P_6	:	
	8	P_8	$A(8)$	
	1	P_1		$A(2)=A(1+1)$
	3	P_3		$A(4)=A(3+1)$
	5	P_5		:
	7	P_7		$A(8)=A(7+1)$
1	3	P_3	$A(3)$	
	7	P_7	$A(7)$	
	1	P_1		$A(3)=A(1+2)$
	5	P_5		$A(7)=A(5+2)$
2	1	P_1	$A(1)$	

リダクション：手続き

Procedure Reduction($a_1, a_2, \ldots, a_N$)
  for j=0 to ($\log_2 N$)-1 do
    for i=1 to N do in parallel /* 各プロセッサ並列 */
      P_i は持っている値 $a(i)$ を $A(i)$ に書く；
      if (i mod 2^{j+1}) = 1 then /* 2^{j+1} で割ると1余り */
        Pi は $A(i+2^j)$ から値 $a(i+2^j)$ を読む；
        $a(i) \leftarrow a(i) \oplus a(i+2^j)$;
    end for
  end for.

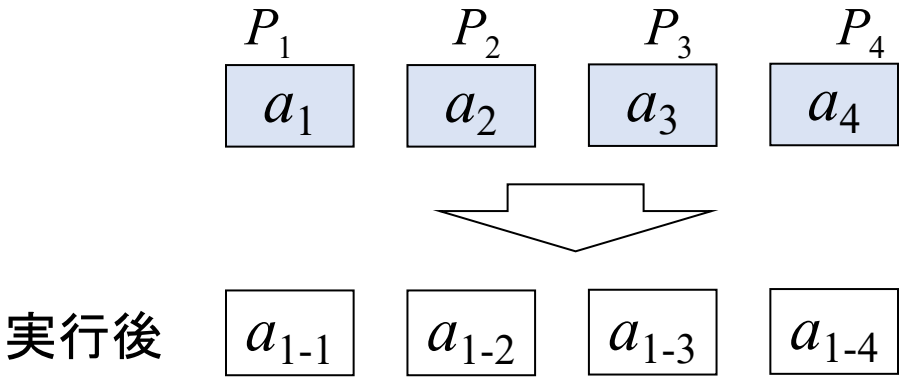
このアルゴリズムの実行ステップ数は $\log_2$ N に比例する。
したがって実行時間 T=O(logN).

$a(i)$: P_i がそのとき持っている値

プレフィックス計算

プロセッサ Pi が値 a_i をもつ。
記号 $\oplus$ が2項演算を表すとき
プレフィックス計算 $a_{1\text{-}i}=a_1 \oplus a_2 \oplus \ldots \oplus a_i$ （1≦i ≦N）

最初のプロセッサから
各プロセッサまでのデータを
2項演算し、結果をそれぞれの
プロセッサで求める。



演算が加算のときプレフィックス和 $a_{1\text{-}i}=a_1 + a_2 + \ldots + a_i$

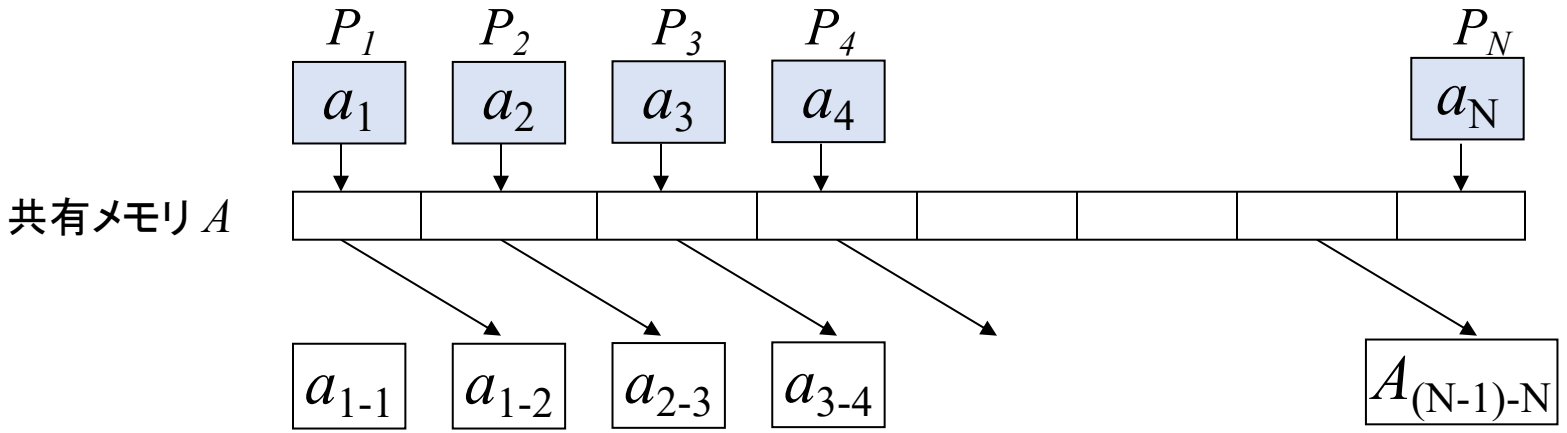
プレフィックス和の計算：方針と手順

■方針

求める部分和の範囲をステップごとに2倍にする。

■手順

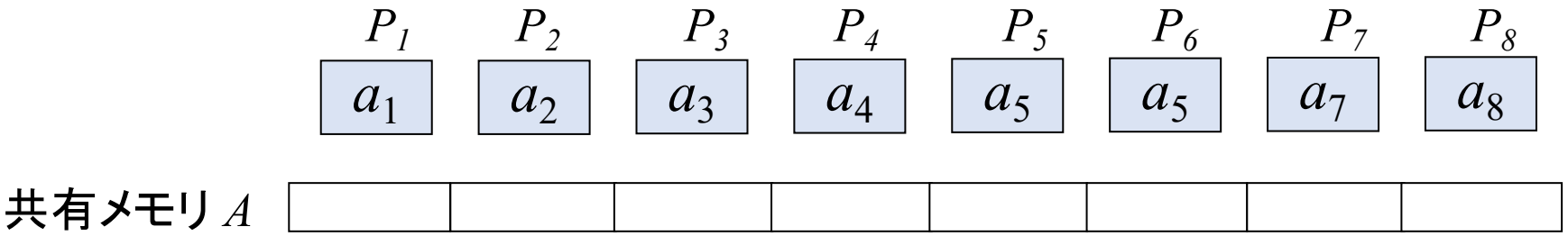
1. 各プロセッサはもっている値を共有メモリAに書く。
2. 各プロセッサはAから他の値（部分和）を読む。
3. 各プロセッサは読んだ値と持っている値を加算する。
4. ステップ1～3を繰り返す。



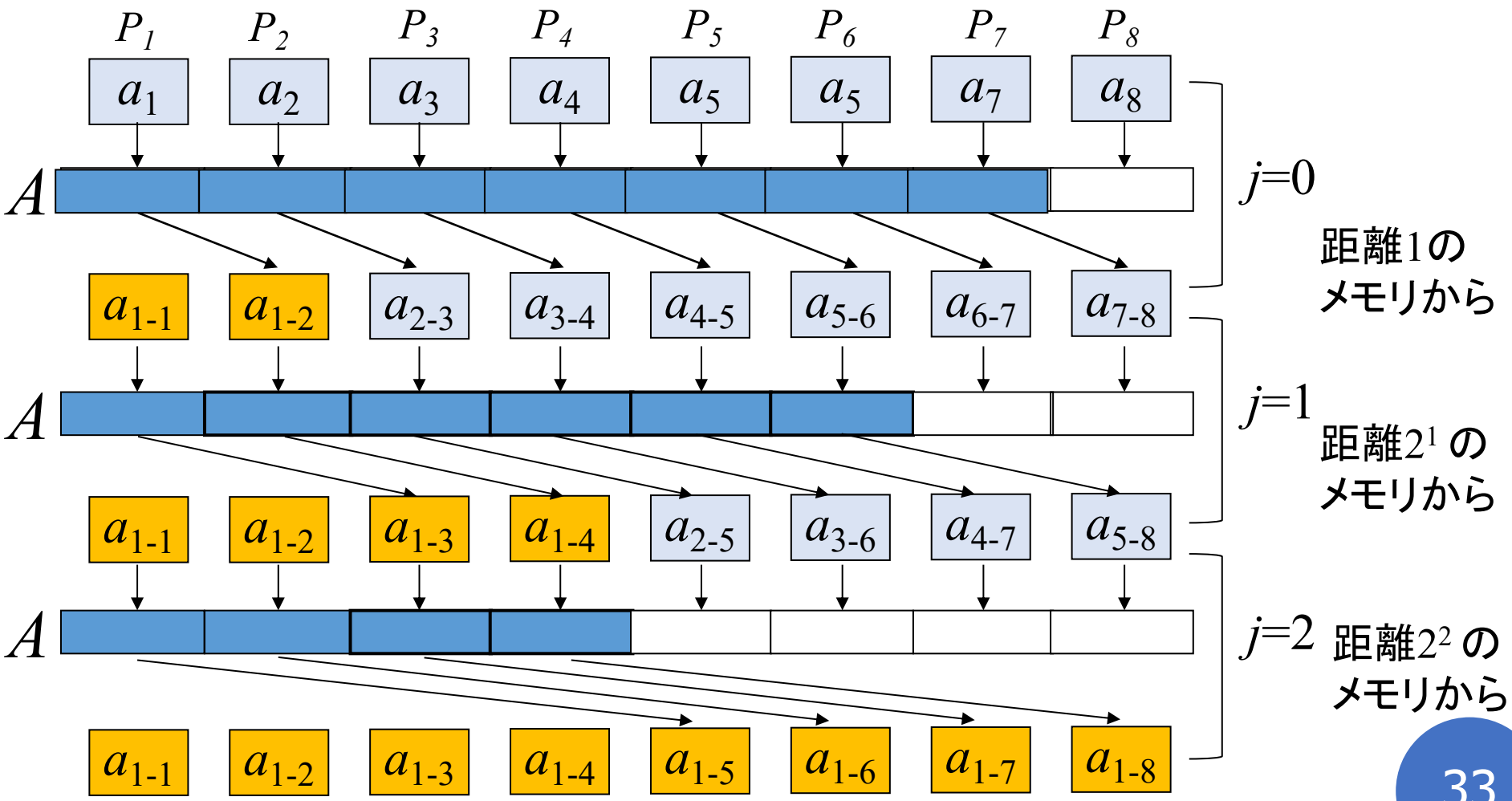
プレフィックス計算の例

- プロセッサ P_i が値 a_i をもつとき、N=8に対して、共有メモリを用いてプレフィックス計算を行う。

- 初期状態：



プレフィックス計算の例（続き）



操作の一般化

■ステップ j でのプロセッサ i の操作

j	i	プロセッサ P_i	書く メモリ位置	読む メモリ位置
0	1 2 :	P_1 P_2 :	A(1) A(2) :	
	2 : i :	P_2 : P_i :		A(1) : A(i-1) :
1	2 :	P_2 :	A(2) :	
	3 : i :	P_3 : P_i :		A(1) : $A(i-2^1)$:
:				

プレフィックス計算：手続き

Procedure Prefix($a_1, a_2, \ldots, a_N$)
  for j = 0 to ($\log_2 N$)-1 do
    for i=1 to N do in parallel /* 各プロセッサ並列 */
      P_i は
      (1) もっている値 $a(i)$ を $A(i)$ に書く；
      (2) $A(i\text{-}2^j)$ から値 $a(i\text{-}2^j)$ を読む；
      (3) $a(i) \leftarrow a(i) \oplus a(i\text{-}2^j)$;
    end for
  end for.

