コンピュータ科学特別講義Ⅳ

Parallel Algorithm Design（#5）

Masato Edahiro

June 8, 2018

Please download handouts before class from
http://www.pdsl.jp/class/utyo2018/

Contents of This Class

- Our Target
  - Understand Systems and Algorithms on "Multi-Core" processors
- Schedule (Tentative)
  - #1 April 6 (=Today) What's "Multi-Core"?
  - #2 April 13 : Parallel Programming Languages (Ex. 1)
  - April 20, 27, May 4, 11, 18: NO CLASS
  - #3 May 25 : Parallel Algorithm Design
  - #4 June 1 (Fri) : Laws on Multi-Core
  - #5 June 8 : Examples of Parallel Algorithms (1) (Ex. 2)
  - June 15: NO CLASS
  - #6 June 22 : Examples of Parallel Algorithms (2)
  - #7 June 29 : Examples of Parallel Algorithms (3)
  - #8 July 6 : Examples of Parallel Algorithms (4)
  - #9 July 13 : Examples of Parallel Algorithms (5) (Ex. 3)
  - (July 20)

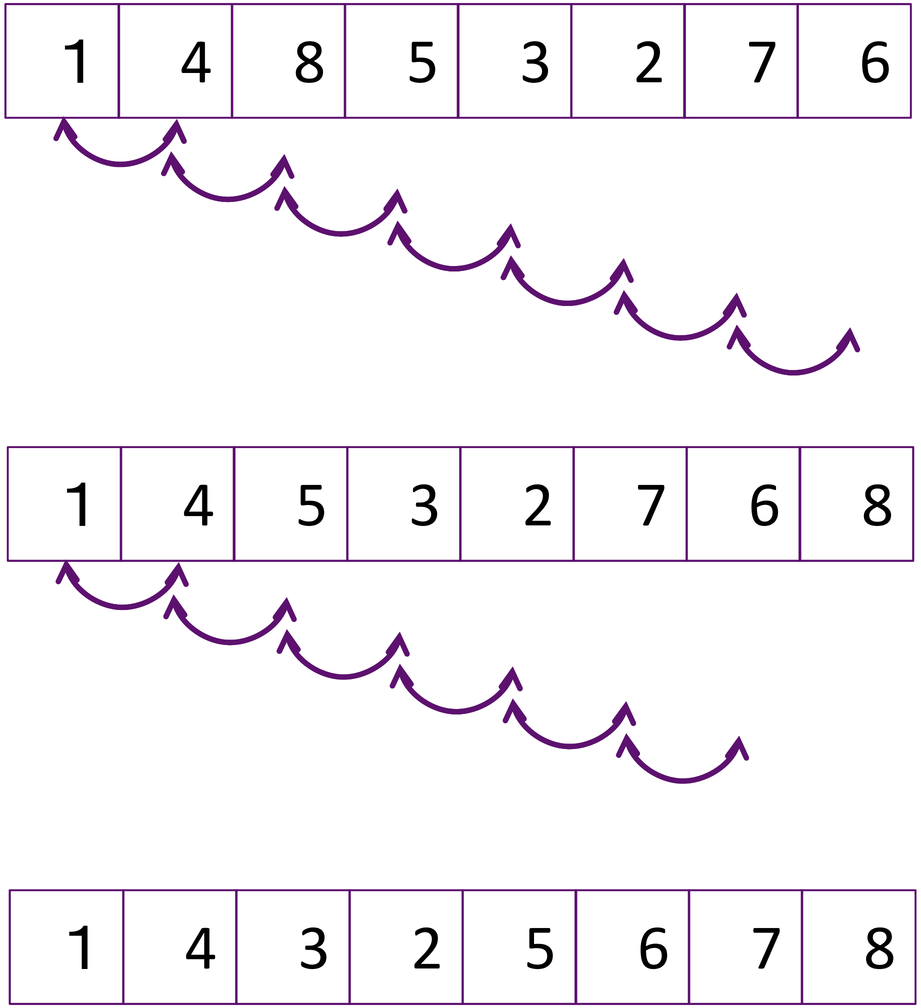
ソート（1）

- 与えられた数の列に対し、昇順（または降順）に並べ替えること

- 今回：実用的アルゴリズム

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

バブルソート



時間計算
複雑度？

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

奇偶転置ソート（並列バブルソート）

1	4	8	5	3	2	7	6

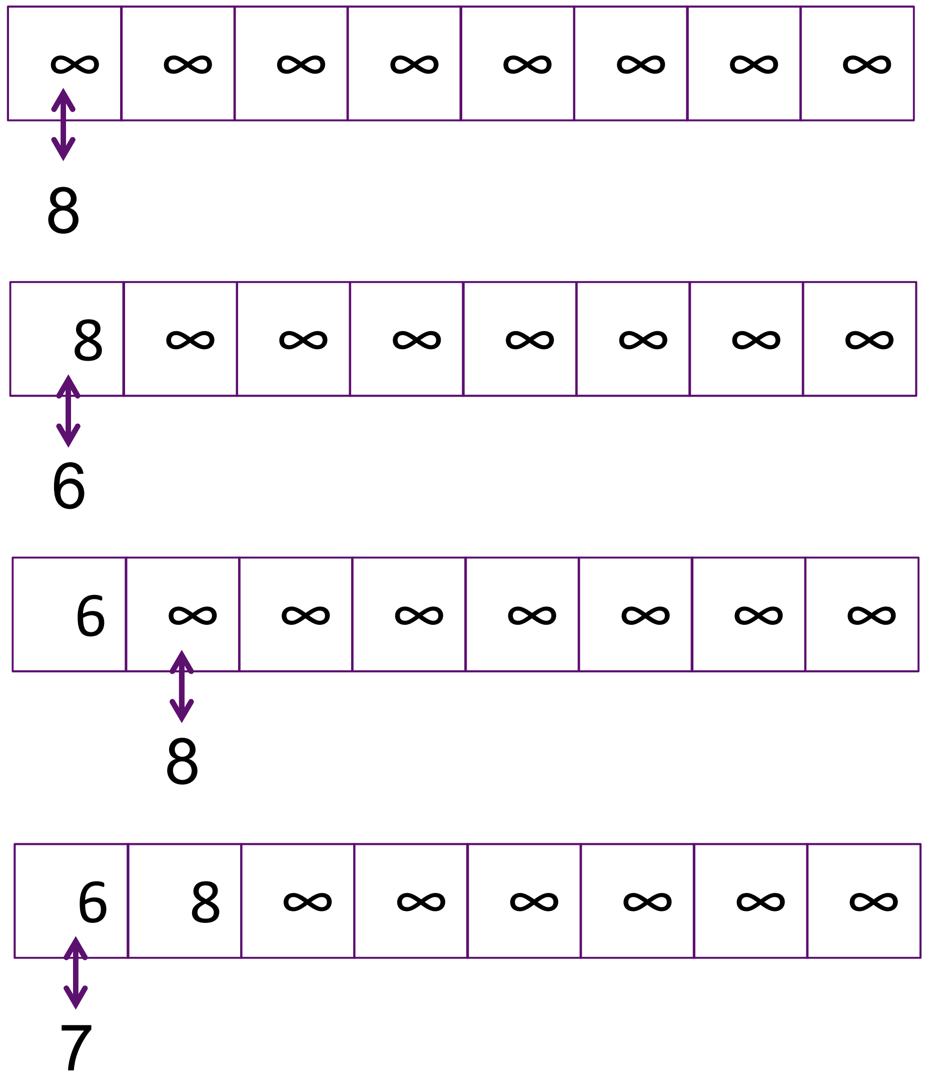
1	4	5	8	2	3	6	7

1	4	5	2	8	3	6	7

1	4	2	5	3	8	6	7

