

# マップレデュース (MapReduce)

## マップレデュース (MapReduce)

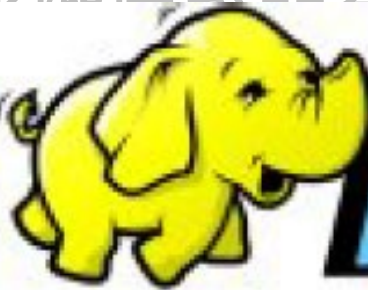
- 分散ファイルシステム (DFS) の上に様々な新しいソフトウェア階層が開発されてきた (急速に進化し, 現在も拡張を続けている. . . )

# マップレデュース (MapReduce)

- 分散ファイルシステム(DFS)の上に様々な新しいソフトウェア階層が開発されてきた(急速に進化し, 現在も拡張を続けている...)
- マップレデュース(MapReduce) と呼ばれる計算形式
  - \* 大量データに対する一般的な計算の多くを効率よく実行できるようになった
  - \* 計算途中でのハードウェア障害にも耐性を持っている
  - \* 効率のよいアルゴリズムのデザインが重要
    - + スーパーコンピュータの並列アルゴリズムと異なる
    - + 通信コストの問題が重要

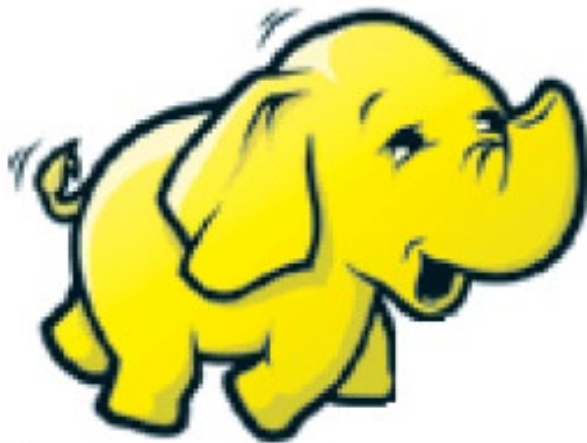
# マップレデュース (MapReduce)

- 分散ファイルシステム(DFS)の上に様々な新しいソフトウェアも拡張を



**hadoop**

形式  
は



いる

- + スーパーコンピュータの並列アルゴリズムと異なる
- + 通信コストの問題が重要

- Hadoop** (Apache Software Foundation) でオープンソース実装

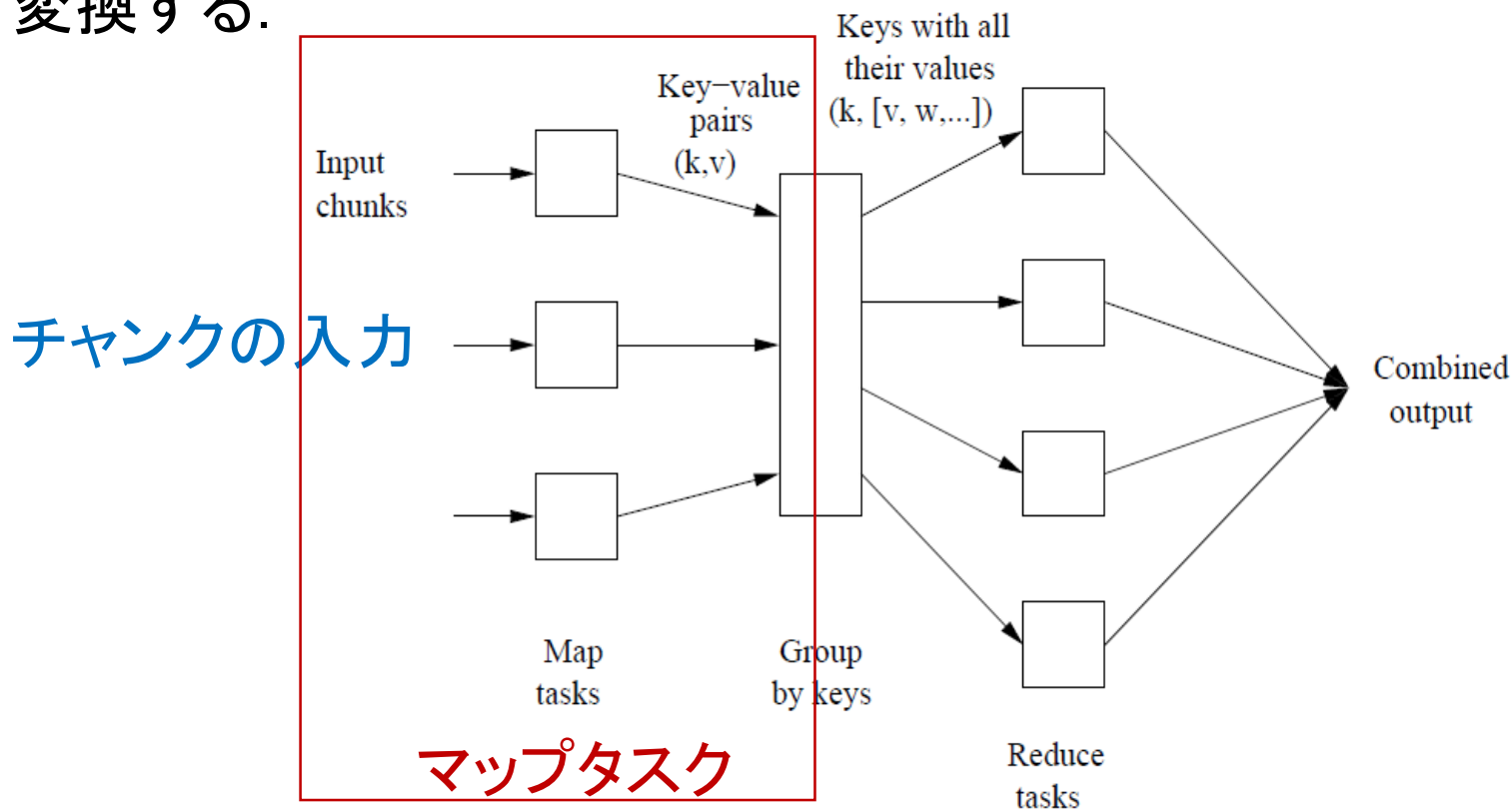
# マップレデュース (MapReduce)

- マップレデュースの実装を使うことで、ハードウェアの障害に強い大規模計算を管理することができる
- ユーザーが記述しなければならない関数は、マップ (Map) とレデュース (Reduce) の2つだけである
- 後はシステムが、並列計算やマップあるいはレデュースを実行するタスクの管理を行う

# マップレデュース (MapReduce) の手順

## 1. マップタスク (Map task)

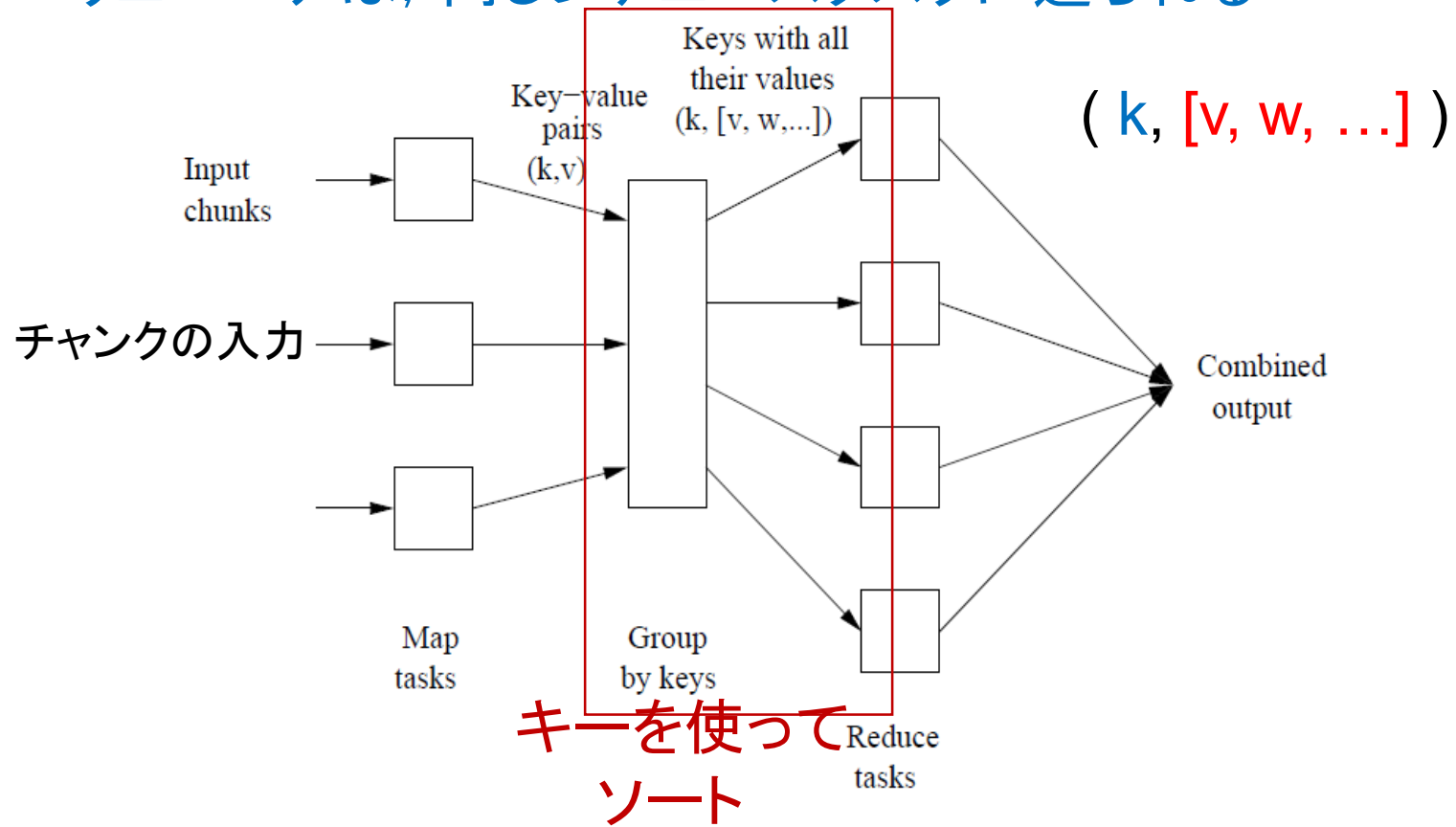
- 各マップタスクに、分散ファイルシステム上の1個以上の**チャンク**が**入力**として与えられる。
- マップタスクは、チャンクを**キーバリュー**(key-value) ペアの集合に変換する。