このアルゴリズムの実行ステップ数は $\log_2 N$ に比例する。
このため実行時間 T=O(logN).

（注）
- $a(i)$: P_i がそのとき持っている値
- ステップ(2)で、メモリAの値を読む必要のないプロセッサの条件は省略

補足

- リダクションやプレフィックス計算で登場した「二項演算」は任意のものではなく、結合律が成り立つものに限られる。
  $(\ (A \oplus B) \oplus C) = (A \oplus (B \oplus C))$

  - OKなもの
    - 和
    - 積
    - 最大値 （全順序集合なら）
    - 最小値

  - ダメなもの
    - 差
    - 除算
    - じゃんけん のように順序によって結果が変わるもの

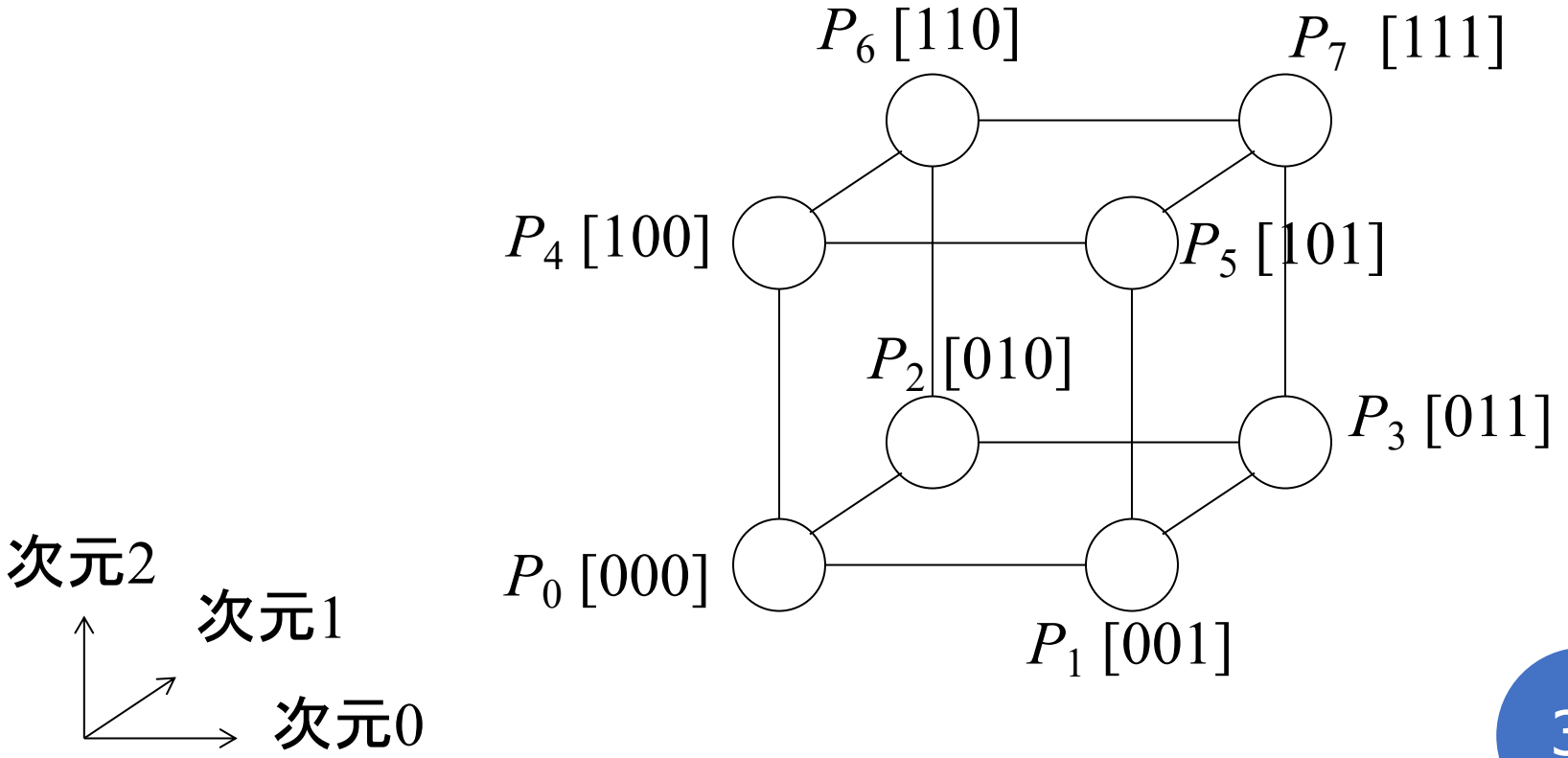
相互結合ネットワーク上でのグループ操作

超立方体結合ネットワーク

超立方体結合モデル

■超立方体結合モデル

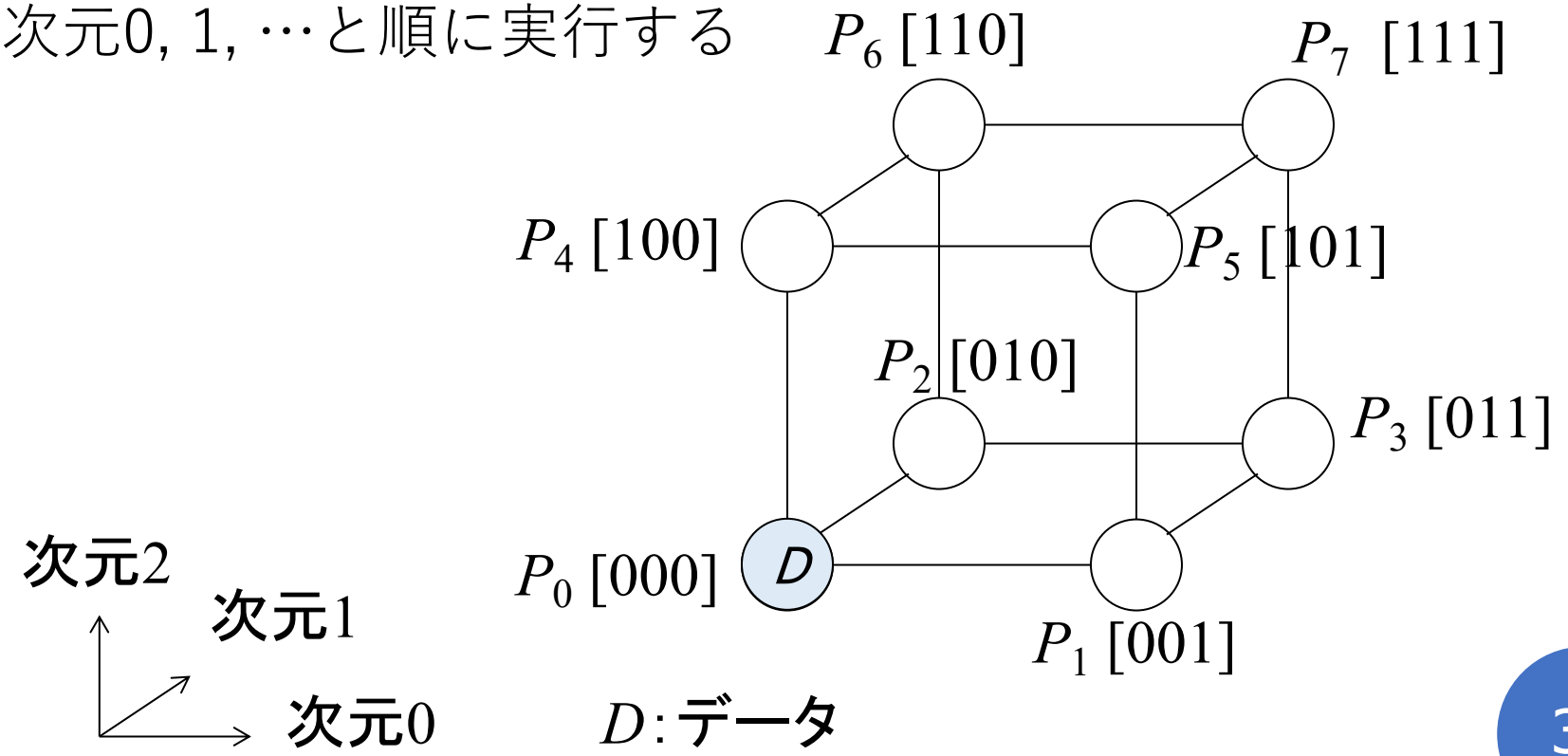
$N=2^3$のとき、プロセッサ $P_i : [u_2u_1u_0]$, $u_i \in \{0,1\}$



