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

Part 0: Introduction
No 2: Parallel Architecture & Sample Programs

Apr 11, 2024

Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

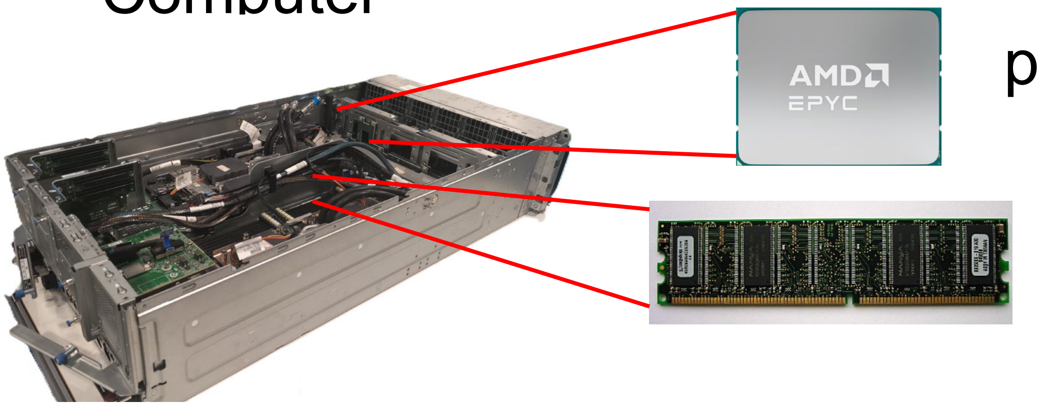
Overview of This Course

- Part 0: Introduction
  - 2 classes ← We are here (2/2)
- Part 1: OpenMP for shared memory programming
  - 4 classes
- Part 2: GPU programming
  - OpenACC and CUDA
  - 4 classes
- Part 3: MPI for distributed memory programming
  - 3 or 4 classes

