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

Part 0: Introduction

No 2: Overview of Course

Apr 13, 2023

Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

Overview of This Course

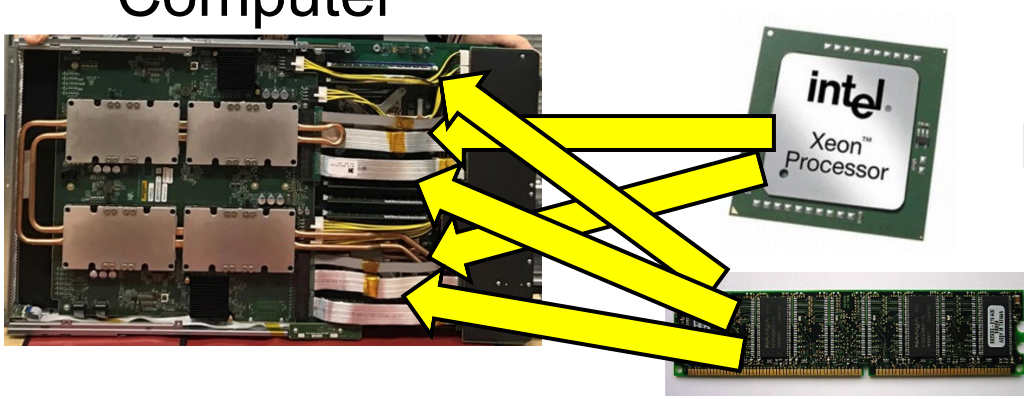
- Part 0: Introduction
  - 2 classes ← We are here (2/2)
- Part 1: OpenMP for shared memory programming
  - 4 classes
- Part 2: GPU programming
  - OpenACC and CUDA
  - 4 classes
- Part 3: MPI for distributed memory programming
  - 3 or 4 classes

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