ブロードキャスト

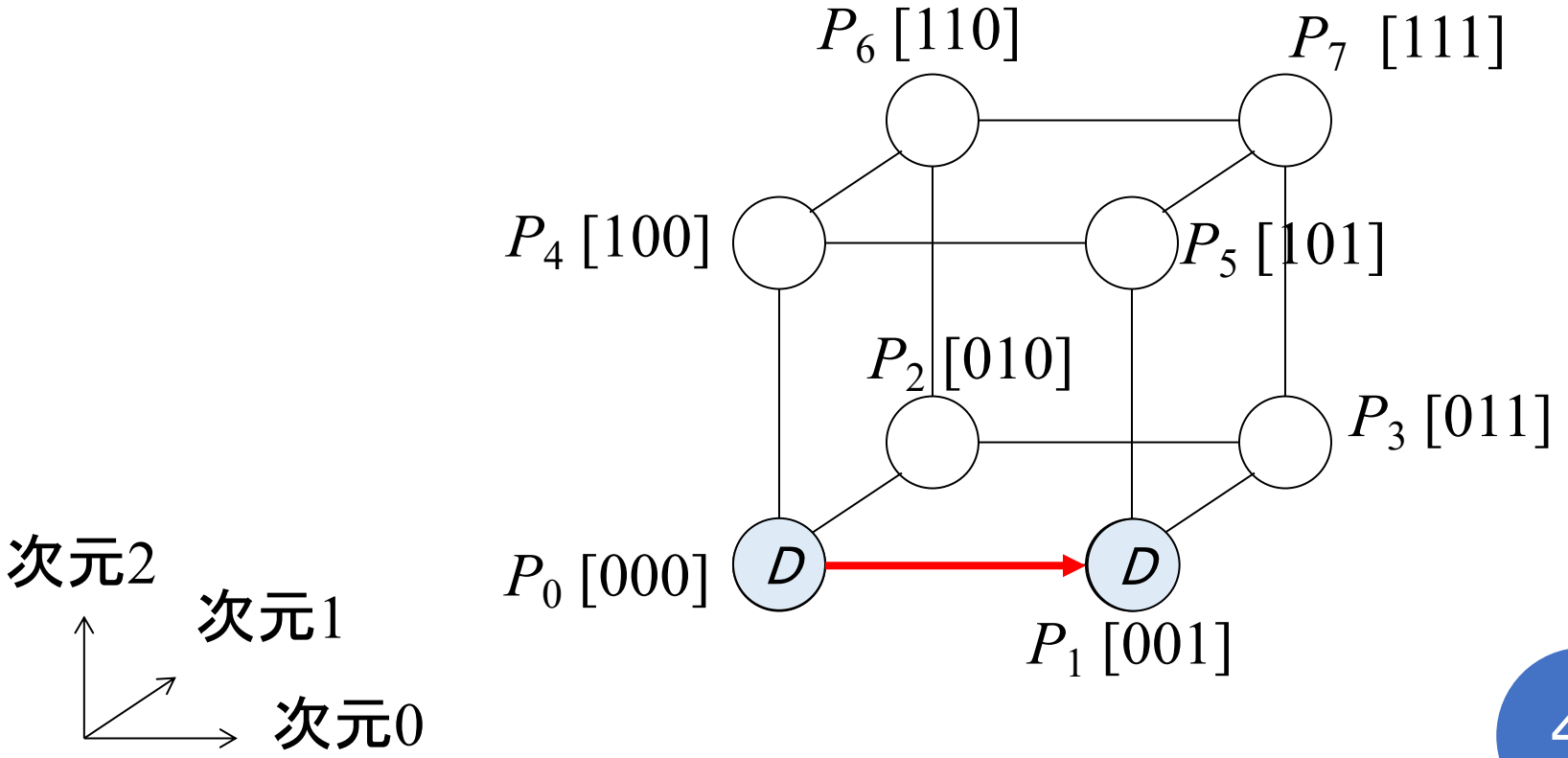
[000]がもつデータDを全プロセッサに配布したい。

- 方針：次元ごとに1対1通信する
- 手順：次元0, 1, …と順に実行する



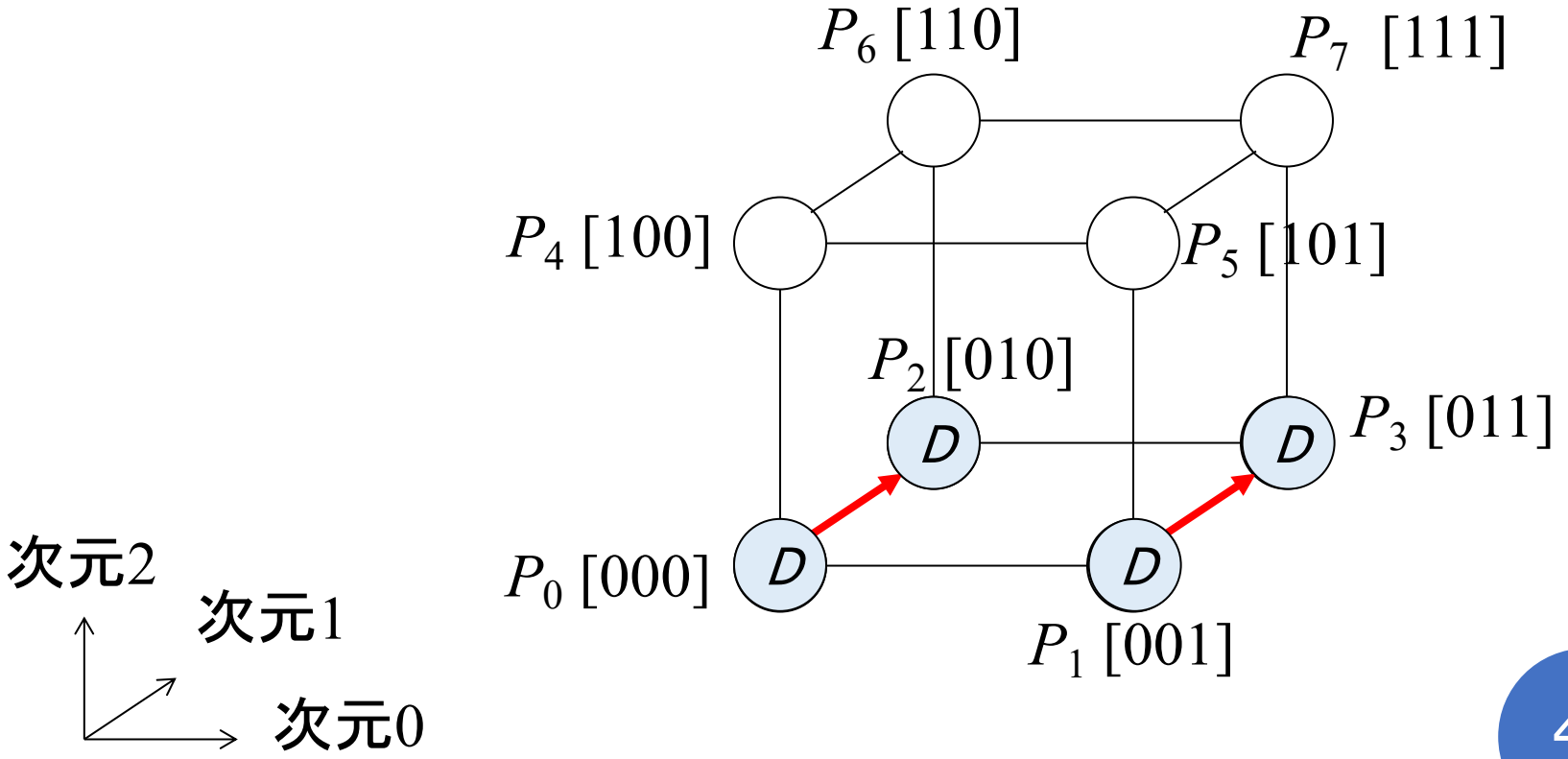
ブロードキャスト:Step1

Step1： 次元0の方向へ1対1通信
$+2^0$ のIDをもつプロセッサへ



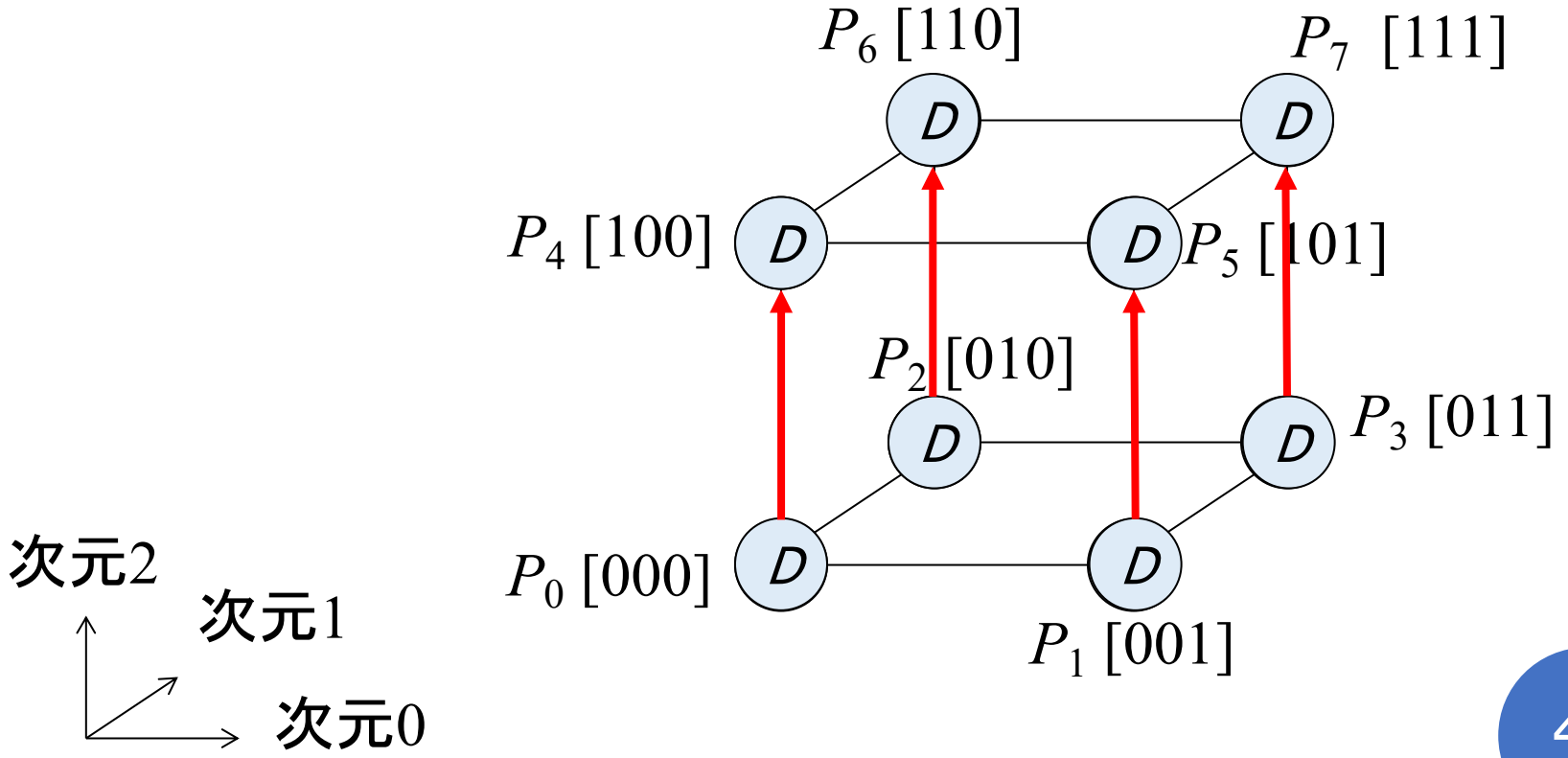
ブロードキャスト:Step2

Step2： 次元1の方向へ1対1通信
$+2^1$のIDをもつプロセッサへ



ブロードキャスト:Step3

Step3： 次元2の方向へ1対1通信
$+2^2$のIDをもつプロセッサへ



ブロードキャスト：手続き

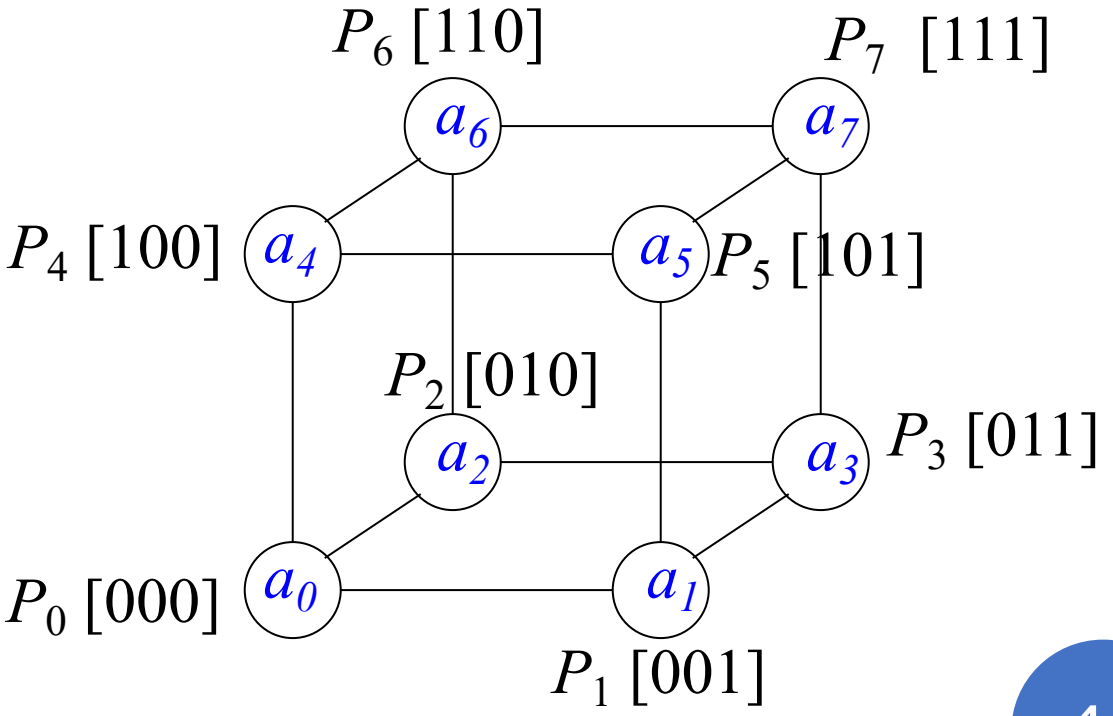
procedure Cube_Broadcast(D, N)
  for j=0 to ($\log_2 N$)-1 do /*次元の順に */
    for i=0 to 2^j-1 do in parallel /* 各プロセッサ並列実行 */
      （対象となるプロセッサならば）
      P_i は D を P_{i+2^j} に送る
    end for
  end for.