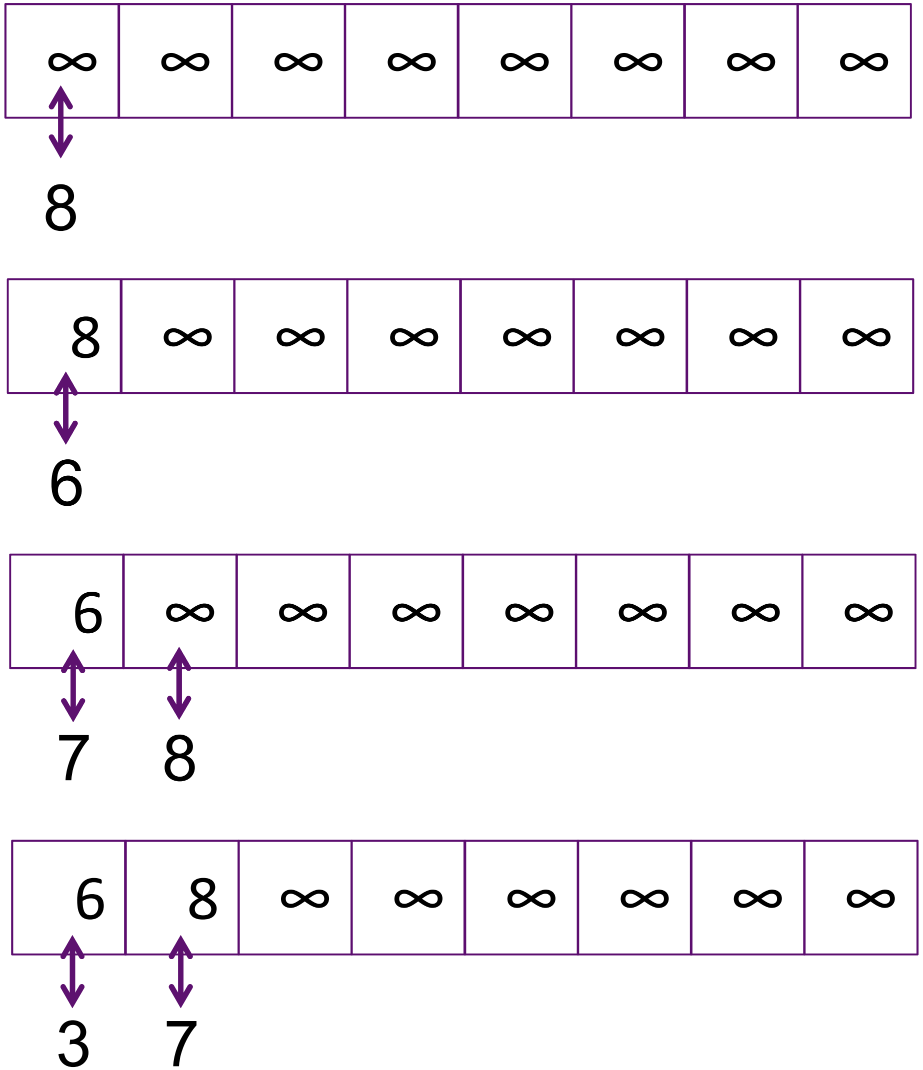
Pプロセッサのときの時間計算複雑度？

挿入ソート



時間計算
複雑度？

並列挿入ソート



Pプロセッサのときの時間計算複雑度？

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Quicksort

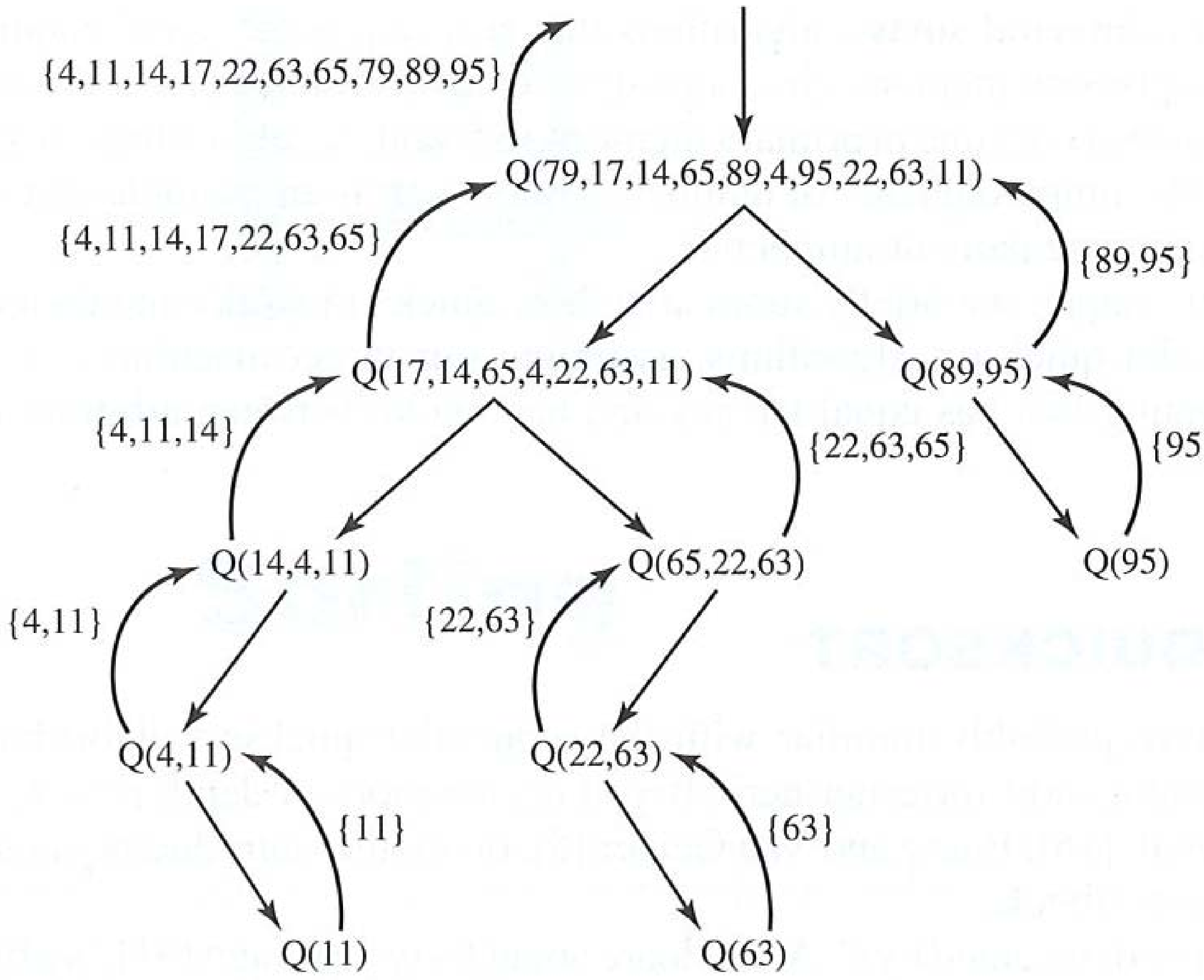


Figure 14.1 Sorting a 10-element list using quicksort. Each Q represents a call to quicksort. The algorithm removes the first element from the list, using it as a pivot to divide the list into two parts. It calls itself recursively to sort the two sublists. (The call is omitted for empty sublists.) It returns the concatenation of the sorted “low list,” the pivot, and the sorted “high list.”

時間計算
複雑度？

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Quicksortとは

key

79 , 17 , 14 , 65 , 89 , 4 , 85 , 22 , 63 , 11 , 33 , 95 , 1

79 , 17 , 14 , 65 , 4 22 , 63 , 11 , 33 , 1 | 89 , 95 , 85

1 , 4 | 79 , 17 , 14 , 65 , 22 , 63 , 11 , 33 | 85 , 89 | 95

1 , 4 , 11 , 14 , 17 , 22 , 33 , 63 , 65 , 79 , 85 , 89 , 95

並列ソート

- （ここでの）ソートの定義
  - ソートされていない値のリストが、プロセッサの主記憶にほぼ同数分散されて記憶されている
  - ソートの完了時は
    - すべてのプロセッサに記憶されている数のリストはソートされている
    - すべての i ($0 \leqq i \leqq p-2$) に対してP_iのリストの最後の値は、P_{i+1}のリストの最初の値よりも同じか小さい
    - 各プロセッサに記憶されている値の数は同数である必要はない

