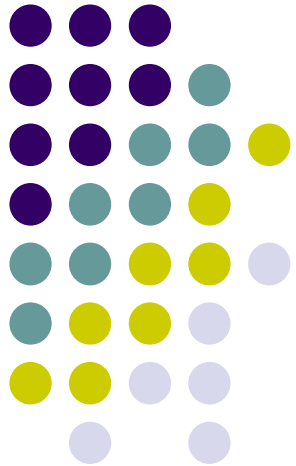
Practical Parallel Computing (実践的並列コンピューティング)

2025 Class No.2

[Introduction Part] (2)

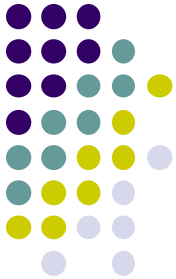
Parallel architecture& Sample programs



Toshio Endo

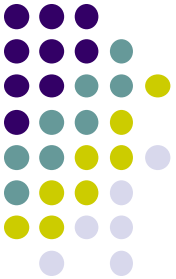
endo@scrc.iir.isct.ac.jp

Overview of This Course



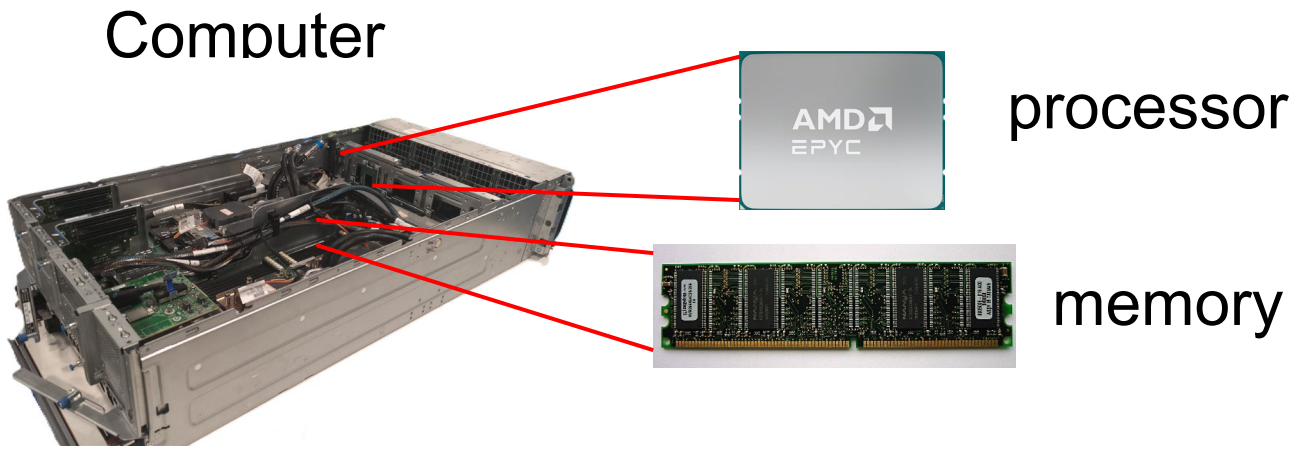
- Introduction Part
  - 2 classes including today ← We are here (2/2)
- OpenMP (OMP) Part
  - 4 classes
  - Report (required)
- OpenACC (ACC) Part
  - 2 classes
  - Report (required)
- CUDA Part
  - 3 classes
  - Report (elective)
- MPI Part
  - 3 classes
  - Report (elective)

Different Parallel Programming Methods



- Why do we learn several programming methods?
  - OpenMP, OpenACC, CUDA, MPI in this lecture

Reason: Programming methods depend on structure of computer hardware (or computer architecture) we will use

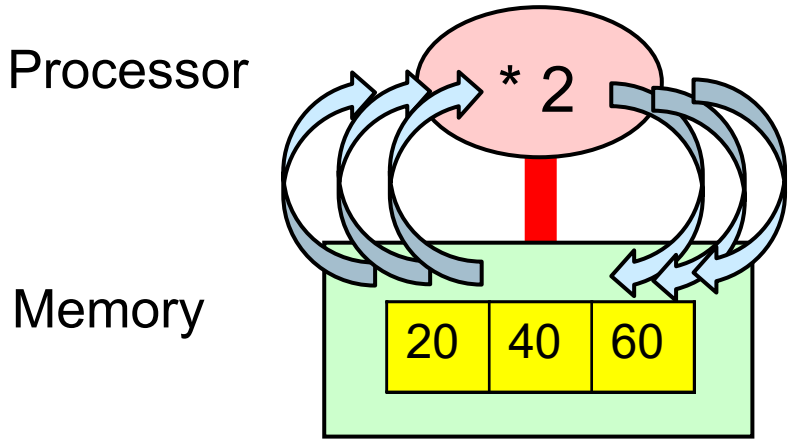


Software Runs on Hardware

- Software = Algorithm + Data
- Hardware (architecture) ≒ Processor + Memory

Note: This is so simplified discussion

Hardware



Software Example

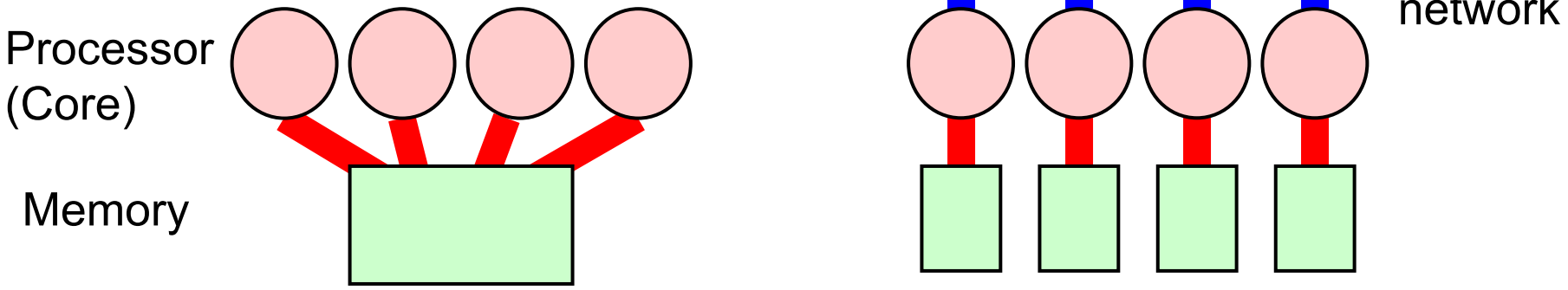
```
int a[3] = {10, 20, 30};
int i;

for (i = 0; i < 3; i++) {
   a[i] = a[i] *2;
}
```

What is Parallel Architecture?