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



Computer

processor

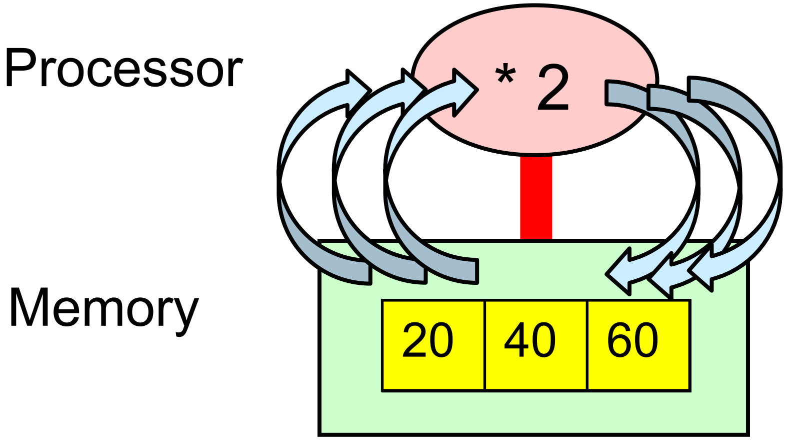
memory

Software Runs on Hardware

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

Note: This is so simplified discussion

Hardware



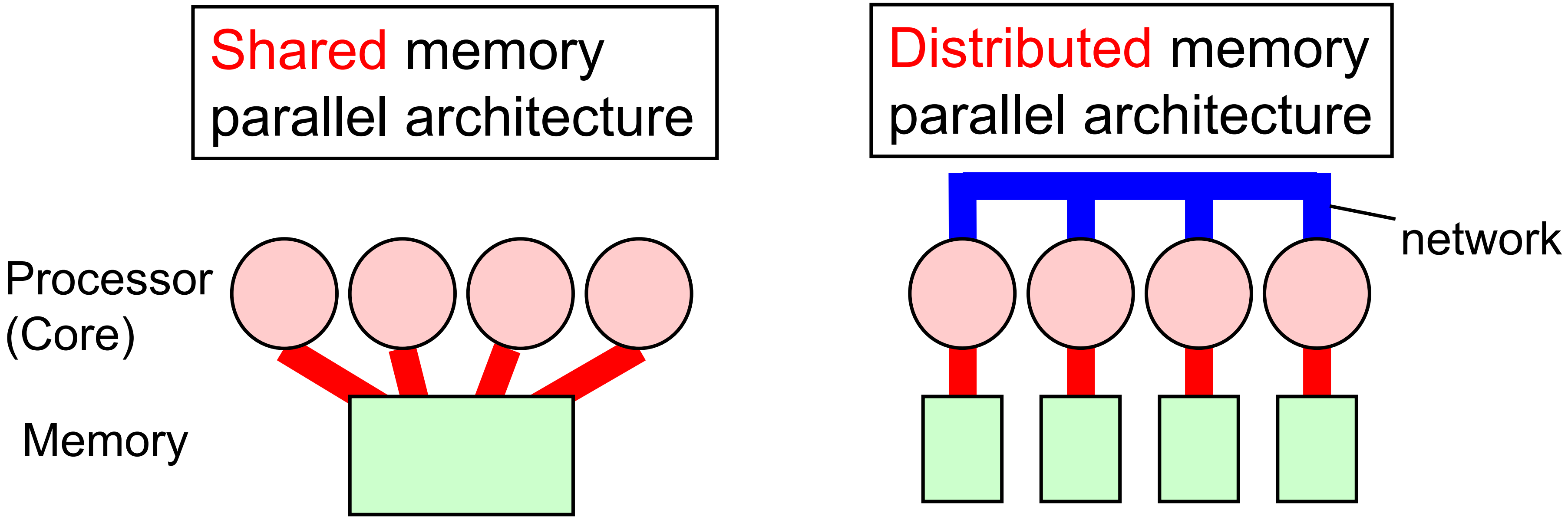
Software Example

```
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

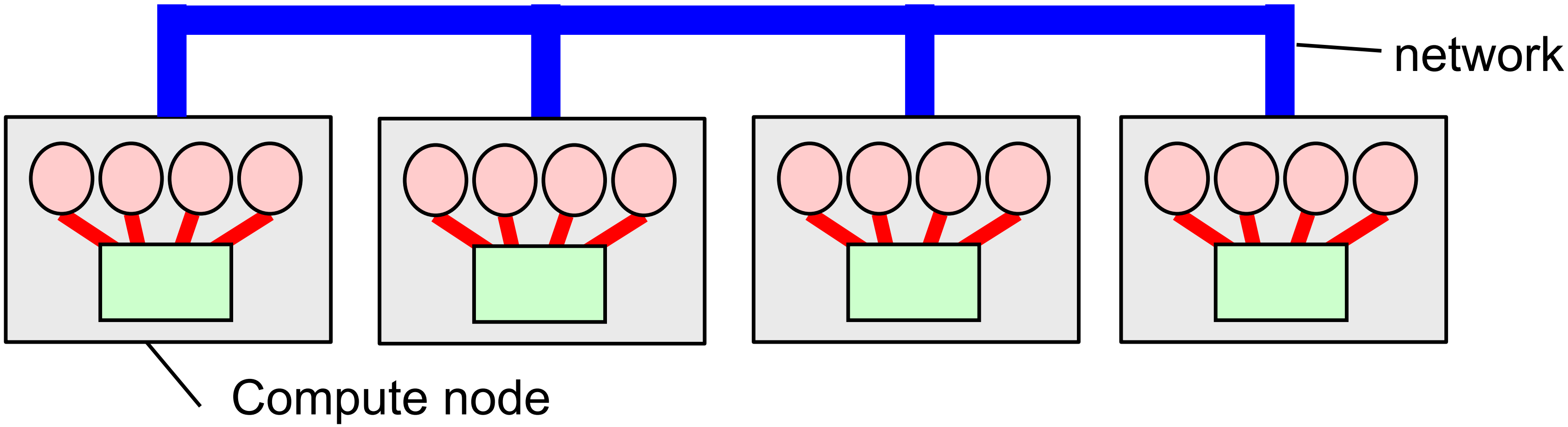
- Different programming methods are used for different architecture

Modern SCs use Both!

Modern SCs are combination of “shared” and “distributed”

“shared memory” in a node

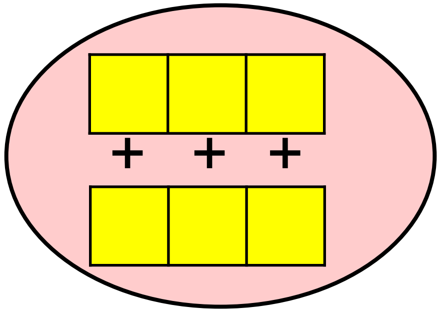
“distributed memory” among nodes, connected by network



※ Moreover, each processor (core) may have *SIMD parallelism* , such as SSE, AVX…

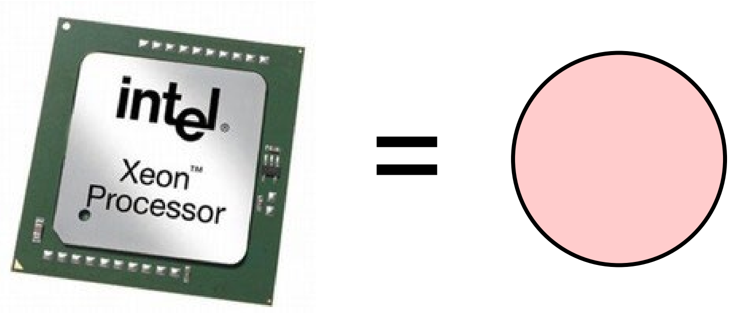