並列Quick Sort

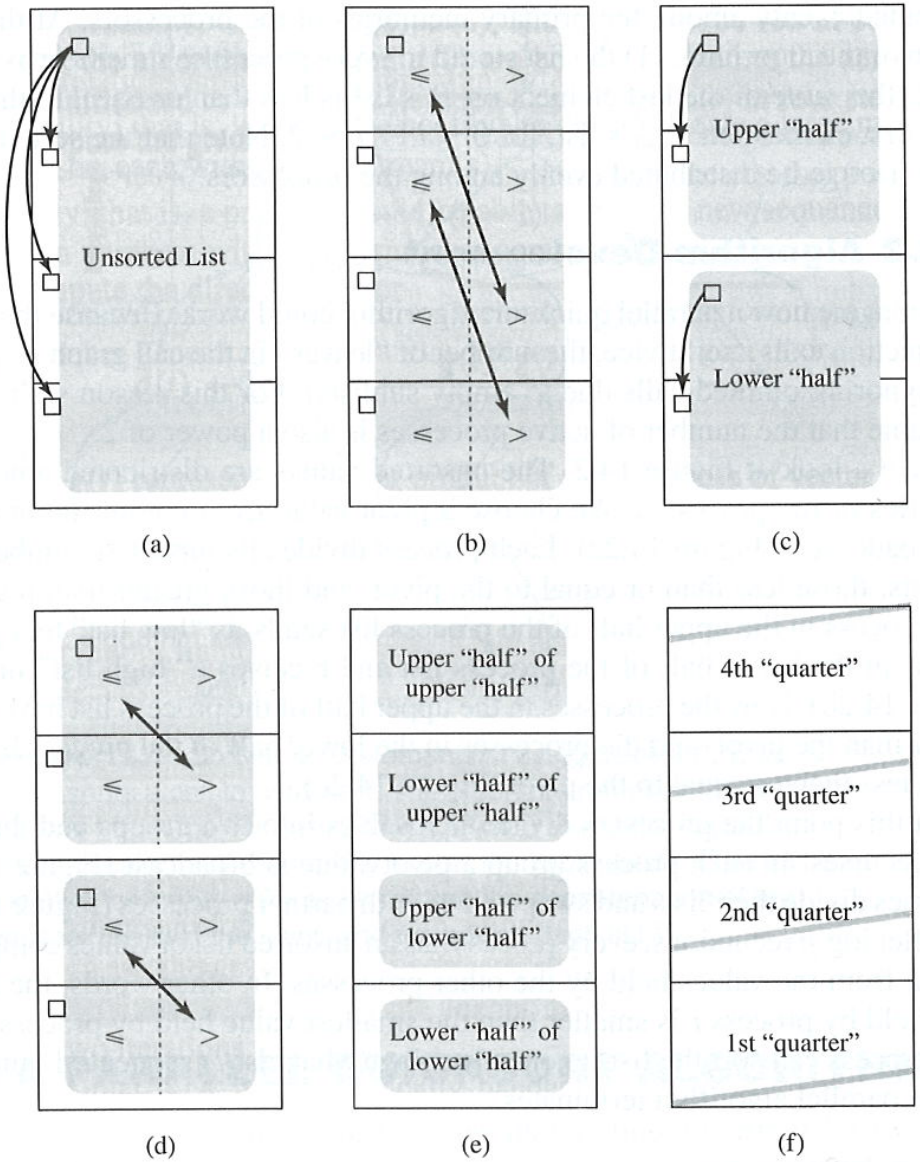


Figure 14.2 High-level view of a parallel quicksort algorithm. (a) Initially the unsorted values are distributed among the memories of all the processes. A single value is chosen as the pivot. The pivot is broadcast to the other processes. (b) Processes use the pivot to divide their numbers into those in the "lower half" and those in the "upper half." Each process in the upper half swaps values with a partner in the lower half. (c) The algorithm recurses. A single value from each "half" is chosen as the pivot for that "half" and broadcast to the other process responsible for that "half." (d) As in step (b), processes use the pivot to divide their numbers. Upper processes swap with lower processes, swapping smaller values for larger values. (e) At this point the largest value held by process i is less than the smallest value held by process $i + 1$. (f) Each process uses quicksort to sort the elements it controls. The list is now sorted.

- 負荷バランスを良くするために、pivotの選択が鍵

Pプロセッサのときの時間計算複雑度？

並列Quicksort (1)

11 , 50 , 53 , 95 , 36 , 67 , 86 , 44	35 , 16 , 81 , 1 , 44 , 23 , 15 , 5
97 , 48 , 16 , 8 , 66 , 96 , 17 , 49	58 , 76 , 54 , 39 , 82 , 47 , 65 , 51

並列Quicksort (2)

11 , 50 , 53 , 44 , 56 , 67 , 86 , 95

49 , 48 , 16 , 8 , 66 , 17 , 96 , 97

35 , 16 , 5 , 1 , 44 , 23 , 15 , 81

58 , 51 , 54 , 39 , 65 , 47 , 82 , 76

11 , 50 , 53 , 44 , 56 , 67 , 49 , 48 , 16 , 8 , 66 , 17

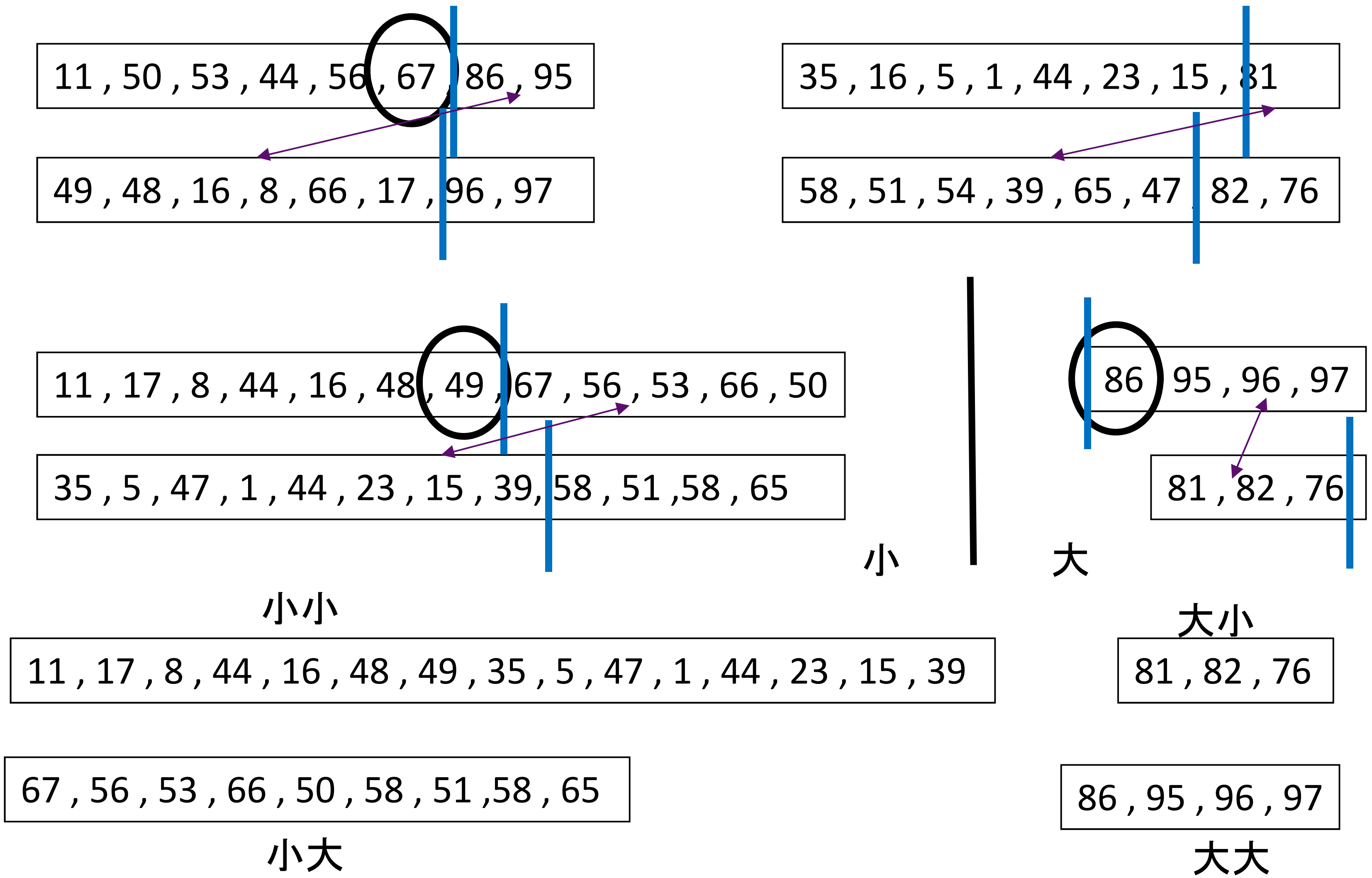
35 , 5 , 1 , 44 , 23 , 15 , 58 , 51 , 54 , 39 , 65 ,47

86 , 95 , 96 , 97

81 , 82 , 76

小 大

並列Quicksort (3)



Hyper Quick Sort

Sort! (または中央値を計算)

Sort! (または中央値を計算)

