コンピュータ科学特別講義Ⅳ　第２回資料

2018. 4. 13. 枝廣

- 並列対応プログラムについて
  - 並列プログラムモデル
    - SIMD, MIMD
  - アーキテクチャモデル
    - クラスタ/AMP (Multi-CPU, Multi-OS、分散メモリ)
    - SMP (Multi-CPU, Single-OS、共有メモリ)
    - (Single-CPU with Multi-ALU, Single-OS)
  - 並列プログラミング言語、ＡＰＩ（Application Program Interface)

- Class WWW: http://www.pdsl.jp/class/utyo2018/

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

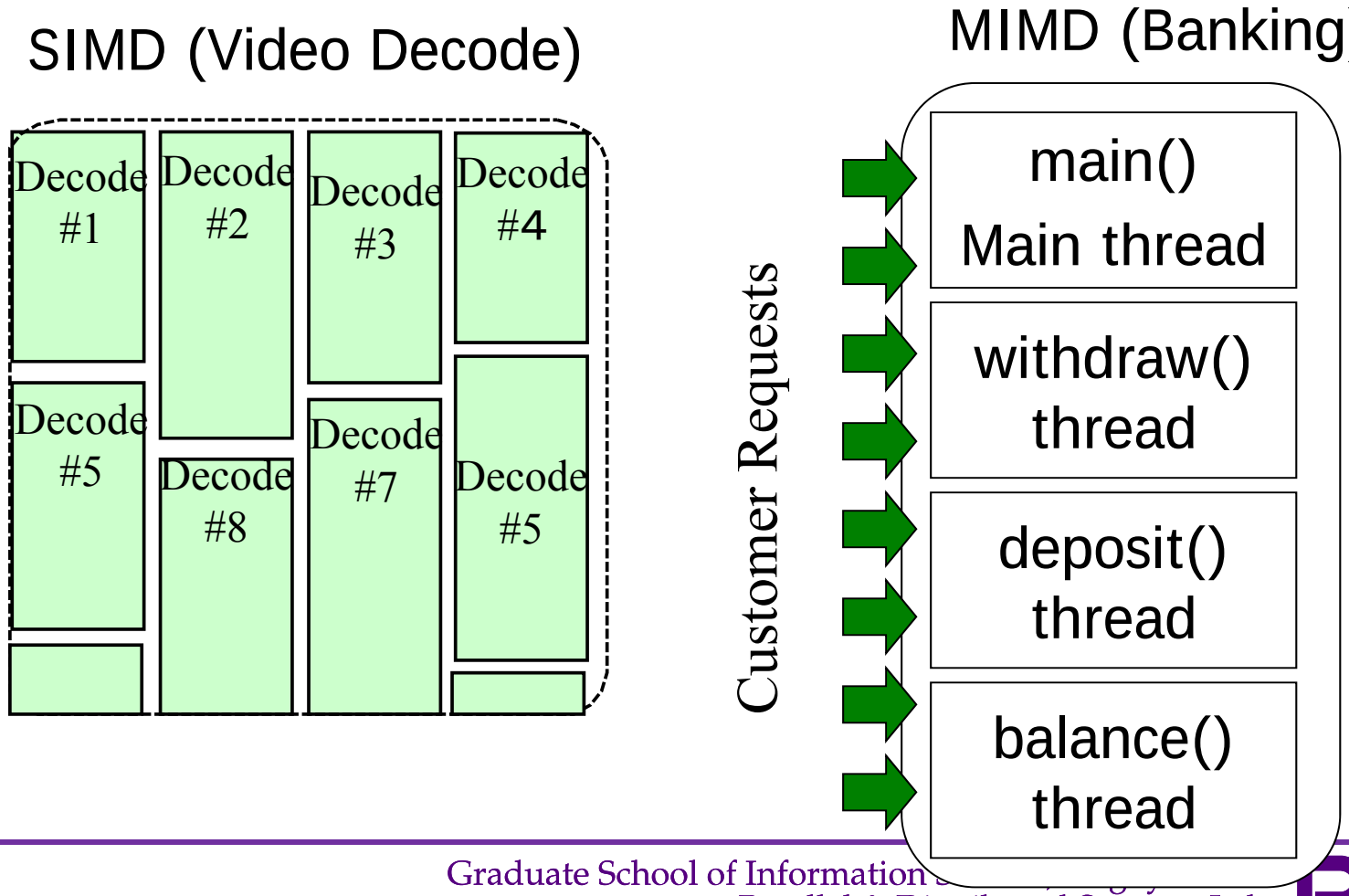
Contents of This Class

- 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)

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

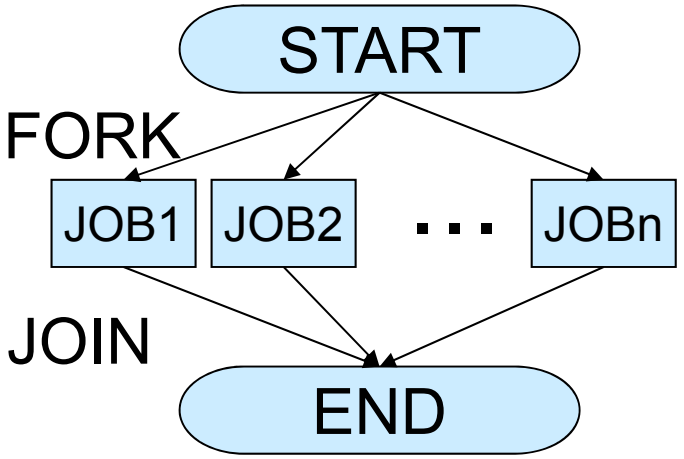
並列プログラムモデル

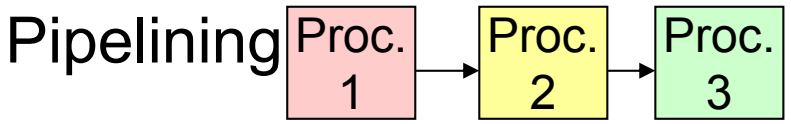
- SIMD (Single Instruction, Multiple Data)
- MIMD (Multiple Instruction, Multiple Data)

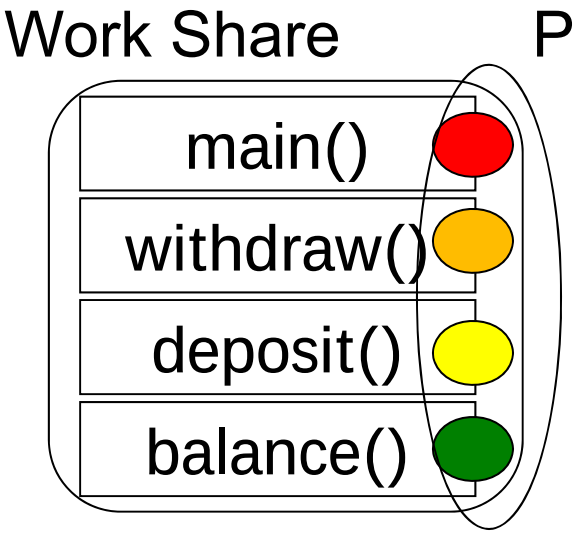


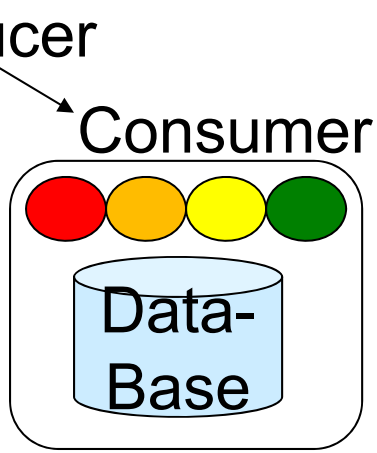
プログラムモデルに関するその他の用語 (分類ではない)