- Parallel architecture has MULTIPLE components
- Two basic types:

Shared memory parallel architecture

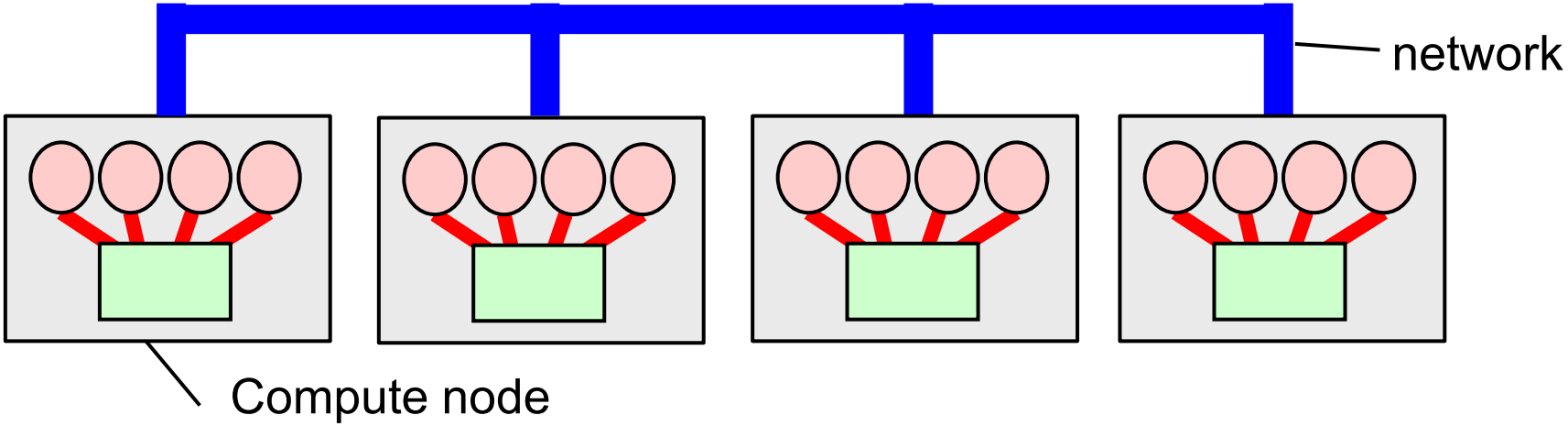
Distributed memory parallel architecture



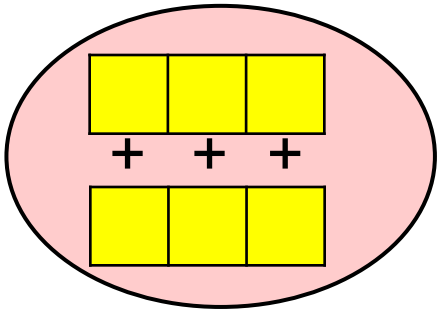
- Different programming methods are used for different architecture

Modern SCs use Both!

Modern SCs are combination of “shared” and “distributed”
“shared memory” in a node
“distributed memory” among nodes, connected by network

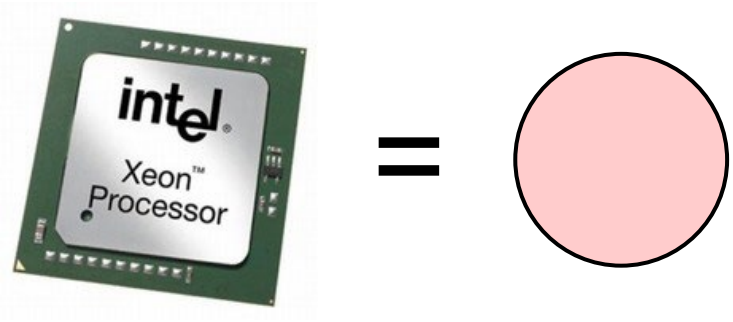


※ Moreover, each processor (core) may have *SIMD parallelism* , such as SSE, AVX…
A processor (core) can do several computations at once
SIMD is out of scope of this class

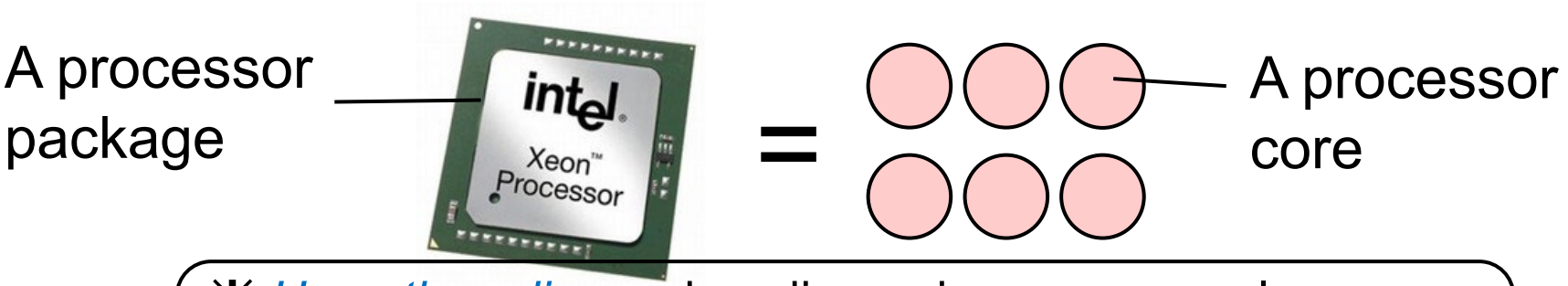


(Confusing) Terminology

- In old days, definition of "processor" was simple



- Since around 2005, "multicore processor" became popular

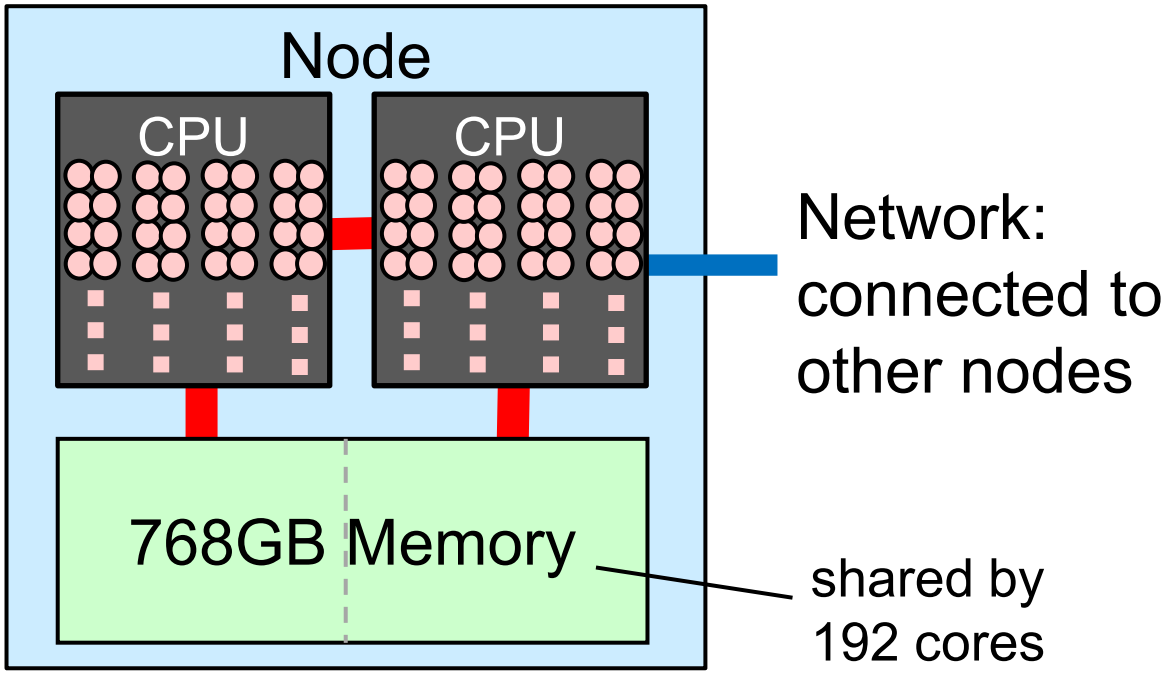


※ *Hyperthreading* makes discussion more complex:
1 physical core = 2 logical cores
In this slide, "core" basically means physical core