# マップレデュース (MapReduce) の手順

## 2. Group by Keys

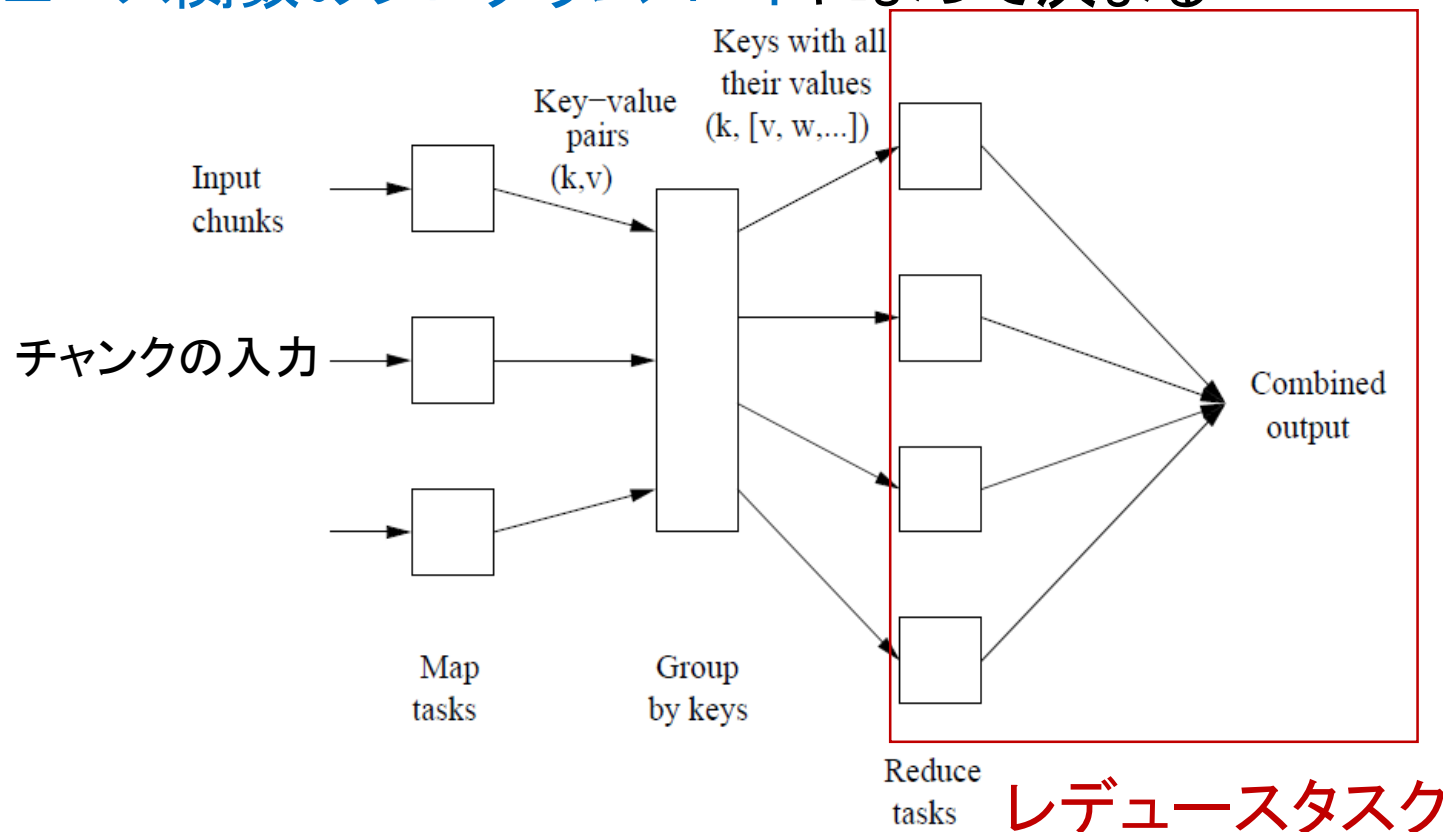
- マップタスクによって生成されたキーバリューペアが、**マスターコントローラー**によって集められ、**キーを使ってソート**される
- キーは、**レデュースタスク全体**にわたり分配され、**同じキーを持つキーバリューペアは、同じレデュースタスクに送られる**



# マップレデュース (MapReduce) の手順

## 3. レデュースタスク (Reduce Task)

- レデュースタスクは、1回につき1個のキーを処理し、**キーに関連付けられたすべてのバリューを何らかのやり方で結合する**
- 値がどのように結合されるかは、ユーザーによって書かれた**レデュース関数のプログラムコード**によって決まる





# MapReduce 計算の例

文書集合に含まれる各語の出現回数を数え上げるタスク

Provided by the  
programmer

## 1. マップタスク

### MAP:

Reads input and  
produces a set  
of key-value  
pairs

The crew of the space  
shuttle Endeavor recently  
returned to Earth as  
ambassadors, harbingers of  
a new era of space  
exploration. Scientists at  
NASA are saying that the  
recent assembly of the  
Dextre bot is the first step in  
a long-term space-based  
man/machine partnership.  
"The work we're doing now  
-- the robotics we're doing -  
is what we're going to  
need .....

(The, 1)  
(crew, 1)  
(of, 1)  
(the, 1)  
(space, 1)  
(shuttle, 1)  
(Endeavor, 1)  
(recently, 1)  
....

Big documents (key, value)

# MapReduce 計算の例

## 2. キーによるグルーピング

Provided by the  
programmer

### MAP:

Reads input and  
produces a set of  
key-value pairs

### Group by key:

Collect all pairs  
with same key

The crew of the space  
shuttle Endeavor recently  
returned to Earth as  
ambassadors, harbingers of  
a new era of space  
exploration. Scientists at  
NASA are saying that the  
recent assembly of the  
Dextre bot is the first step in  
a long-term space-based  
man/machine partnership.  
"The work we're doing now  
-- the robotics we're doing -  
is what we're going to  
need .....

Big documents

(The, 1)  
(crew, 1)  
(of, 1)  
(the, 1)  
(space, 1)  
(shuttle, 1)  
(Endeavor, 1)  
(recently, 1)  
....

(key, value)

(crew, 1)  
(crew, 1)  
(space, 1)  
(the, 1)  
(the, 1)  
(the, 1)  
(shuttle, 1)  
(recently, 1)  
...

(key, value)

key 集約(aggregation)

( the, [1,1,1] )

キーとそれに関連付け  
られたバリューのリスト

すべてのマップタスク  
から同一のキーに対  
するキーバリューペア

# MapReduce 計算の例

## 3. レデュースタスク

Provided by the  
programmer

### MAP:

Read input and  
produces a set of  
key-value pairs

(The, 1)  
(crew, 1)  
(of, 1)  
(the, 1)  
(space, 1)  
(shuttle, 1)  
(Endeavor, 1)  
(recently, 1)  
....

(key, value)

### Group by key:

Collect all pairs  
with same key

(crew, 1)  
(crew, 1)  
(space, 1)  
(the, 1)  
(the, 1)  
(the, 1)  
(shuttle, 1)  
(recently, 1)  
...

(key, value)

Provided by the  
programmer

### Reduce:

Collect all values  
belonging to the  
key and output

( the, [1,1,1] )  
(crew, 2)  
(space, 1)  
(the, 3)  
(shuttle, 1)  
(recently, 1)  
...

(key, value)

Only sequential reads

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need .....

Big documents

単語のキー  $w$ , 単語の出現回数  $m$

# MapReduce 計算の概要

## MAP:

Reads input and produces a set of key-value pairs

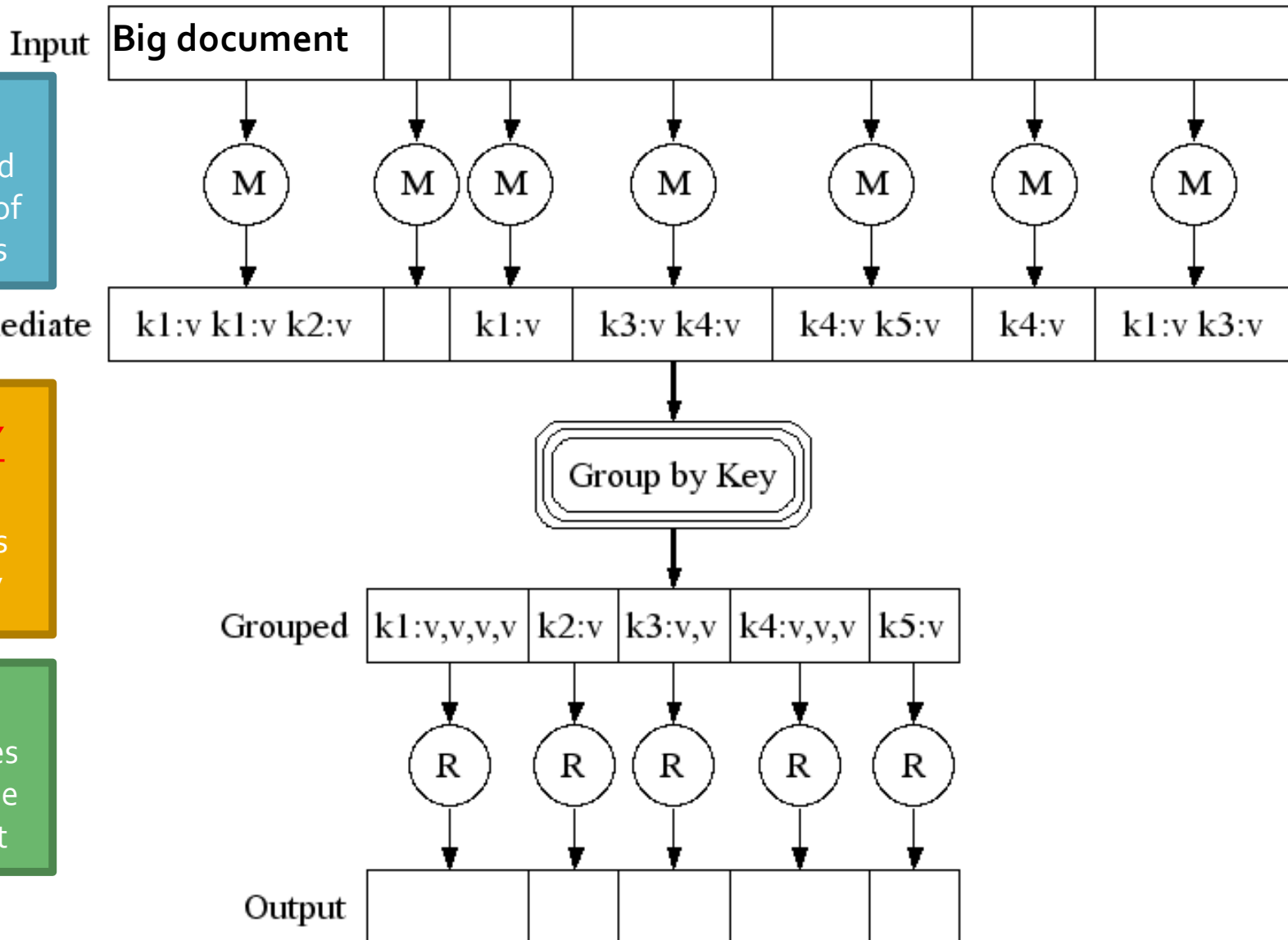
Intermediate

## GROUP BY KEY:

Collect all pairs with same key

## REDUCE:

Collect all values belonging to the key and output



# コンバイナー (Combiners)

- レデュース関数が**結合性 (associative)**と**可換性 (commutative)**を満たす場合、**レデュースタスクの一部をマップタスクの中で行うことが可能**
  - \* **結合性 (associative)**: 演算を施す順番は、被演算子の並びの順を変えない限り、結果に影響を与えない  $(a+b)+c = a+(b+c)$
  - \* **可換性 (commutative)**: 二つの被演算子の現れる位置を入れ替えても結果が変わらない  $a+b = b+a$