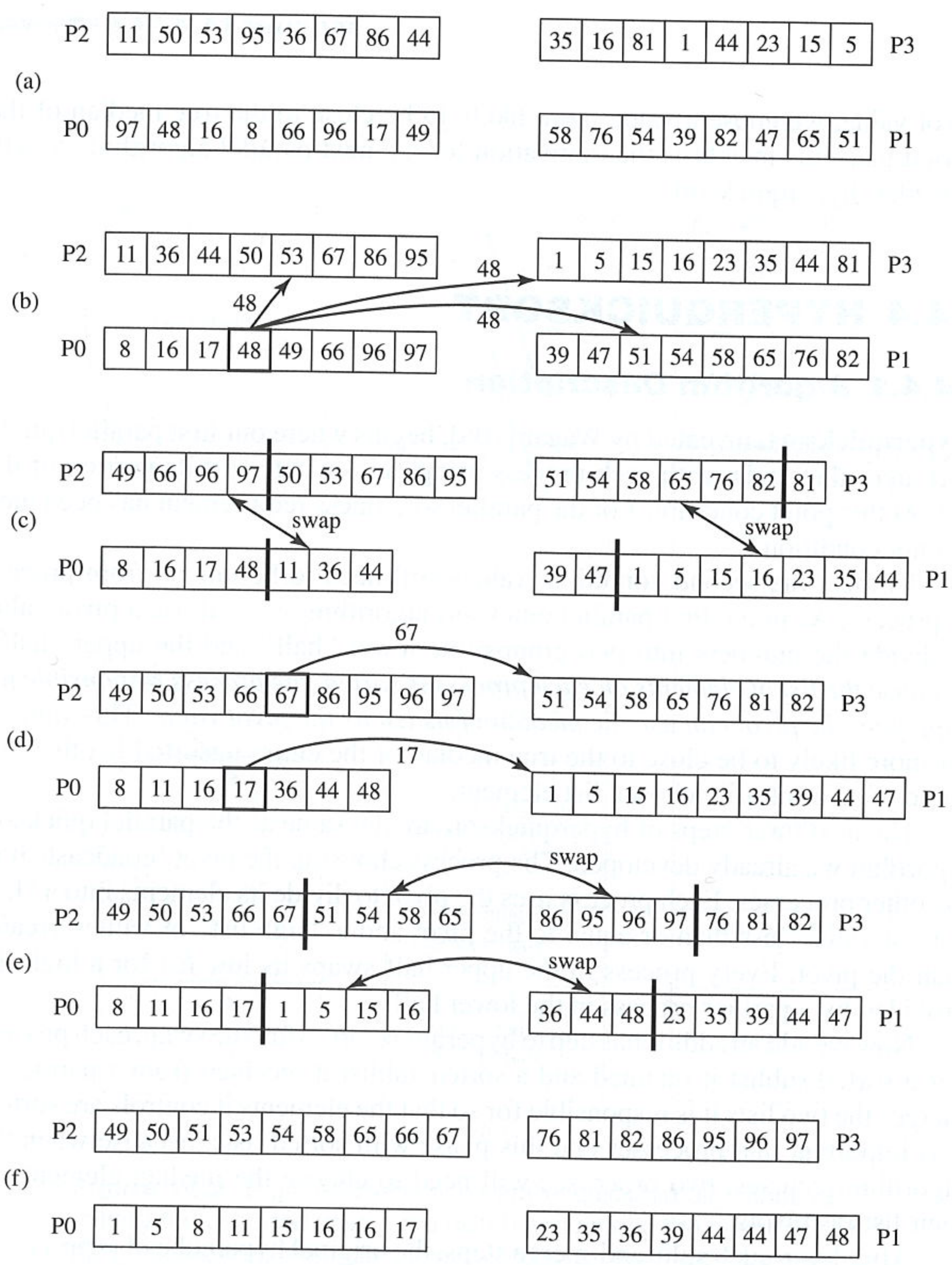


Figure 14.3 Illustration of the hyperquicksort algorithm. In this example 32 elements are being sorted on four processes logically organized as a two-dimensional hypercube. (a) Initially, each process has eight numbers. (b) Each process sorts its own list using quicksort. Process 0 broadcasts its median value, 48, to the other processes. (c) Processes in the lower half of the hypercube send values greater than 48 to processes in the upper half. The processes in the upper half send down values less than or equal to 48. (d) Each process merges the numbers it kept with the numbers it received. Process 0 broadcasts its median value to process 1, and process 2 broadcasts its median value to process 3. (e) Processes swap values across another hypercube dimension. (f) Each process merges the numbers it kept with the numbers it received. At this point the list is sorted.

Hyper Quicksort (1)

11 , 50 , 53 , 95 , 36 , 67 , 86 , 44

35 , 16 , 81 , 1 , 44 , 23 , 15 , 5

97 , 48 , 16 , 8 , 66 , 96 , 17 , 49

58 , 76 , 54 , 39 , 82 , 47 , 65 , 51

Hyper Quicksort (2)

11 , 36 , 44 , 50 , 53 , 67 , 86 , 95

1 , 5 , 15 , 16 , 23 , 35 , 44 , 81

8 , 16 , 17 , 48 , 49 , 66 , 96 , 97

39 , 47 , 51 , 54 , 58 , 65 , 76 , 82

小 11 , 36 , 44 , 8 , 16 , 17 , 48

1 , 5 , 15 , 16 , 23 , 35 , 44 , 39 , 47

大 50 , 53 , 67 , 86 , 95 , 49 , 66 , 96 , 97

51 , 54 , 58 , 65 , 76 , 82 , 81

Hyper Quicksort (3)

11 , 36 , 44 , 50 , 53 , 67 , 86 , 95

1 , 5 , 15 , 16 , 23 , 35 , 44 , 81

8 , 16 , 17 , 48 , 49 , 66 , 96 , 97

39 , 47 , 51 , 54 , 58 , 65 , 76 , 82

小

8 , 11 , 16 , 17 , 36 , 44 , 48

1 , 5 , 15 , 16 , 23 , 35 ,39 , 44 , 47

大

49 , 50 , 53 , 66 , 67 , 86 , 95 , 96 , 97

51 , 54 , 58 , 65 , 76 , 81 , 82

Hyper Quicksort (4)

11 , 36 , 44 , 50 , 53 , 67 , 86 , 95

1 , 5 , 15 , 16 , 23 , 35 , 44 , 81

8 , 16 , 17 , 48 , 49 , 66 , 96 , 97

39 , 47 , 51 , 54 , 58 , 65 , 76 , 82

小小 8 , 11 , 16 , 17 , 1 , 5 , 16

36 , 44 , 48 , 23 , 35 ,39 , 44 , 47 小大

大小 49 , 50 , 53 , 66 , 67 , 51 , 54 58 , 65

86 , 95 , 96 , 97 , 76 , 81 , 82 大大

規則的サンプリングによる並列ソート

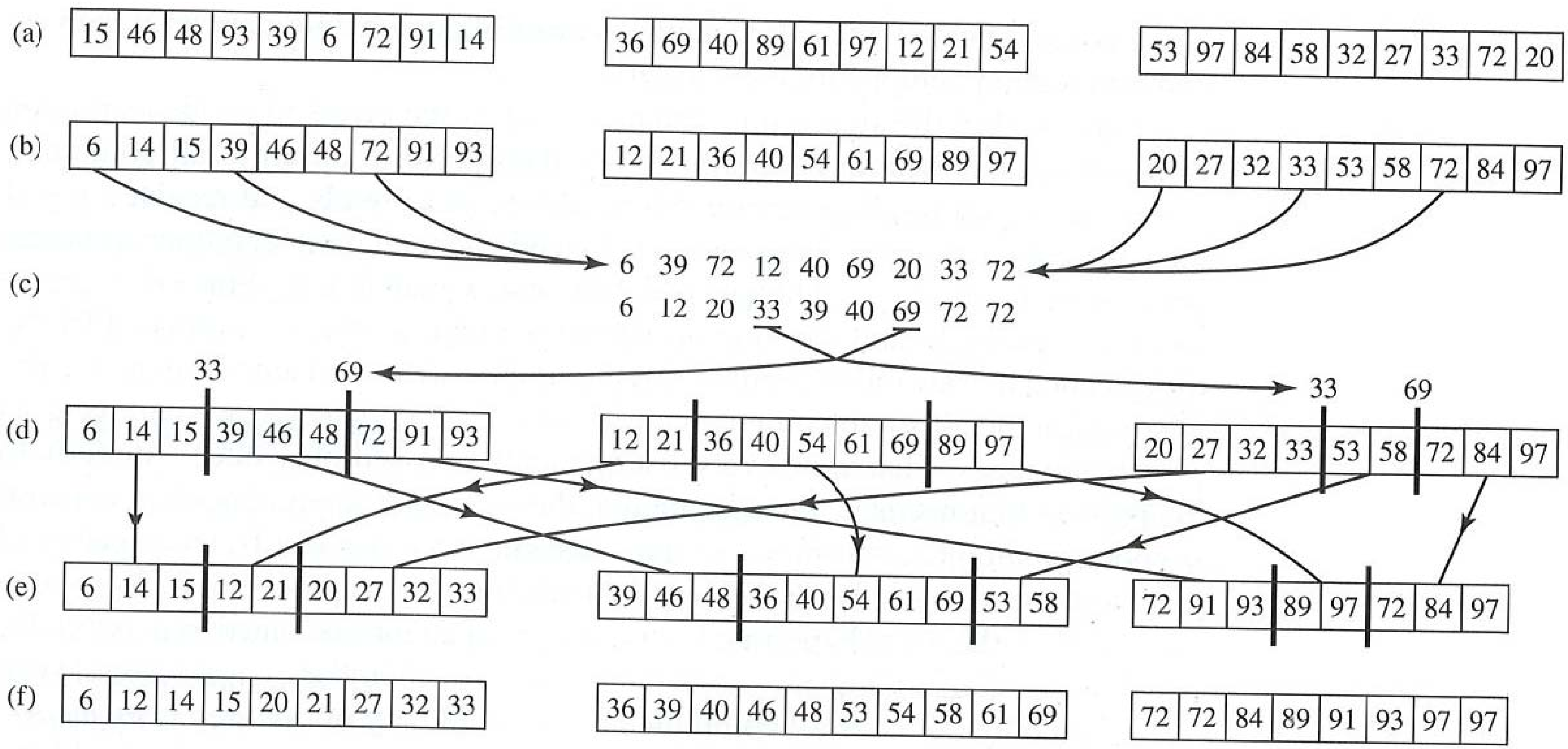


Figure 14.5 This example illustrates how three processes would sort 27 elements using the PSRS algorithm. (a) Original unsorted list of 27 elements is divided among three processes. (b) Each process sorts its share of the list using sequential quicksort. (c) Each process selects regular samples from its sorted sublist. A single process gathers these samples, sorts them, and broadcasts pivot elements from the sorted list of samples to the other processes. (d) Processes use pivot elements computed in step (c) to divide their sorted sublists into three parts. (e) Processes perform an all-to-all communication to migrate the sorted sublist parts to the correct processes. (f) Each process merges its sorted sublists.