A TSUBAME4 Node (1)

- 2 processor packages (CPU) × 96 cores
  - ➔ A TSUBAME4 node has 192 cores

=

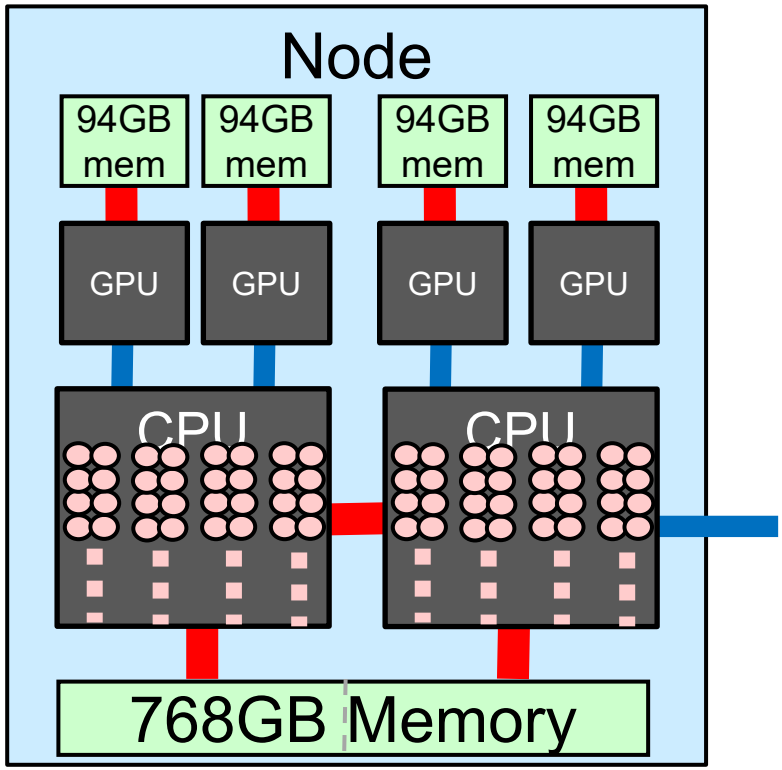


- GPUs are (still) omitted in this figure

A TSUBAME3 Node (2)