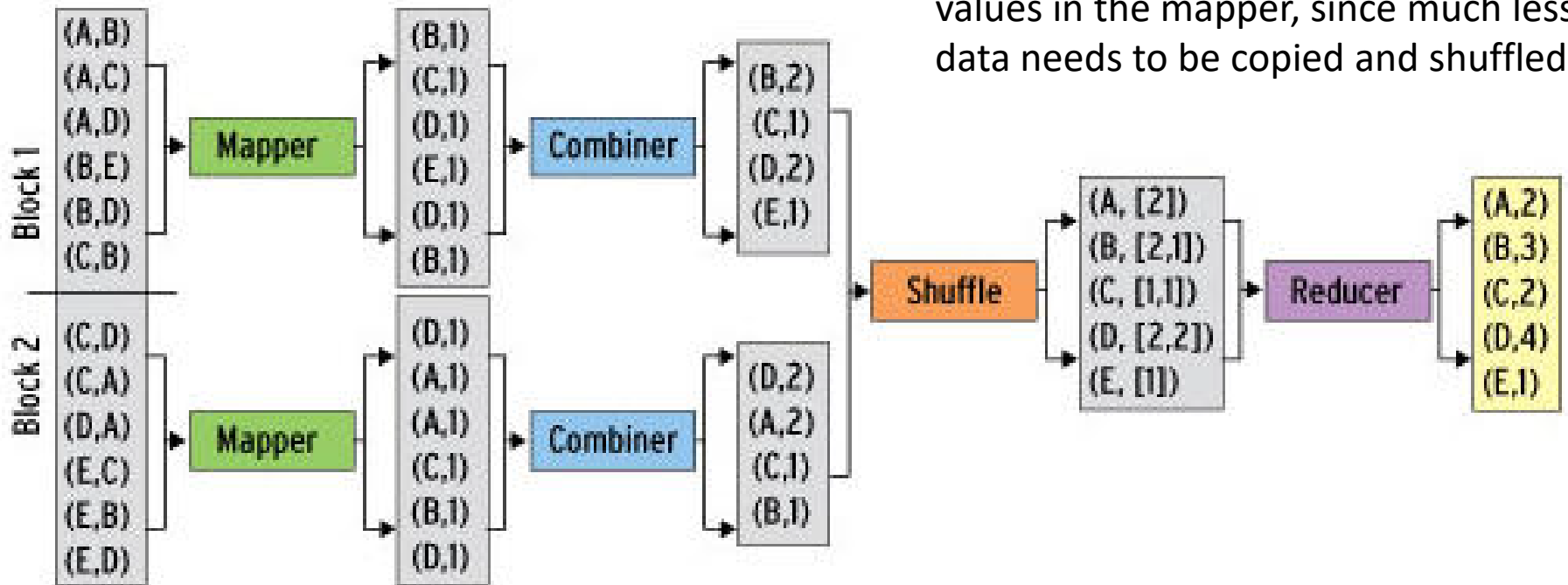
単語出現回数の例で行われるバリューの加算は結合性と可換性を持つ演算の例である。数のリスト  $[V_1, V_2, \dots, V_n]$  をどのように構成するかに関係なく、その合計は同じである。

# コンバイナー(Combiners)

- レデュース関数が**結合性 (associative)**と**可換性(commutative)**を満たす場合、**レデュースタスクの一部をマップタスクの中で行う**ことが可能

$$\text{combine}(k, \text{list}(v_1)) \rightarrow (k, v_2)$$

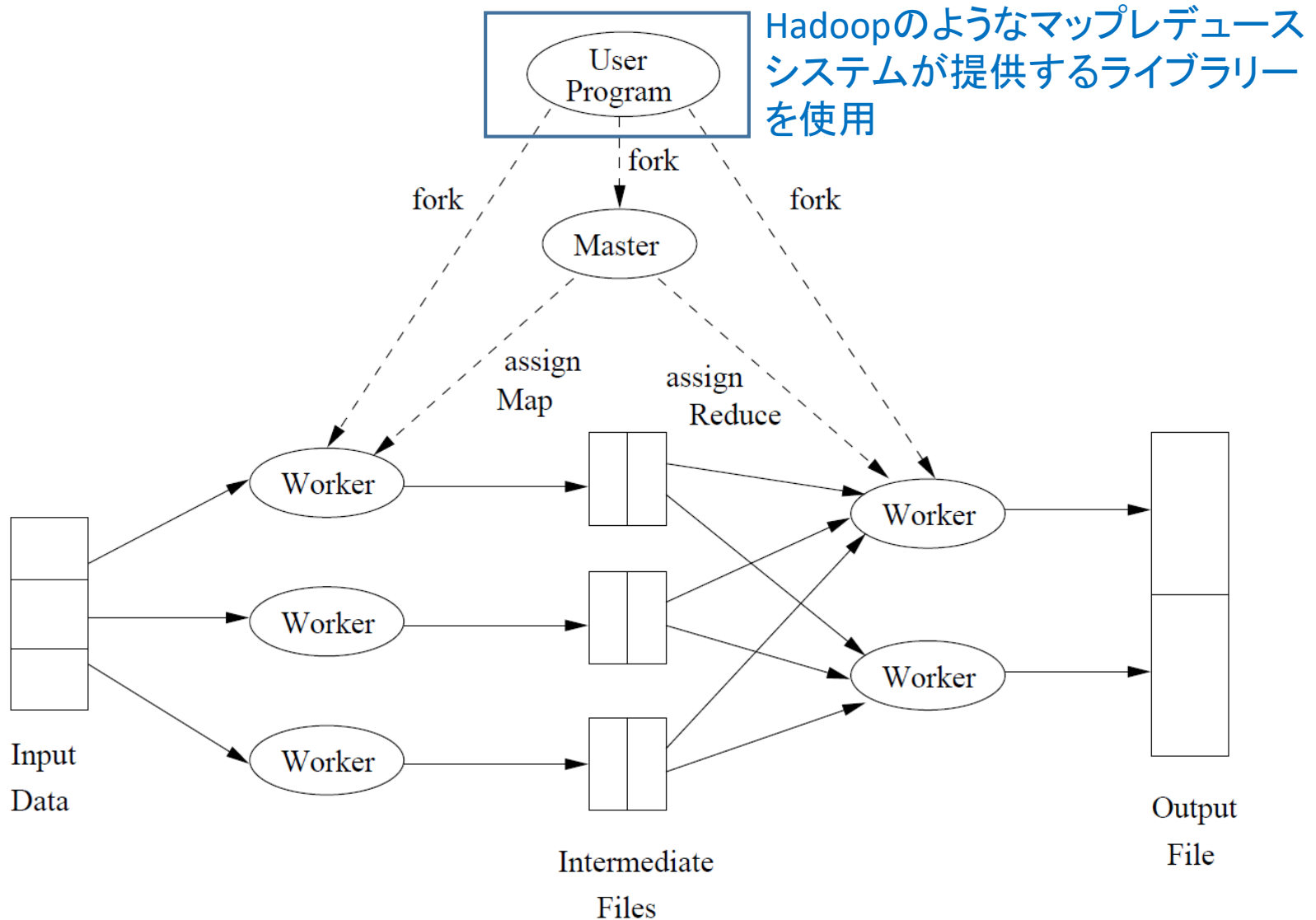
save network time by pre-aggregating values in the mapper, since much less data needs to be copied and shuffled