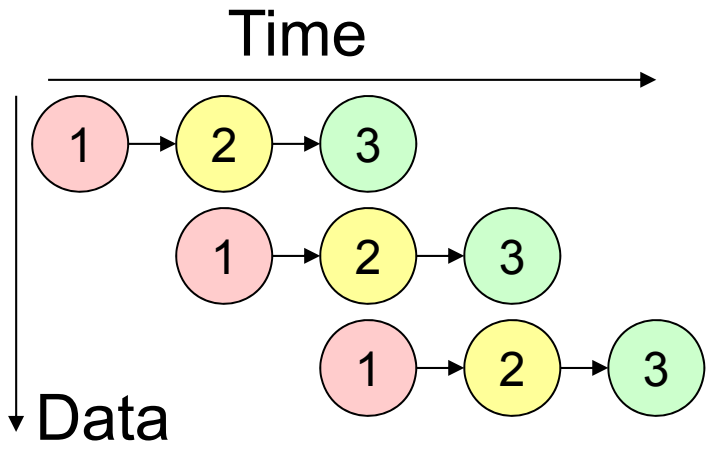
- データ並列 (= SIMD)
- Fork-Join
- ワークシェアリング
- 生産者－消費者
- パイプライン











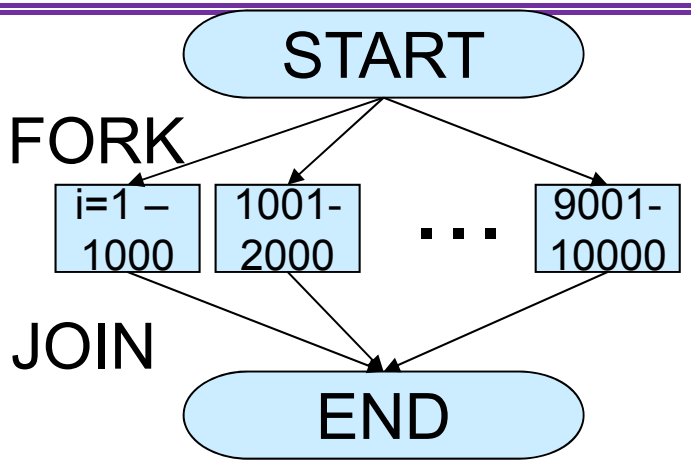
Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

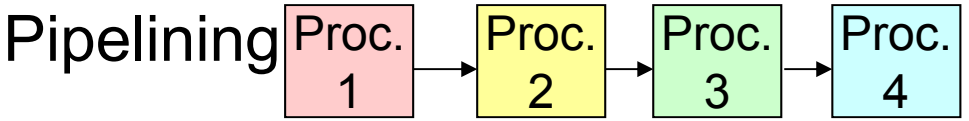
Example 1

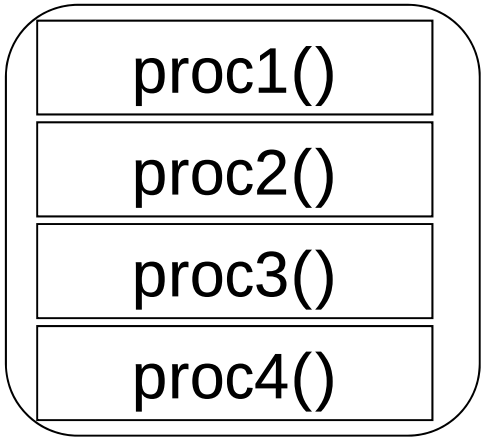
```
for (i=0; i<10000; i++) {
    proc1(i);
    proc2(i);
    proc3(i);
    proc4(i);
}
```