A processor (core) can do several computations at once

SIMD is out of scope of this class

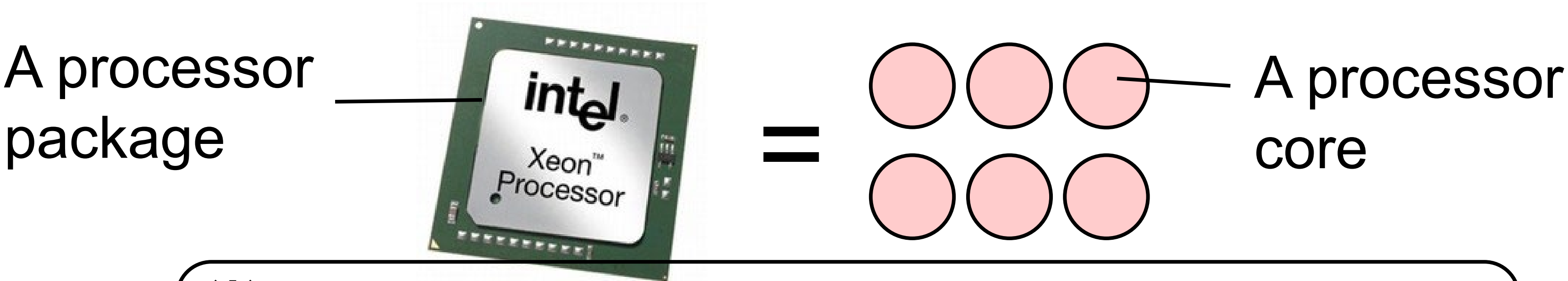


(Confusing) Terminology

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



- Since around 2005, "multicore processor" became popular

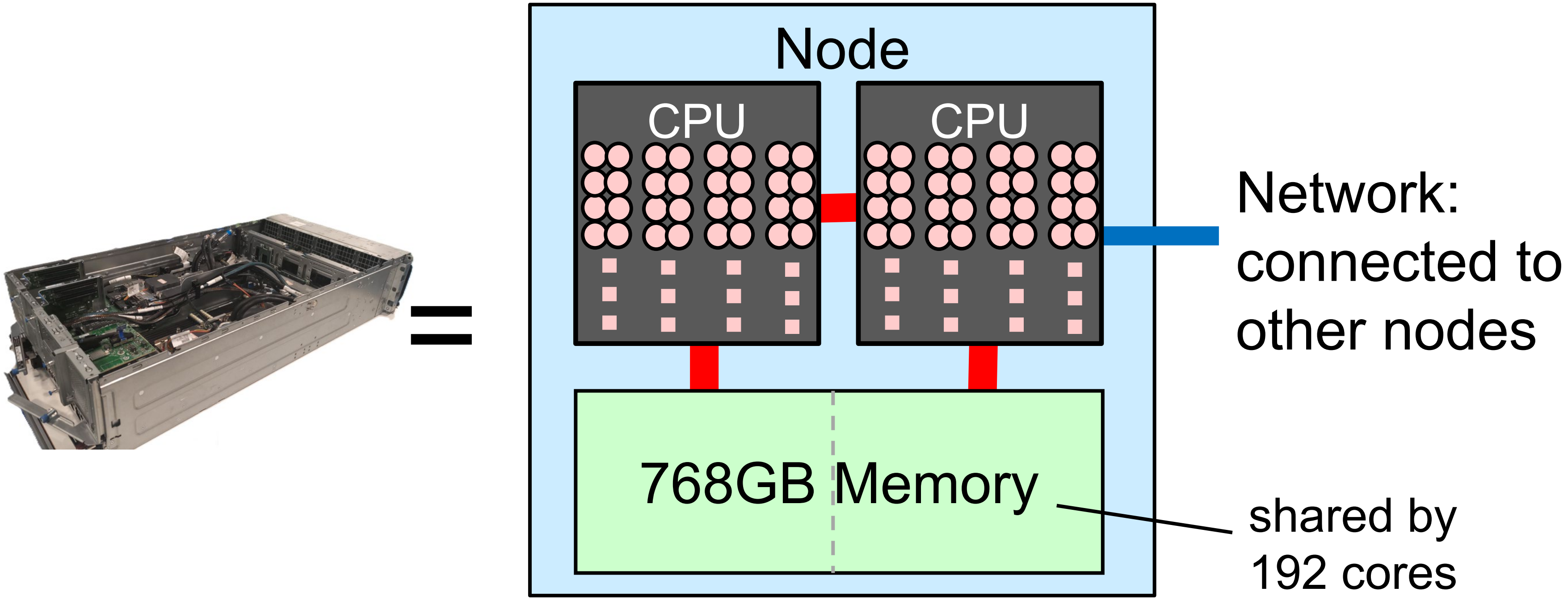


※ *Hyperthreading* makes discussion more complex:
1 physical core = 2 logical cores
In this slide, "core" basically means physical core

A TSUBAME4 Node (1)

- 2 processor packages (CPU) × 96 cores

  ➔ A TSUBAME4 node has 192 cores



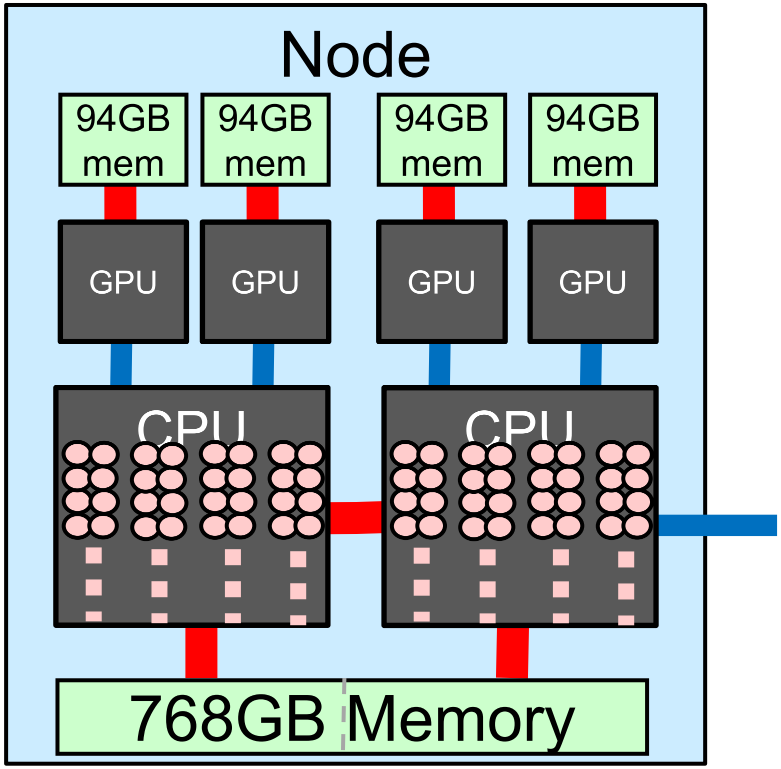
Network: connected to other nodes

shared by 192 cores

- GPUs are (still) omitted in this figure

A TSUBAME3 Node (2)

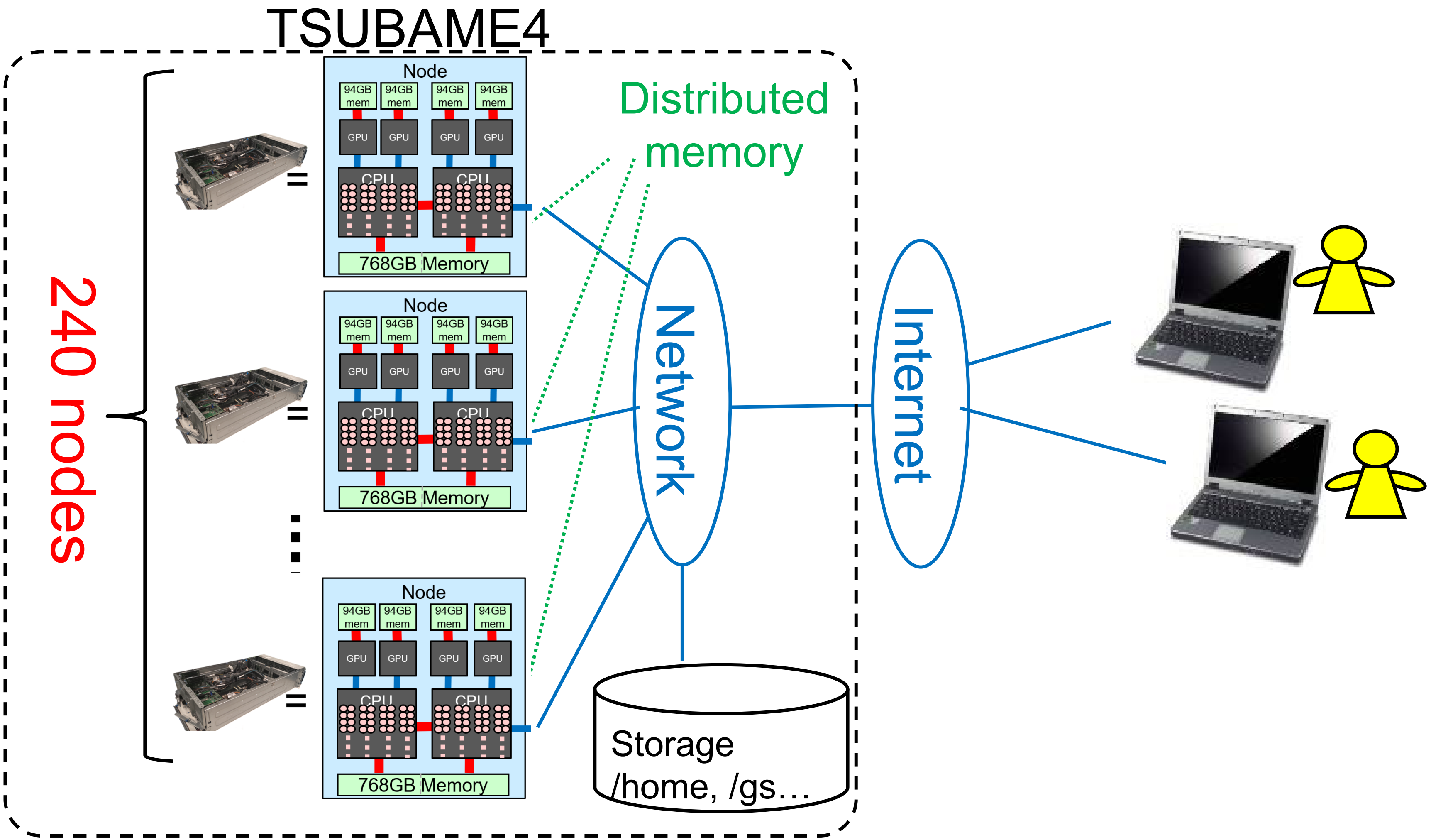
- A node has 2 CPUs + 4 GPUs
  - Each GPU (H100) has 132SMs = 16,896 cores