注意:この場合でも、グルーピングと集約を行い、結果をレデュースタスクに渡す処理は依然として必要である

# マップレデュース実行の詳細

- プロセス, タスク, ファイルの相互関係

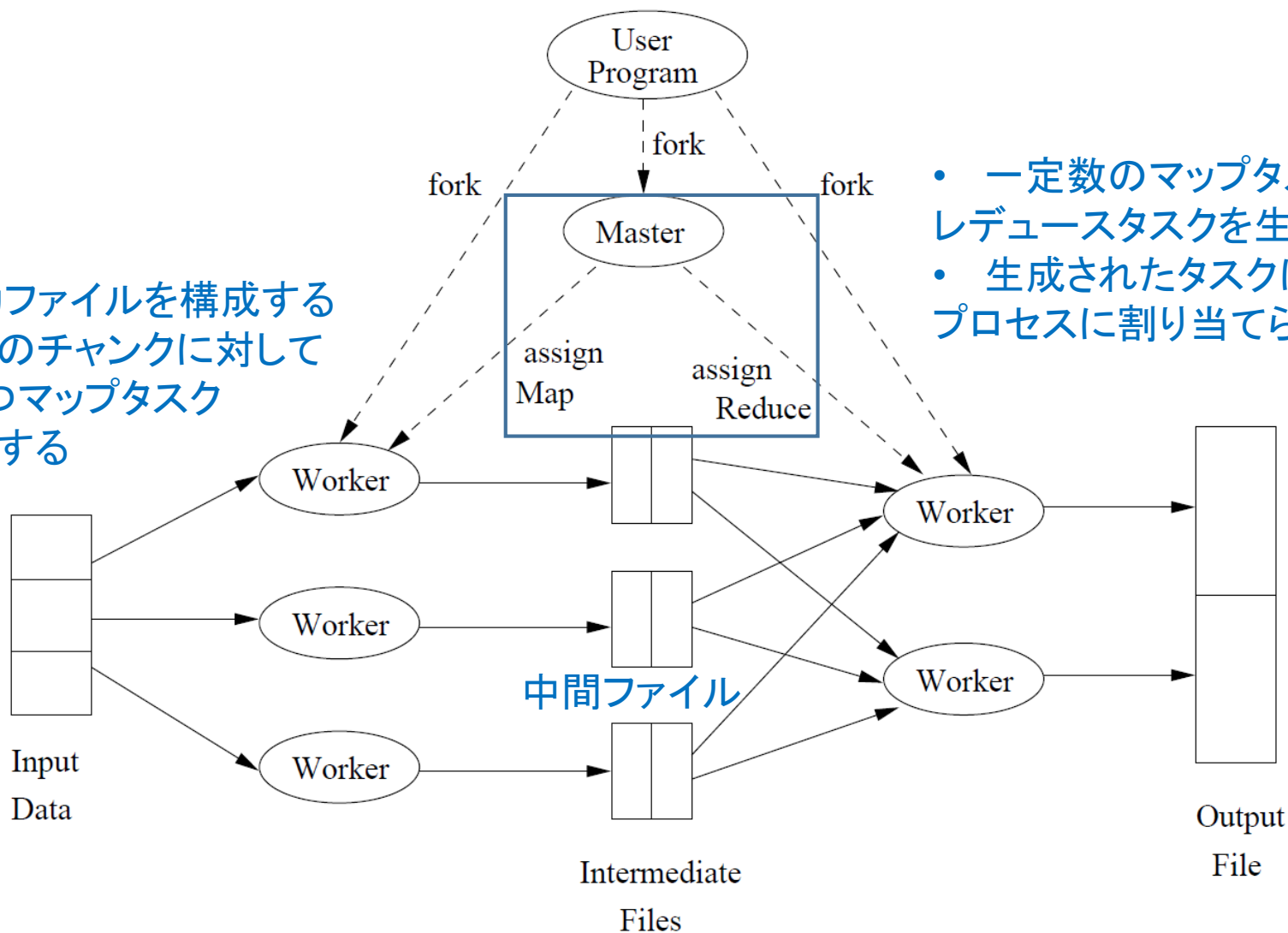


# マップレデュース実行の詳細

## • マスターコントローラー (Master)

- 入力ファイルを構成するすべてのチャンクに対して1つずつマップタスクを生成する

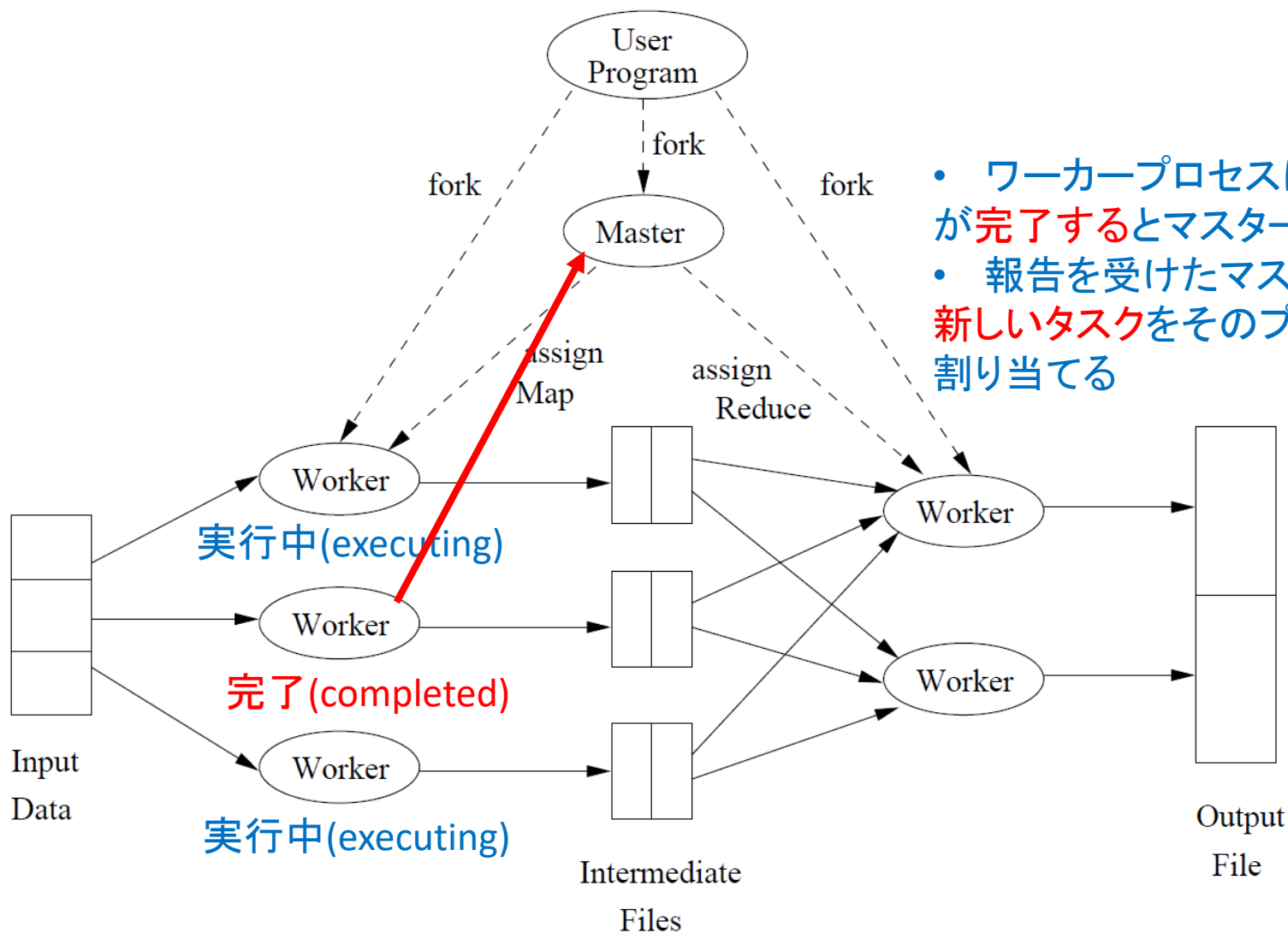
- 一定数のマップタスクとレデュースタスクを生成する
- 生成されたタスクはワーカープロセスに割り当てられる





# マップレデュース実行の詳細

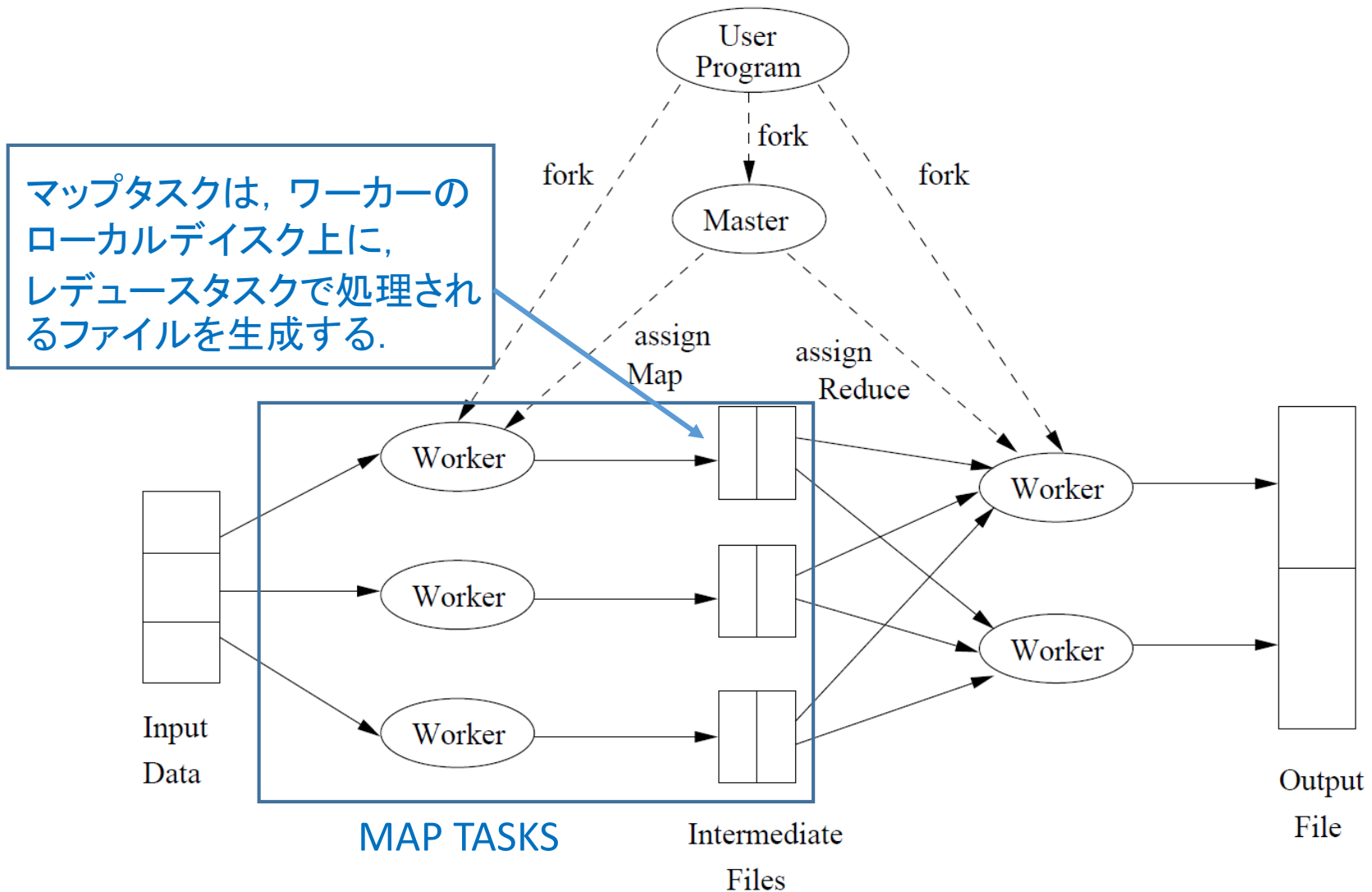
- マスターコントローラー (Master)