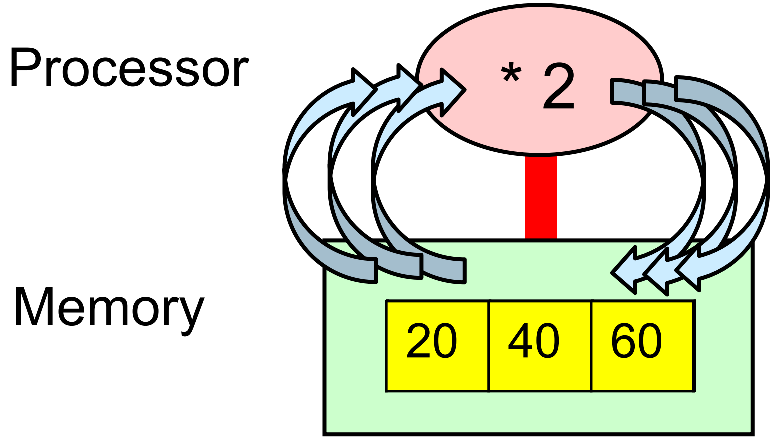
memory

Software Runs on Hardware

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

Note: This is so simplified discussion

Hardware



Software Example

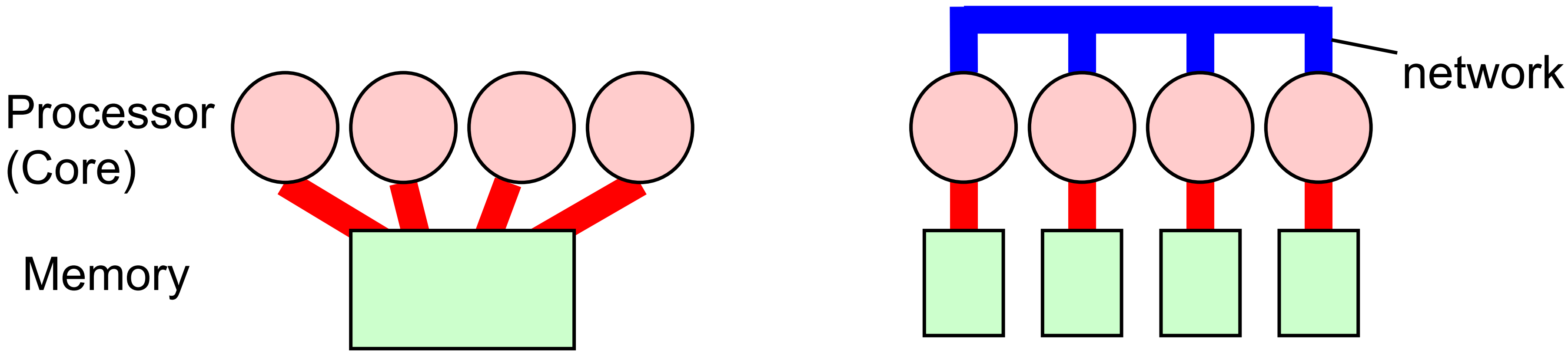
```
int a[3] = {10, 20, 30};
int i;

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

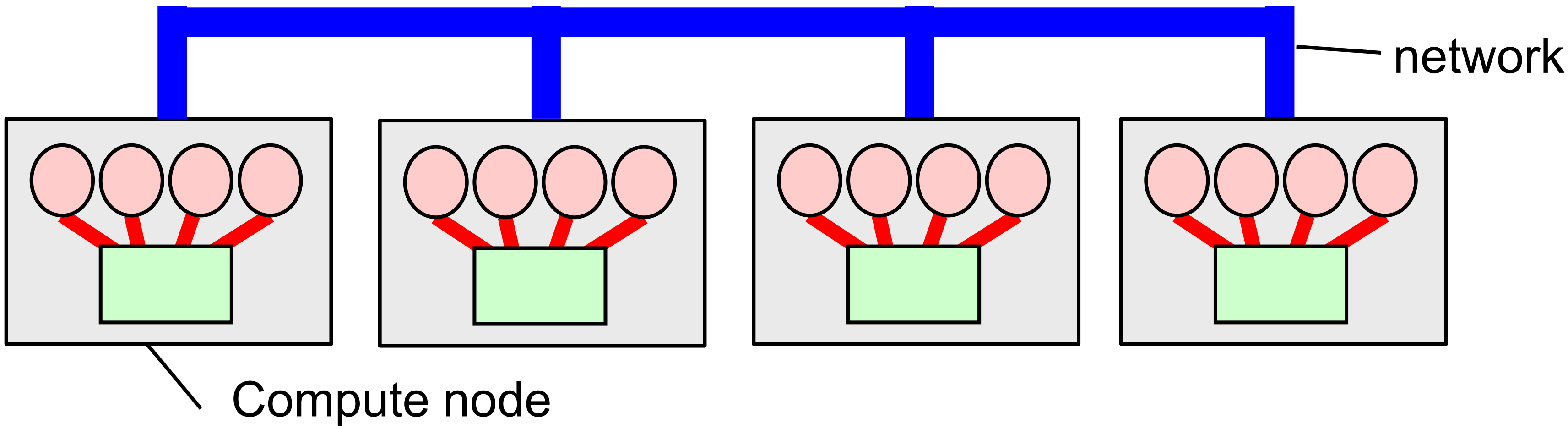
Distributed memory parallel architecture



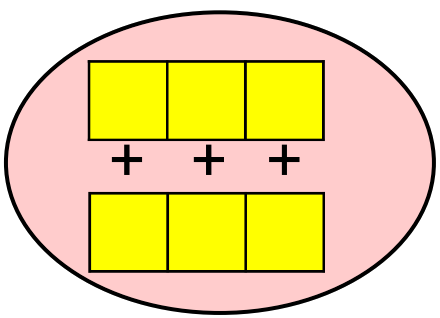
- Different programming methods are used for different architecture

Modern SCs use Both!