- A node has 2 CPUs + 4 GPUs
  - Each GPU (H100) has 132SMs = 16,896 cores

=

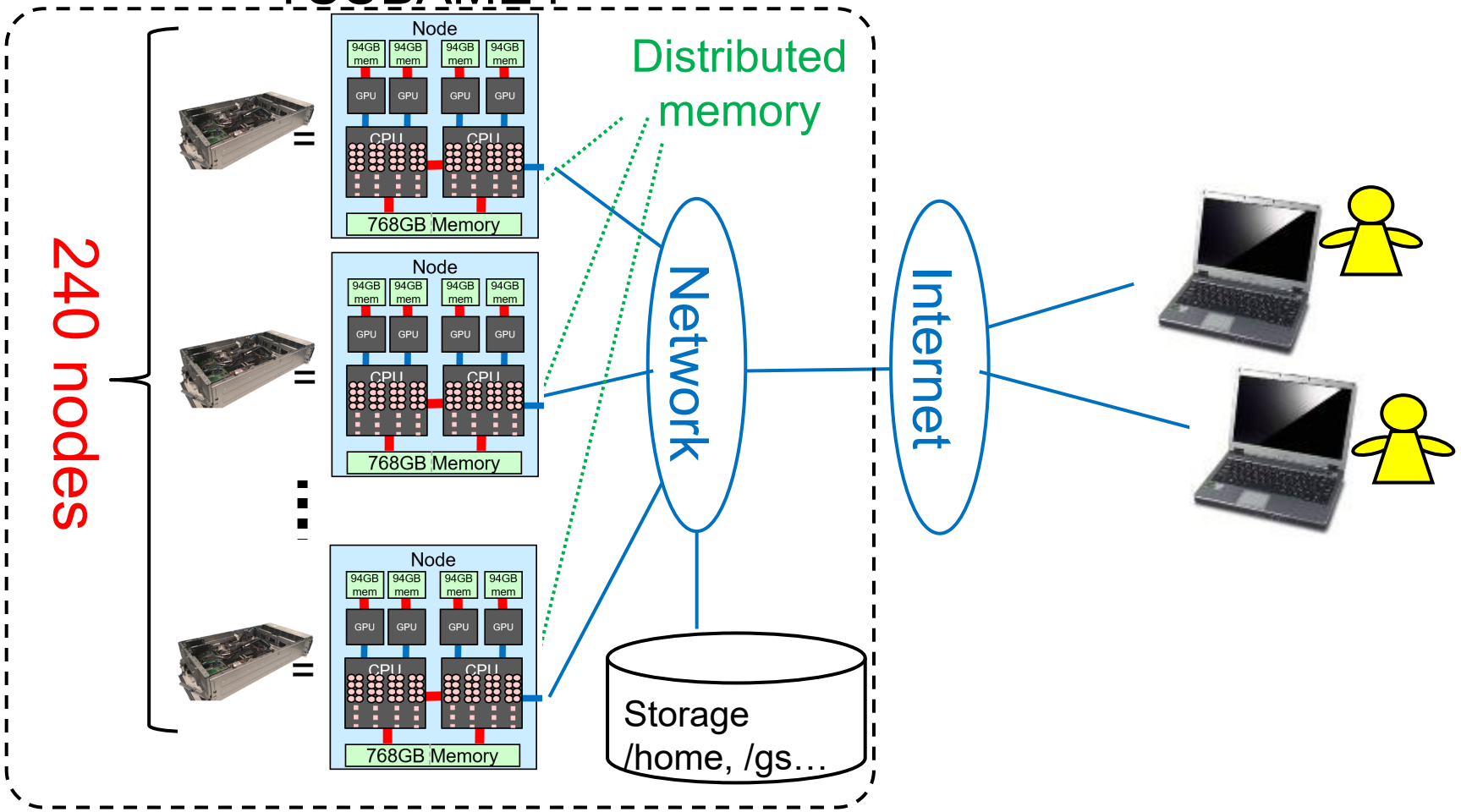


Network: connected to other nodes

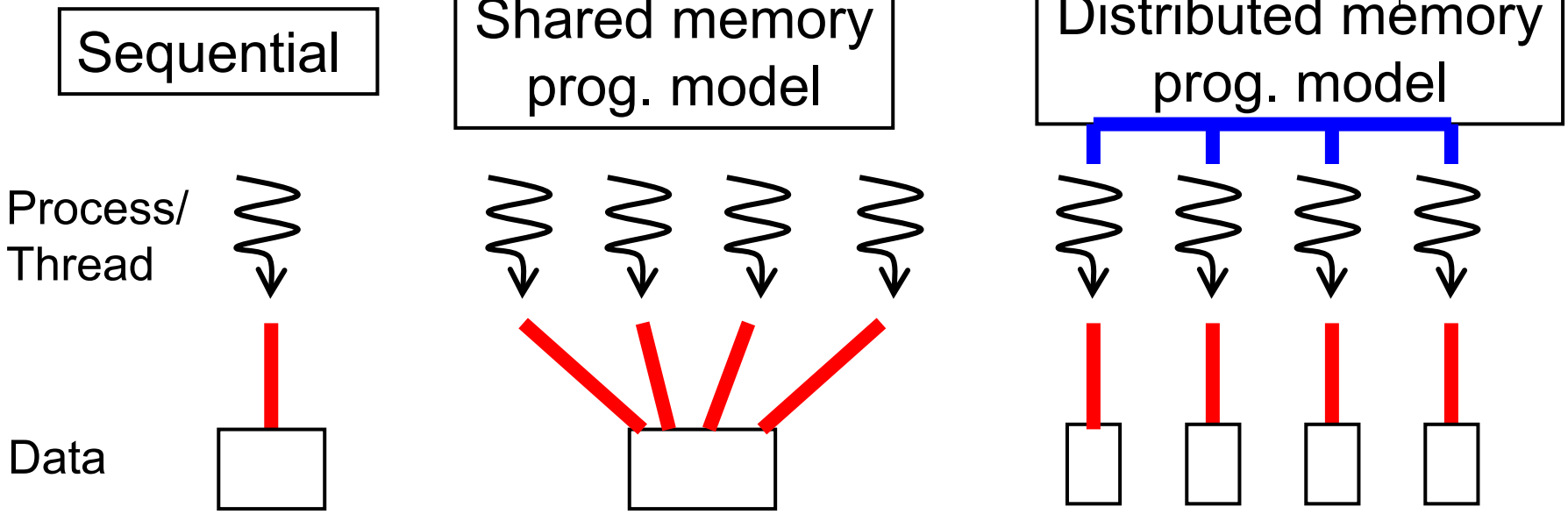
TSUBAME4 System

- 240 nodes (and storage) are connected by fast network

TSUBAME4



Classification of Parallel Programming Models

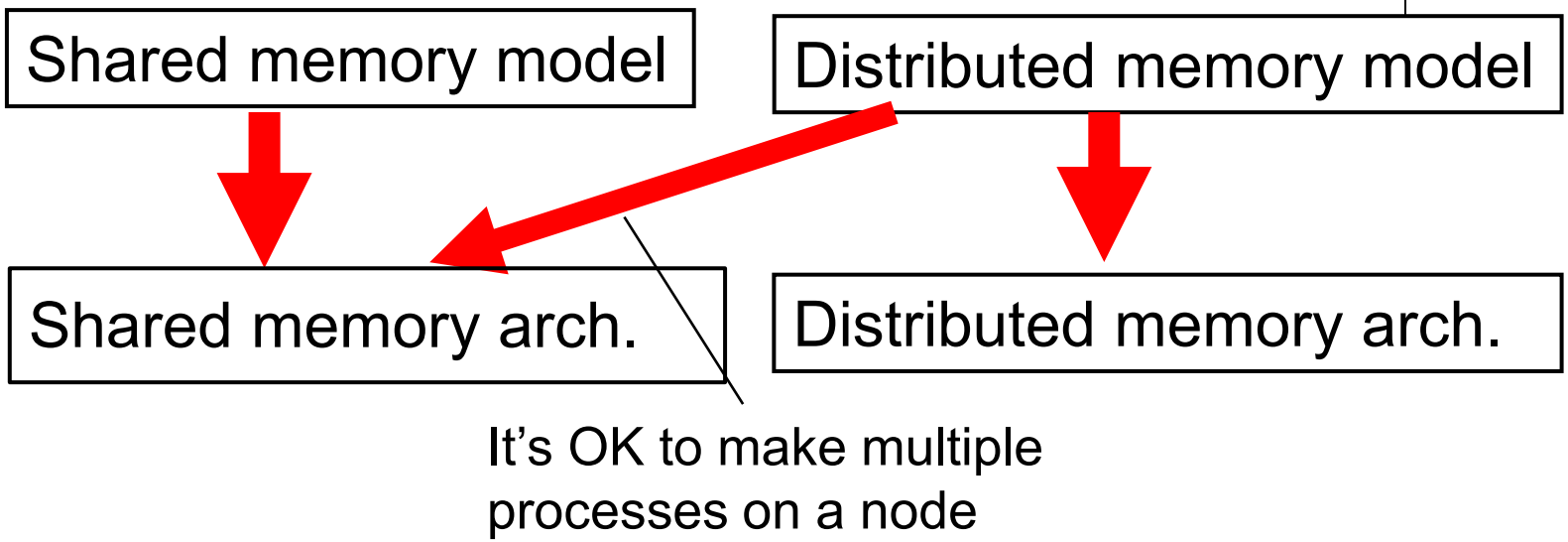


Programming without parallelsim

Threads have access to shared data
- OpenMP
- pthread
- Java thread…

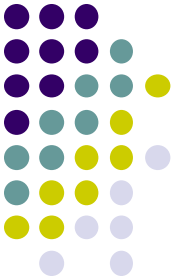
Need communication among processes
- MPI
- socket
- Hadoop, Spark…