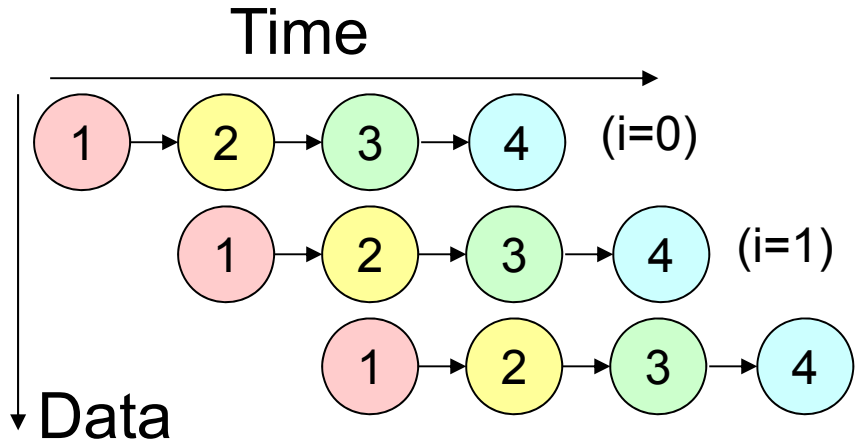
Example 1

- for (i=0; i<10000; i++) {
  proc1(i);
  proc2(i);
  proc3(i);
  proc4(i);
  }

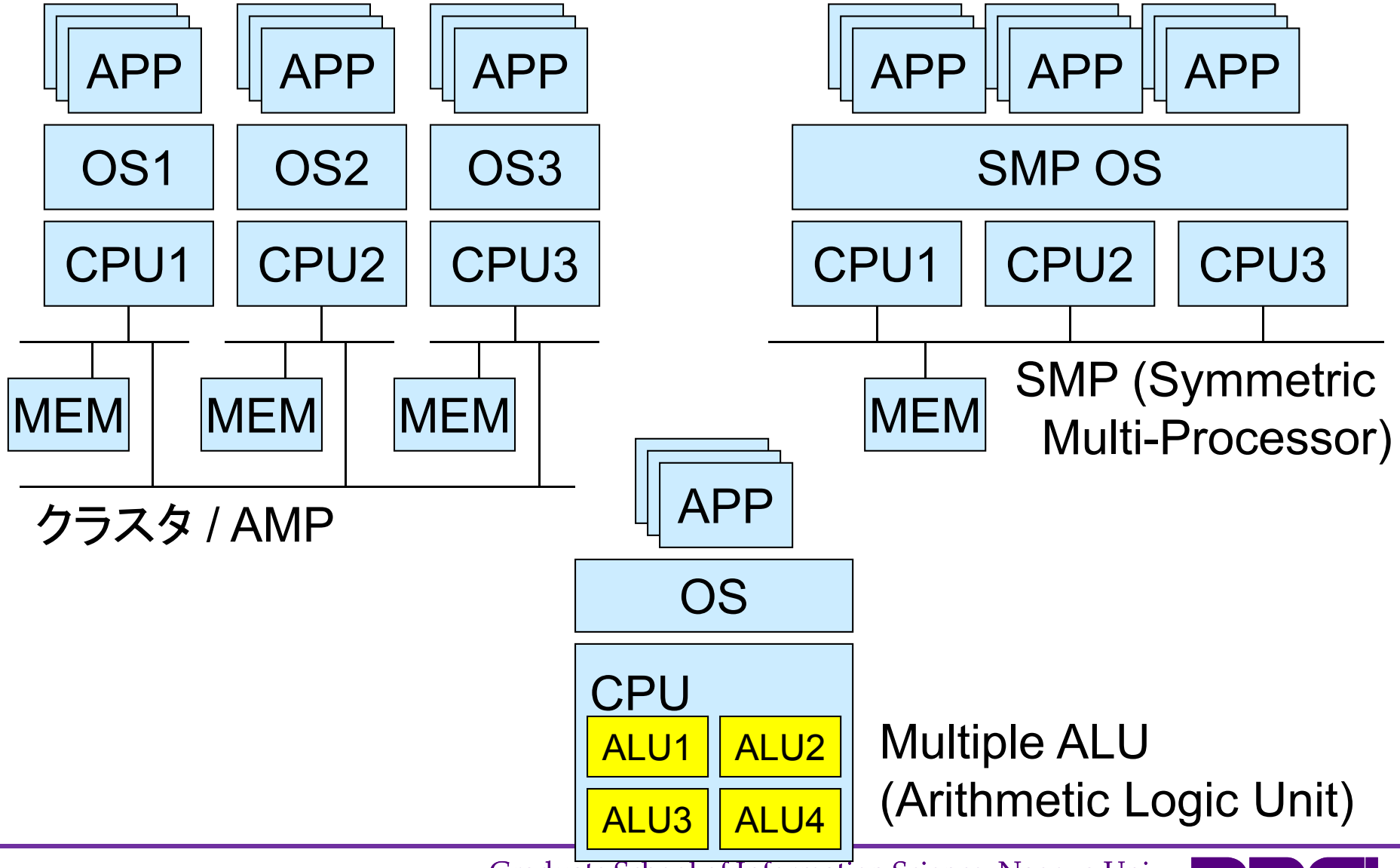




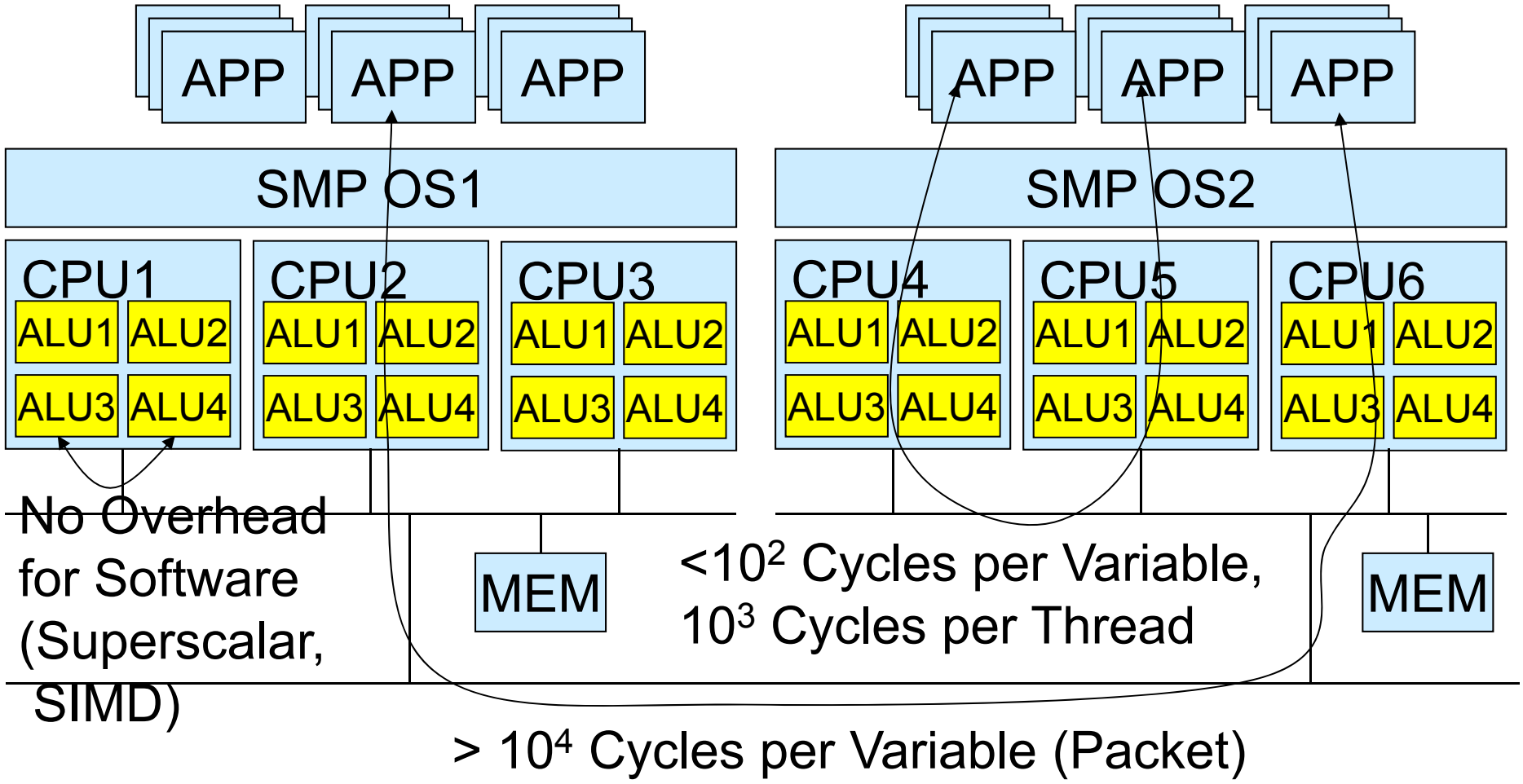




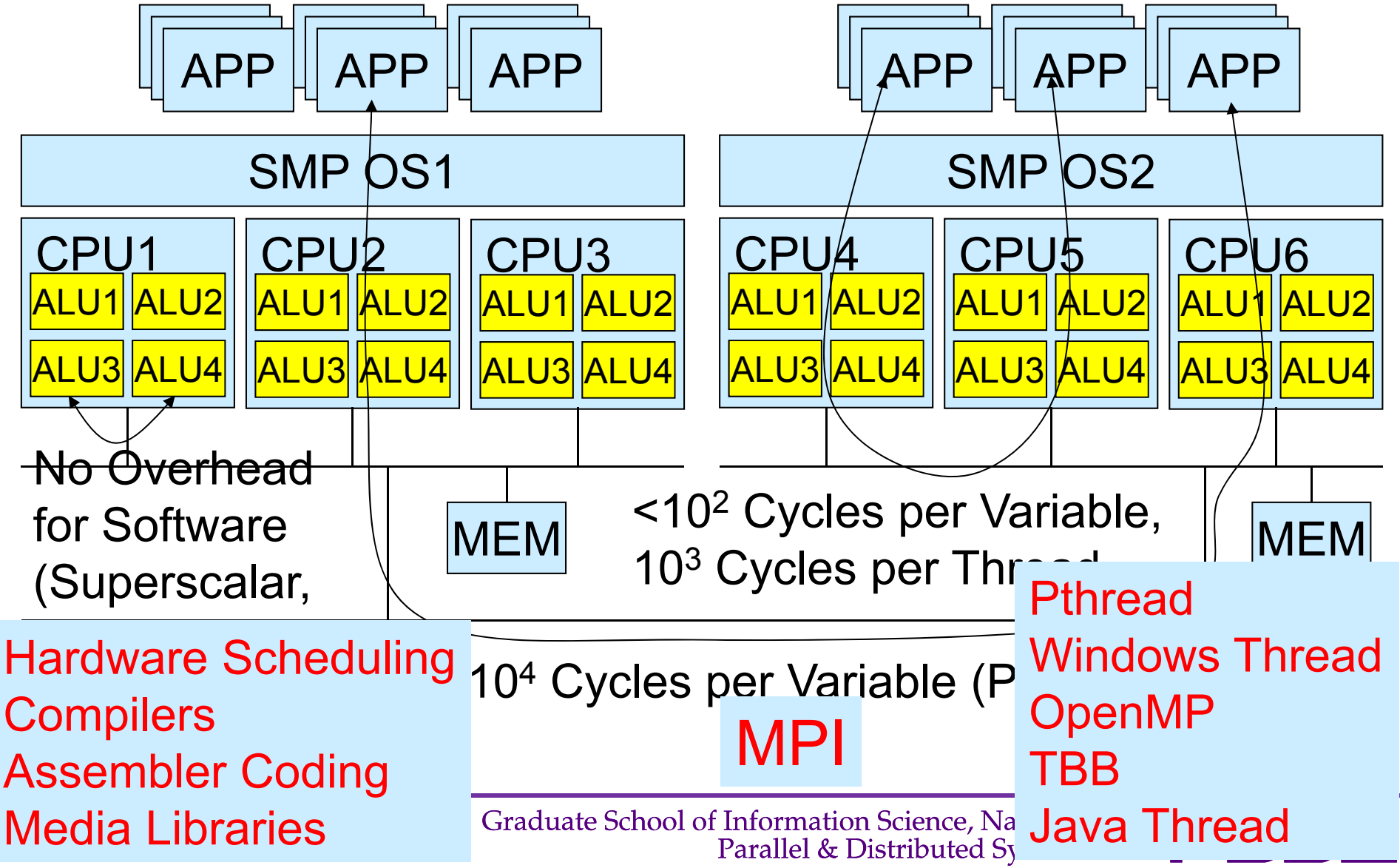
アーキテクチャモデル


APP
APP
APP
OS1
OS2
OS3
CPU1
CPU2
CPU3
MEM
MEM
MEM
クラスタ / AMP
APP
APP
APP
SMP OS
CPU1
CPU2
CPU3
MEM
SMP (Symmetric
Multi-Processor)
APP
OS
CPU
ALU1
ALU2
ALU3
ALU4
Multiple ALU
(Arithmetic Logic Unit)

将来のマルチコア、メニーコアの可能性として



並列プログラムモデル、ＡＰＩ



OS スレッドライブラリ

- pthread
  - IEEE POSIX Section 1003.1c
    POSIX: Portable Operating System Interface
  - Nichols, Buttlar, and Farrell: Pthreads Programming, O'REILLY, 1998.
  - Linuxなどで標準
  - pthread_create, pthread_join
- Windows Thread API
  - CreateThread, WaitForMultipleObjects

Example2: Calculate Primes

```
#include <stdio.h>
#include <math.h>

#define DATA_NUM 100

int main() {
    BOOL primes[DATA_NUM];
    int i;
```

```
    /* Check */
    for (i = 0; i < DATA_NUM; i++) {
        primes[i] = TRUE;
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (primes[j] && i % j == 0) {
                primes[i] = FALSE;
                break;
            }
    }

    /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

If primes[i] is TRUE (j is a prime), and (i % j == 0) (i is multiple number of j), i is not a prime.
If j is not a prime, we don't have to check if I is multiple number of j.
Why?

Pthread (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <pthread.h>

#define THREAD_NUM 3
#define DATA_NUM 100

typedef struct _thread_arg
{
    int id;
    bool *primes;
} thread_arg_t;
```

Calc Primes

マルチコアCPUのための
並列プログラミング
（秀和システムズ）より

```
void thread_func(void *arg) {
    thread_arg_t* targ = (thread_arg_t *)arg;
    int c_start, c_end, range, limit;
    int i, j;

    /* Determine Range of Values to be Checked */
    range = (DATA_NUM - 2) / THREAD_NUM + 1;
    c_start = 2 + targ->id *range;
    c_end = 2 + (targ->id+1) *range;
    if (c_end > DATA_NUM) c_end = DATA_NUM;

    /* Check */
    for (i = c_start; i < c_end; i++) {
        limit = (int)sqrt((double) i);
        for (j = 2; j <= limit; j++)
            if (targ->primes[j] && i % j == 0) {
                targ->primes[i] = false;
                break;
            }
    }
    return;
}
```

```
int main() {
    pthread_t
handle[THREAD_NUM];
    thread_arg_t
targ[THREAD_NUM];