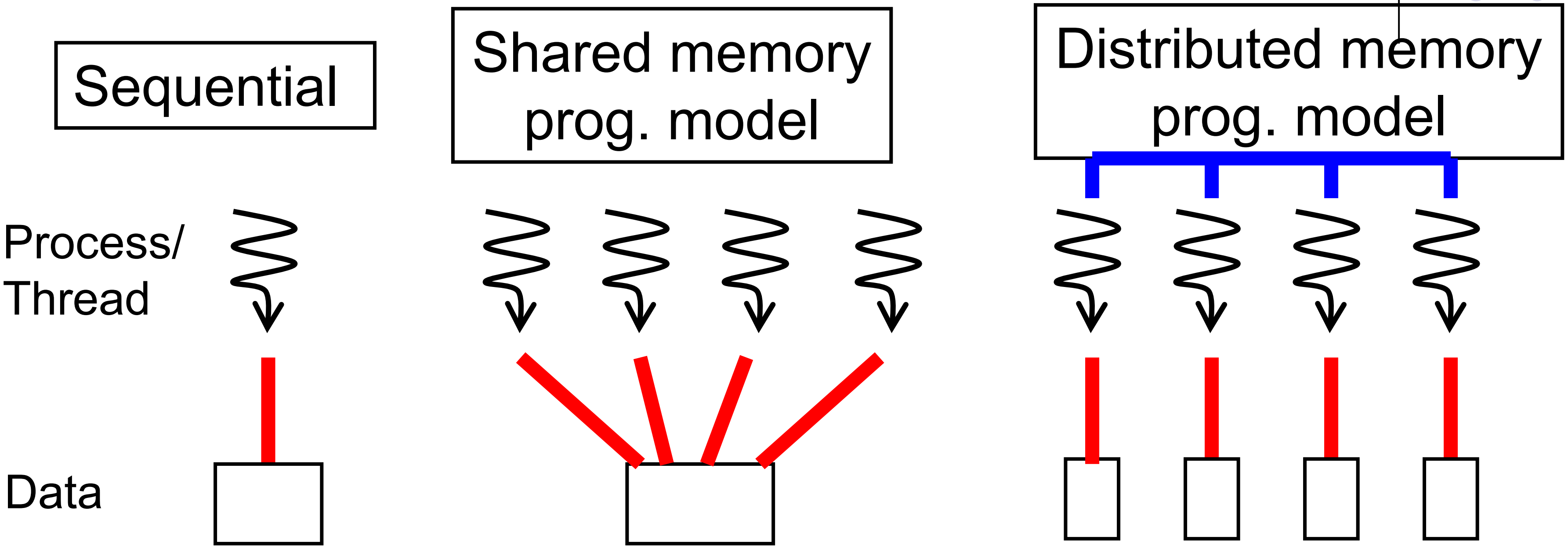
Network: connected to other nodes

TSUBAME4 System

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



Classification of Parallel Programming Models



Programming without parallelsim

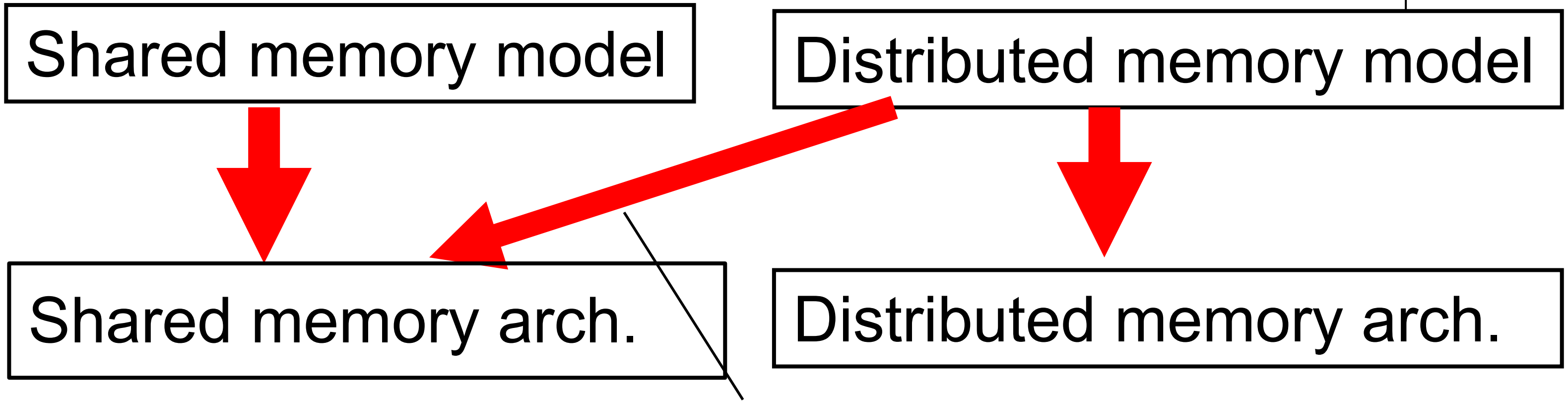
Threads have access to shared data

- OpenMP
- pthread
- Java thread…

Need communication among processes

- MPI
- socket
- Hadoop, Spark…

Programming Models on Architecture



Shared memory model

Distributed memory model

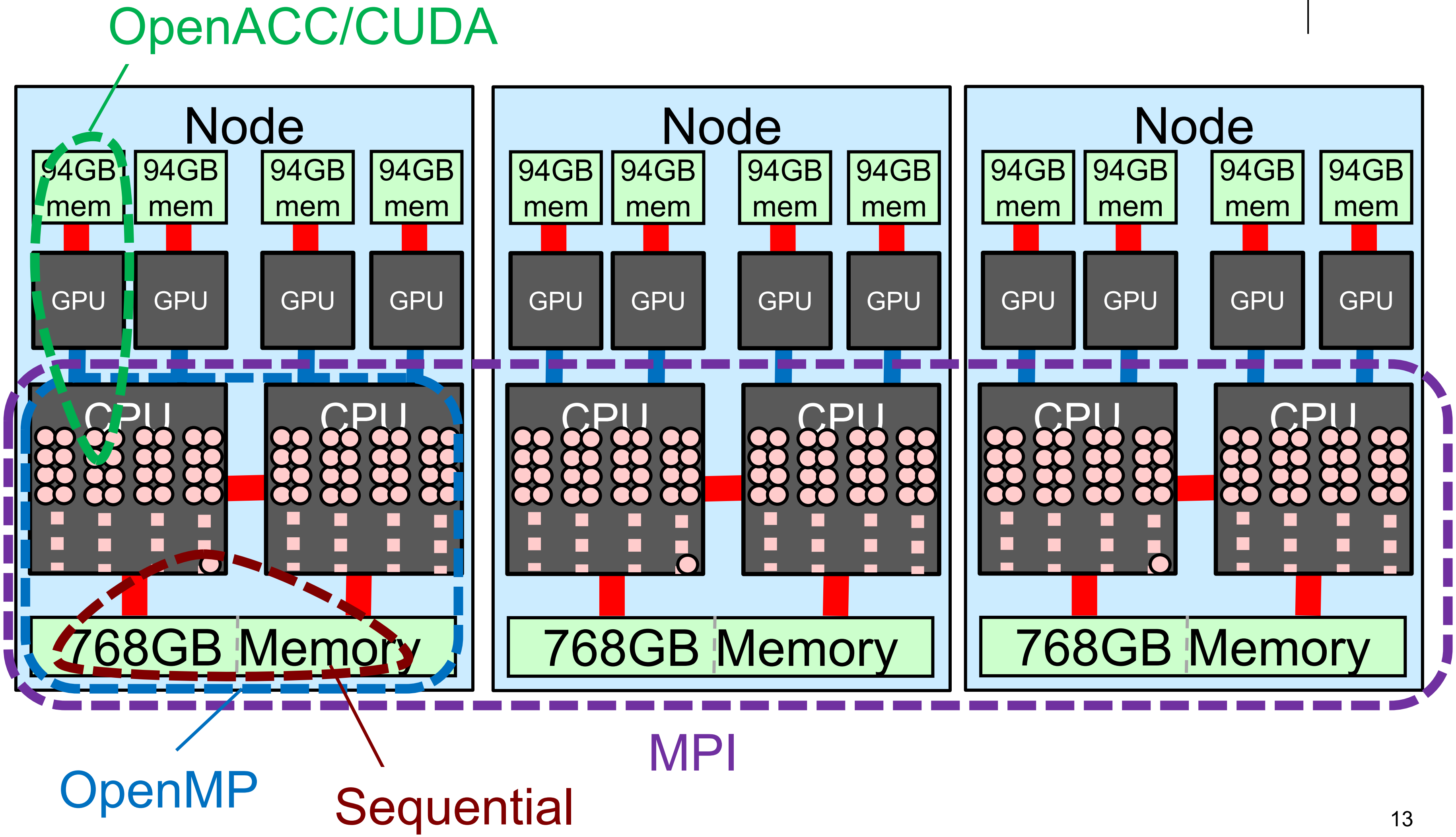
Shared memory arch.

Distributed memory arch.

It’s OK to make multiple processes on a node

- Shared memory model (Part 1) can use only cores in a single node (up to 192 cores on TSUBAME4)
- Distributed memory model (Part 3) supports large scale parallelism (192x240=46,080 cores on TSUBAME4)