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

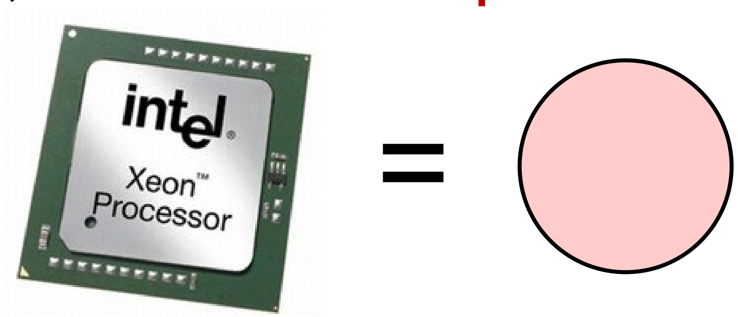


※ Moreover, each processor (core) may have *SIMD parallelism* , such as SSE, AVX…
A processor (core) can do several computations at once
SIMD is out of scope of this class

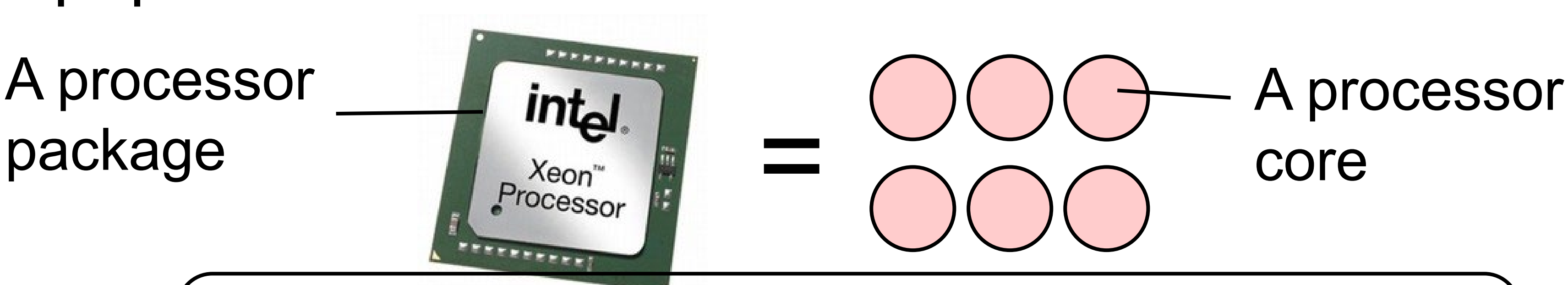


(Confusing) Terminology

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



- Since around 2005, "multicore processor" became popular

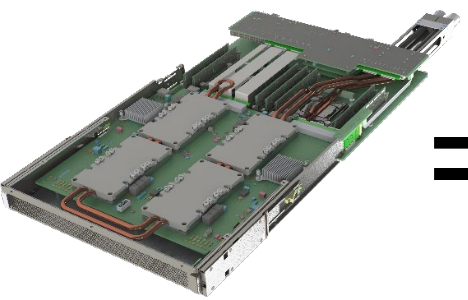