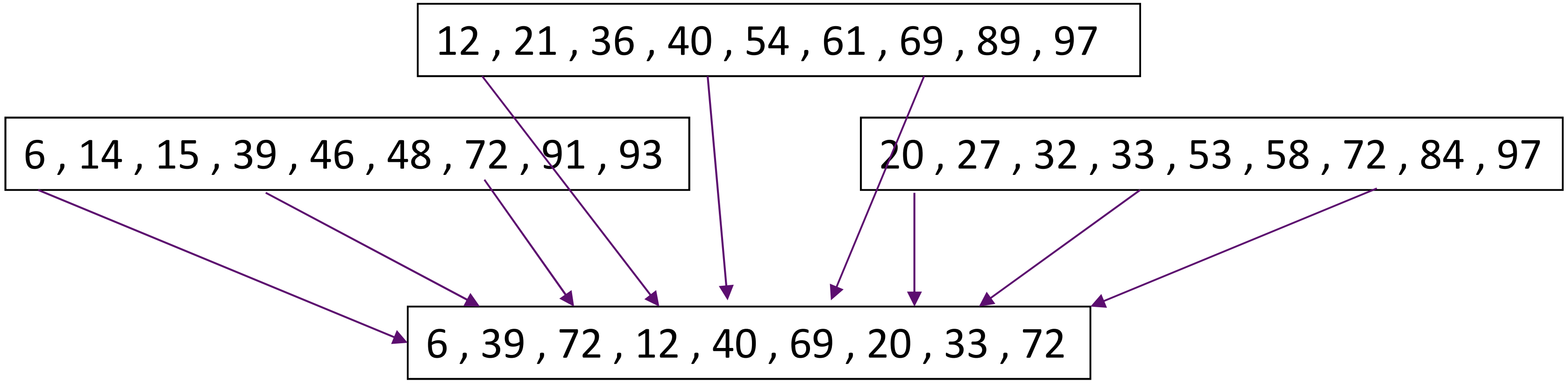
規則的サンプリングによる並列ソート (1)

36 , 69 , 40 , 89 , 61 , 97 , 12 , 21 , 54

15 , 46 , 48 , 93 , 39 , 6 , 72 , 91 , 14

53 , 97 , 84 , 58 , 32 , 27 , 33 , 72 ,20

規則的サンプリングによる並列ソート (2)



規則的サンプリングによる並列ソート (3)