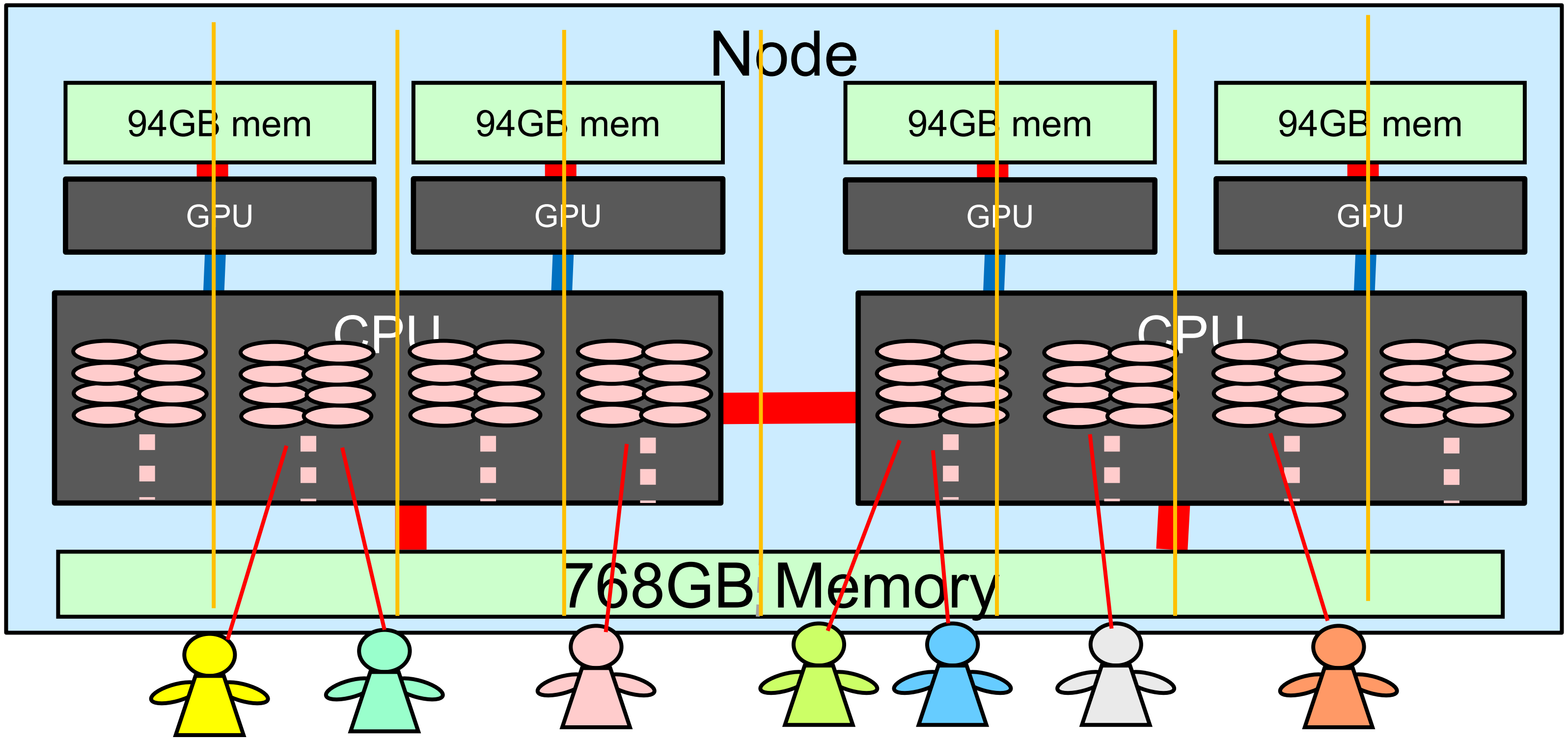
Parallel Programming Methods on TSUBAME



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 /gs/bs/tga-ppcomp/24/ directory
  - You have to a member of tga-ppcomp group
    - If “ls /gs/bs/tga-ppcomp/24” works well, you are a member
  - There are sub-directories per sample
- Sequential (non-parallel) sample programs are
  - mm: matrix multiplication
  - pi: approximation of pi (π)
  - diffusion: simple simulation of diffusion phenomena
  - fib: Fibonacci number
  - sort: quick-sort sample

Make Copies of Sample In Case of mm

- Samples in /gs/… are “*read-only*”, so make copies of samples into somewhere in your home directory
- Compile programs inside your home directory

Example:

Create a directory ~/ppc24 (name is arbitrary, do once)