    bool primes[DATA_NUM];
    int i;

    /* Initialize */
    for (i = 0; i < DATA_NUM; i++)
        primes[i] = true;

    /* Start */
    for (i = 0; i < THREAD_NUM;
i++) {
        targ[i].id = i;
        targ[i].primes = primes;
        pthread_create(&handle[i],
NULL, (void*)thread_func,
(void*)&targ[i]);
    }
```

```
    /* Wait for All Threads */
    for (i = 0; i < THREAD_NUM; i++)
        pthread_join(handle[i], NULL);

    /* Output */
    for (i = 2; i < DATA_NUM; i++)
        if (primes[i])
            printf("%d ", i);
    printf("¥n");
    return 0;
```

Pthread (2/2)

Windows thread (1/2)

```
#include <stdio.h>
#include <windows.h>
#include <math.h>

#define THREAD_NUM 3
#define DATA_NUM 100

typedef struct _thread_arg
{
    int id;
    BOOL *primes;
} thread_arg_t;
```

```
void thread_func(void *arg) {
    thread_arg_t* targ = (thread_arg_t *)arg;
    int c_start, c_end, range, limit;
    int i, j;

    /* Determine Range of Values to be Checked */
    range = (DATA_NUM - 2) / THREAD_NUM + 1;
    c_start = 2 + targ->id * range;
    c_end = 2 + (targ->id + 1) * range;
    if (c_end > DATA_NUM) c_end = DATA_NUM;

    /* Check */
    for (i = c_start; i < c_end; i++) {
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (targ->primes[j] && i % j == 0) {
                targ->primes[i] = FALSE;
                break;
            }
    }
    return;
}
```

Calc Primes

マルチコアCPUのための並列プログラミング
（秀和システムズ）より

```
int main() {
    HANDLE handle[THREAD_NUM];
    thread_arg_t targ[THREAD_NUM];
    BOOL primes[DATA_NUM];
    int i;

    for (i = 0; i < DATA_NUM; i++) {
        primes[i] = TRUE;
    }

    for (i = 0; i < THREAD_NUM; i++) {
        targ[i].id = i;
        targ[i].primes = primes;
        handle[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)thread
_func, (void *)&targ[i], 0, NULL);
    }
```

```
WaitForMultipleObjects(THREAD_NUM,
handle, TRUE, INFINITE);

 /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

Windows thread (2/2)

OpenMP

- OS スレッドライブラリは低レベル
  - プログラマはアーキテクチャを考慮し、粒度や負荷分散を考えながら自分でプログラムを切って記載する必要がある。
- OpenMP
  - C/C++/FORTRANの指示行として並列を記載
  - USのコンパイラベンダが集まって開発
  - PC向けの開発環境などでサポートされている
  - http://www.openmp.org/
  - Fork-Join Model
  - 粒度はランタイムによって決められる

OpenMP

```
#include <stdio.h>
#include <math.h>
#include <omp.h>

#define DATA_NUM 100

int main() {
    BOOL primes[DATA_NUM];
    int i,j;