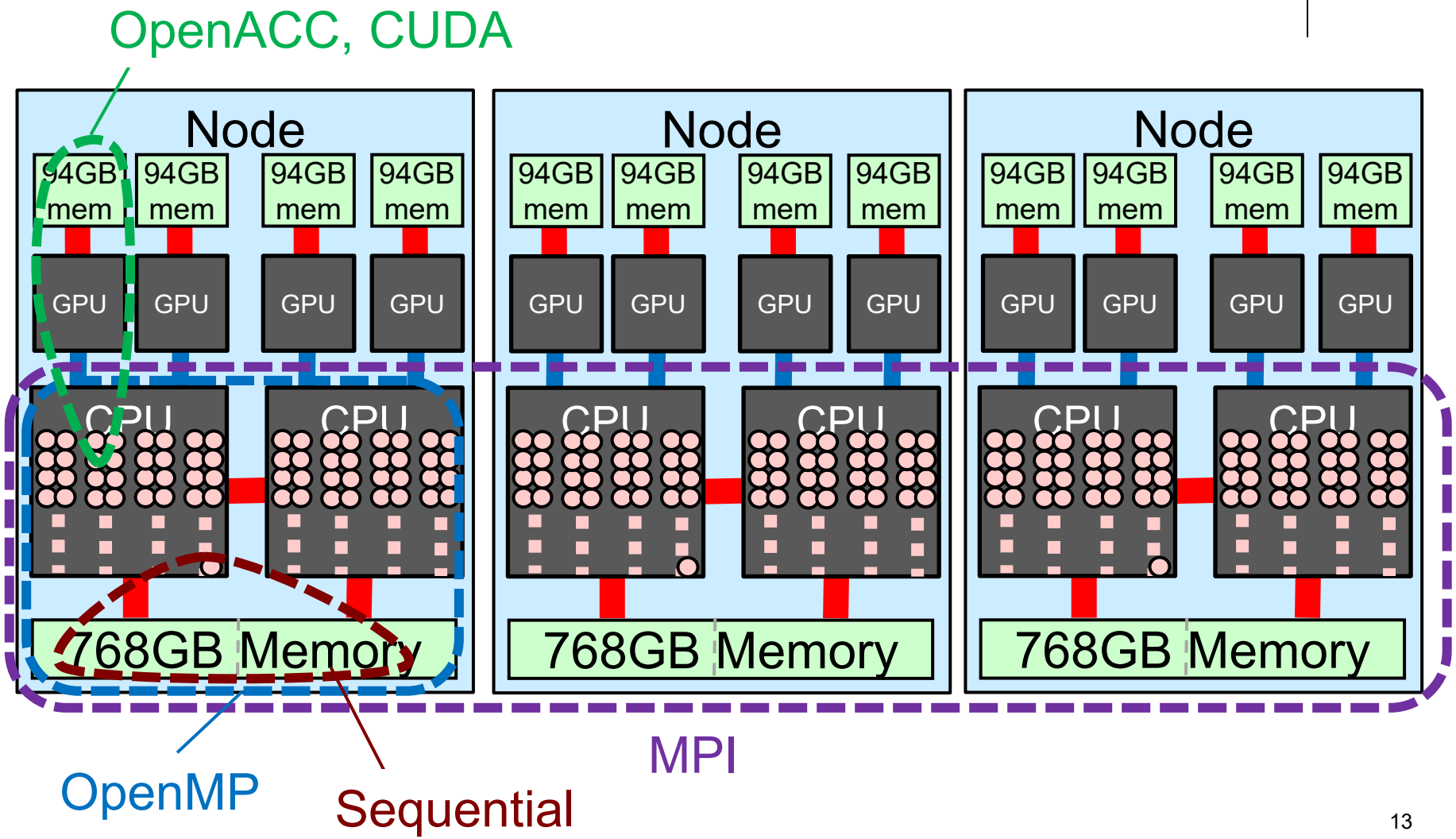
Programming Models on Architecture



- Shared memory model (OMP Part) can use only cores in a single node (up to 192 cores on TSUBAME4)
- Distributed memory model (MPI Part) supports large scale parallelism (192x240=46,080 cores on TSUBAME4)

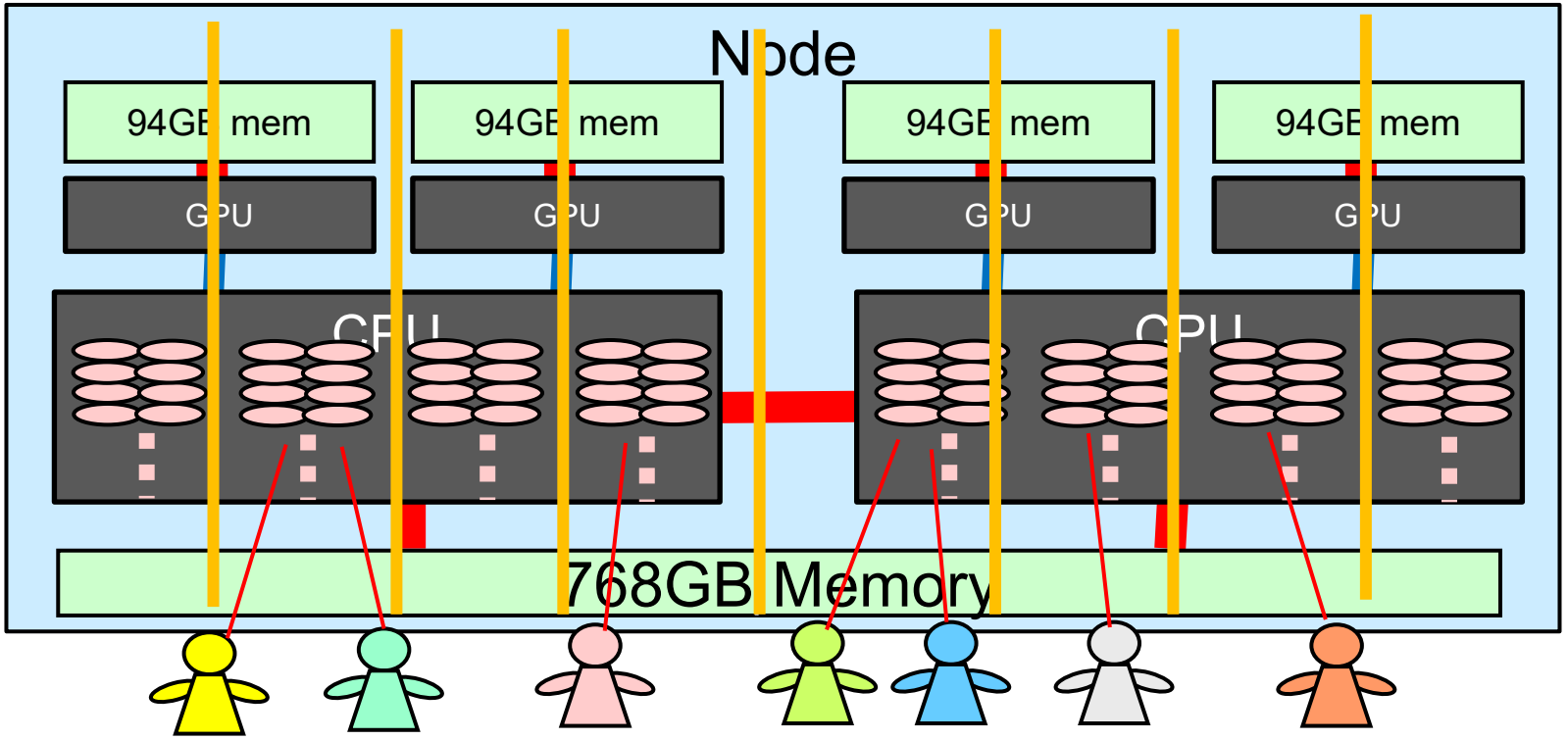
Parallel Programming Methods on TSUBAME





Standard route | Web-only route

TSUBAME Interactive Node



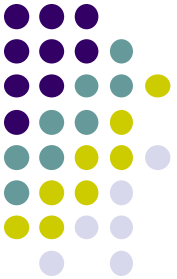
A node is partitioned into 8, each of which has