```
mkdir ~/ppc24
```

Copy the “mm” sample as ~/ppc24/mm

```
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/mm .
```

don’t forget space & dot

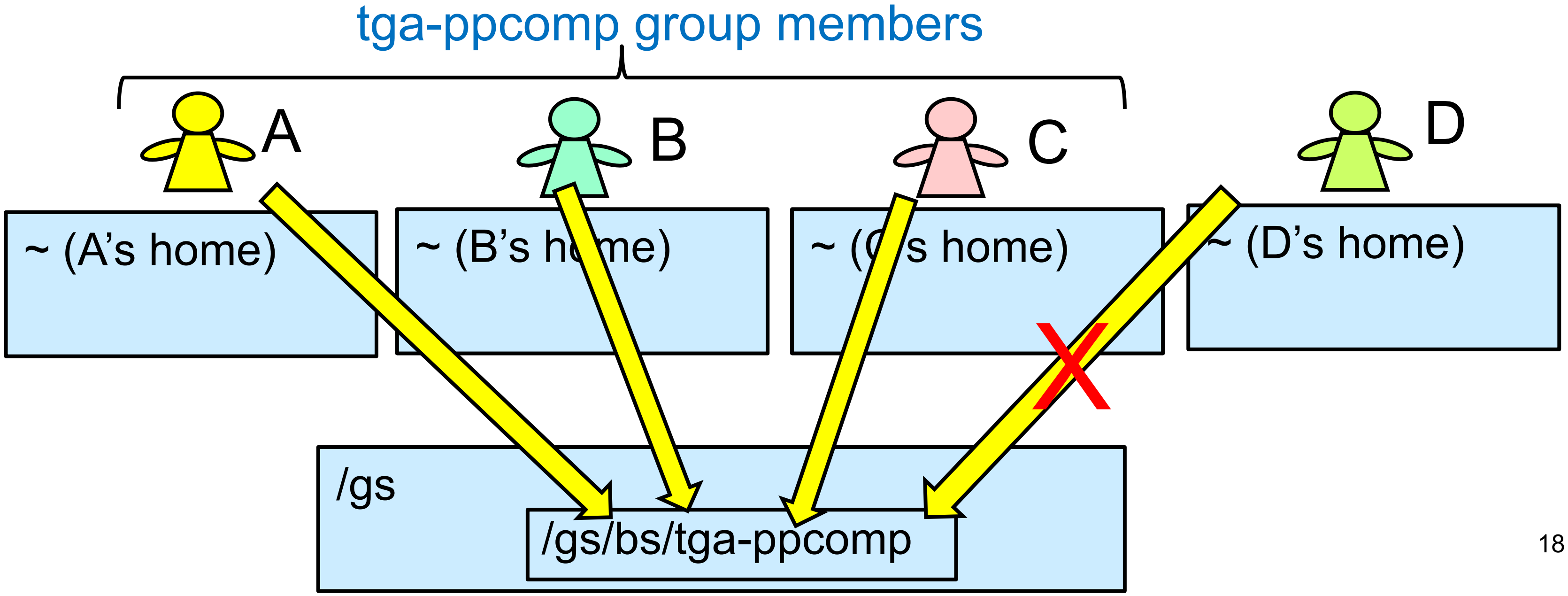
Executing Sample
In Case of mm

```
cd ~/ppc24
[make sure that you have copied mm directory]
cd 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

Disk Storage of TSUBAME

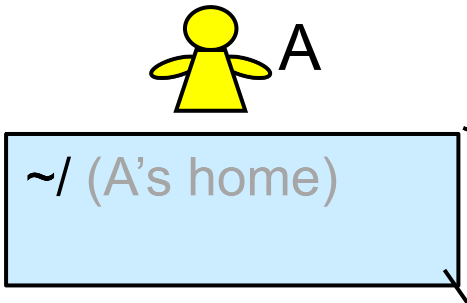
- ~/… (home directory) is assigned to each user
  - Actually, ~ is an alias to /home/??/uX00000 (username)
- /gs/… are shared by multiple users
  - /gs/bs/tga-ppcomp/… is shared by tga-ppcomp members
- Above storage can be accessed from all computing nodes and log-in nodes in TSUBAME