    /* Initialize */
#pragma omp parallel for
    for (i = 0; i < DATA_NUM; i++)
        primes[i] = TRUE;
```

Calc Primes

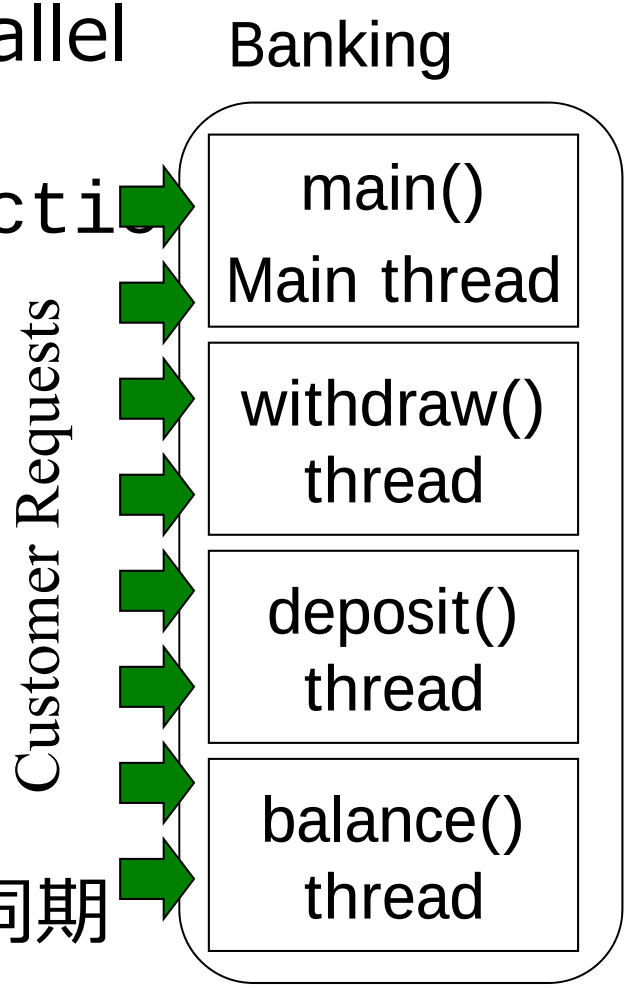
```
    /* Check */
#pragma omp parallel for
    for (i = 0; i < DATA_NUM; i++) {
        primes[i] = TRUE;
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (primes[j] && i % j == 0) {
                primes[i] = FALSE;
                break;
            }
    }

    /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

Software for SMP （OpenMP)

- Example of OpenMP (Banking)
  - Execute ‘section’s in Parallel within ‘sections’ block

```
#pragma omp parallel sections
{
#pragma omp section
   main();
#pragma omp section
   withdraw();
#pragma omp section
   deposit();
#pragma omp section
   balance();
}
```

  - ‘sections’ブロックの‘}’で同期 (すべてのsectionは’}’で同期)

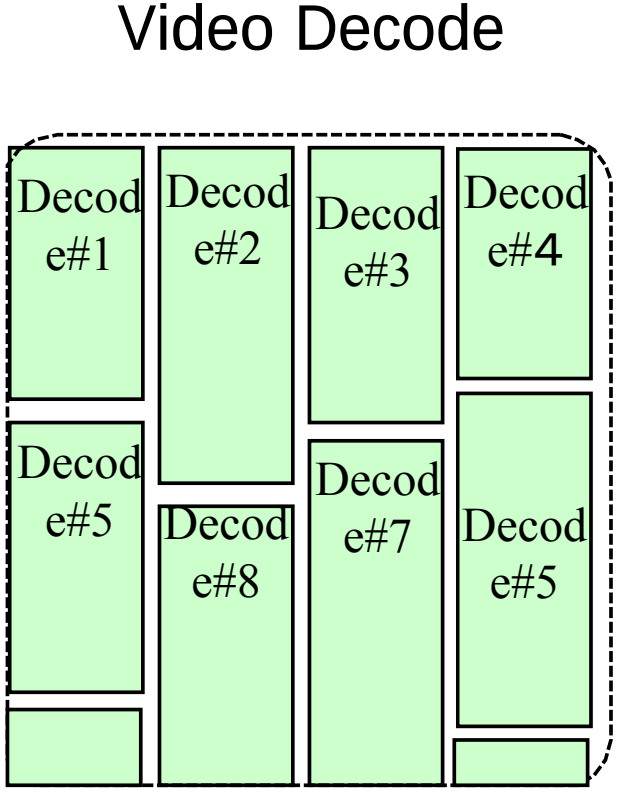


Software for SMP (OpenMP)

- Example of OpenMP (Video Decode)
  - for-loop with 'for' Directive is executed in Parallel

```
#pragma omp parallel for
    for(i=1; i<=N; i++)
       Decode#i;
```

  - その他の指示文
    - 総和
    - バリア
    - アトミック
    - …



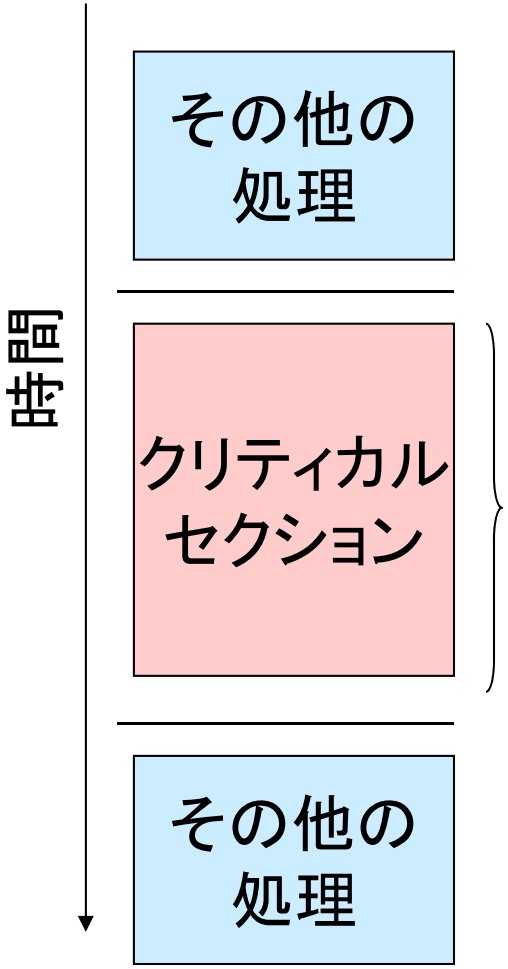
その他の並列言語、ＡＰＩ

- Java Thread
  - Java言語の中で定義
  - OS Threadと同程度の低レベル
- TBB (Threading Building Block)
  - By Intel, for C++
  - OpenMPに近い
  - より自由度の大きいスレッドスケジュールが可能
  - J. Reinders: “Intel Threading Building Blocks”, O’REILLY, 2007