12 , 21 , 36 , 40 , 54 , 61 , 69 , 89 , 97

6 , 14 , 15 , 39 , 46 , 48 , 72 , 91 , 93

20 , 27 , 32 , 33 , 53 , 58 , 72 , 84 , 97

6 , 12 , 20 , 33 , 39 , 40 , 69 , 72 , 72

12 , 21 , 36 , 40 , 54 , 61 , 69 , 89 , 97

6 , 14 , 15 , 39 , 46 , 48 , 72 , 91 , 93

20 , 27 , 32 , 33 , 53 , 58 , 72 , 84 , 97

中

39 , 46 , 48 , 36 , 40 , 54 ,61 , 69 , 89 ,97

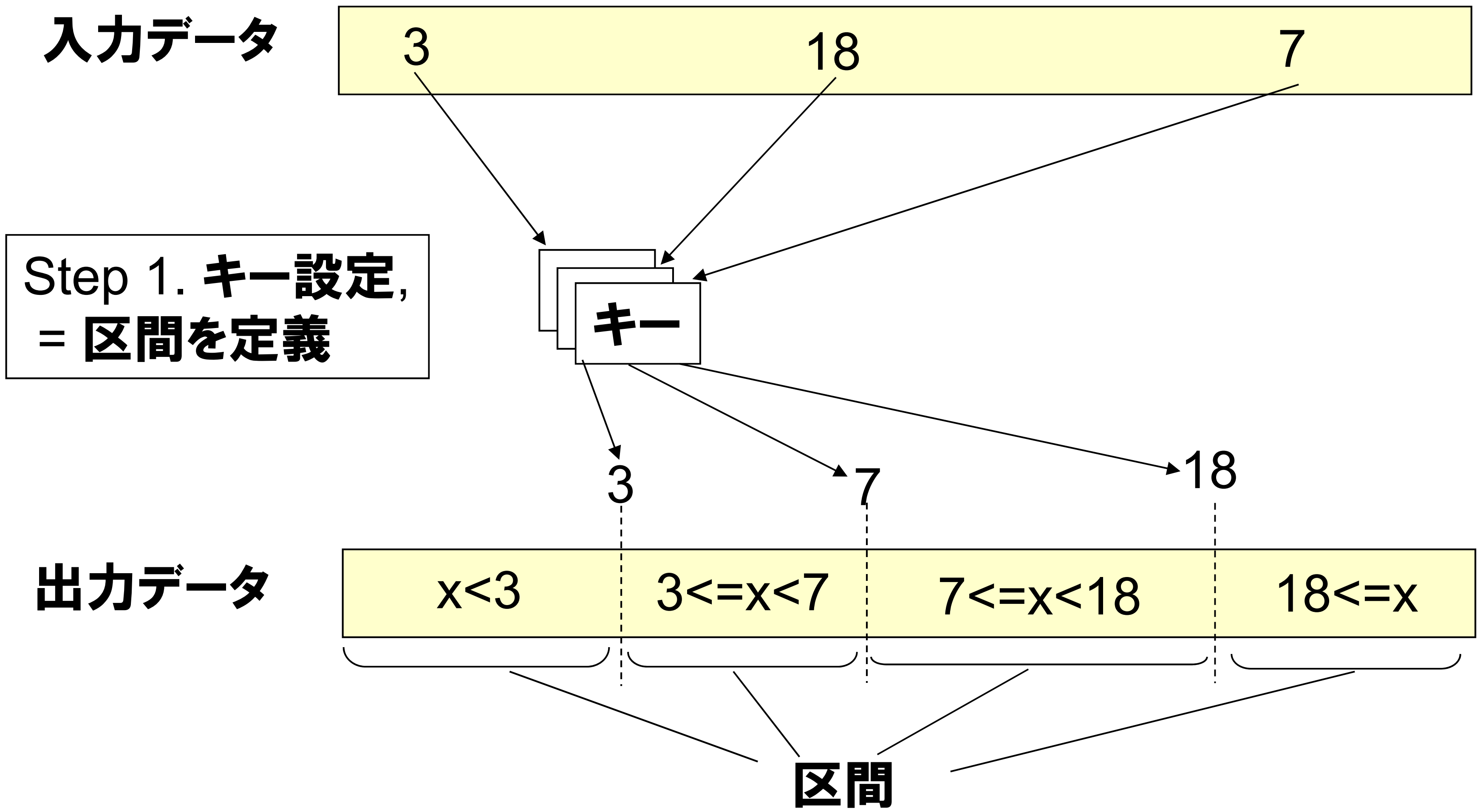
小

6 , 14 , 15 , 12 , 21 , 20 , 27 , 32 , 33

大

72 , 91 , 93 , 89 , 97 , 72 , 84 , 97

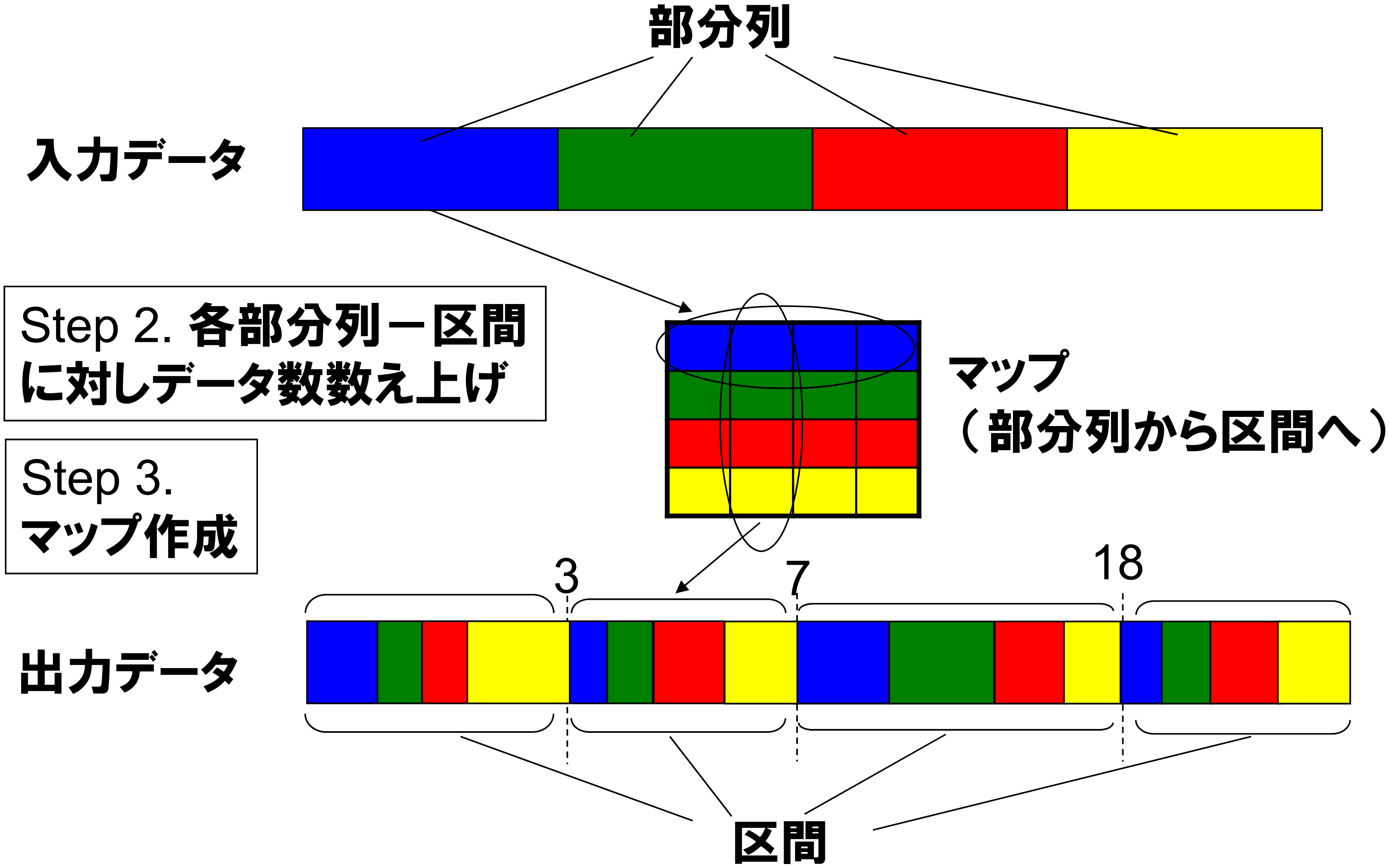
Map Sort --- 基本アイデア (1)



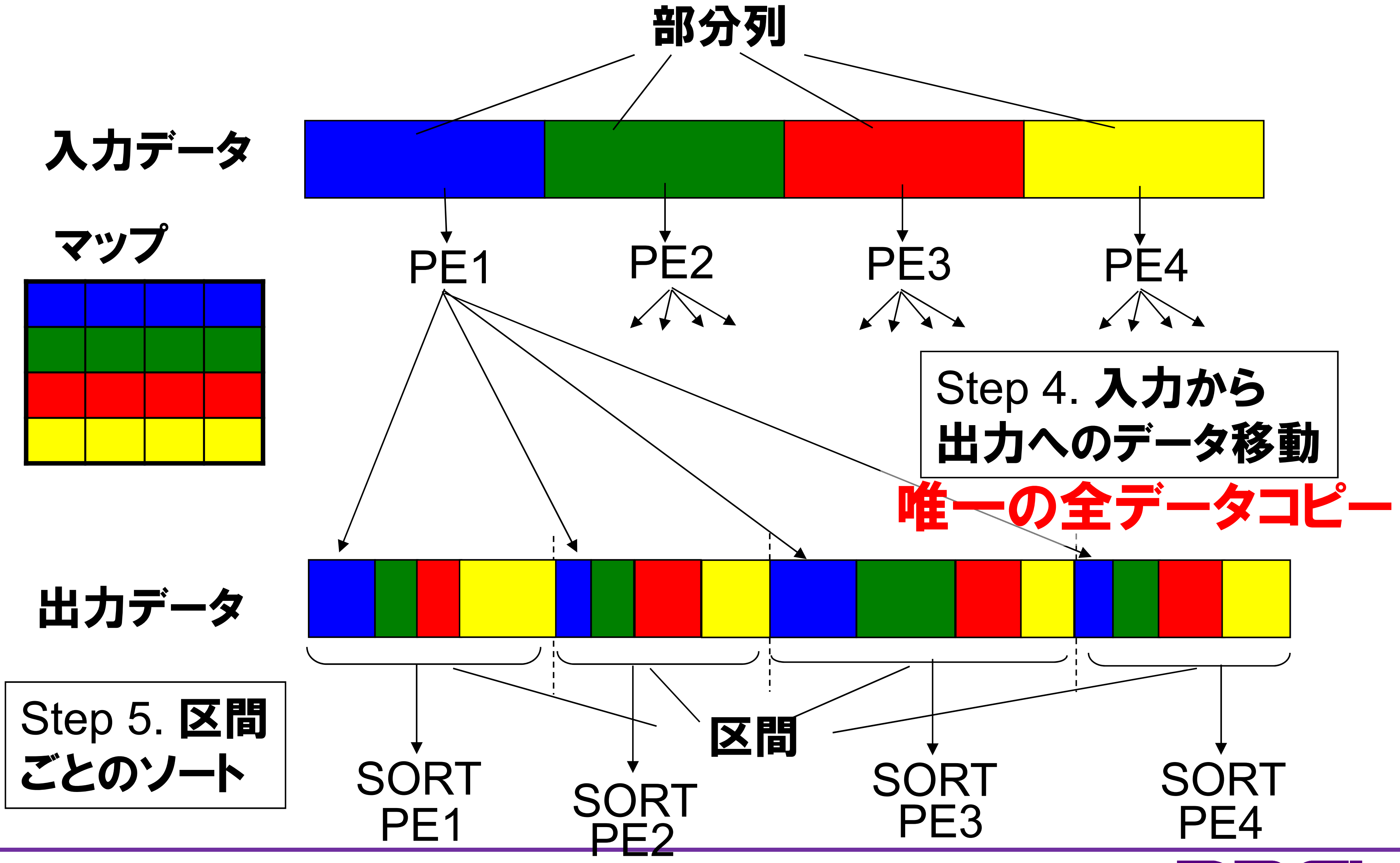
Map Sort --- 基本アイデア (2)

Step 2. 各部分列ー区間に対しデータ数数え上げ

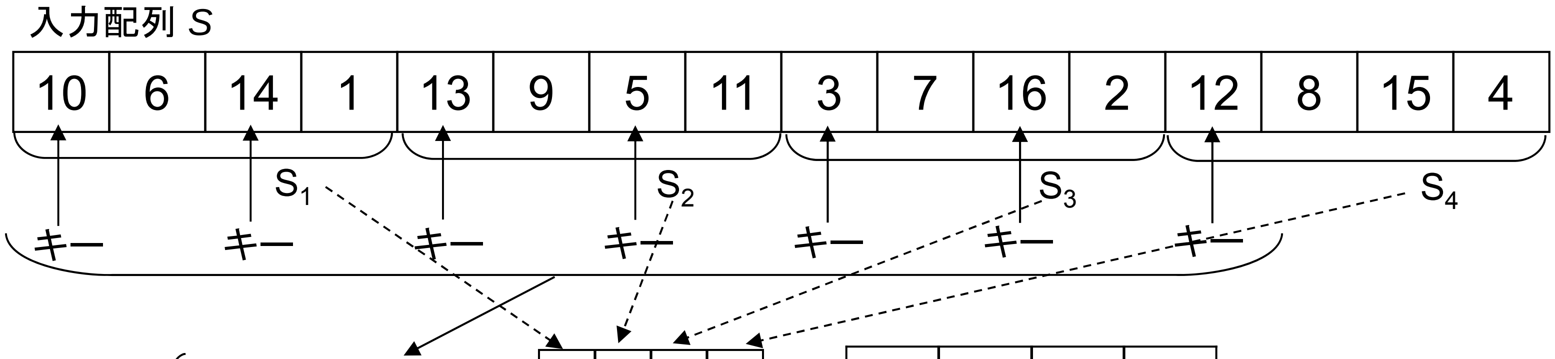
Step 3. マップ作成



Map Sort --- 基本アイデア (3)



Map Sort --- 例 (1)