- 1/8 node = 24 CPU cores + 96GB memory
  + 0.5 GPU (7680cores+46GB mem)

A user can use one partition

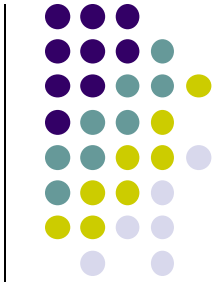
A partition may be shared by several users → you may suffer from slow down

Sample Programs in this Lecture



- Samples are at github (from 2025)
  - https://github.com/toshioendo/ppcomp-ex
  - Sub-directories: base/ omp/ acc/ cuda/ mpi/
- Base (non-parallel) sample programs are
  - base/mm: matrix multiplication
  - base/pi: approximation of pi (π)
  - base/diffusion: simple simulation of diffusion phenomena
  - base/fib: Fibonacci number
  - base/qsort: quick-sort sample
  - base/bsort: bitonic-sort sample

Make Copies of Sample In Case of mm



- Samples are in github site
  - Please make your copy on TSUBAME by the following commands

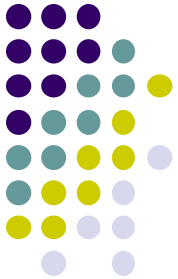
Example command:

- This make the copy on top of home directory. You can make sub-directory if you want

```
cd
git clone https://github.com/toshioendo/ppcomp-ex.git
```

➔ ppcomp-ex directory is created and all samples are stored. Try ls command

Executing Sample In Case of mm



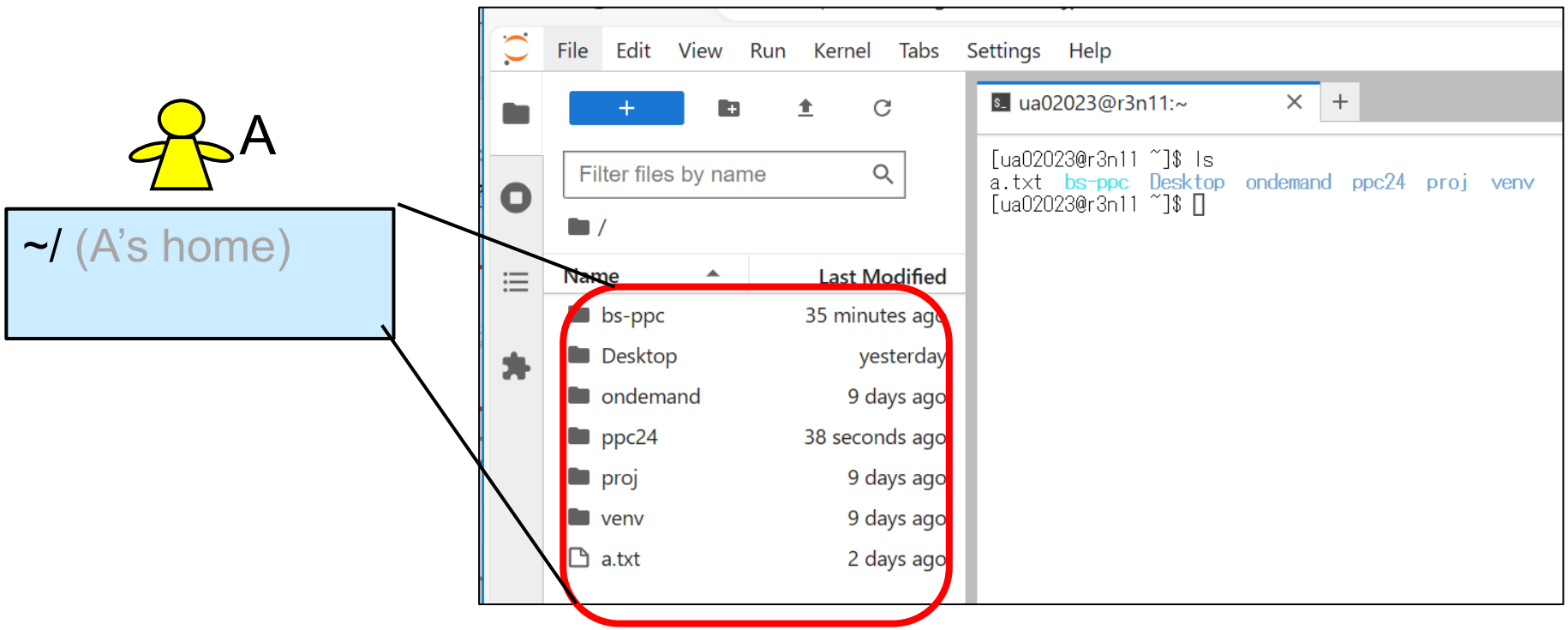
```
cd      [If you copied sample in sub-directory, go there]
cd ppcomp-ex/base/mm
ls
[you will see 3 files of mm.c, Makefile, job.sh]
make    [this creates an executable file “mm”]
./mm 2000 2000 2000
[this is the execution of mm sample]
```

- grey texts are comments; do not type

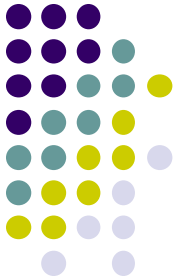
Web-only route

Notes in Web-Only Route

- In Jupyter lab screen, the folder tree shows your home (~/)



Using Sample Programs (3) Executing Samples



Before execution, please do cd and make properly for each sample

- mm

  ```
  ./mm 2000 2000 2000
  ```

  Options are matrix sizes *m,n,k*

- pi

  ```
  ./pi 100000000
  ```

  10^8

  Option is number of samples *n*

- diffusion

  ```
  ./diffusion 20
  ```

  Option is number of time steps *nt*

- fib

  ```
  ./fib 40
  ```

  Option is sequence index *n*

- qsort

  ```
  ./qsort 10000000
  ```

  10^7

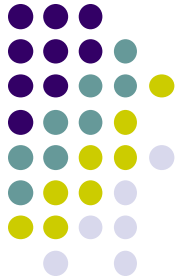
  Option is array length *n* to be sorted

- bsort

  ```
  ./bsort 10000000
  ```

  Option is array length *n* to be sorted

How Do We Edit C Programs?