- ワーカープロセスは、タスクが完了するとマスターに報告
- 報告を受けたマスターは、新しいタスクをそのプロセスに割り当てる

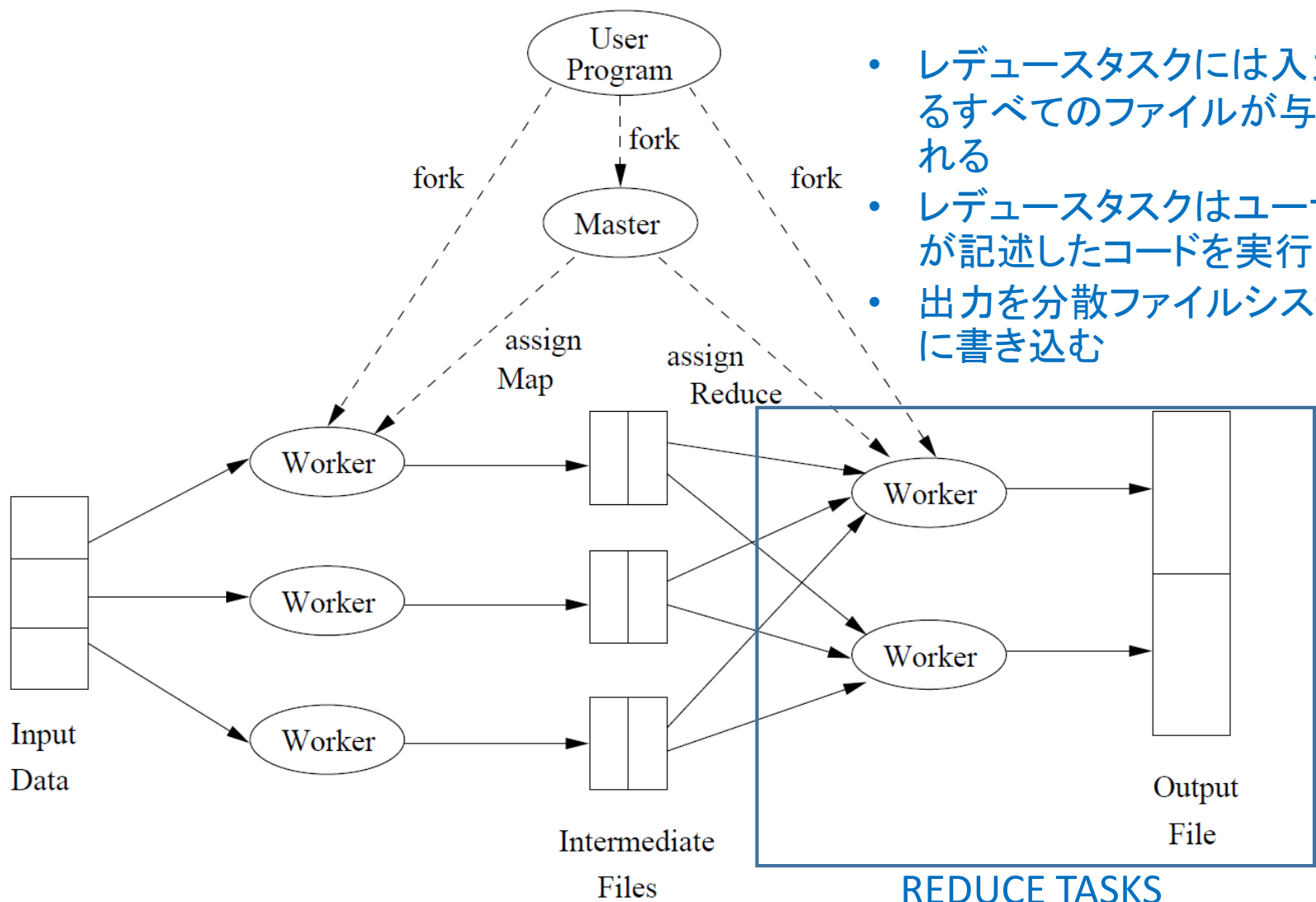
# マップレデュース実行の詳細

## • マップタスク



# マップレデュース実行の詳細

## • レデュースタスク



- レデュースタスクには入力となるすべてのファイルが与えられる
- レデュースタスクはユーザーが記述したコードを実行
- 出力を分散ファイルシステムに書き込む

# ノード障害の扱い

- 最悪の事態: ?

## ノード障害の扱い

- 最悪の事態: マスターが実行されている計算ノードが障害を起こす → マップレデュースジョブ全体を再実行

他の障害はマスターによって管理され、  
最終的にはマップレデュースタスクは  
完了する

# ノード障害の扱い

マップワーカーが動いている計算ノードが障害を起こした場合：

1. 障害はマスターによって検出される
2. このワーカーに割り当てられたすべてのマップタスクが(すでに完了したものも含めて)再実行される
3. マスターはマップタスクの現在の状態を“アイドル(idle)”にセット
4. 別のワーカーが利用可能になったら、タスクの再スケジュールを行う
5. マスターはレデュースタスクに対してマップタスクからの入力の変更に知らせる必要がある

# ノード障害の扱い

レデュースワーカーが動いている計算ノードが障害を起こした場合：

1. マスターはこのワーカーに現在“**実行中(executing)**”のレデュースタスクの状態を“**アイドル(idle)**”にセットする
2. それらのタスクは別のレデュースワーカー上で再スケジュールされる

# マップレデュースを用いたアルゴリズム (Algorithms Using MapReduce)

## 1. マップレデュースによる行列ベクトル積



# マップレデュースによる行列ベクトル積

- **サーチエンジン**において行われる**ウェブページのランキング**の処理で必要(行列・ベクトルのサイズは**数百億**を超える)
- **疎な行列**を想定(平均して各行あたり10から15の0でない要素を持つ;**ページ間のリンク**を表現している)

The diagram illustrates the matrix-vector multiplication  $M \cdot v = x$ . On the left, a purple square matrix  $M$  is shown with a small white square at the intersection of row  $i$  and column  $j$ , labeled  $m_{ij}$ . Below the matrix is the label  $M$  and the text  $n \times n$  行列 (nは何百億). In the center, a green vertical vector  $v$  is shown with a horizontal line at the  $j$ -th element, labeled  $v_j$ . To the right of  $v$  is an equals sign, followed by another green vertical vector  $x$  with a horizontal line at the  $i$ -th element, labeled  $x_i$ . To the right of  $x$  is the equation  $x_i = \sum_{j=1}^n m_{ij} v_j$ .

$$M \cdot v = x$$
$$x_i = \sum_{j=1}^n m_{ij} v_j$$

$n \times n$  行列  
(nは何百億)

# マップレデュースによる行列ベクトル積

- ベクトル $v$ が主記憶に収まる場合
- ベクトル $v$ が各マップタスクへの入力の一部である
- 行列 $M$  とベクトル $v$ とはDFS上のファイルに格納される

行列要素 3つ組 $(i, j, m_{ij})$ を使う

$$\begin{pmatrix} 0 & 1.5 \\ 0.77 & 0 \end{pmatrix}$$

$(1, 2, 1.5)$
$(2, 1, 0.77)$
...

ベクトルも

$$\begin{pmatrix} 1.3 \\ 2.5 \\ 3.7 \\ \dots \end{pmatrix}$$

$(1, 1.3)$
$(2, 2.5)$
$(3, 3.7)$
...

1	1.3
2	2.5
3	3.7
...	...
	...

ファイル上での  
位置を使う場合

# マップレデュースによる行列ベクトル積

## マップ関数

1. マップタスクはそれぞれベクトル $\mathbf{v}$ 全体と行列 $\mathbf{M}$ からのチャンクを入力にとる
2. 各入力行列要素から、以下の通りキーバリューペアが生成される

$$\mathbf{M} = \begin{pmatrix} 0 & 10 & 0 \\ 0 & 20 & 50 \\ 0 & 0 & 30 \end{pmatrix}$$

$$\mathbf{v} = \begin{pmatrix} 1.3 \\ 2.5 \\ 3.7 \end{pmatrix}$$

$i$	$j$	$m_{ij}$		$i$	$m_{ij}v_j$
1	2	10	→	1	$10 \times 2.5$
2	2	20		2	$20 \times 2.5$

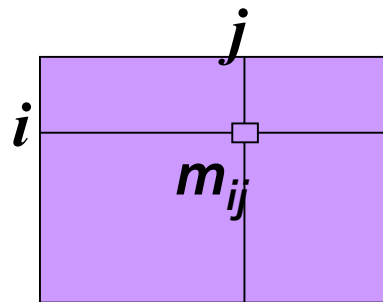
Map Task 1

入力

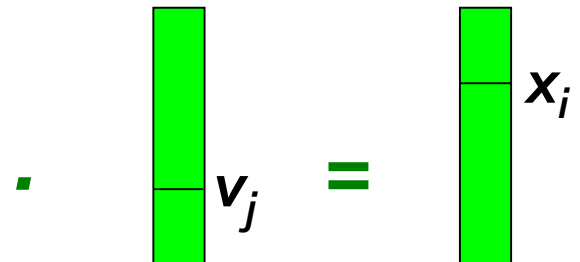
キーバリューペア

2	3	50	→	2	$50 \times 3.7$
3	3	30		3	$30 \times 3.7$

Map Task 2



$\mathbf{M}$



$\mathbf{M} \cdot \mathbf{v} = \mathbf{x}$

$$x_i = \sum_{j=1}^n m_{ij} v_j$$

# マップレデュースによる行列ベクトル積

- レデュース関数

レデュースタスクは単純にキー  $i$  に関連付けられた値を足し合わせる

$i$      $m_{ij}v_j$

Map  
Task 1

(1,  $10 \times 2.5$ )  
(2,  $20 \times 2.5$ )

キーバリューペア

Map  
Task 2

(2,  $50 \times 3.7$ )  
(3,  $30 \times 3.7$ )

キーごとにグルーピング

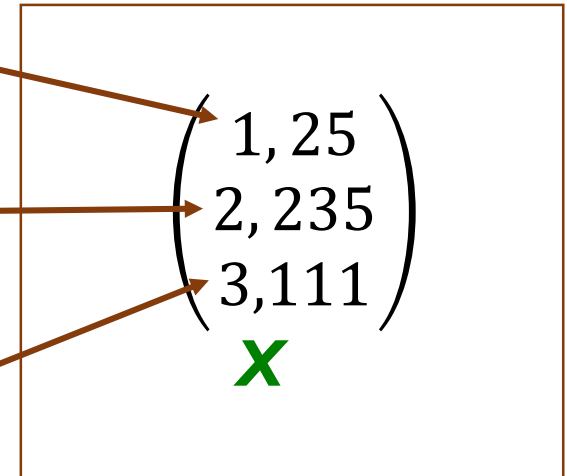
Reduce Tasks

(1,  $10 \times 2.5$ )

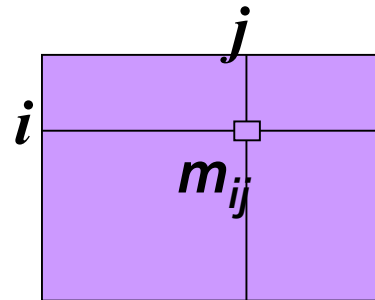
(2,  $20 \times 2.5$ )

(2,  $50 \times 3.7$ )