区間

1	x<3	1	0	1	0	0	-	1	-
2	3<=x<5	0	0	1	1	-	-	2	3
3	5<=x<10	1	2	1	1	4	5	7	8
4	10<=x<12	1	1	0	0	9	10	-	-
5	12<=x<13	0	0	0	1	-	-	-	11
6	13<=x<14	0	1	0	0	-	12	-	-
7	14<=x<16	1	0	0	1	13	-	-	14
8	16<=x	0	0	1	0	-	-	15	-

Step 1. キー設定, = 区間を定義

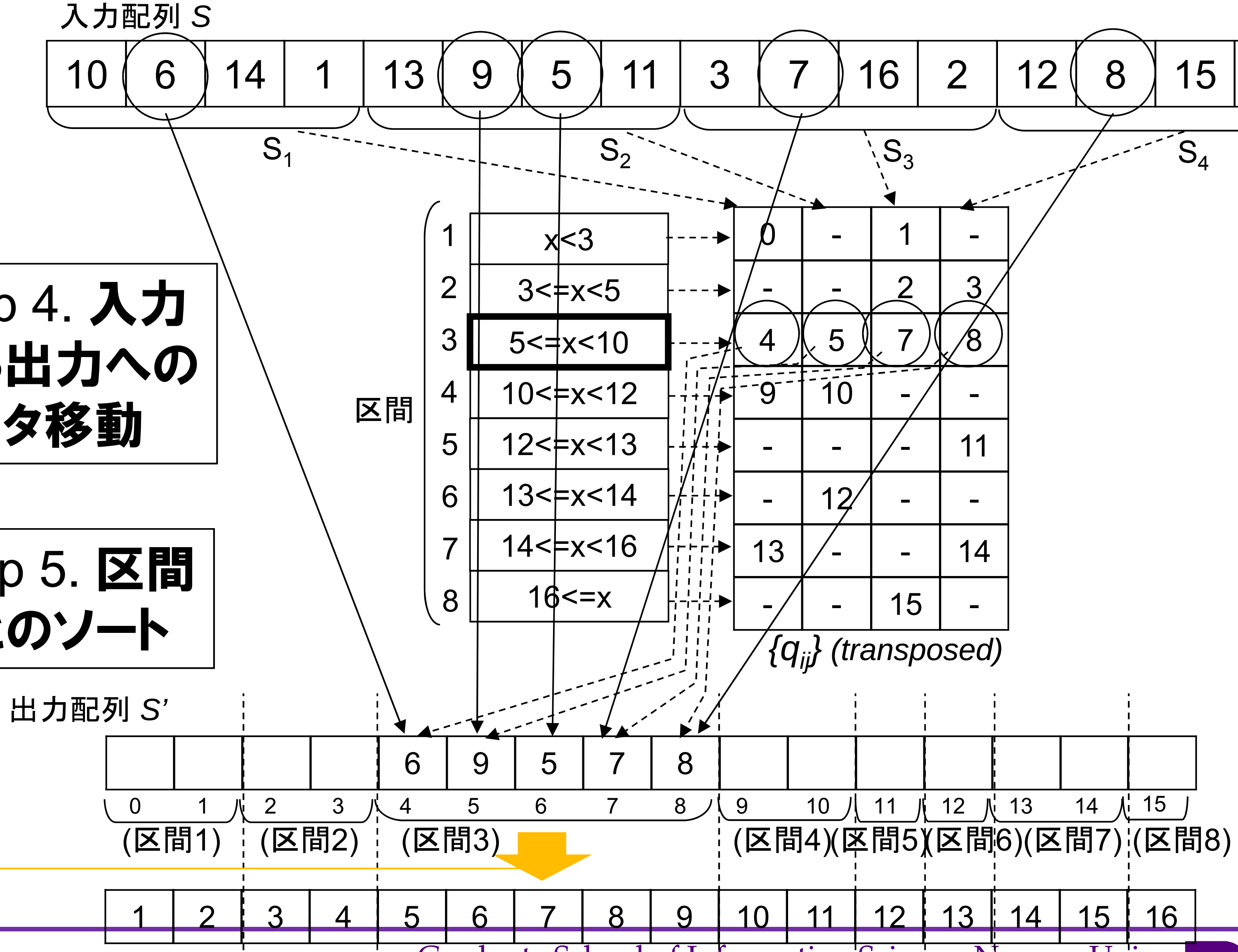
Step 2. 各部分列ー区間に対しデータ数数え上げ

Step 3. マップ作成

Map Sort --- 例 (2)