There are several ways. The best way is up to you

1. Using editors on Linux

   [1a] vim

   [1b] emacs

   NOTE: emacs is not good on web route, since Ctrl+s may not work well

2. Using editors on your PC
   - You need to copy the file into PC, edit on your PC, and copy it to TSUBAME again
     - scp command on your PC, or WinSCP can be used
     - Drag&drop

   Web-only route

3. Using Jupyter's editor

   Web-only route

“mm” sample: Matrix Multiply

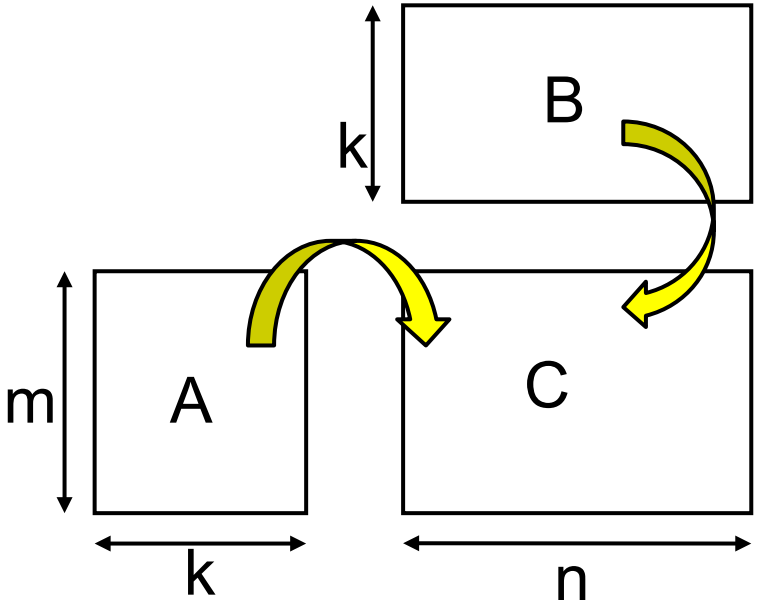
ppcomp-ex/base/mm/ directory

A: a (m × k) matrix

B: a (k × n) matrix

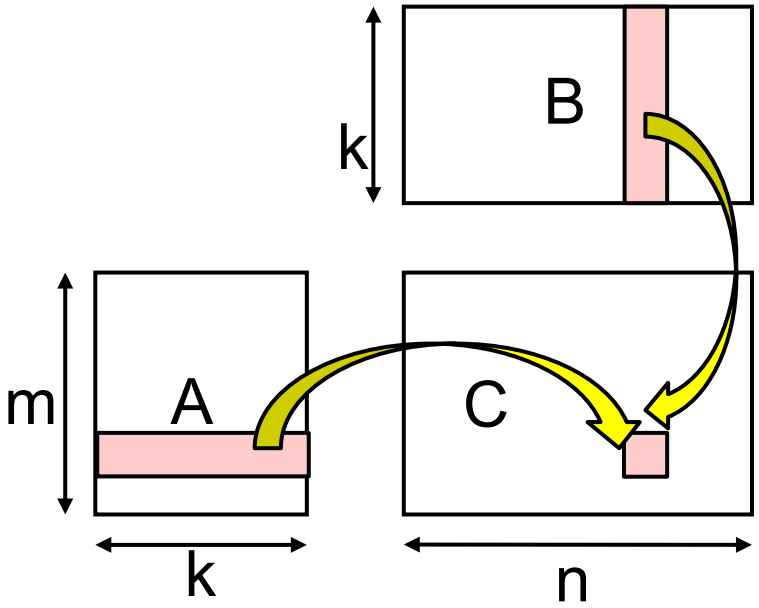
C: a (m × n) matrix

C ← A B



- This sample supports variable matrix sizes
- Execution: ./mm [m] [n] [k]
  - cf) ./mm 2000 2000 2000
  - cf) ./mm 1000 3000 1000

Matrix Multiply Algorithm (1)



$C_{i,j}$ is defined as the dot product of

- A's i-th row
- B's j-th column

The algorithm uses triply-nested loop

```
for (i = 0; i < m; i++) {          ←For each row in C
  for (j = 0; j < n; j++) {        ←For each column in C
    for (l = 0; l < k; l++) {      ←For dot product
      Ci,j += Ai,l * Bl,j;
    } } }
```

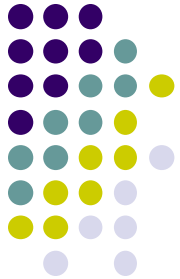
Matrix Multiply Algorithm (2)

```
for (i = 0; i < m; i++) {          ←For each row in C
  for (j = 0; j < n; j++) {        ←For each column in C
    for (l = 0; l < k; l++) {      ←For dot product
      Ci,j += Ai,l * Bl,j;
    } } }
```

- The innermost statement is executed for *mnk* times
- Compute Complexity : O(mnk)
  - Computation speed (Flops) is obtained as 2mnk/t, where *t* is execution time

The innermost statement includes 2 (floating point) calculations: *, +

Variable Length Arrays in (Classical) C Language



- double C[n]; raises an error. How do we do?
- void *malloc(size_t size);

  ⇒ Allocates a memory region of *size* bytes from “heap region”, and returns its head pointer
- When it becomes unnecessary, it should be discarded with free() function

A fixed length array

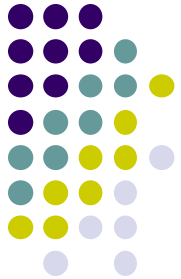
```
double C[5];

… C[i] can be used …
```

A variable length array

```
double *C;
C = (double *)malloc(sizeof(double)*n);

… C[i] can be used …

free(C);
```

array length

How We Do for Multiple Dimensional Arrays



double C[m][n]; raises an error. How do we do?

Not in a straightforward way. Instead, we do either of:

(1) Use a pointer of pointers

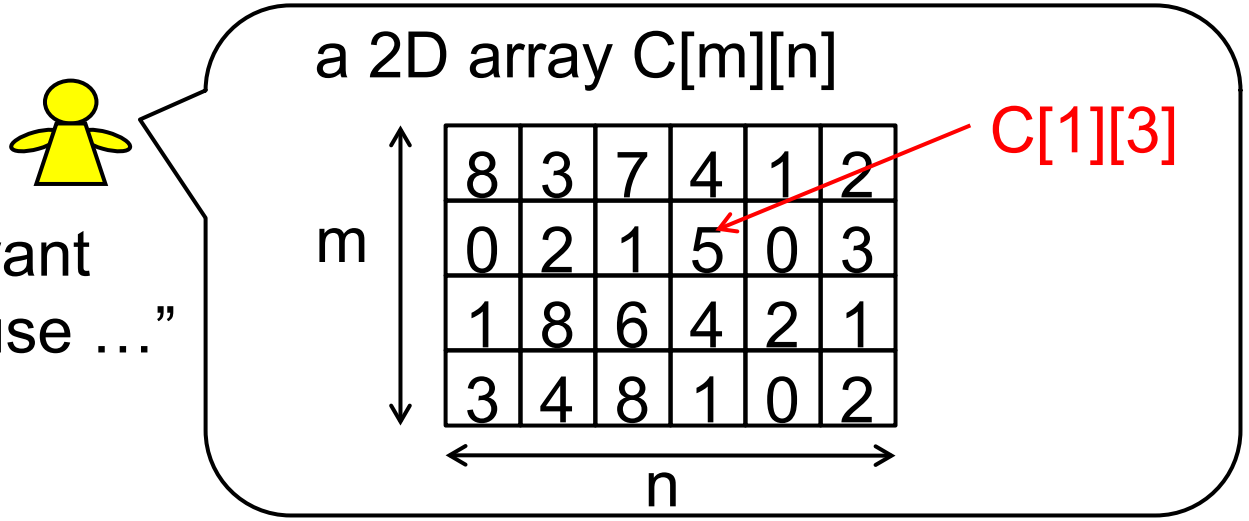
- We *malloc* m 1D arrays for every row (each has n length)
- We malloc 1D array of m length to store the above pointers