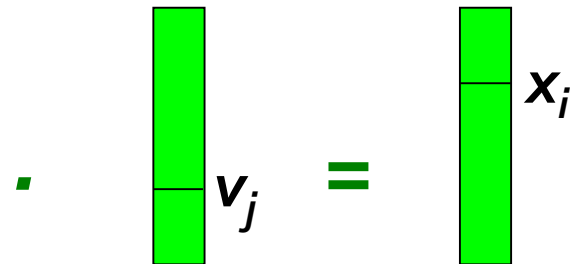
(3,  $30 \times 3.7$ )



$$x_i = \sum_{j=1}^n m_{ij} v_j$$



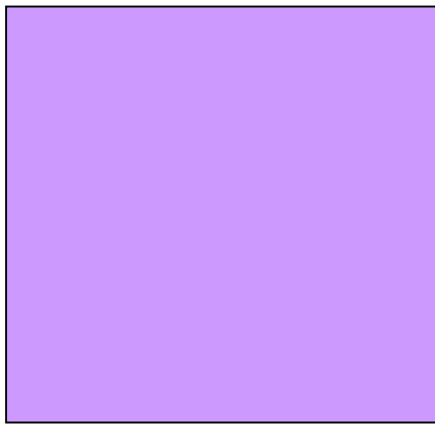
**M**



**v = x**

## ベクトル $v$ が主記憶に収まらない場合

- ベクトル $v$ を主記憶に乗せないと多数のディスクアクセスが発生．．．
- 代替案？



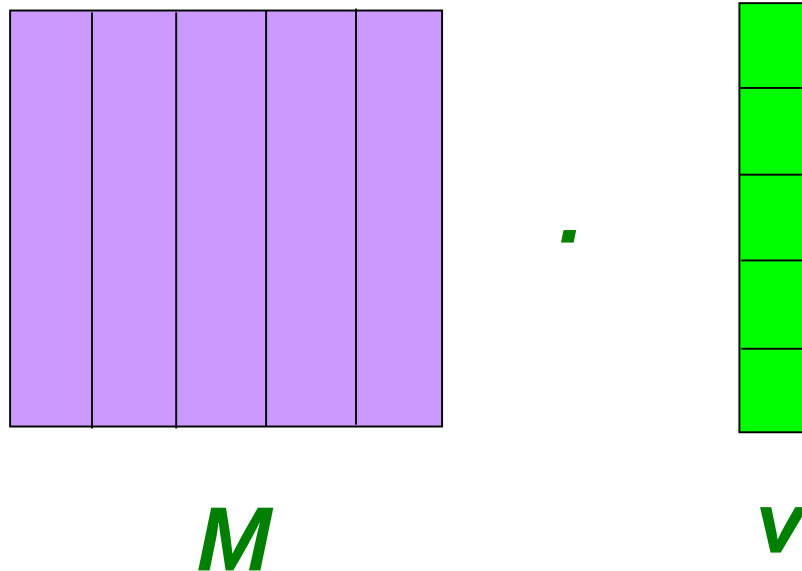
$M$



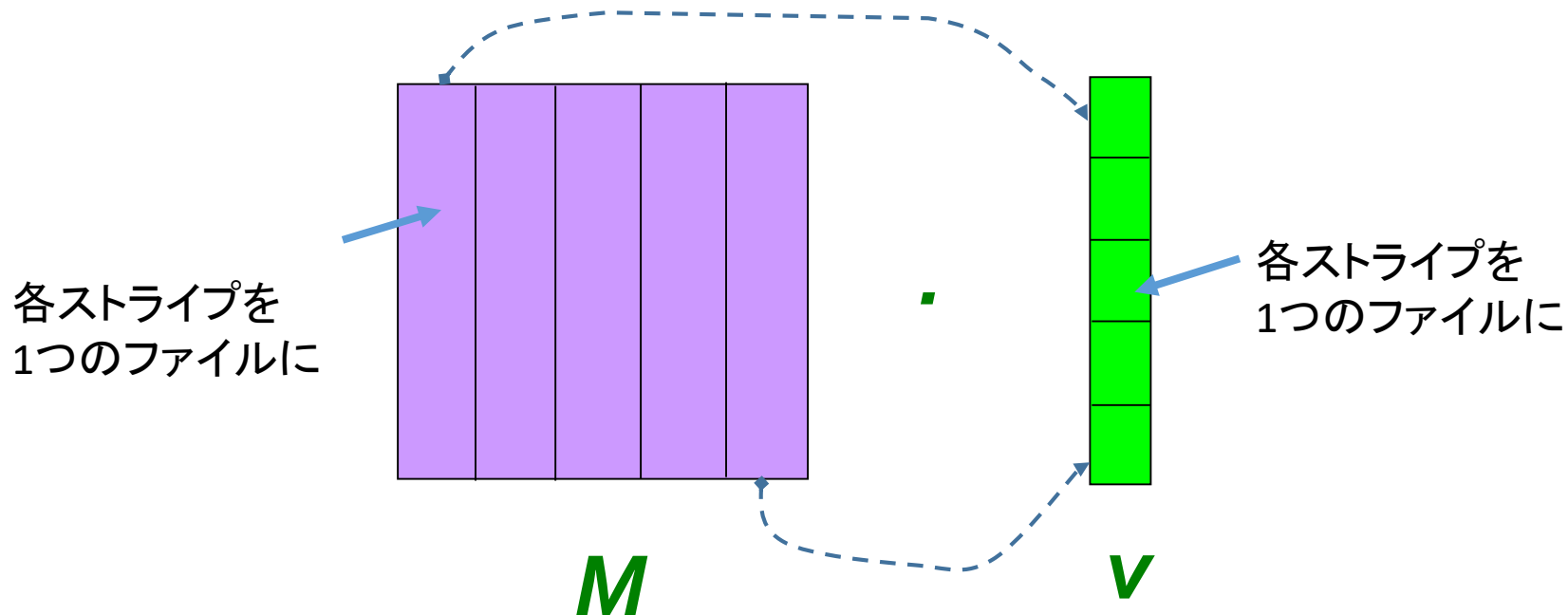
$v$

.

# ベクトル $v$ が主記憶に収まらない場合



# ベクトル $v$ が主記憶に収まらない場合



- マッピングタスクの入力

各マッピングタスクには行列のストライプの1つを含むチャンクが割り当てられ、対応するベクトルのストライプ全体が渡される

# マップレデュースを用いたアルゴリズム (Algorithms Using MapReduce)

## 2. 関係代数の演算 (Relational-Algebra Operations)



# 関係代数の演算 (Relational-Algebra Operations)

- 用語の説明

関係 (Relation) :

- \* 属性 (Attribute) と呼ばれる列のヘッダーを持った表 (テーブル)
- \* 関係における行要素はタプル (tuple) と呼ばれる

商品テーブル (関係)

$R$

商品番号	商品名	単価
1001	Chair	2000
1002	Table	20000
1003	Sofa	30000
1004	Carpet	11000

# 関係代数の演算 (Relational-Algebra Operations)

- 用語の説明

## 関係 (Relation) :

- \* 属性 (Attribute) と呼ばれる列のヘッダーを持った表 (テーブル)
- \* 関係における行要素はタプル (tuple) と呼ばれる
- \*  $R(A_1, A_2, \dots, A_n)$   $\Rightarrow$  関係名が  $R$  であり, その属性が  $A_1, A_2, \dots, A_n$
- \* 関係に含まれる属性の集合をスキーマ (schema) と呼ぶ

商品テーブル (関係)

$R$	商品番号	商品名	単価
	1001	Chair	2000
	1002	Table	20000
	1003	Sofa	30000
	1004	Carpet	11000

関係 (Relation)	$\longleftrightarrow$	表 (table)
属性 (Attribute)	$\longleftrightarrow$	列 (column)
タプル (Tuple)	$\longleftrightarrow$	行 (row)

# 関係代数の演算 (Relational-Algebra Operations)

## • 用語の説明

### 関係 (Relation) :

- \* 属性 (Attribute) と呼ばれる列のヘッダーを持った表 (テーブル)
- \* 関係における行要素はタプル (tuple) と呼ばれる
- \*  $R(A_1, A_2, \dots, A_n) \Rightarrow$  関係名が  $R$  であり, その属性が  $A_1, A_2, \dots, A_n$
- \* 関係に含まれる属性の集合をスキーマ (schema) と呼ぶ

商品テーブル (関係)

$R$

商品番号	商品名	単価
1001	Chair	2000
1002	Table	20000
1003	Sofa	30000
1004	Carpet	11000

Relation *Links*

From	To
url1	url2
url1	url3
url2	url3
url2	url4
...	...

関係 (Relation)  $\longleftrightarrow$  表 (table)  
属性 (Attribute)  $\longleftrightarrow$  列 (column)  
タプル (Tuple)  $\longleftrightarrow$  行 (row)

***Links (From, To)***

最初のURLが2つ目のURLへのリンクを持つ (URLの対の集合)

# 関係代数の演算 (Relational-Algebra Operations)

- 選択 (Selection)

関係中の各タプルに**条件C**を適用し、**Cを満たすタプルだけ**を出力として生成する (**条件を満たす行**を抜き出す)

商品テーブル

$R$

商品番号	商品名	単価
1001	Chair	2000
1002	Table	20000
1003	Sofa	30000
1004	Carpet	11000

条件C  
単価が1万2000円未満



選択 (Selection)

$\sigma_C(R)$

商品番号	商品名	単価
1001	Chair	2000
1004	Carpet	11000

# 関係代数の演算 (Relational-Algebra Operations)

- 射影 (Projection)

関係中の属性の部分集合  $S$  に含まれる属性と一致するタプルの構成要素のみを生成する (特定の列を抜き出す)

$R$

顧客番号	名前	住所	メールアドレス
101	K. Yamada	Hiroshima 123	yamada@
102	P. Tanaka	Tokyo 456	tanaka@
103	F. Suzuki	Osaka 789	Suzuki@



射影 (Projection)

部分集合  $S$   
顧客番号, 住所

$\pi_S(R)$

顧客番号	住所
101	Hiroshima 123
102	Tokyo 456
103	Osaka 789

# 関係代数の演算 (Relational-Algebra Operations)

- 結合 (Join)  
条件に基づいて2つのテーブルを結びつける

$R$	顧客	
	顧客番号	名前
	101	K. Yamada
	102	P. Tanaka
$S$	注文	
	顧客番号	注文番号
	102	1111
	101	2222
	102	3333

# 関係代数の演算 (Relational-Algebra Operations)

- 結合 (Join)

条件に基づいて2つのテーブルを結びつける  $R$  と  $S$  の直積

$R$	顧客	
	顧客番号	名前
	101	K. Yamada
	102	P. Tanaka
$S$	注文	
	顧客番号	注文番号
	102	1111
	101	2222
	102	3333

顧客番号	名前	顧客番号	注文番号
101	K. Yamada	102	1111
101	K. Yamada	101	2222
101	K. Yamada	102	3333
102	P. Tanaka	102	1111
102	P. Tanaka	101	2222
102	P. Tanaka	102	3333
103	F. Suzuki	102	1111
103	F. Suzuki	101	2222
103	F. Suzuki	102	3333

# 関係代数の演算 (Relational-Algebra Operations)

- 結合 (Join)

条件に基づいて2つのテーブルを結びつける  $R$  と  $S$  の直積

$R$	顧客	
	顧客番号	名前
	101	K. Yamada
	102	P. Tanaka
$S$	注文	
	顧客番号	注文番号
	102	1111
	101	2222
	102	3333

顧客番号	名前	顧客番号	注文番号
101	K. Yamada	102	1111
101	K. Yamada	101	2222
101	K. Yamada	102	3333
102	P. Tanaka	102	1111
102	P. Tanaka	101	2222
102	P. Tanaka	102	3333
103	F. Suzuki	102	1111
103	F. Suzuki	101	2222
103	F. Suzuki	102	3333



# 関係代数の演算 (Relational-Algebra Operations)

- 自然結合 (Natural Join)
- 2つの関係に対して、それぞれの関係に含まれるタプルの対を比較する
  - あるタプル対が2つのスキーマに共通するすべての属性において一致する場合、どちらかのスキーマだけにある属性と、2つのタプルに共通の属性を要素として持つタプルを生成する