排他制御に関する用語

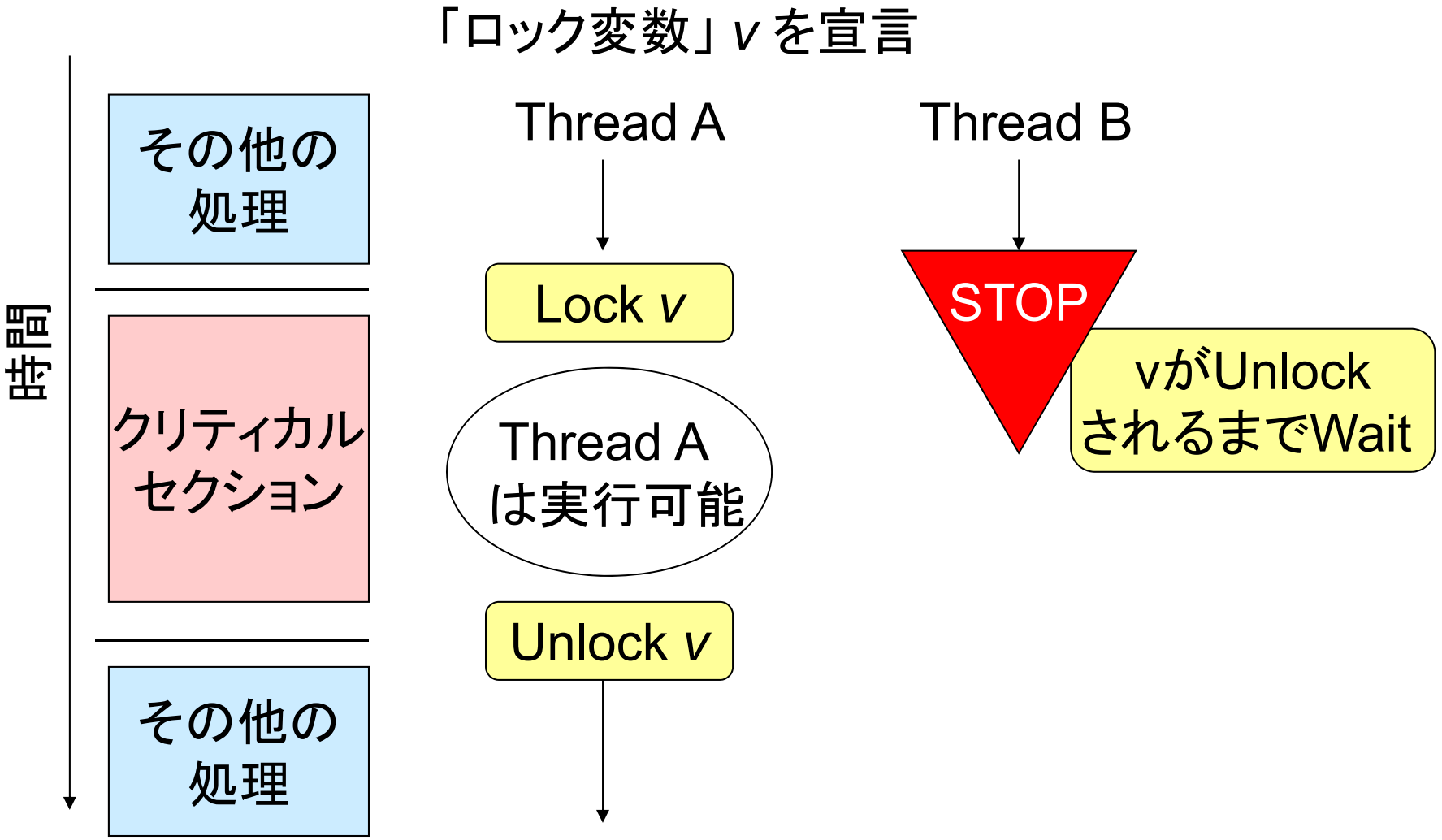
- クリティカルセクション
  - 一度に一つのプロセスまたはスレッドのみが実行可能なプログラムの部分
  - 例：グローバル変数の書換 (素数の数のカウント)
- 共有リソース
  - メモリ、周辺デバイスなど

排他制御



一度に一つのプロセス（スレッド）のみが実行可能
例：グローバル変数の書換
　　共有リソースの利用

Lock - Unlock


「ロック変数」v を宣言
時間
その他の処理
クリティカルセクション
その他の処理
Thread A
Lock v
Thread A は実行可能
Unlock v
Thread B
STOP
vがUnlockされるまでWait

排他制御の例

- Mutex (= Mutual Exclusion)
  - ある変数のLock/Unlock
- セマフォ
  - リソースが複数ある場合に利用
  - 利用可能なリソース数を保持し、リソースが残っている限りプログラムはクリティカルセクションに入れる
  - Mutexはリソース数が一つの特殊ケースと考えられる

pthread, POSIX セマフォ

- pthread mutex
  - pthread_mutex_init
    - ロック変数の初期化
  - pthread_mutex_lock, pthread_mutex_unlock
  - pthread_destroy
- POSIX セマフォ
  - sem_init
  - sem_wait, sem_post
  - sem_destroy

Windows Thread API

- クリティカルセクション
  - InitializeCriticalSection
  - EnterCriticalSection, LeaveCriticalSection
  - DeleteCriticalSection
- セマフォ
  - CreateSemaphore
  - WaitForSingleObject, ReleaseSemaphore
  - CloseHandle

Example2': Calculate Primes and count # of Primes

```
#include <stdio.h>
#include <math.h>

#define DATA_NUM 100

int main() {
    BOOL primes[DATA_NUM];
    int I, count;
```

```
    /* Check */
    for (i = 0; i < DATA_NUM; i++) {
        primes[i] = TRUE;
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (primes[j] && i % j == 0) {
                primes[i] = FALSE;
                break;
            }
    if (j > limit) count++;
    }

    /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

Pthread (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <pthread.h>

#define THREAD_NUM 3
#define DATA_NUM 100

typedef struct _thread_arg {
    int id;
    bool *primes;
    pthread_mutex_t *mutex;
} thread_arg_t;

int count;
```

Calc Primes

```
void thread_func(void *arg) {
    thread_arg_t* targ = (thread_arg_t *)arg;
    int c_start, c_end, range, limit;
    int i, j;

    /* Determine Range of Values to be Checked */
    range = (DATA_NUM - 2) / THREAD_NUM + 1;
    c_start = 2 + targ->id *range;
    c_end = 2 + (targ->id+1) *range;
    if (c_end > DATA_NUM) c_end = DATA_NUM;

    /* Check */
    for (i = c_start; i < c_end; i++) {
        limit = (int)sqrt((double) i);
        for (j = 2; j <= limit; j++)
            if (targ->primes[j] && i % j == 0) {
                targ->primes[i] = false;
                break;
            }
        if(j > limit) {
            pthread_mutex_lock(targ->mutex);
            count++;
            pthread_mutex_unlock(targ->mutex);
        }
    }
    return;
}
```

```
int main() {
    pthread_t handle[THREAD_NUM];
    thread_arg_t targ[THREAD_NUM];
    bool primes[DATA_NUM];
    int i;
    pthread_mutex_t mutex;

    /* Initialize */
    for (i = 0; i < DATA_NUM; i++)
        primes[i] = true;

    /* Initialize mutex variable */
    pthread_mutex_init(&mutex,
NULL);

    /* Start */
    for (i = 0; i < THREAD_NUM; i++) {
        targ[i].id = i;
        targ[i].primes = primes;
        targ[i].mutex = &mutex;
        pthread_create(&handle[i], NULL,
(void*)thread_func, (void*)&targ[i]);
    }
```

```
    /* Wait for All Threads */
    for (i = 0; i < THREAD_NUM; i++)
        pthread_join(handle[i], NULL);

    /* Destroy Mutex Variable */
    pthread_mutex_destroy(&mutex);

    /* Output */
    for (i = 2; i < DATA_NUM; i++)
        if (primes[i])
            printf("%d ", i);
    printf("¥n");
    return 0;
```

Pthread (2/2)

Windows thread (1/2)

```
#include <stdio.h>
#include <windows.h>
#include <math.h>

#define THREAD_NUM 3
#define DATA_NUM 100

typedef struct _thread_arg {
    int id;
    BOOL *primes;
CRITICAL_SECTION *cs;
} thread_arg_t;

int count;
```

Calc Primes

```
void thread_func(void *arg) {
    thread_arg_t* targ = (thread_arg_t *)arg;
    int c_start, c_end, range, limit;
    int i, j;

    /* Determine Range of Values to be Checked */
    range = (DATA_NUM - 2) / THREAD_NUM + 1;
    c_start = 2 + targ->id * range;
    c_end = 2 + (targ->id + 1) * range;
    if (c_end > DATA_NUM) c_end = DATA_NUM;

    /* Check */
    for (i = c_start; i < c_end; i++) {
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (targ->primes[j] && i % j == 0) {
                targ->primes[i] = FALSE;
                break;
            }
      if(j > limit) {
            EnterCriticalSection(targ->cs);
            count++;
            LeaveCriticalSection(targ->cs);
        }
    }
    return;
}
```

```
int main() {
    HANDLE handle[THREAD_NUM];
    thread_arg_t targ[THREAD_NUM];
    BOOL primes[DATA_NUM];
    int i;
    CRITICAL_SECTION cs;

    for (i = 0; i < DATA_NUM; i++) {
        primes[i] = TRUE;
    }

 /* Initialize critical section variable */
    InitializeCriticalSection(&cs);

    for (i = 0; i < THREAD_NUM; i++) {
        targ[i].id = i;
        targ[i].primes = primes;
        targ[i].mutex = &cs;
        handle[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)thread_
func, (void *)&targ[i], 0, NULL);
    }
```