※ *Hyperthreading* makes discussion more complex:
1 physical core = 2 logical cores
In this slide, "core" basically means physical core

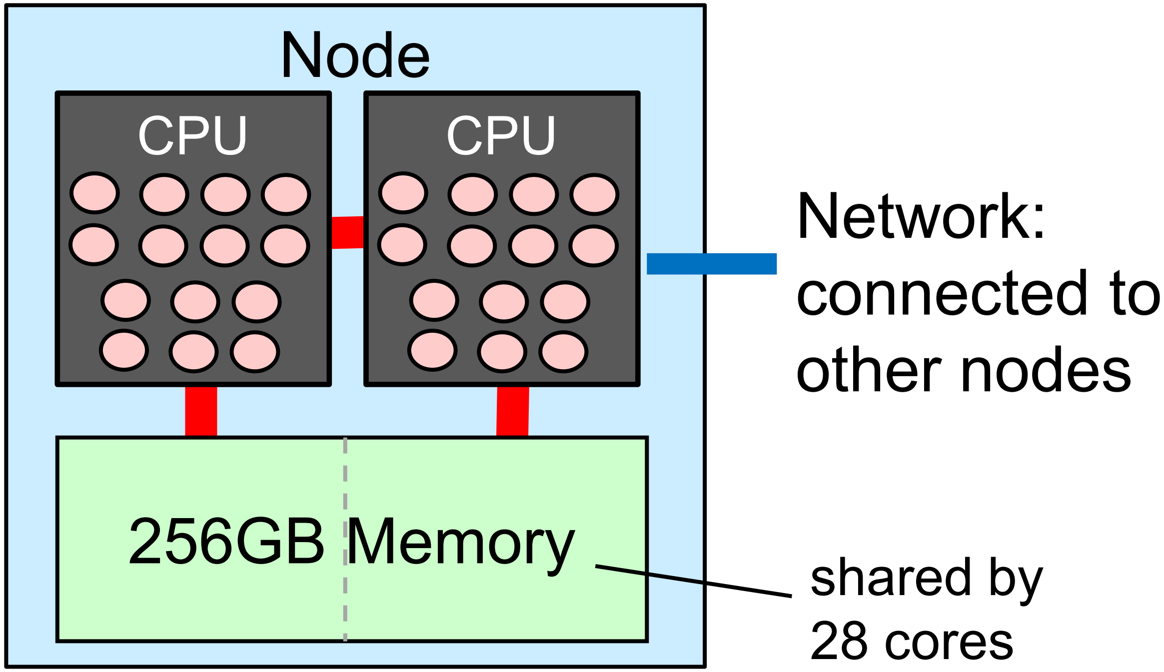
A TSUBAME3 Node (1)

- 2 processor packages (CPU) × 14 cores

  ➔ A TSUBAME3 node has 28 cores