$R$ 注文者	顧客番号	名前
	101	K. Yamada
	102	P. Tanaka
	103	F. Suzuki

$S$ 注文	顧客番号	注文番号
	102	1111
	101	2222
	102	3333

$R$  と  $S$  の自然結合

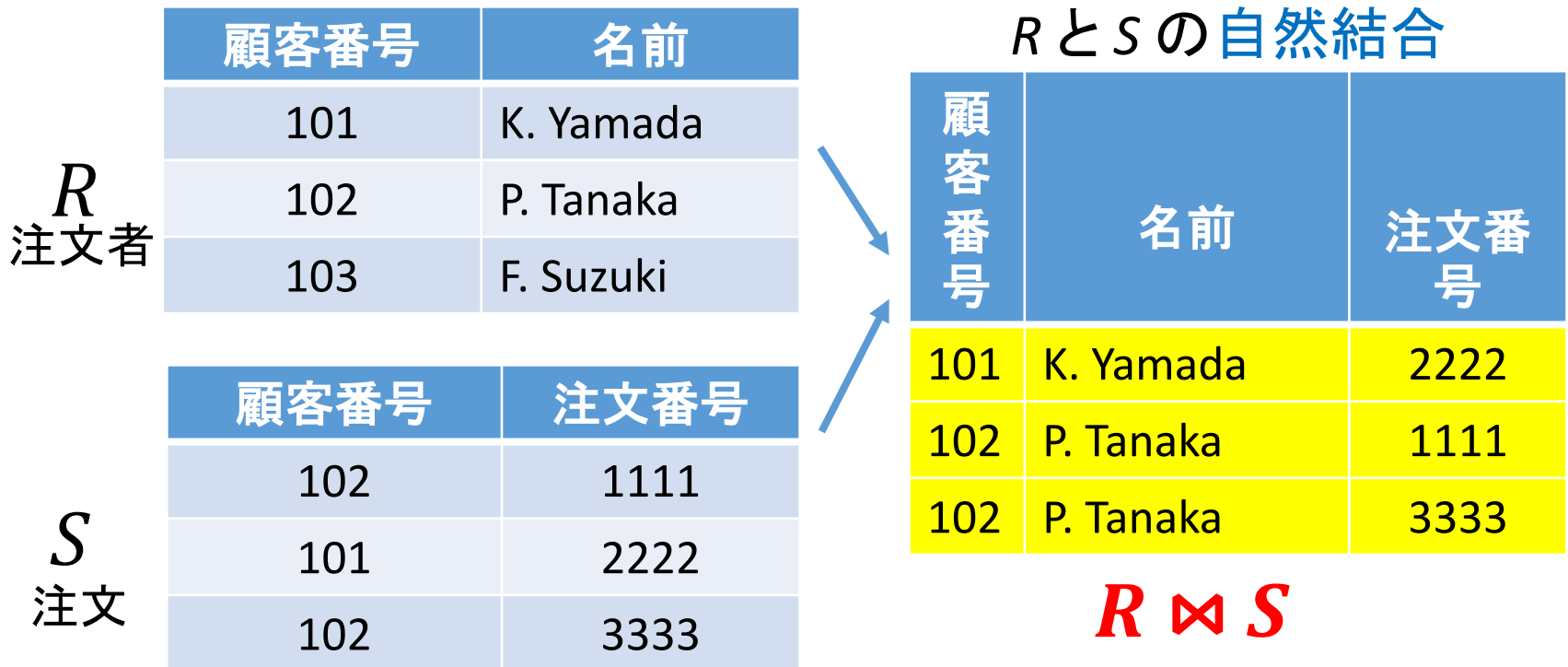
顧客番号	名前	注文番号
101	K. Yamada	2222
102	P. Tanaka	1111
102	P. Tanaka	3333

$R \bowtie S$

# 関係代数の演算 (Relational-Algebra Operations)

- 自然結合 (Natural Join)

3. もしタプル対が共通の属性において1つでも一致しない要素を持っている場合には, このタプル対からは何も生成しない(直積の結果でハイライトされなかった行)



## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation *Links*

From	To
url1	url2
url1	url3
url2	url3
url2	url4
...	...

ヒント: 自然結合を使う

*Links(From, To)*

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation *Links*

From	To
url1	url2
url1	url3
url2	url3
url2	url4
...	...

*Links(From, To)*

関係 *Links* とそれ自身の自然結合をとればよい!

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation  $L1$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation  $L2$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

$L1 \bowtie L2$

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation  $L1$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation  $L2$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

長さ2のすべてのパス

$L1 \bowtie L2$

$(url1, url2, url3)$

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

# 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation  $L1$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation  $L2$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

長さ2のすべてのパス

$L1 \bowtie L2$

$(url1, url2, url3)$

$(url1, url2, url4)$

...

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation  $L1$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation  $L2$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

$L1 \bowtie L2$

$(url1, url2, url3)$   
 $(url1, url2, url4)$

...

長さ2のすべてのパスではなく,  $u$  から  $w$  への少なくとも1個の長さ2のパスを持つURLの対  $(u, w)$  だけが欲しい場合?



## 自然結合の例

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
⇒  $u$  から  $v$ ,  $v$  から  $w$  へのリンクがあるような  
URLの3つ組  $(u, v, w)$  を見つけたい

Relation  $L1$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation  $L2$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

$L1 \bowtie L2$

$\begin{matrix} u & v & w \\ (url1, url2, url3) \\ (url1, url2, url4) \end{matrix}$

...

$\pi_{u,w}(L1 \bowtie L2)$

$(url1, url3)$

$(url1, url4)$

...

長さ2のすべてのパスではなく,  $u$  から  $w$  への  
少なくとも1個の長さ2のパスを持つURLの対  $(u, w)$   
だけが欲しい場合

# グルーピング(Grouping) と集約(Aggregation)

1. 関係 $R$ に対して, グルーピング属性(grouping attribute) と呼ばれる属性集合 $G$ を選び, その値に応じてタプルを分割する

Relation *Friends*

User	Friend
John	Peter
John	Mary
Steven	John
Steven	Hanako
John	Sally
...	...

*Friends(User, Friend)*

サイトの経営者:

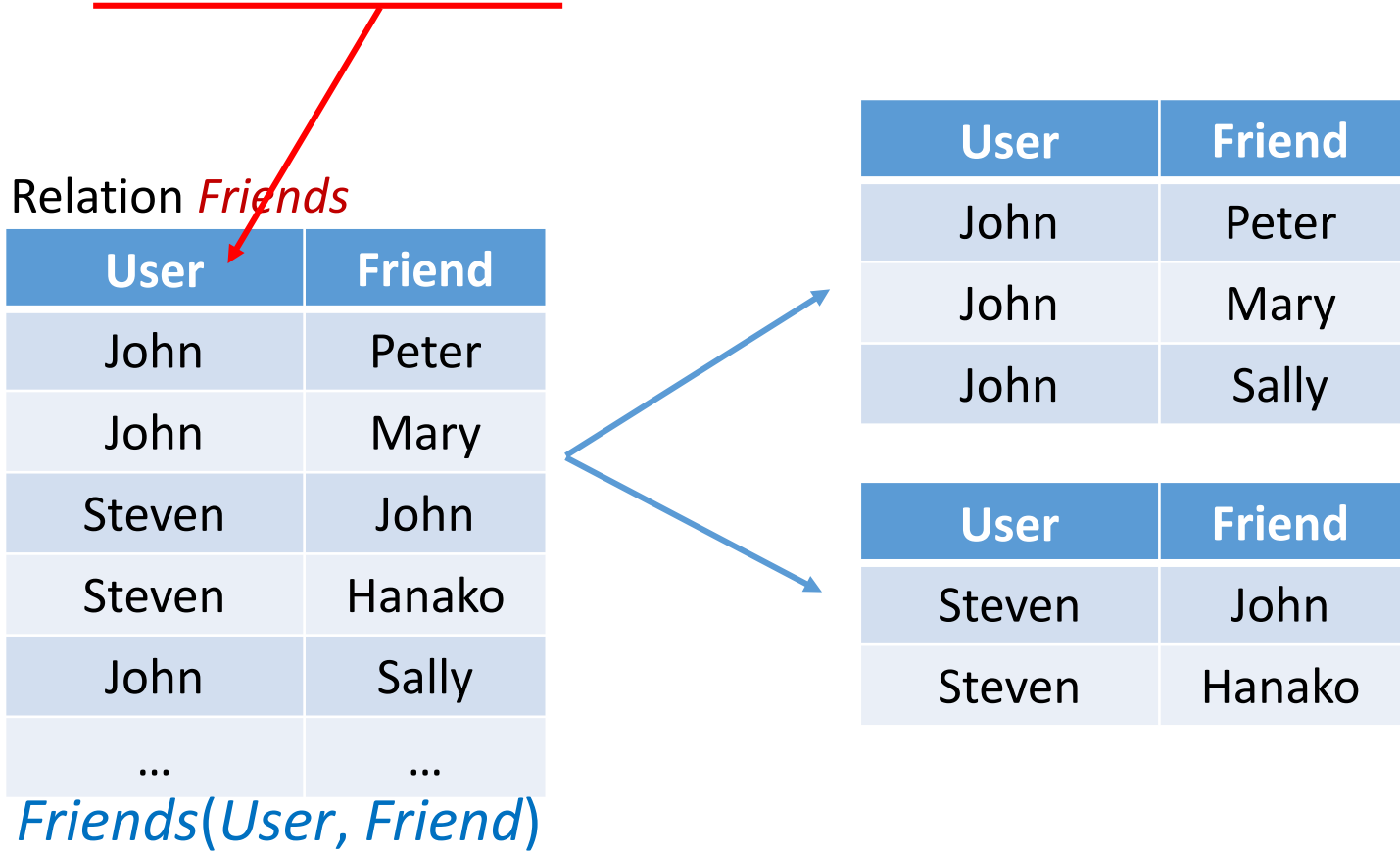
ユーザーが友達を何人持っているか  
に関する統計をとりたい

各ユーザーの友達の数进行計算する

# グルーピング(Grouping) と集約(Aggregation)

1. 関係 $R$ に対して, グルーピング属性(grouping attribute) と呼ばれる属性集合 $G$ を選び, その値に応じてタプルを分割する

Relation *Friends*



User	Friend
John	Peter
John	Mary
Steven	John
Steven	Hanako
John	Sally
...	...

*Friends(User, Friend)*

User	Friend
John	Peter
John	Mary
John	Sally

User	Friend
Steven	John
Steven	Hanako

# グルーピング(Grouping) と集約(Aggregation)

2. 分割されたそれぞれのグループのに対し, Gとは別の属性の値を**集約**する

集約の例: SUM, COUNT, AVG, MIN, MAX

Relation *Friends*

User	Friend
John	Peter
John	Mary
Steven	John
Steven	Hanako
John	Sally
...	...

*Friends(User, Friend)*

User	Friend
John	Peter
John	Mary
John	Sally

User	Friend
Steven	John
Steven	Hanako

User	Friends
John	3

集約: COUNT

User	Friends
Steven	2

(John, 3)  
(Steven, 2)  
...