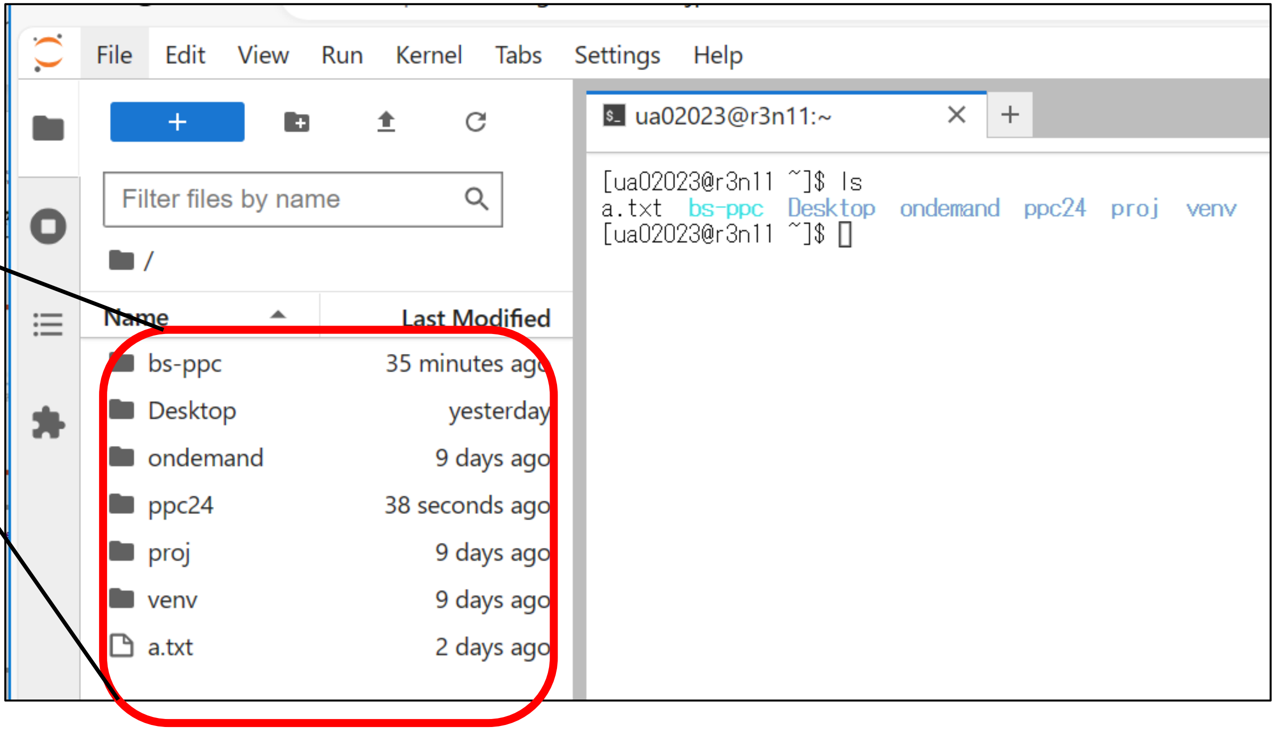


Web-only route

Notes in Web-Only Route

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





Name	Last Modified
bs-ppc	35 minutes ago
Desktop	yesterday
ondemand	9 days ago
ppc24	38 seconds ago
proj	9 days ago
venv	9 days ago
a.txt	2 days ago

```
[ua02023@r3n11 ~]$ ls
a.txt  bs-ppc  Desktop  ondemand  ppc24  proj  venv
[ua02023@r3n11 ~]$
```

Make Copies of Other Samples

Do once for each sample.

"pi" sample

```
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/pi .
```

"diffusion" sample

```
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/diffusion .
```

"fib" sample

```
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/fib .
```

"sort" sample

```
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/sort .
```