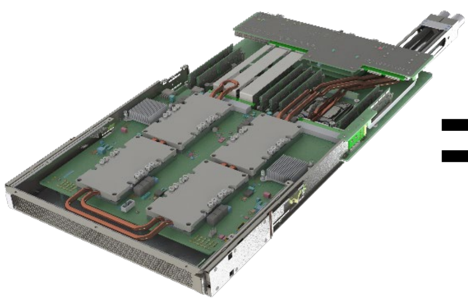
=



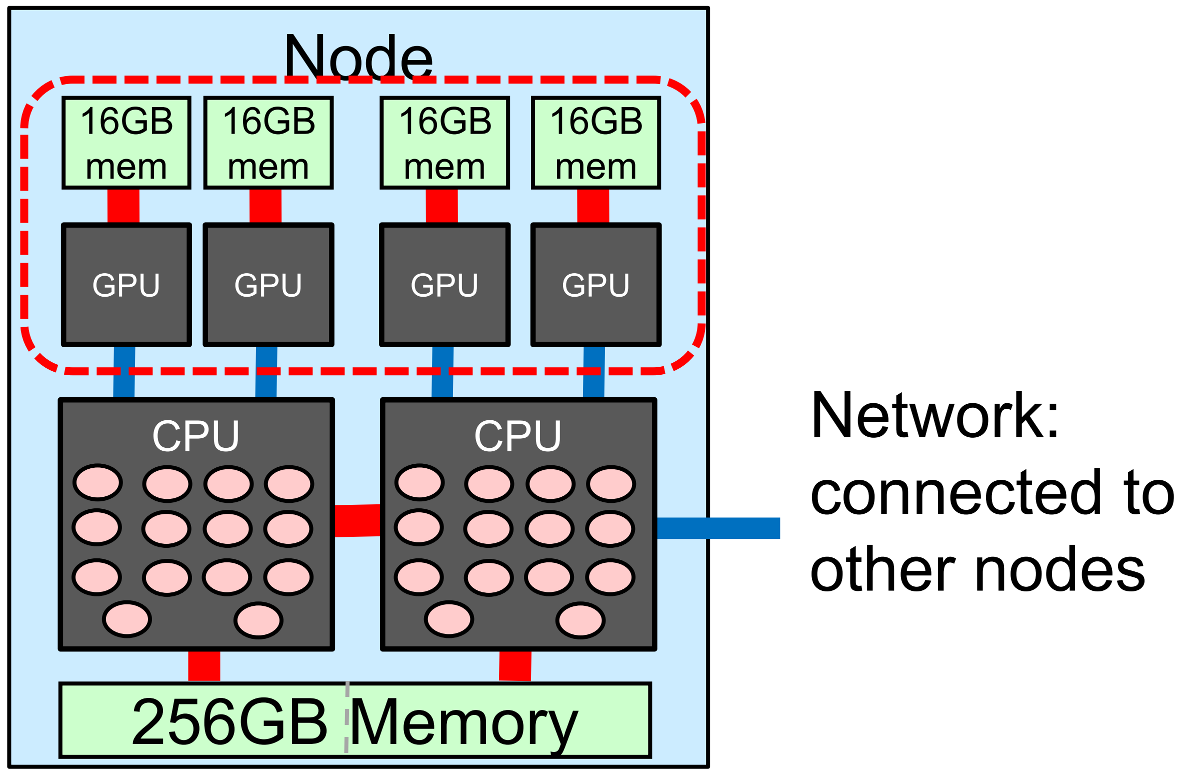
- GPUs are (still) omitted in this figure

A TSUBAME3 Node (2)

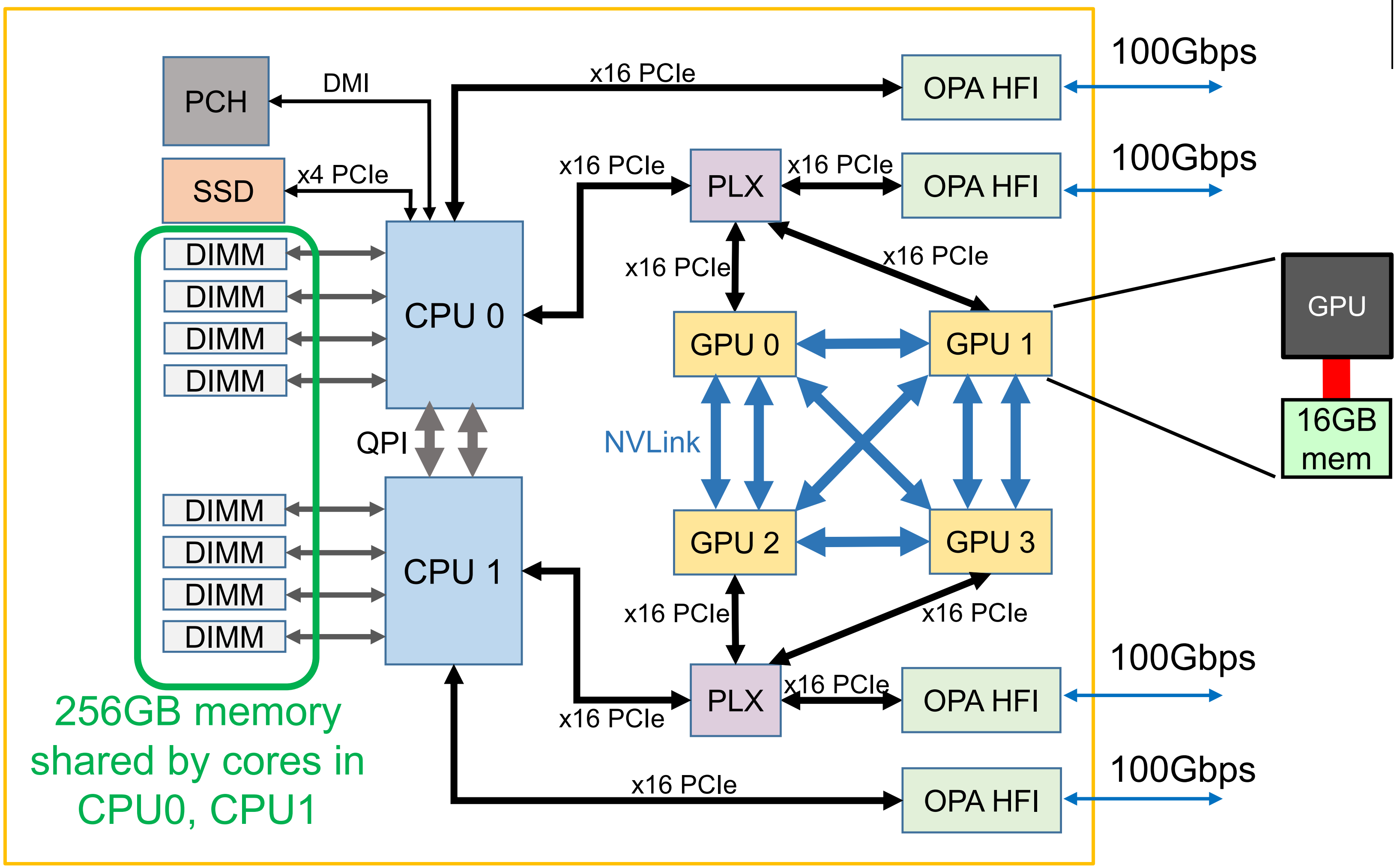
- A node has 2 CPUs + 4 GPUs