# グルーピング (Grouping) と集約 (Aggregation)

2. 分割されたそれぞれのグループのに対し, Gとは別の属性の値を**集約**する

集約の例: SUM, COUNT, AVG, MIN, MAX

Relation *Friends*

User	Friend
John	Peter
John	Mary
Steven	John
Steven	Hanako
John	Sally
...	...

*Friends*(User, Friend)

User	Friend
John	Peter
John	Mary
John	Sally

User	Friend
Steven	John
Steven	Hanako

User	Friends
John	3

集約: COUNT

User	Friends
Steven	2

$$\gamma_{\mathbf{X}}(R) = \gamma_{\mathbf{G}, \theta(\mathbf{A})}(R)$$

$$\gamma_{\mathbf{User}, \mathbf{COUNT}(\mathbf{Friend})}(\mathit{Friends})$$

# マップレデュースを用いたアルゴリズム (Algorithms Using MapReduce)

## 3. マップレデュースによる関係代数演算の計算 (Computing Relational-Algebra Operations by MapReduce)

# マップレデュースによる選択の計算

## 選択 $\sigma_C(R)$ のマップレデュース実装

- マッピング関数

- $R$ 中の各タプル $t$ に対し,  $C$ を満たすかどうかを調べる
- もし満たすなら, キーバリューペア( $t, t$ )を出力

$R$	商品番号	商品名	単価	<u>条件C: 単価が1万2000円未満</u>	
	1001	Chair	2000	$\sigma_C(R)$	
	1002	Table	20000	<u>Key: <math>t</math></u>	<u>Value: <math>t</math></u>
	1003	Sofa	30000	( (1001, Chair, 2000), (1001, Chair, 2000) )	
	1004	Carpet	11000	( (1004, Carpet, 11000), (1004, Carpet, 11000) )	

- レデュース関数

何もしない(各キーバリューペアをそのまま出力する)

# マップレデュースによる射影の計算

## 射影 $\pi_S(R)$ のマップレデュース実装

- マッピング関数

$R$ 中の各タプル $t$ に対し,  $S$ に含まれない属性から構成要素を $t$ から取り除いた $t'$ を生成する.

出力は, キーバリューペア( $t'$ ,  $t'$ )である

$R$

注文番号	名前	メールアドレス	商品
101	Yamada	yamada@	Book A
102	Tanaka	tanaka@	Book B
103	Yamada	yamada@	Book C

部分集合 $S$ : 名前, メールアドレス

Key:  $t'$

Value:  $t'$

( (Yamada, yamada@), (Yamada, yamada@) )

( (Tanaka, tanaka@), (Tanaka, tanaka@) )

( (Yamada, yamada@), (Yamada, yamada@) )



# マップレデュースによる射影の計算

## 射影 $\pi_S(R)$ のマップレデュース実装

マップタスクの出力

Key:  $t'$

Value:  $t'$

( (Yamada, yamada@), (Yamada, yamada@) )

( (Tanaka, tanaka@), (Tanaka, tanaka@) )

( (Yamada, yamada@), (Yamada, yamada@) )

システムによって、キーごとにグルーピングされると

重複

( (Yamada, yamada@), [(Yamada, yamada@), (Yamada, yamada@)] )      ( $t'$ , [ $t'$ ,  $t'$ ])

( (Tanaka, tanaka@), (Tanaka, tanaka@) )      ( $t'$ ,  $t'$ )

# マップレデュースによる射影の計算

## 射影 $\pi_S(R)$ のマップレデュース実装

マップタスクの出力

Key:  $t'$

Value:  $t'$

( (Yamada, yamada@), (Yamada, yamada@) )

( (Tanaka, tanaka@), (Tanaka, tanaka@) )

( (Yamada, yamada@), (Yamada, yamada@) )

システムによって、キーごとにグルーピングされると

重複

( (Yamada, yamada@), [(Yamada, yamada@), (Yamada, yamada@)] )      ( $t'$ , [ $t'$ ,  $t'$ ])

( (Tanaka, tanaka@), (Tanaka, tanaka@) )      ( $t'$ ,  $t'$ )

- レデュース関数

重複を取り除く

( $t'$ , [ $t'$ ,  $t'$ , ...,  $t'$ ])  $\rightarrow$  ( $t'$ ,  $t'$ )

( (Yamada, yamada@), (Yamada, yamada@) )

# マップレデュースによる自然結合の計算

- 例: 関係 *Links* を用いて, ウェブ上の長さ2のパスを求めよう  
uからv, vからwへのリンクがあるようなURLの3つ組  $(u, v, w)$  を見つけたい

Relation *L1*

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L1(u, v)$

Relation *L2*

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

$L2(v, w)$

$L1 \bowtie L2$

最初のURLが2つ目のURLへの  
リンクを持つ(URLの対の集合)

# マップレデュースによる自然結合の計算

- マッピング関数

L1中の各タプル(u, v)に対し, キーバリューペア(v, (L1, u))を生成する

L2中の各タプル(v, w)に対し, キーバリューペア(v, (L2, w))を生成する

$L1(u, v)$

u	v
url1	url2
url1	url3
url2	url3
url2	url4
...	...

(v, (L1, u))

(url2, (L1, url1))  
(url3, (L1, url1))  
(url3, (L1, url2))  
(url4, (L1, url2))

$L2(v, w)$

v	w
url1	url2
url1	url3
url2	url3
url2	url4
...	...

(v, (L2, w))

(url1, (L2, url2))  
(url1, (L2, url3))  
(url2, (L2, url3))  
(url2, (L2, url4))

# マップレデュースによる自然結合の計算

## マップタスクの出力

$(v, (L1, u))$	$(v, (L2, w))$
$(url2, (L1, url1))$	$(url1, (L2, url2))$
$(url3, (L1, url1))$	$(url1, (L2, url3))$
$(url3, (L1, url2))$	$(url2, (L2, url3))$
$(url4, (L1, url2))$	$(url2, (L2, url4))$

システムによって、キーごとにグルーピングされると

$(url1, [(L2, url2), (L2, url3)])$
$(url2, [(L1, url1), (L2, url3), (L2, url4)])$
$(url3, [(L1, url1), (L1, url2)])$
$(url4, (L1, url2))$

# マップレデュースによる自然結合の計算

システムによって、キーごとにグルーピングされると

(url1, [ (L2, url2), (L2, url3) ])

(url2, [ (L1, url1), (L2, url3), (L2, url4) ])

(url3, [ (L1, url1), (L1, url2) ])

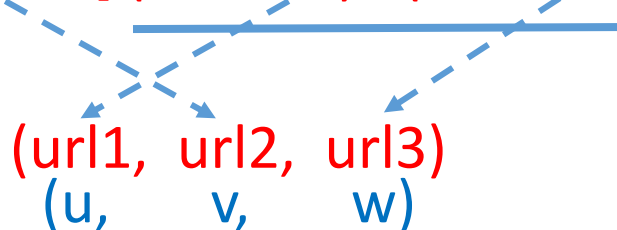
(url4, (L1, url2) )

- レデュース関数

各キーvに対して、最初の要素が(L1, u)から、2番目の要素が(L2, w)からなるすべての対の組み合わせから、出力タプル(u, v, w)を作成する

(url2, [ (L1, url1), (L2, url3), (L2, url4) ])

(url1, url2, url3)  
(u, v, w)



# マップレデュースによる自然結合の計算

システムによって、キーごとにグルーピングされると

(url1, [ (L2, url2), (L2, url3) ])

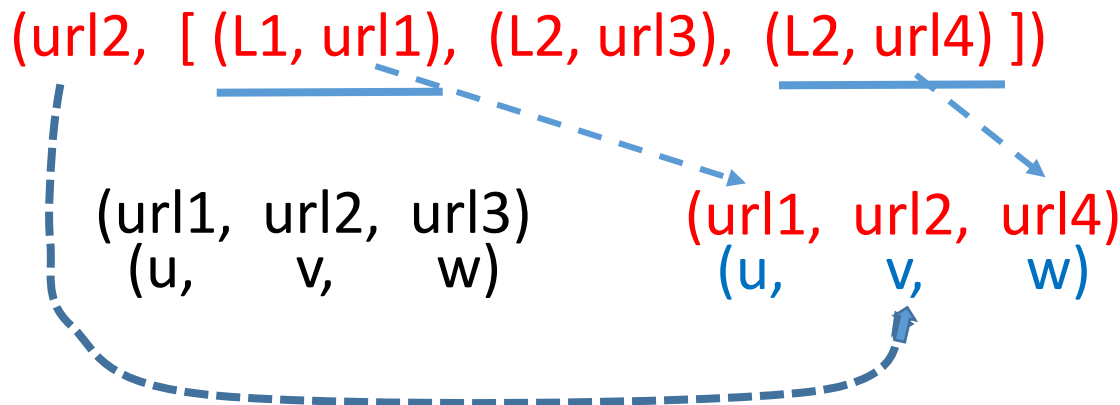
(url2, [ (L1, url1), (L2, url3), (L2, url4) ])

(url3, [ (L1, url1), (L1, url2) ])

(url4, (L1, url2) )

- レデュース関数

各キー $v$ に対して、最初の要素が $(L1, u)$ から、2番目の要素が $(L2, w)$ からなるすべての対の組み合わせから、出力タプル $(u, v, w)$ を作成する



# マップレデュースによるグルーピングと集約

- 関係 $R(A, B, C)$ に対して演算  $\gamma_{A, \theta(B)}(R)$  を適用する

A: グルーピング属性    B: 集約属性    集約  $\theta(B)$  : SUM

C: グルーピングも, 集約も行わない属性

R	顧客番号 A	単価 B	商品名 C
	1001	1000	Book A
	1001	2000	Book B
	1002	1500	Book C
	1002	1000	Book D
	1003	1000	Book E

マップ関数がグルーピングを行う  
レデュース関数が集約を行う



# マップレデュースによるグルーピングと集約

- マップ関数

各タプル(a, b, c)に対して, キーバリューペア(a, b)を生成する

R	顧客番号 A	単価 B	商品名 C	Key	Value
	1001	1000	Book A	(1001, 1000)	
	1001	2000	Book B	(1001, 2000)	
	1002	1500	Book C	(1002, 1500)	
	1002	1000	Book D	(1002, 1000)	
	1003	1000	Book E	(1003, 1000)	

システムによって, キーごとにグルーピングされると

Key a    関連付けられた属性BのValueのリスト

(1001, [1000, 2000])  
(1002, [1500, 1000])  
(1003, 1000)

# マップレデュースによるグルーピングと集約

システムによって、キーごとにグルーピングされると

Key a 関連付けられた属性BのValueのリスト

(1001, [1000, 2000])

(1002, [1500, 1000])

(1003, 1000)

- レデュース関数

- \* 各キーaがグループを表現しています

- \* Key aに関連付けられた属性BのValueのリスト[b<sub>1</sub>, b<sub>2</sub>,...,b<sub>n</sub>]に  
集約化演算子 $\theta(B)$ を適用する

出力:     a          $x = \theta(B)$

(1001, 1000+2000)

(1002, 1500+1000)

(1003, 1000)

THE END