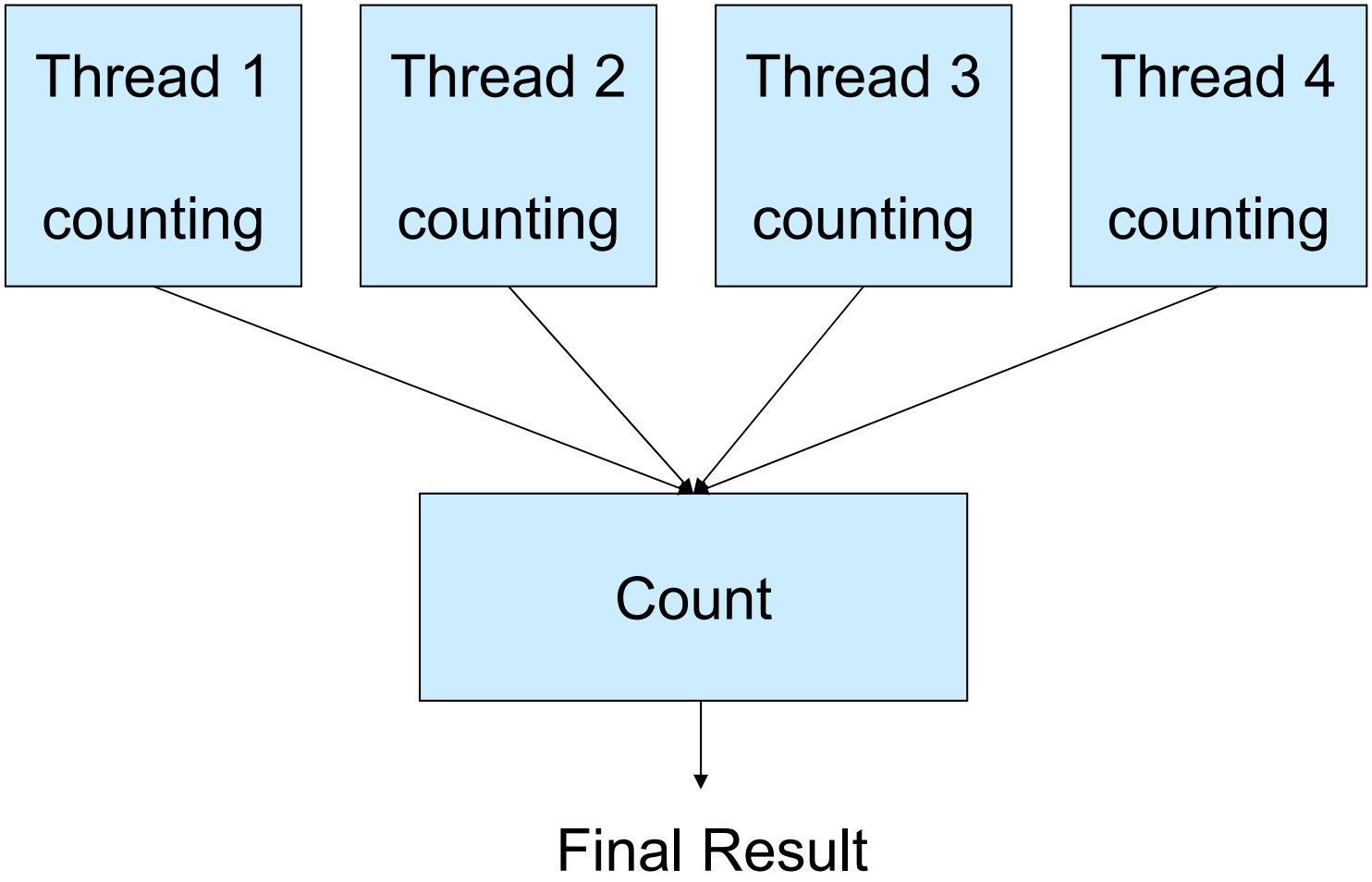
```
    WaitForMultipleObjects(THREAD_NUM,
handle, TRUE, INFINITE);

    /* Destroy critical section Variable */
   DeleteCriticalSection(&cs);

     /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

Windows thread (2/2)

OpenMP

- Clause
  - 付加情報
  - private, shared (変数)
  - reduction (演算)
- #pragma omp critical
- #pragma omp atomic
  - ある文に対するクリティカルセクション

Reduction


Thread 1
counting
Thread 2
counting
Thread 3
counting
Thread 4
counting
Count
Final Result

OpenMP

```
#include <stdio.h>
#include <math.h>
#include <omp.h>

#define DATA_NUM 100

int main() {
    BOOL
primes[DATA_NUM];
    int i,j, count;

 /* Initialize */
#pragma omp parallel for
    for (i = 0; i < DATA_NUM;
i++)
        primes[i] = TRUE;
```

```
    /* Check */
#pragma omp parallel for reduction(+;count)
private(limit, j)
    for (i = 0; i < DATA_NUM; i++) {
        limit = (int)sqrt((double)i);
        for (j = 2; j <= limit; j++)
            if (primes[j] && i % j == 0) {
                primes[i] = FALSE;
                break;
            }
        if (j > limit) count++;
    }