アルゴリズムの実行ステップ数は $\log_2 N$ に比例する。
実行時間 T=O(log N)

リダクション

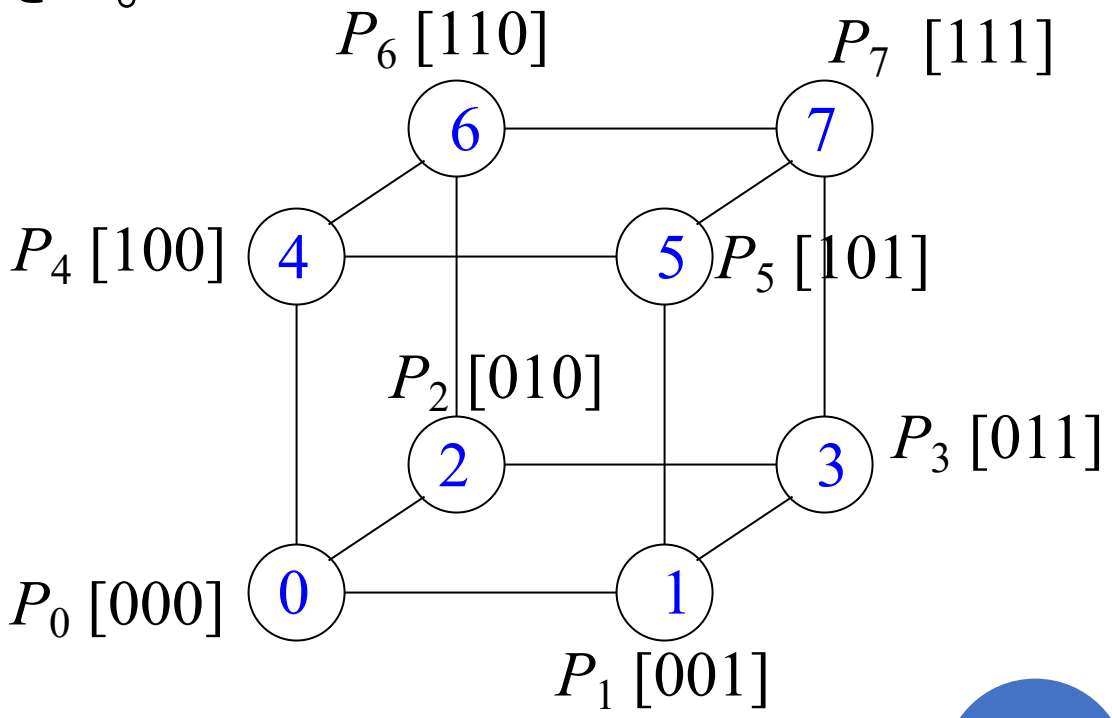
- プロセッサ P_i が値 a_i をもち、記号 $\oplus$ が2項演算を表すとき、N個の数のリダクションは $a_{0,N-1} = a_0 \oplus a_1 \oplus \cdots \oplus a_{N-1}$

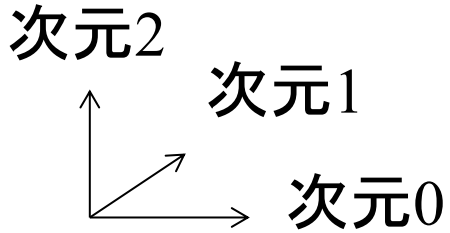
- リダクションの結果を P_0 に求めたい。



リダクション

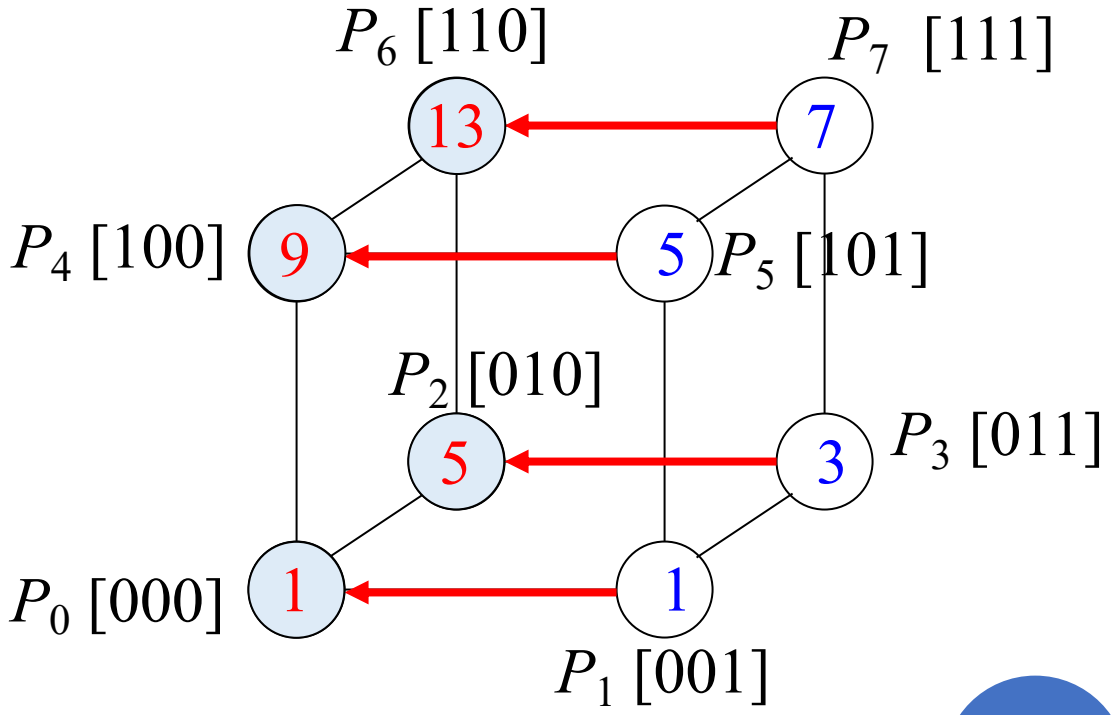
- 超立方体結合された8プロセッサ P_i $(0 \leqq i \leqq 7)$が
  それぞれ値 $a_i=i$ をもつとき、
  8個の数の総和を P_0 に求めたい。
  （つまり演算は＋）

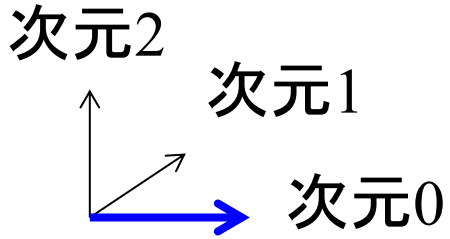




リダクション：Step1

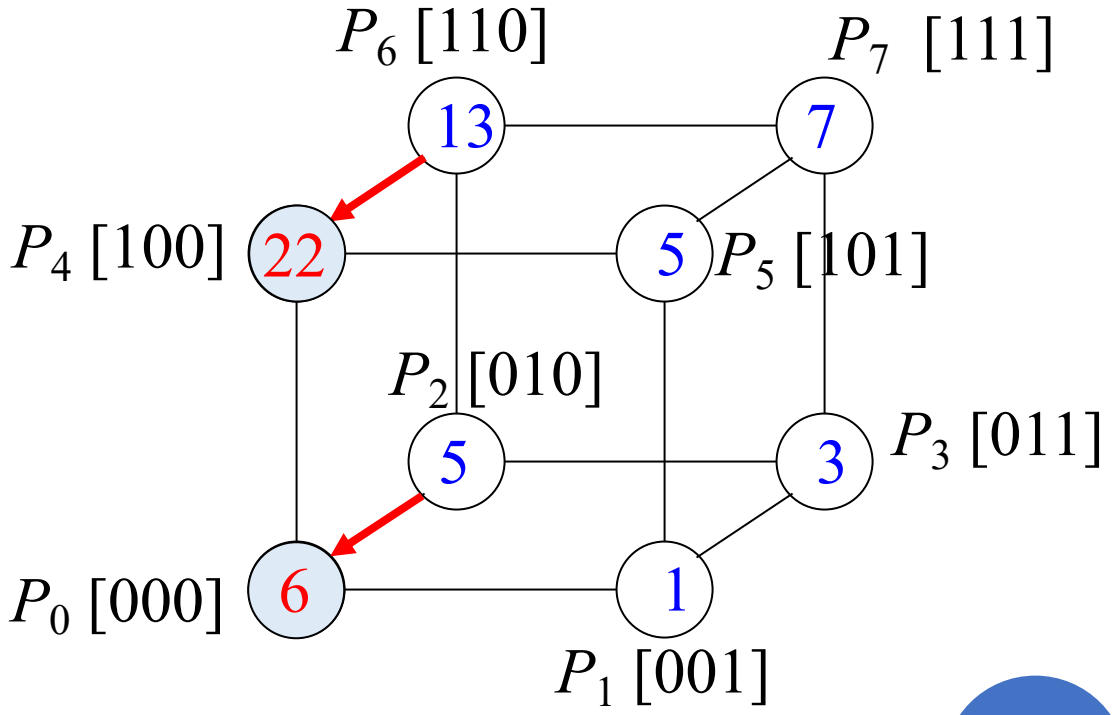
■Step1： 次元0の方向にユニキャスト。
プロセッサIDが -2^0 異なるプロセッサへ送信。
受信プロセッサでは
元々持っていた値と
受信した値で加算。