(2) Use a 1D array with length of $m \times n$

(mm sample uses this method)

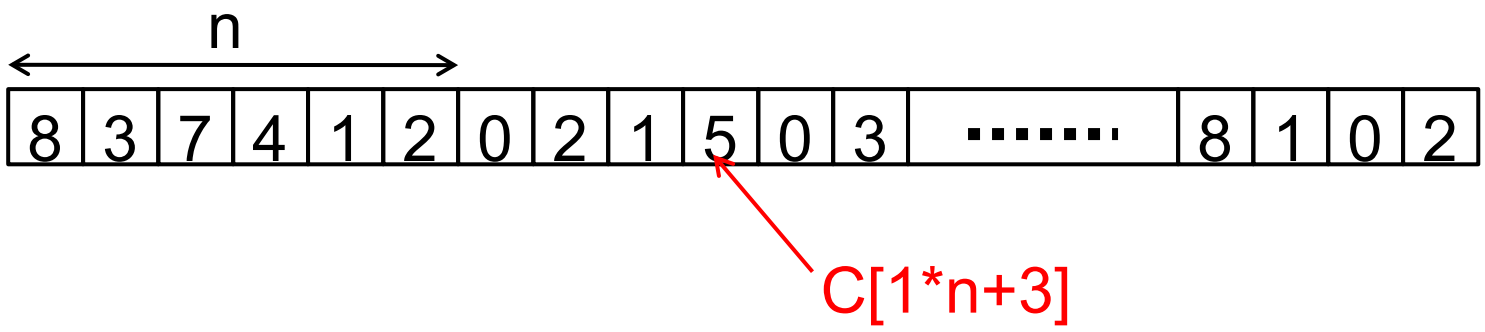
- To access an array element, we should use C[i*n+j] or C[i+j*m], instead of C[i][j]

Express a 2D array using a 1D array



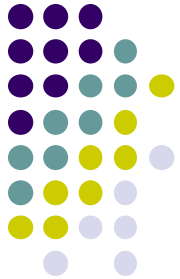
Expressions in C language (Example)

```
double *C;   C = malloc(sizeof(double)*m*n);
```



In this case, an element $C_{i,j}$ is C[i*n+j]

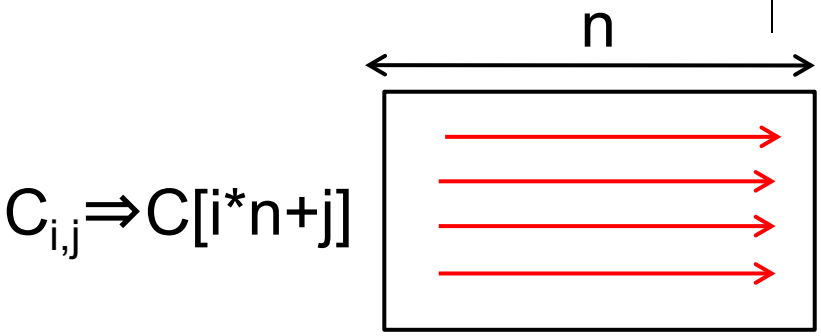
Two Data Formats



Row major format

- More natural for C programmers

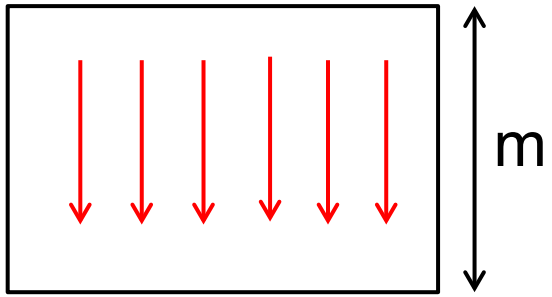
$C_{i,j} \Rightarrow C[i*n+j]$



Column major format

- BLAS library
- mm sample uses this

$C_{i,j} \Rightarrow C[i+j*m]$



- We have more choices for 3D, 4D… arrays

[Q] Does the format affect the execution speed?

Actual Codes in mm Sample

```
for (i = 0; i < m; i++) {
  for (j = 0; j < n; j++) {
    for (l = 0; l < k; l++) {
      Ci,j += Ai,l * Bl,j;
    } } }
```

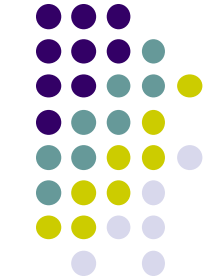
IJL order

```
for (j = 0; j < n; j++) {
  for (l = 0; l < k; l++) {
    double blj = B[l+j*k];
    for (i = 0; i < m; i++) {
      double ail = A[i+l*m];
      C[i+j*m] += ail*blj;
    }}}
```

Change (2):
JLI order is used
(a bit faster)

Change (1):
Matrix elements as
1D array elements

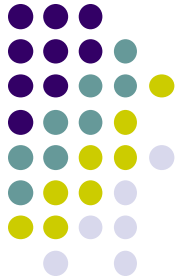
Time Measurement in Samples



- gettimeofday() function is used
  - It provides wall-clock time, not CPU time
  - Time resolution is better than clock()
  - In some programs, time_diff_sec() function is defined

```
#include <stdio.h>
#include <sys/time.h>
        :
{
    struct timeval st, et;
    double sec;
    gettimeofday(&st, NULL); /* Starting time */
    ･･･Part for measurement ･･･
    gettimeofday(&et, NULL); /* Finishing time */
    sec = (et.tv_sec-st.tv_sec)+
          (et.tv_usec-st.tv_usec)/1000000.0;
    /* us is difference between st & et in microseconds */
}
```

If You Have Not Done This Yet



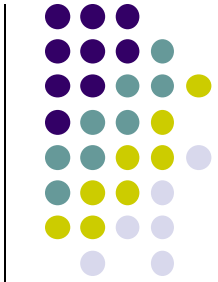
Please do the followings as soon as possible

- Please make your account on TSUBAME
- Please inform the account name via Science Tokyo LMS
  - Please see “Class #1: Today’s homework”

Then we will invite you to the TSUBAME group, please click URL and accept the invitation

その後、TSUBAMEグループへの招待を送ります。メール中のURLをクリックして参加承諾してください

Next Class: Introduction to OpenMP



- Shared memory parallel programming API
- Extensions to C/C++, Fortran
- Includes directives& library functions
  - Directives: #pragma omp ~~

```
  int i;
#pragma omp parallel for
  for (i = 0; i < 100; i++) {
        a[i] = b[i]+c[i];
  }
```