    /* Output */
    for (i = 2; i < DATA_NUM; i++) {
        if (primes[i] == 1) printf("%d ", i);
    }
    printf("¥n");
    return 0;
}
```

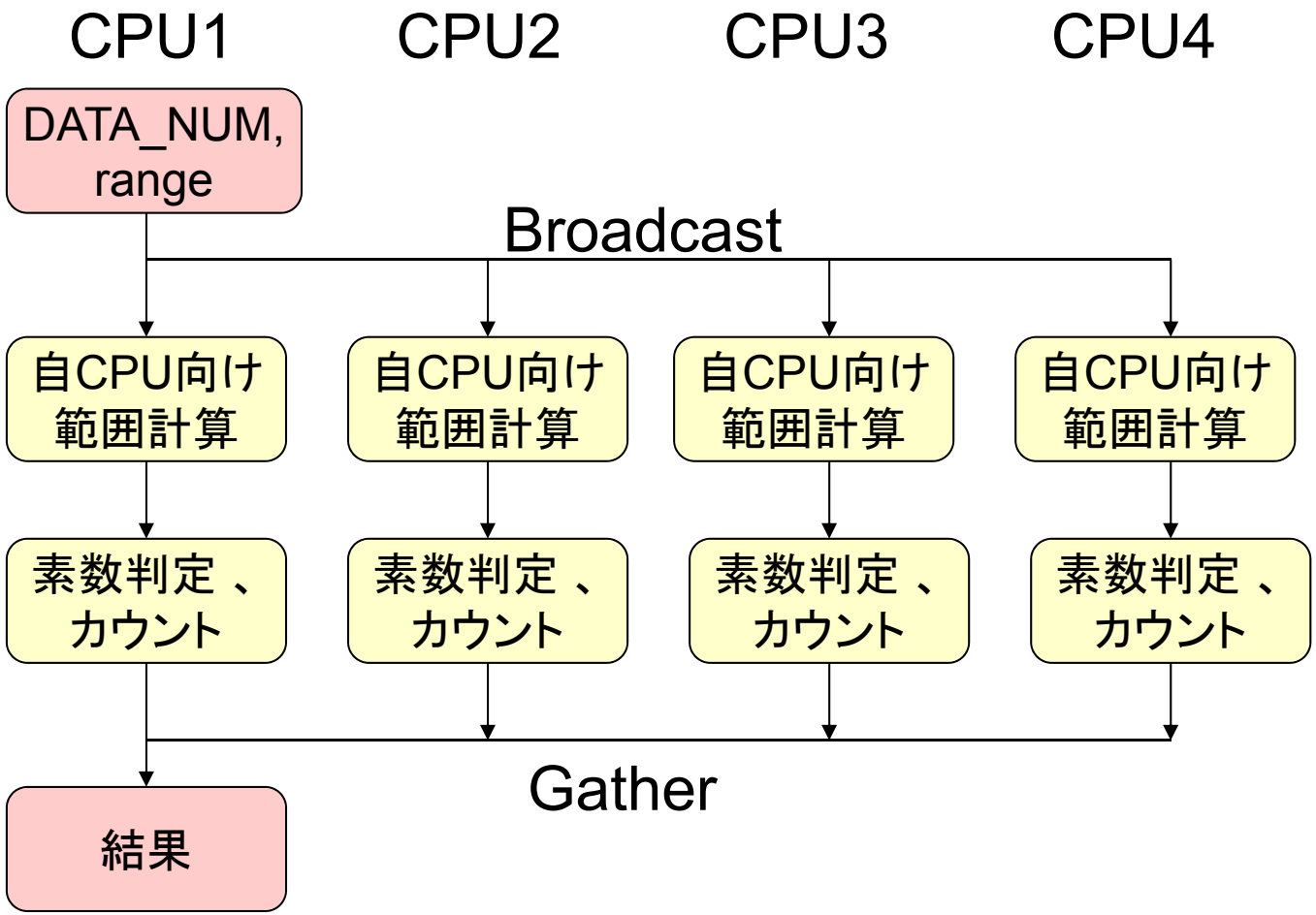
Calc Primes

MPI (Message Passing Interface)

- http://www.mpi-forum.org/
- http://www-unix.mcs.anl.gov/mpi/
- 分散メモリアーキテクチャ向けの低レベルのプロセス間通信API
  - MPI_SEND, MPI_RECV
  - MPI_BCAST, MPI_GATHER, MPI_SCATTER

- 将来のマルチコア、メニコアは分散システムになるといわれている。そのため（共有メモリシステムに対するOpenMPやTBBのような）言語やＡＰＩが今後考えられると思われる

素数計算の例


CPU1
CPU2
CPU3
CPU4
DATA_NUM, range
Broadcast
自CPU向け範囲計算
自CPU向け範囲計算
自CPU向け範囲計算
自CPU向け範囲計算
素数判定、カウント
素数判定、カウント
素数判定、カウント
素数判定、カウント
Gather
結果

典型的な関数の例

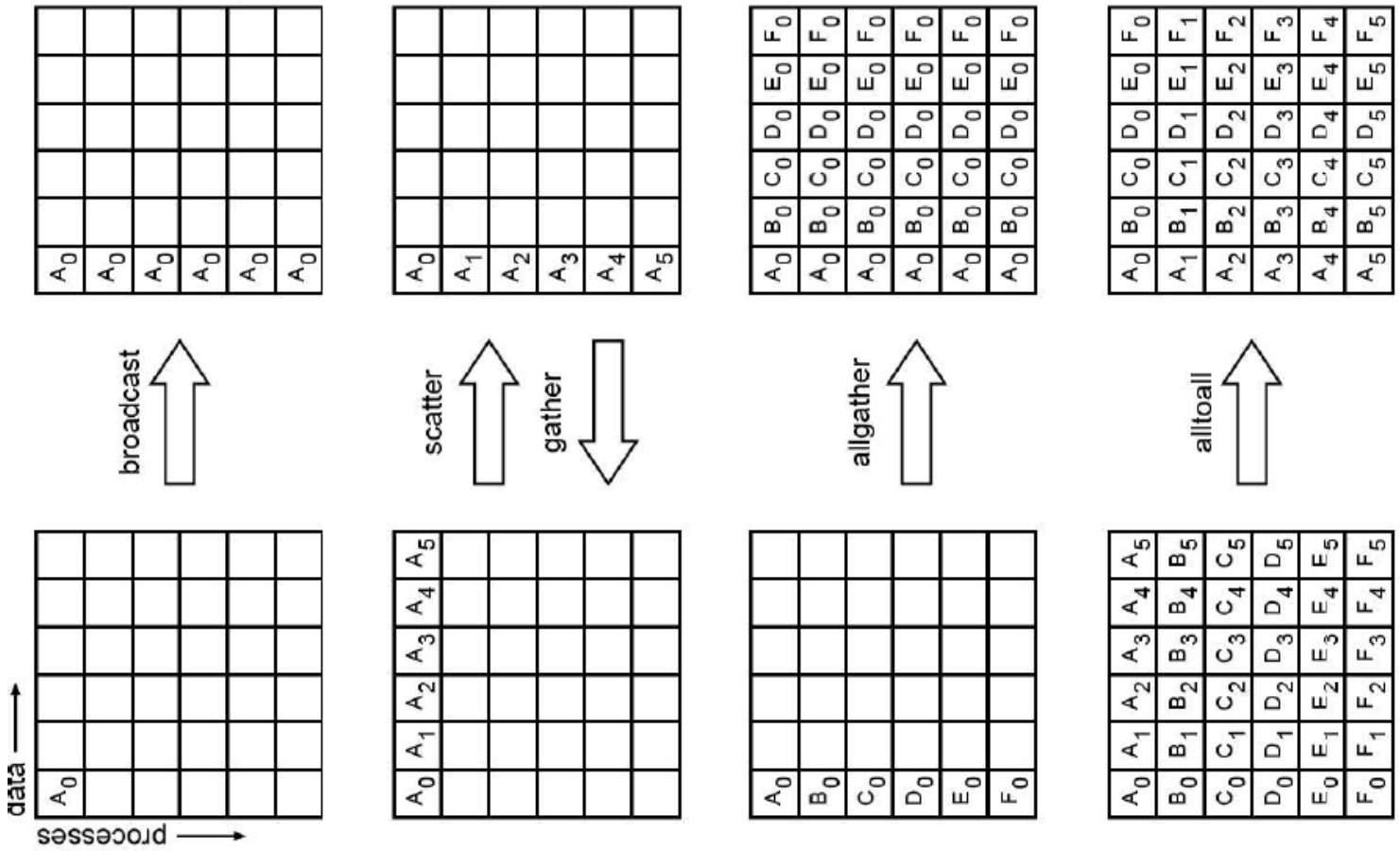


図 4.1: 6 プロセスのグループの集団通信関数の図解。それぞれの場合において、各行はあるプロセスの中のデータ位置を表している。すなわち、ブロードキャストでは、はじめに第 1 プロセスだけがデータ A_0 を持っているが、ブロードキャスト後、全プロセスがそのデータ A_0 を持つ。

Exercise 1

- Set Up OpenMP Environment (OpenMP環境を立ち上げよ)
  - Submit a screen copy to show an evidence (証拠として画面コピーを提出せよ) (to eda@ertl.jp)
  - Deadline: May 24, 2018
- Resource
  - https://computing.llnl.gov/tutorials/openMP/
  - https://computing.llnl.gov/tutorials/openMP/exercise.html
  - Compile options
    - Intel compiler: /Qopenmp
    - Visual C++: /openmp (or check an appropriate box in compile option selection)
    - Gcc: -fopenmp
- You can use “Example for Ex. 1” on Class WWW
- If you don’t have a PC/server with multi-core processors, mail to me (eda@ertl.jp) .