Step 4. **入力から出力へのデータ移動**

Step 5. **区間ごとのソート**

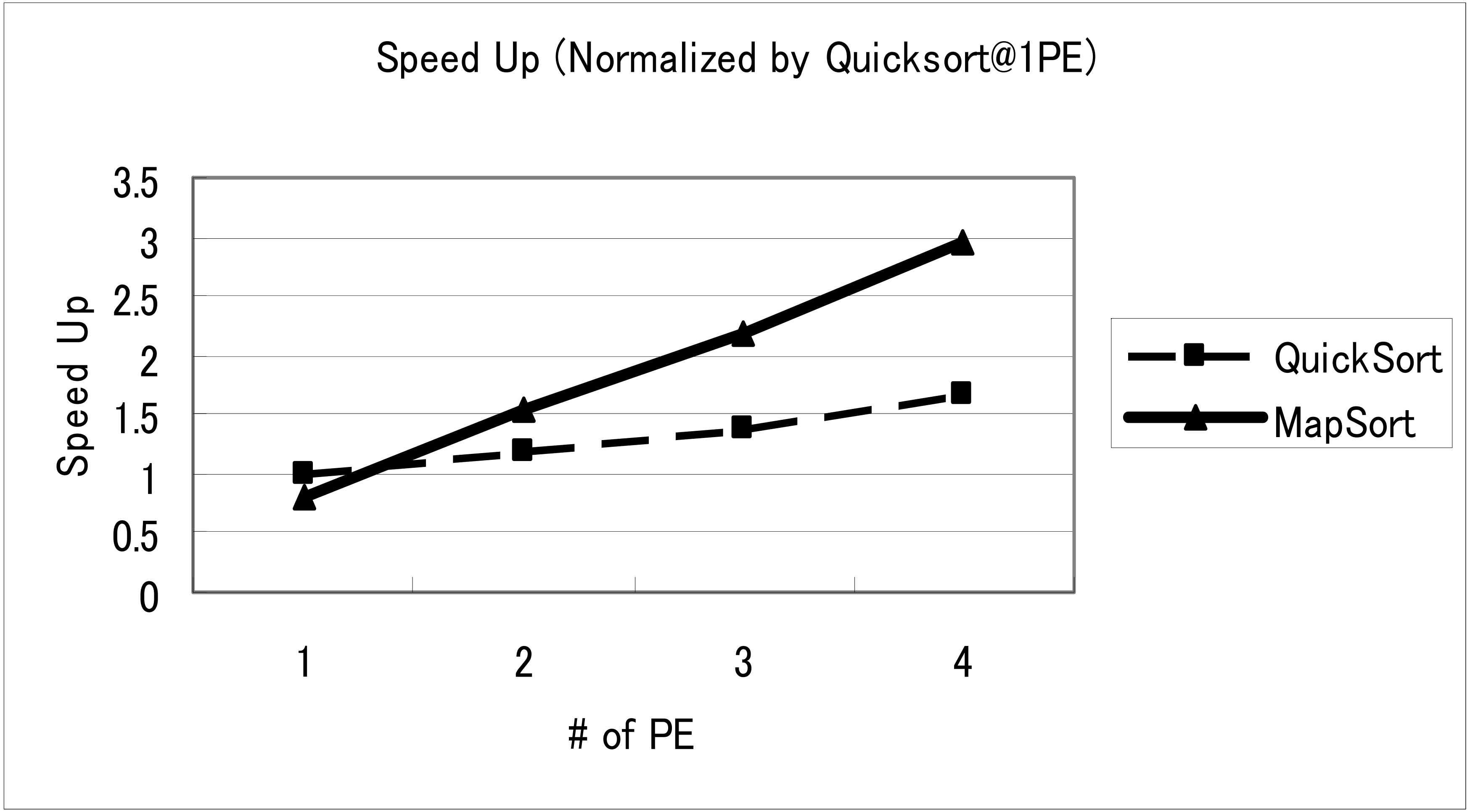


Complexity

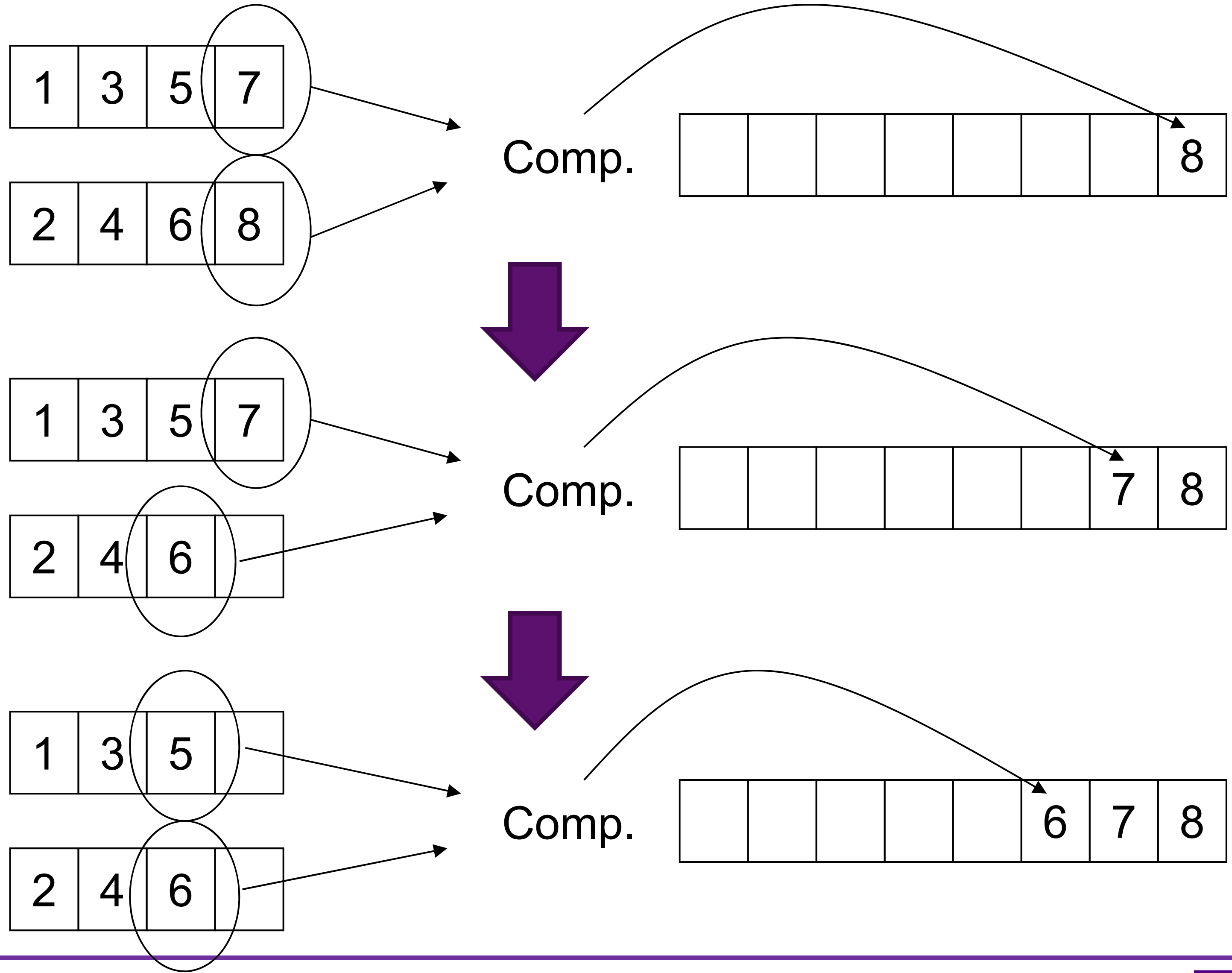
- Assuming L, $M=O(P)$, P: # of PEs,
  P: constant in practice (up to $O(\sqrt{N})$)
- Space Complexity: $O(N+P^2)$
  - Input/Output Array: $O(N)$, Map: $O(P^2)$
- Time Complexity: $O((N/P) \log N)$ with P PEs
  - Find Keys & Define Intervals: $O(P \log P)$
  - Count Data: $O((N/P) \log P)$
  - Construct Map: $O(P)$
  - Move Data from Input to Output: $O(N/P)$
  - Sort on Intervals:
    $O((N/P) \log (N/P)) = O((N/P) \log N)$

Map Sort --- 評価結果

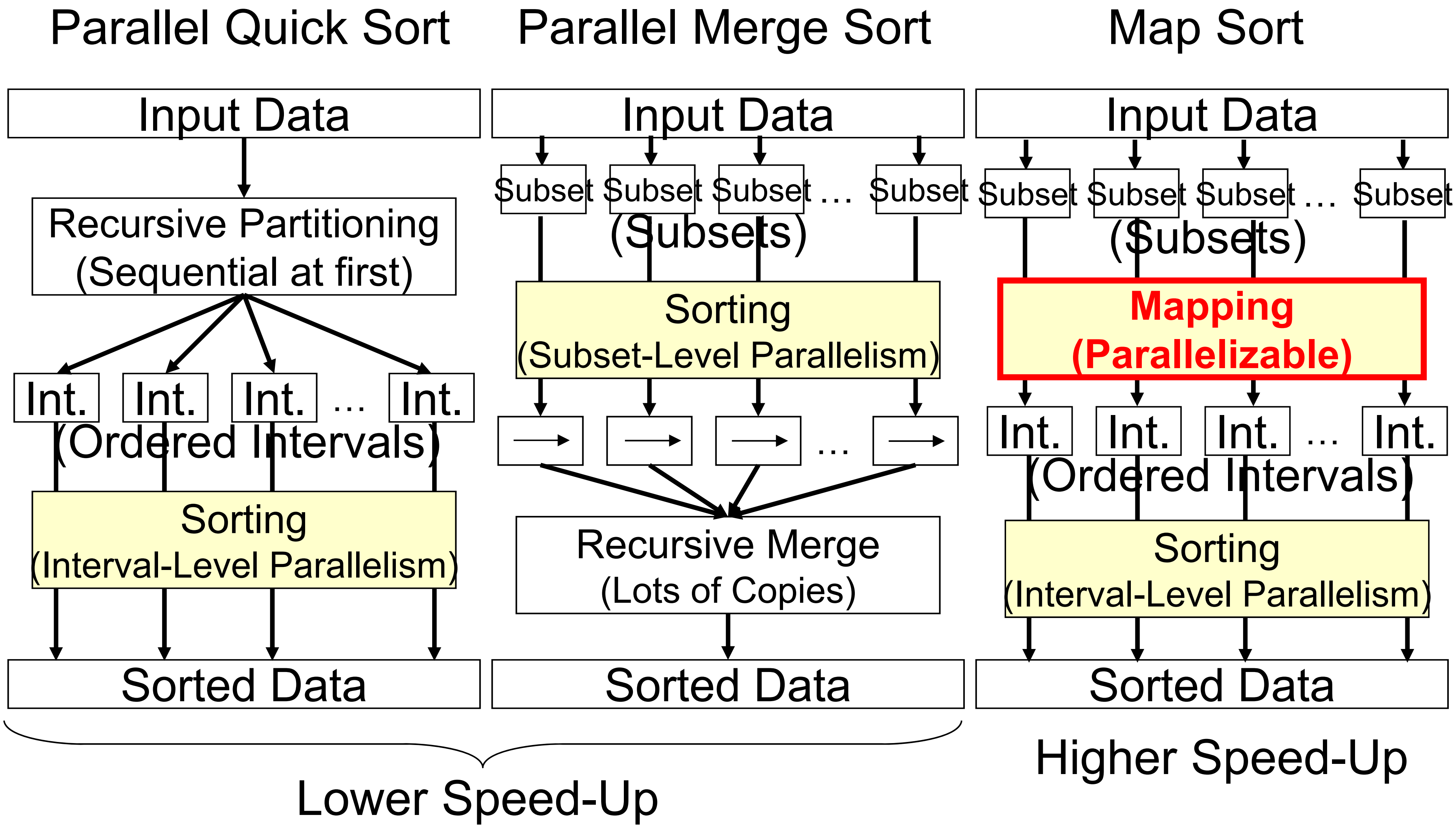
並列クイックソートとの比較



単一プロセッサ上のマージソート



3種の並列ソートアルゴリズム



Exercise 2

- 並列プログラミング　または　レポート課題
  - 並列プログラミング
    - EX2forDownload.zip (次頁)
  - レポート課題
    - 枝廣にメールで聞くこと

Exercise 2 (EX2IforDownload.zip)

ソフトウェアを並列化し、スケーラビリティを確認する

1. EX2IforDownload の中のreadme.txtを読み、main.cを理解し、自分の環境でビルドして実行する。プログラムは枝廣の環境でのみ確認している。（かつ、今年初登場。バグがあるかもしれない。）
2. プログラムを並列化し、実行する
3. 画像のサイズを変える、かつ、スレッド数を変えることによりスケーラビリティを評価する。
   (スケーラビリティとは、プロセッサ数（スレッド数）を増やした時の性能向上のことをいう。一般に、画像サイズが大きい方が高いスケーラビリティを実現しやすい。）
   そして、作成したソフトウェアの中での、アムダールの法則の意味での並列化可能部分の割合を計算する。
4. プログラム、および、スケーラビリティと並列化可能な部分の割合を記載したレポートをeda@ertl.jp に提出する。(〆切は７月１２日)