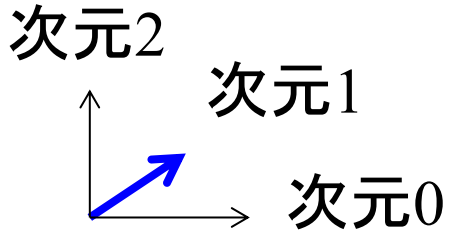




リダクション：Step2

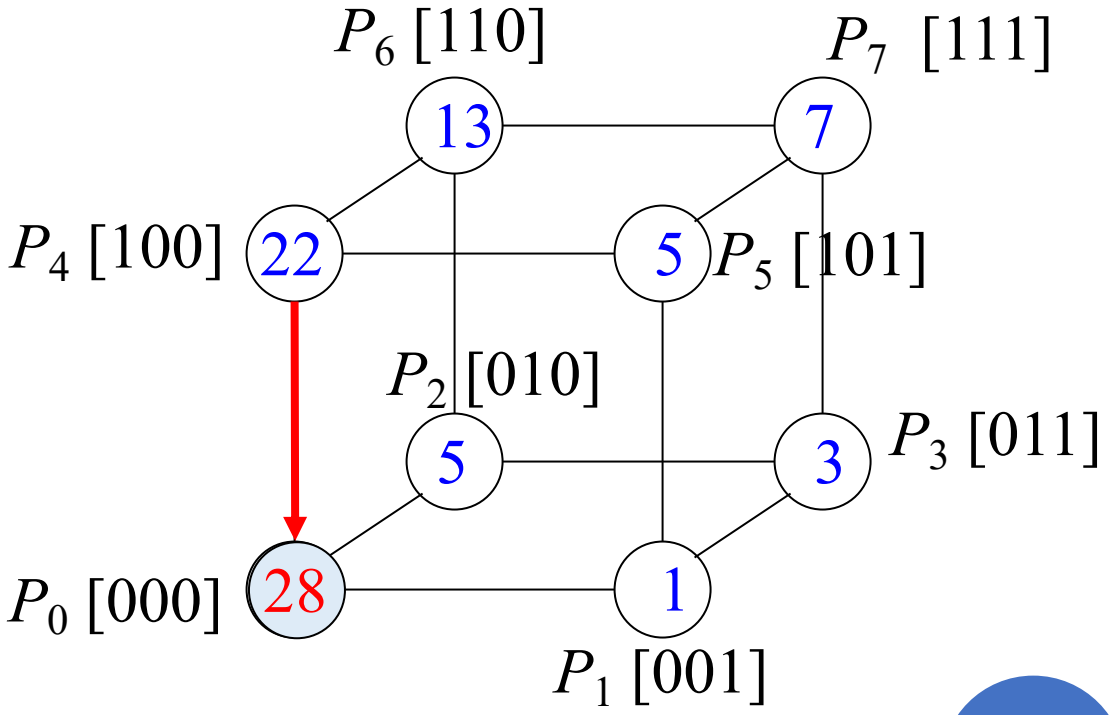
■Step2：次元1方向にユニキャスト。
プロセッサIDが -2^1 異なるプロセッサへ送信。
受信プロセッサでは
元々持っていた値と
受信した値を加算。

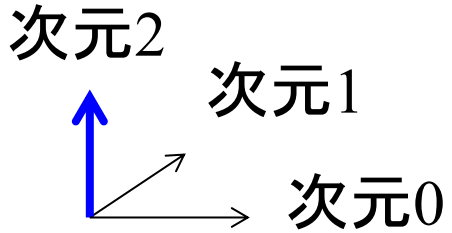




リダクション：Step3

■Step3：次元2の方向にユニキャスト。
プロセッサIDが -2^2 異なるプロセッサへ送信。
受信プロセッサでは
元々持っていた値と
受信した値を加算。





リダクションのデータの流れ

step	P_0 [000]	P_1 [001]	P_2 [010]	P_3 [011]	P_4 [100]	P_5 [101]	P_6 [110]	P_7 [111]
初期状態	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
1	$a_{0\text{-}1}$		$a_{2\text{-}3}$		$a_{4\text{-}5}$		$a_{6\text{-}7}$	
2	$a_{0\text{-}3}$				$a_{4\text{-}7}$			
3	$a_{0\text{-}7}$							

リダクション：手続き

procedure Cube_Reduction($a_1, a_2, \ldots, a_N$)

for j=0 to ($\log_2 N$)-1 do /*次元の向き*/

for i=0 to N-1 do in parallel /* 各プロセッサ並列実行 */

(i mod 2^{j+1})=0 であるP_i は

P_{i+2^j} から値 a_{i+2^j}を受け取り，$a_i \leftarrow a_i \oplus a_{i+2^j}$

end for

end for.

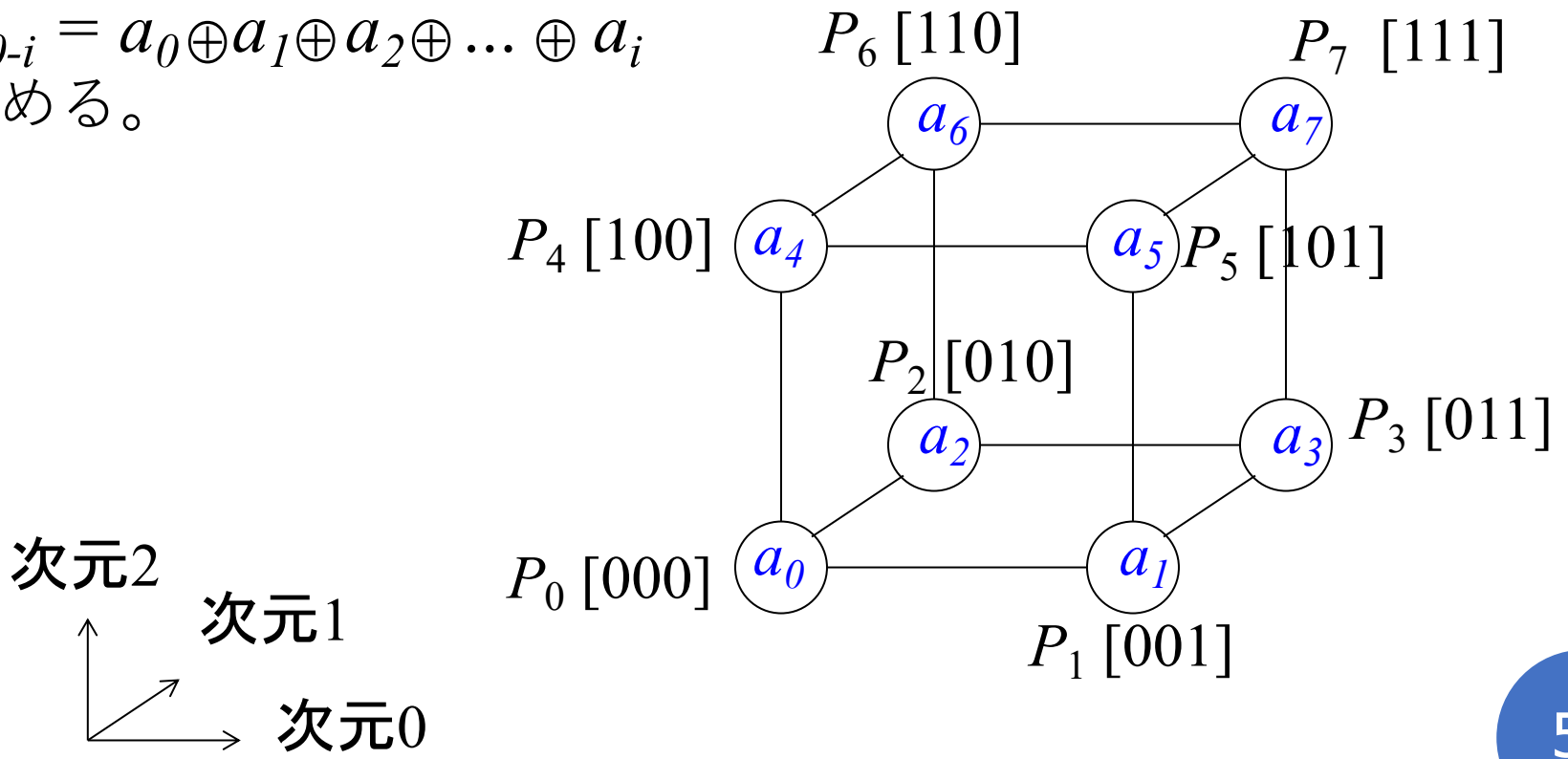
アルゴリズムの実行ステップ数は $\log_2 N$ に比例する。
実行時間 T=O(log N)

a_i: P_i がそのとき持っている値

プレフィックス計算

プロセッサ P_i が値 a_i をもち、記号 $\oplus$ が2項演算を表すとき、
各プロセッサでプレフィックス計算

$a_{0\text{-}i} = a_0 \oplus a_1 \oplus a_2 \oplus \ldots \oplus a_i$
を求める。



プレフィックス和の計算

■プロセッサ P_i が値 i をもっているとき、
各プロセッサでプレフィックス和を求める。

$a_{0\text{-}0}=a_0=0$

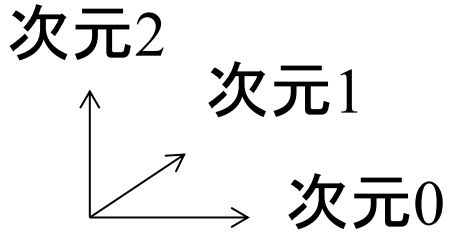
$a_{0\text{-}1}=a_0+a_1=1$

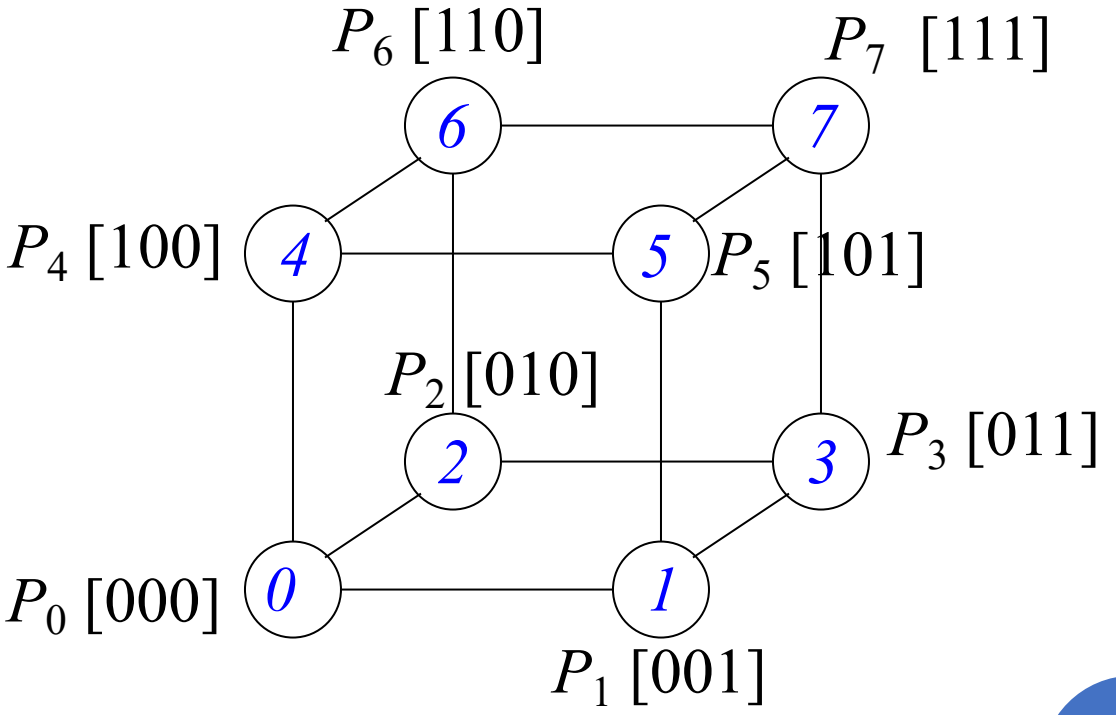
$a_{0\text{-}2}=a_0+a_1+a_2=3$

$a_{0\text{-}3}=a_0+a_1+a_2+a_3=6$

:

$a_{0\text{-}7}=a_0+a_1+\ \ldots\ +a_7=28$

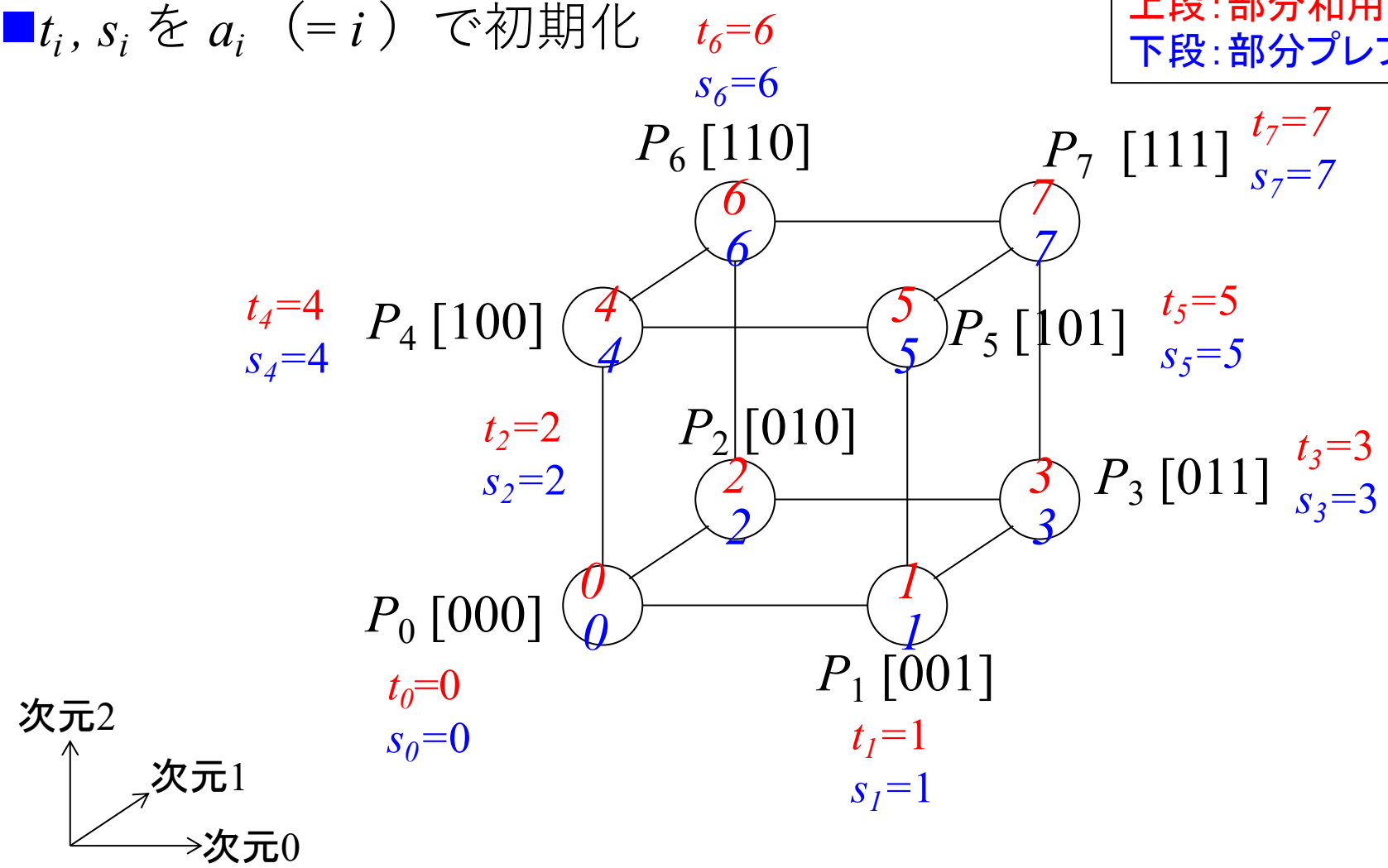




プレフィックス和の計算：初期化

■t_i, s_i を a_i （$= i$）で初期化

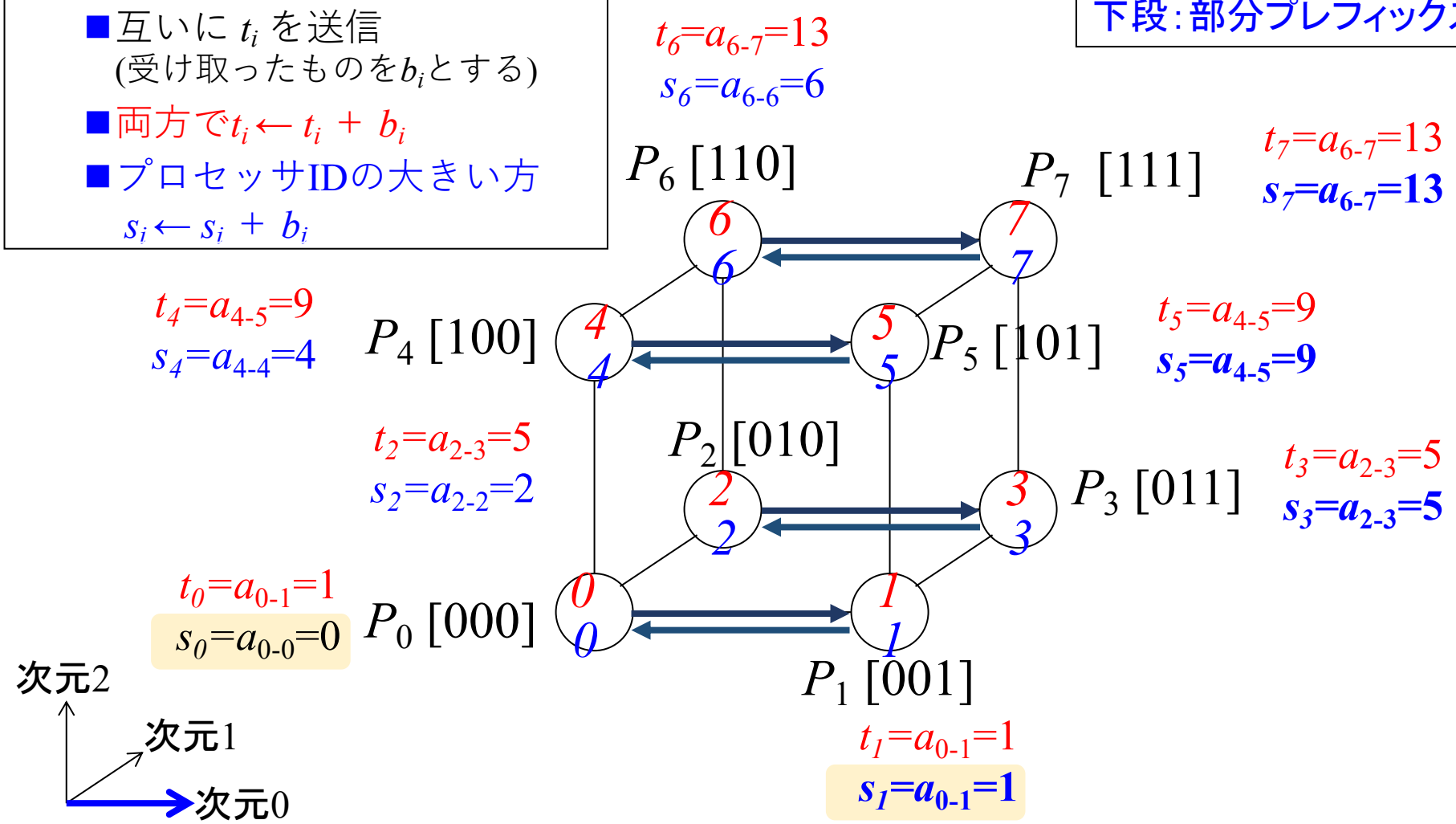
上段：部分和用 t_i
下段：部分プレフィックス和 用s_i



プレフィックス和の計算：Step1

■Step1： 次元0でユニキャスト
- ■互いに t_i を送信
  (受け取ったものをb_iとする)
- ■両方で$t_i \leftarrow t_i + b_i$
- ■プロセッサIDの大きい方
  $s_i \leftarrow s_i + b_i$

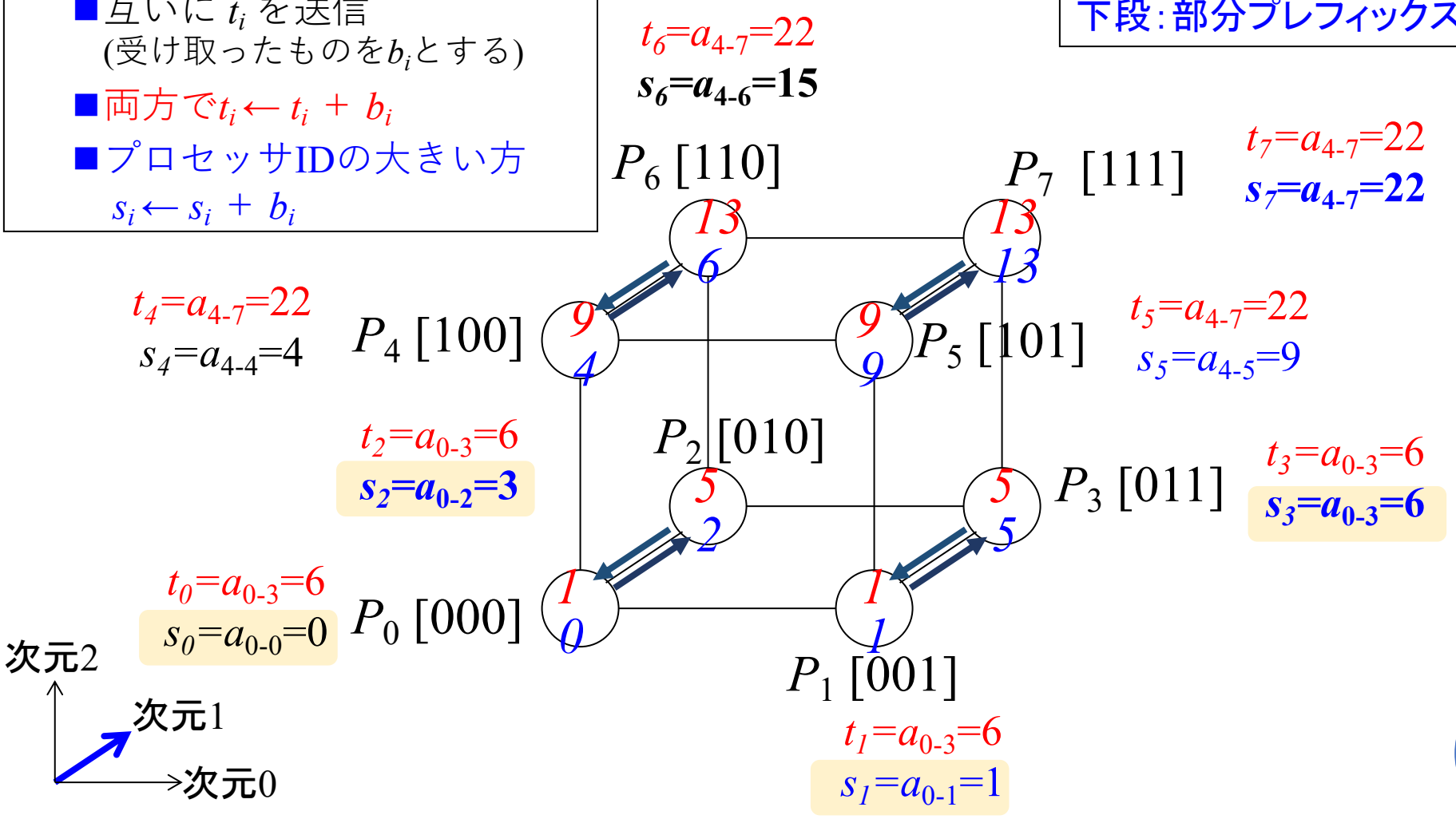
上段：部分和用 t_i
下段：部分プレフィックス和用s_i



プレフィックス和の計算：Step2

■Step2： 次元1でユニキャスト
- ■互いに t_i を送信
  (受け取ったものをb_iとする)
- ■両方で$t_i \leftarrow t_i + b_i$
- ■プロセッサIDの大きい方
  $s_i \leftarrow s_i + b_i$