  - Each GPU (Tesla P100) has 56SMXs = 3,584 cores



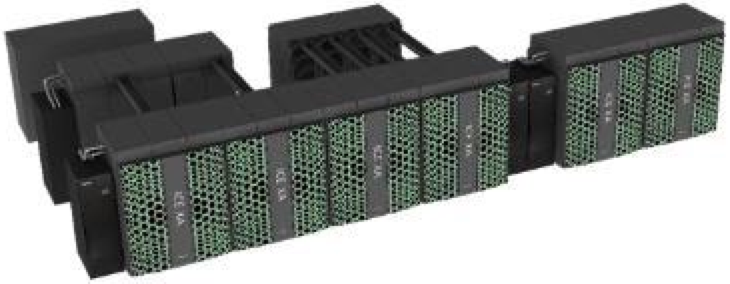
=



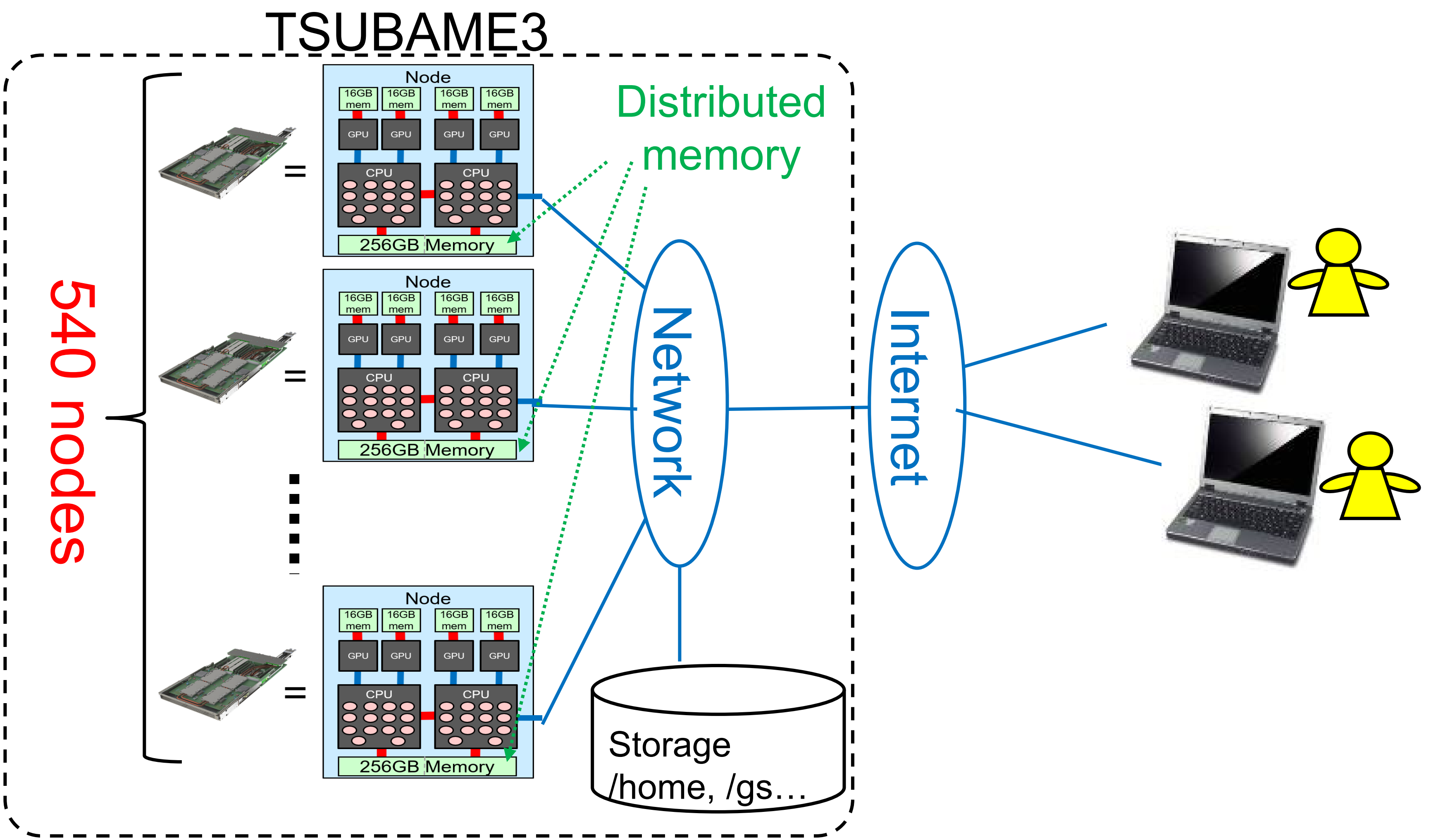
A TSUBAME3 Node in More Detail



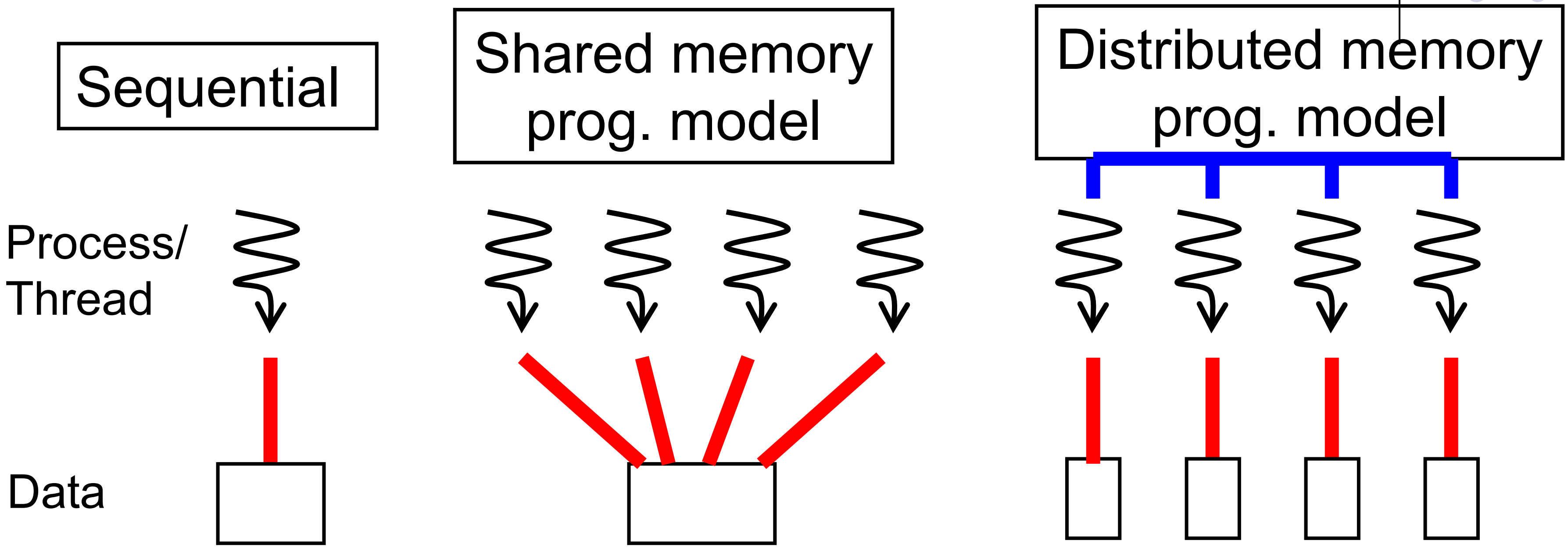
TSUBAME3 System



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