Using Sample Programs (3) Executing Samples

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

- mm

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

Options are matrix sizes *m,n,k*

- pi

10^8

```
./pi 100000000
```

Option is number of samples *n*

- diffusion

```
./diffusion 20
```

Option is number of time steps *nt*

- fib

```
./fib 40
```

Option is sequence index *n*

- sort

10^7

```
./sort 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 does 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

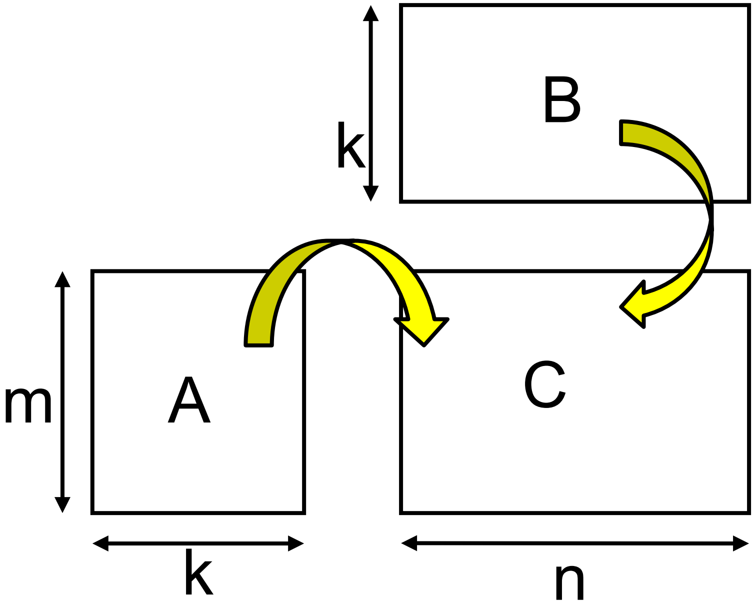
Original version is at /gs/bs/tga-ppcomp/24/mm/

A: a (m × k) matrix

B: a (k × n) matrix

C: a (m × n) matrix

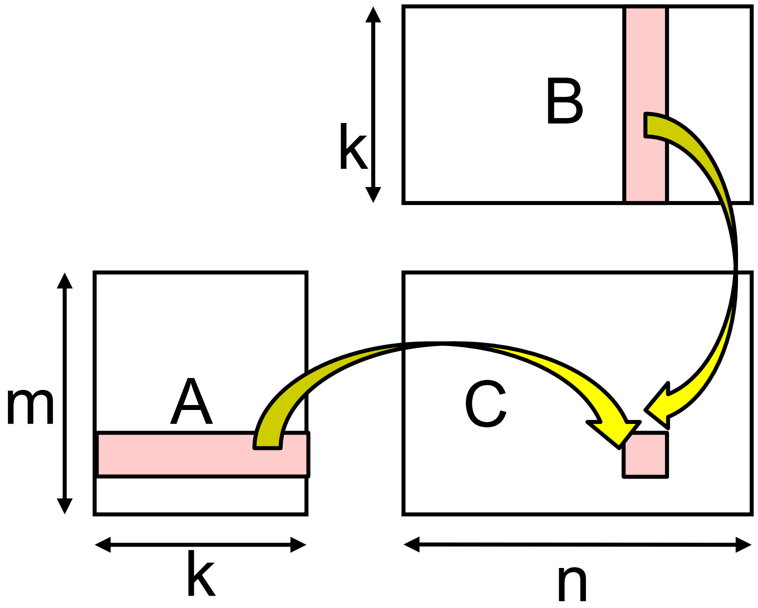
$C \leftarrow A\ B$



- 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

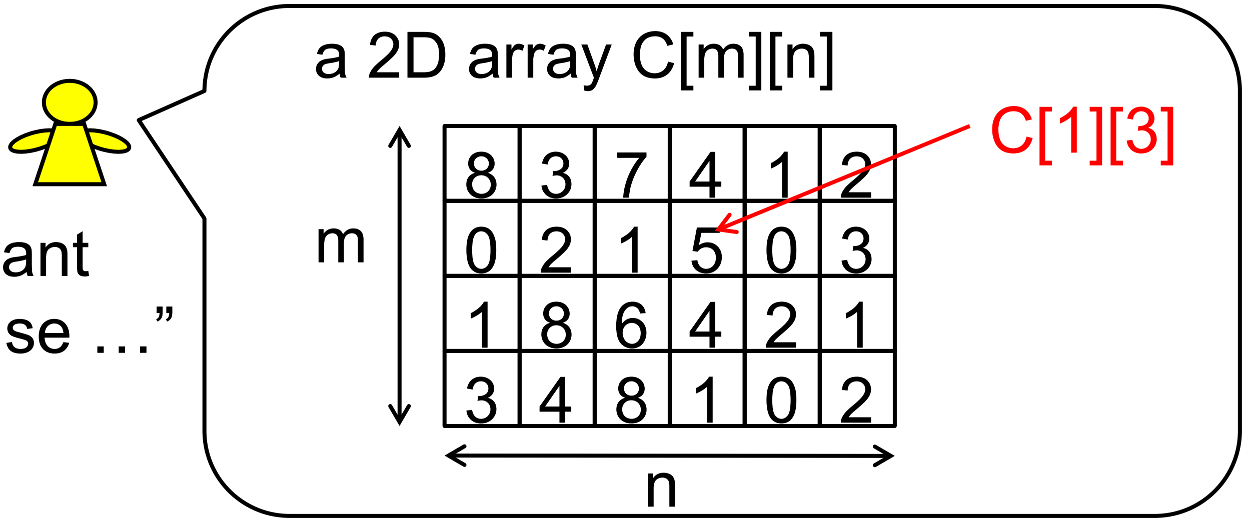
- We *malloc* m 1D arrays for every row (each has n length)
- We malloc 1D array of m length to store the above pointers

(2) Use a 1D array with length of m × 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



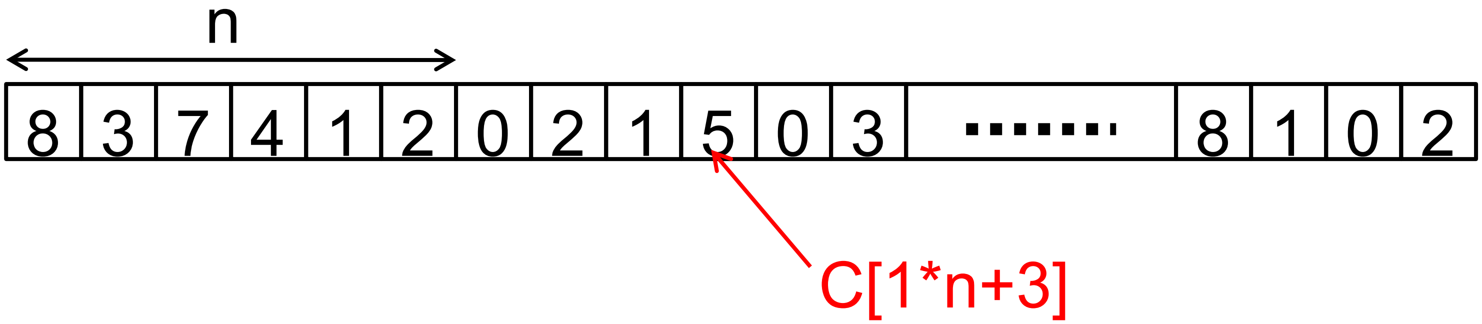
“I want to use …”

a 2D array C[m][n]

8	3	7	4	1	2
0	2	1	5	0	3
1	8	6	4	2	1
3	4	8	1	0	2

Expressions in C language (Example)

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



8	3	7	4	1	2	0	2	1	5	0	3	…….	8	1	0	2

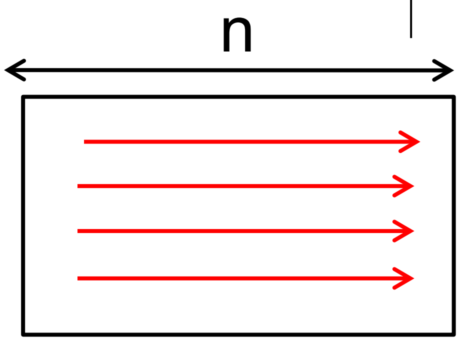
In this case, an element $C_{i,j}$ is C[i*n+j]

Two Data Formats

<u>Row major format</u>

- More natural for C programmers

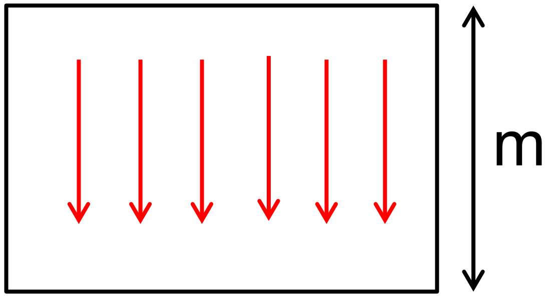
$C_{i,j} \Rightarrow C[i*n+j]$



<u>Column major format</u>

- 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()
  - Note: newer clock_gettime() is more recommended

```
#include <stdio.h>
#include <sys/time.h>
      :
{
   struct timeval st, et;
   long us;
   gettimeofday(&st, NULL); /* Starting time */
   ･･･Part for measurement ･･･
   gettimeofday(&et, NULL); /* Finishing time */
   us = (et.tv_sec-st.tv_sec)*1000000+
        (et.tv_usec-st.tv_usec);
   /* 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 send an e-mail to ppcomp@el.gsic.titech.ac.jp

Subject: TSUBAME3 ppcomp account
To: ppcomp@el.gsic.titech.ac.jp

Department name:
School year:
Name:
Your TSUBAME account name:

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

If You Have Not Done This Yet

Please do 1&2 as soon as possible. We accept the mail later.
1と2をできるだけ早期に行ってください。それより後でも受け付けます

1. (If you are new to TSUBAME) please make your account on TSUBAME
   (まだ作ったことがなければ) TSUBAMEアカウントを作成してください
2. Please send an e-mail after account creation
   作成してから下記のようなe-mailを送ってください

To: ppcomp@el.gsic.titech.ac.jp
Subject: TSUBAME4 ppcomp account

Department name:
Student ID:
Name:
TSUBAME account name:

If You Have Not Done This Yet (cont)

3. You will receive an invitation e-mail to tga-ppcomp TSUBAME group. Please read it and accept the invitation.

tga-ppcomp TSUBAMEグループへの招待e-mailが届くはずです。指示に従って招待を受けてください。

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];
  }
```