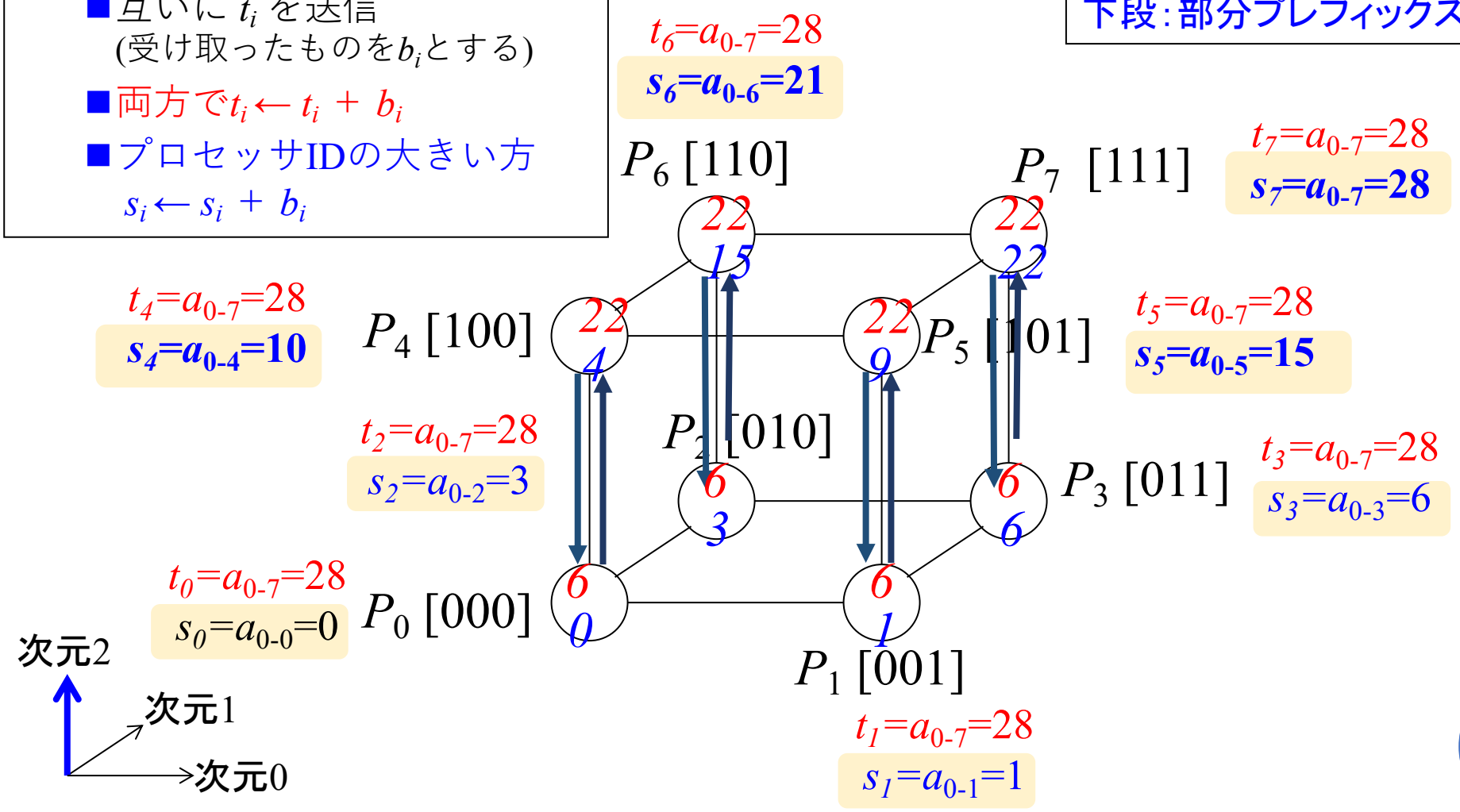
上段：部分和用 t_i
下段：部分プレフィックス和用s_i



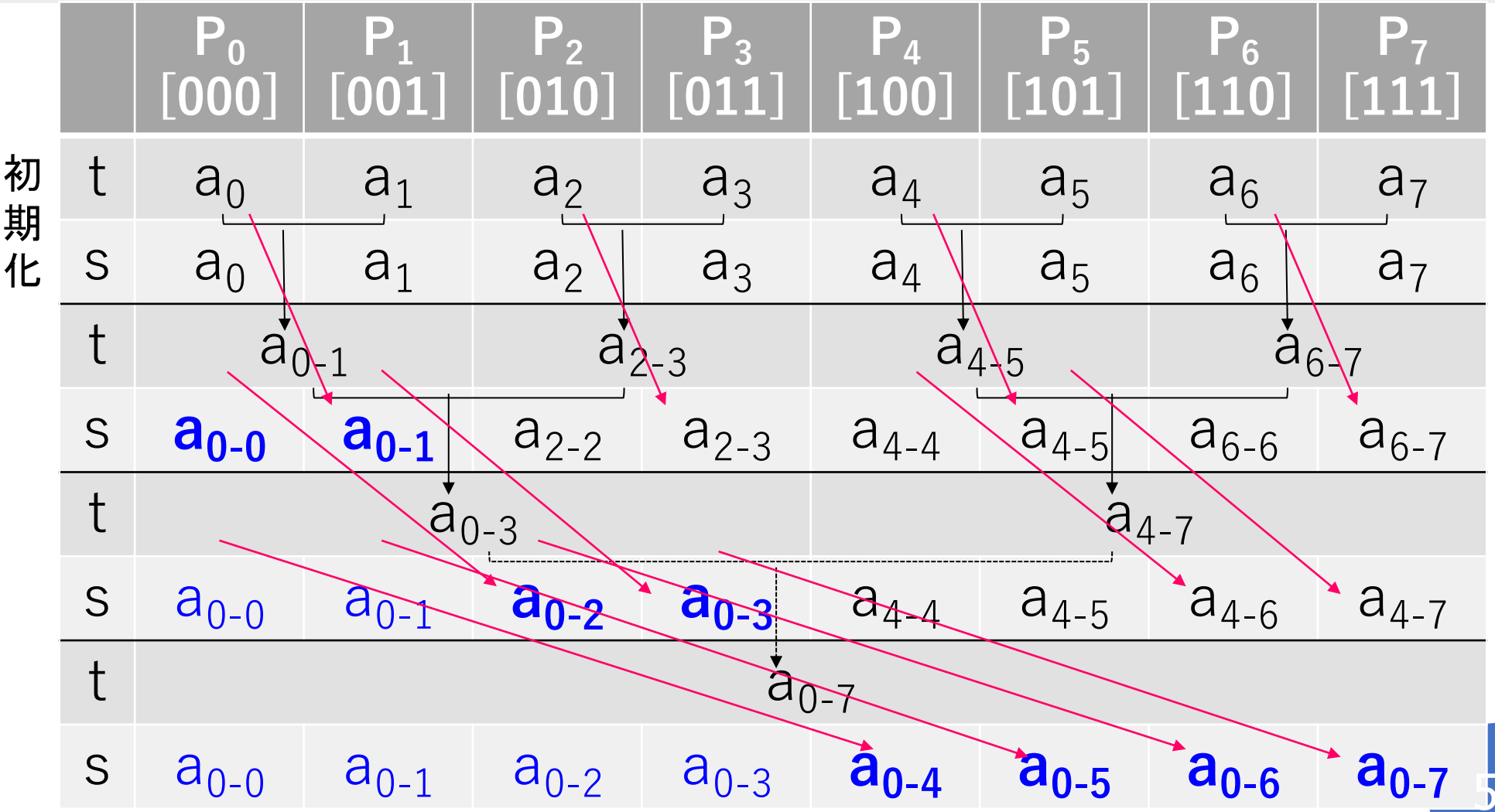
プレフィックス和の計算：Step3

■Step3： 次元2でユニキャスト
- ■互いに t_i を送信
  (受け取ったものをb_iとする)
- ■両方で$t_i \leftarrow t_i + b_i$
- ■プロセッサIDの大きい方
  $s_i \leftarrow s_i + b_i$

上段：部分和用 t_i
下段：部分プレフィックス和用s_i



プレフィックス計算の一般化



プレフィックス計算：手続き

procedure Cube_Prefix_Sum($a_0,a_1,\ldots,a_{N-1}$)
  for i=0 to N-1 do in parallel
    $s_i \leftarrow a_i$; $t_i \leftarrow a_i$ // 初期化
  end for
  for j=0 to ($\log_2 N$)-1 do
    for i=0 to N-1 do in parallel /* 各プロセッサで並列実行 */
      $k \leftarrow i$ のjビット目を反転したものとする。 P_i , $i=[i_{n-1}\ldots i_j\ldots i_0]$
      (1) P_i と P_k は互いに t_i と t_k を送信する。
      (2) P_iで
        $b_i \leftarrow t_k$; // b_i ：隣接プロセッサから送られた部分和
        $t_i \leftarrow t_i \oplus b_i$; // 持っている部分和と加算して更新
        if i_j=1 then // iのj-bit目が1なら（＝IDが大きい方なら）
          $s_i \leftarrow s_i \oplus b_i$;
    end for
  end for.

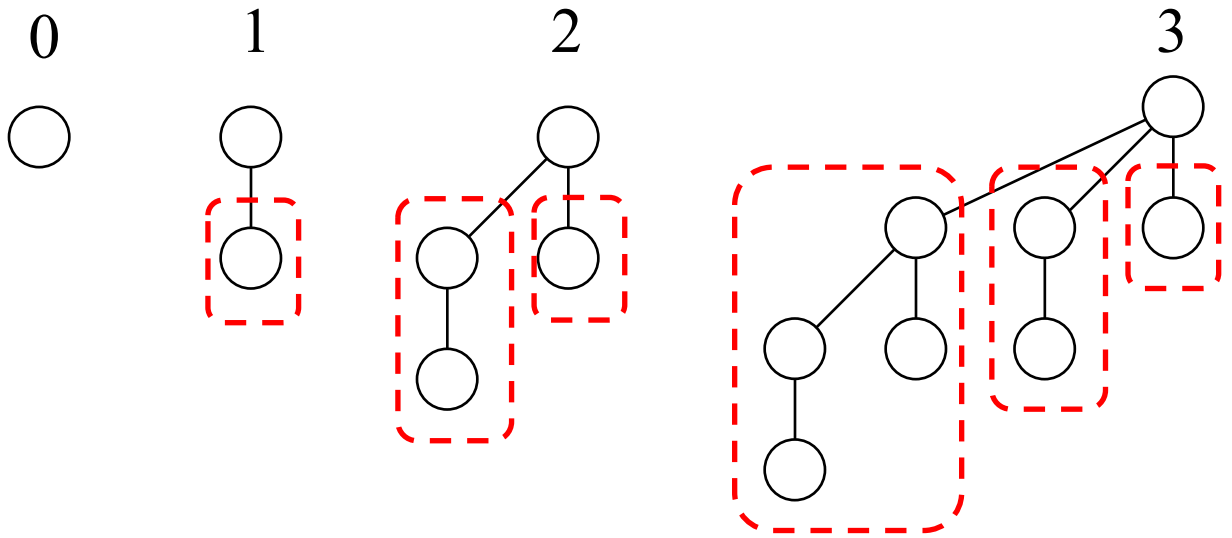
t_i : P_iの部分計算
s_i : P_iの部分プレフィックス

アルゴリズムの実行ステップ数は $\log_2 N$ に比例する。実行時間 T=O(log N)

2項木による通信

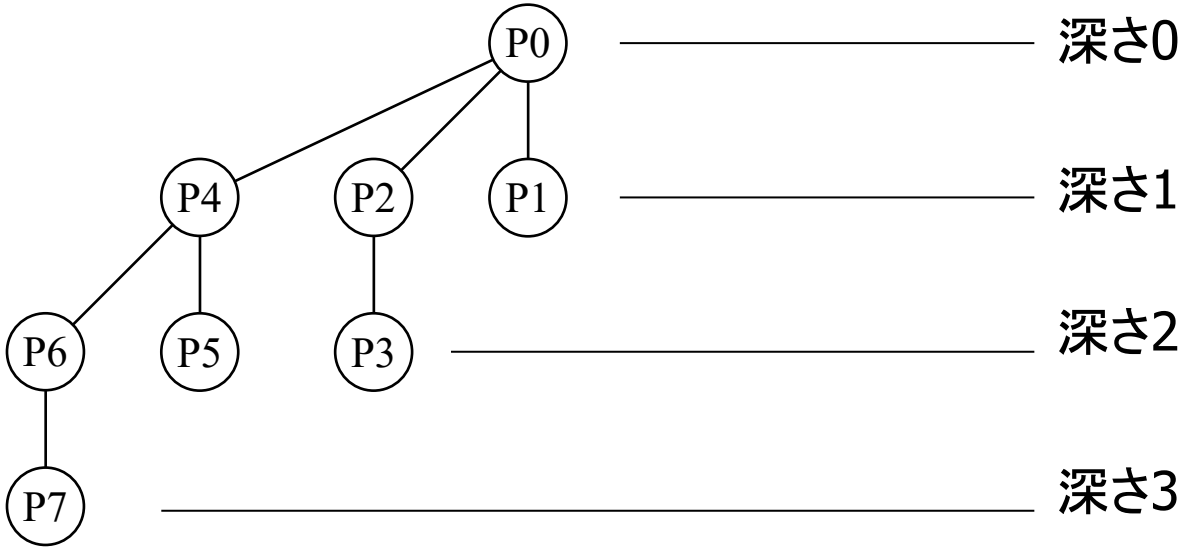
2項木(binominal tree)とは

- [再帰的定義]
  - レベル 0 の二項木は、１つのノードをもつ。
  - レベル k の二項木は、１つの根をもち、その子は
    それぞれレベル k-1, k-2, …, 2, 1, 0の二項木の親である。



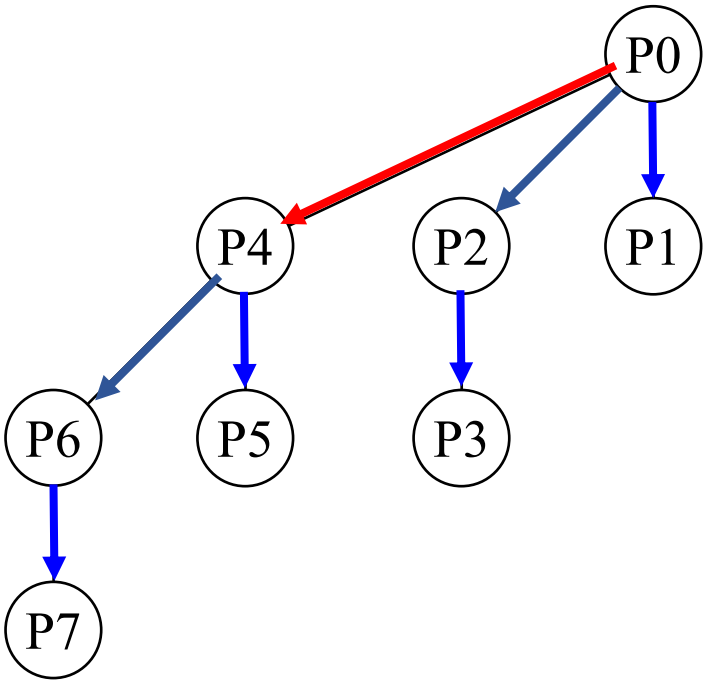
2項木の性質

- $N=2^k$ 個（k は0以上の整数）のノードをもつ2項木では以下の性質が成り立つ。
  - 木の深さは k である。
  - 深さ i $(i=0,1,\cdots,k)$のノードは、${}_kC_i$ 個ある。
  - N個のノードをもつ2項木の最大次数は$\log_2 N$である。

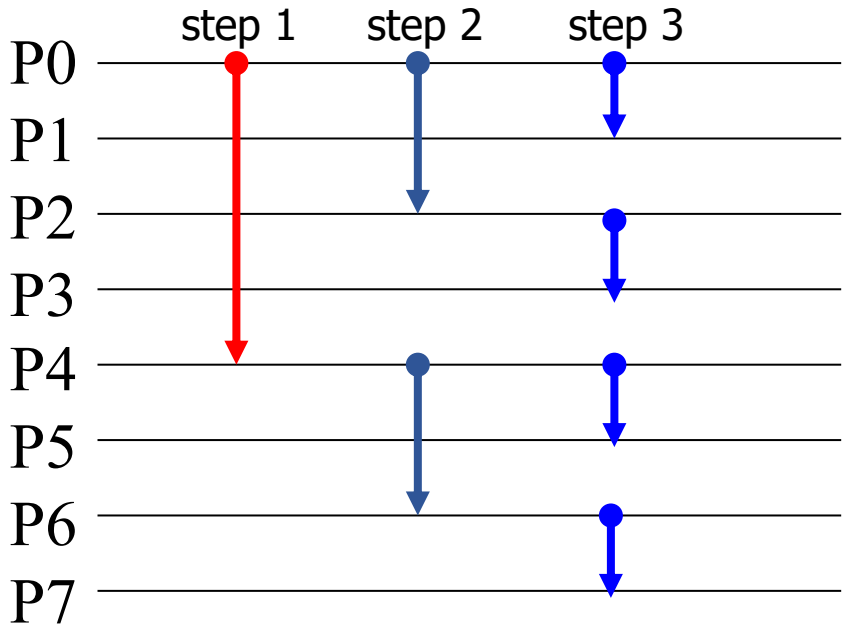


2項木によるデータ分配

■8ノードより成る2項木の根からのデータ分配



2項木によるデータ分配

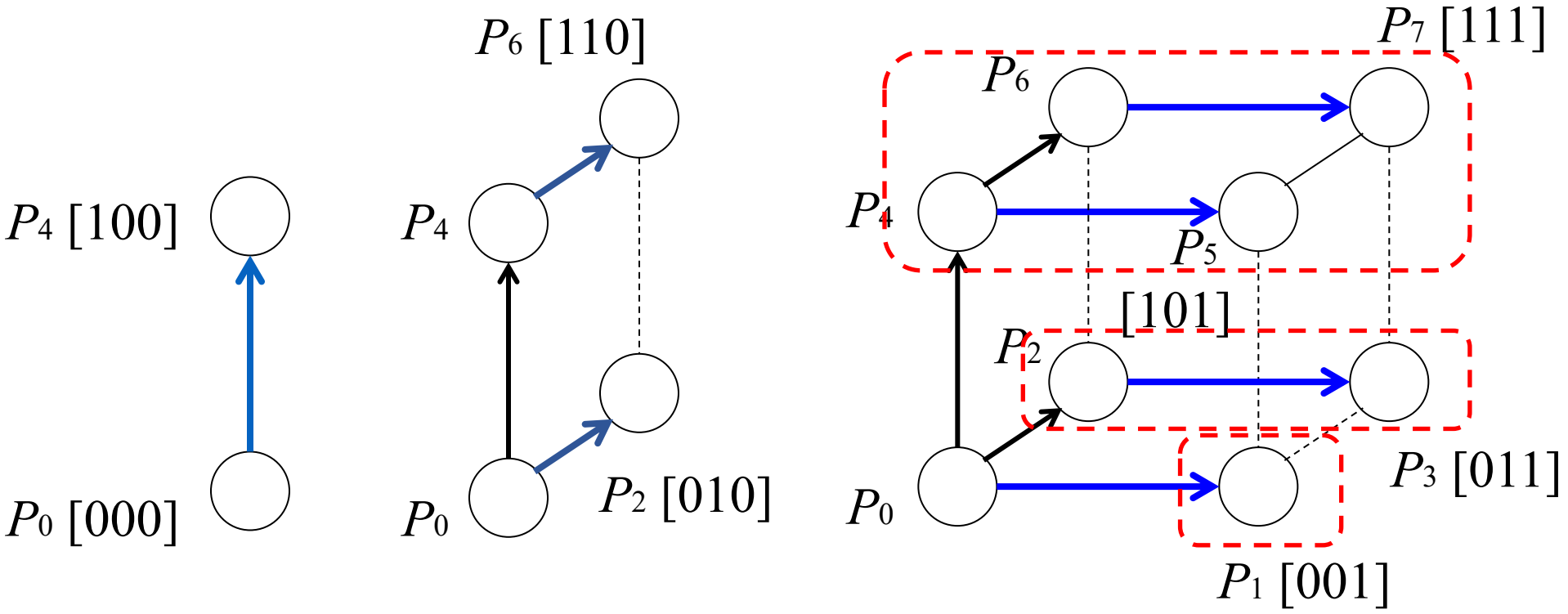


ステップごとのデータ分配

8ノードの2項木を用いたデータ分配のステップ数は3

2項木によるデータ分配（続き）

■2項木によるデータ分配を超立方体結合に射影



超立方体結合モデルにおいて
上位の次元から順にデータを分配した場合に相当

2項木によるリダクションなど

- 2項木を用いて基本的なグループ操作ができる
  - ブロードキャスト
  - 分配
  - 収集
  - リダクション