Classification of Parallel Programming Models



Programming without parallelsim

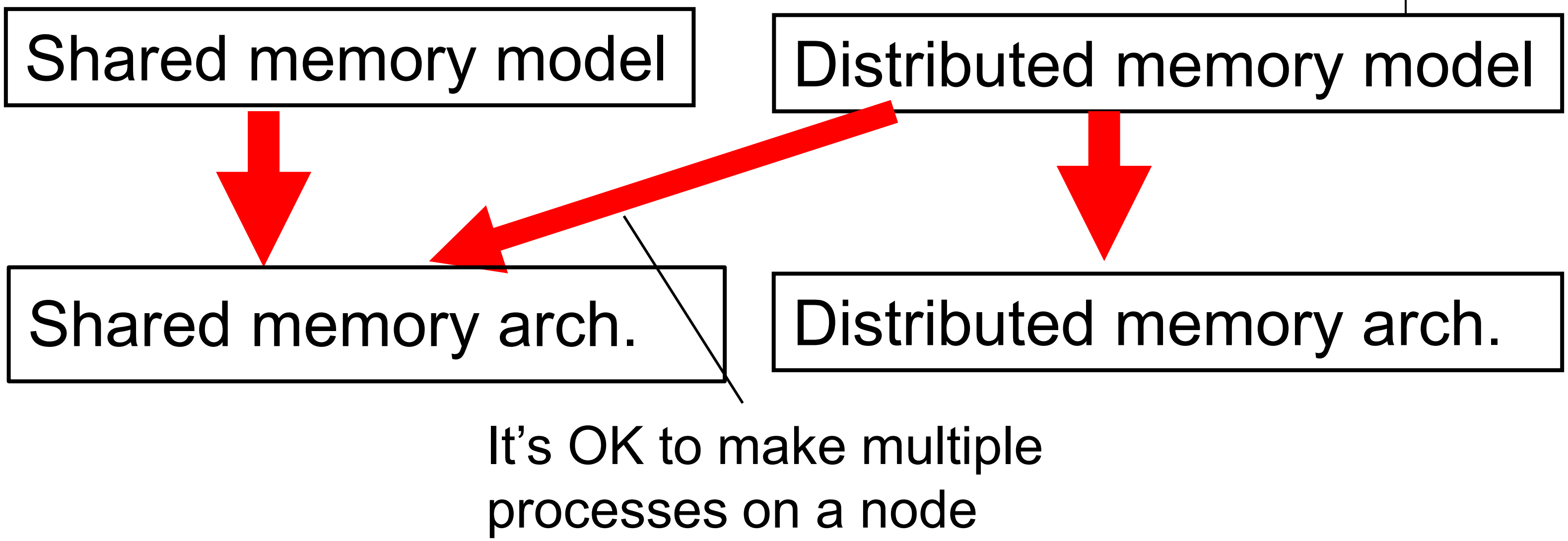
Threads have access to shared data

- OpenMP
- pthread
- Java thread…

Need communication among processes

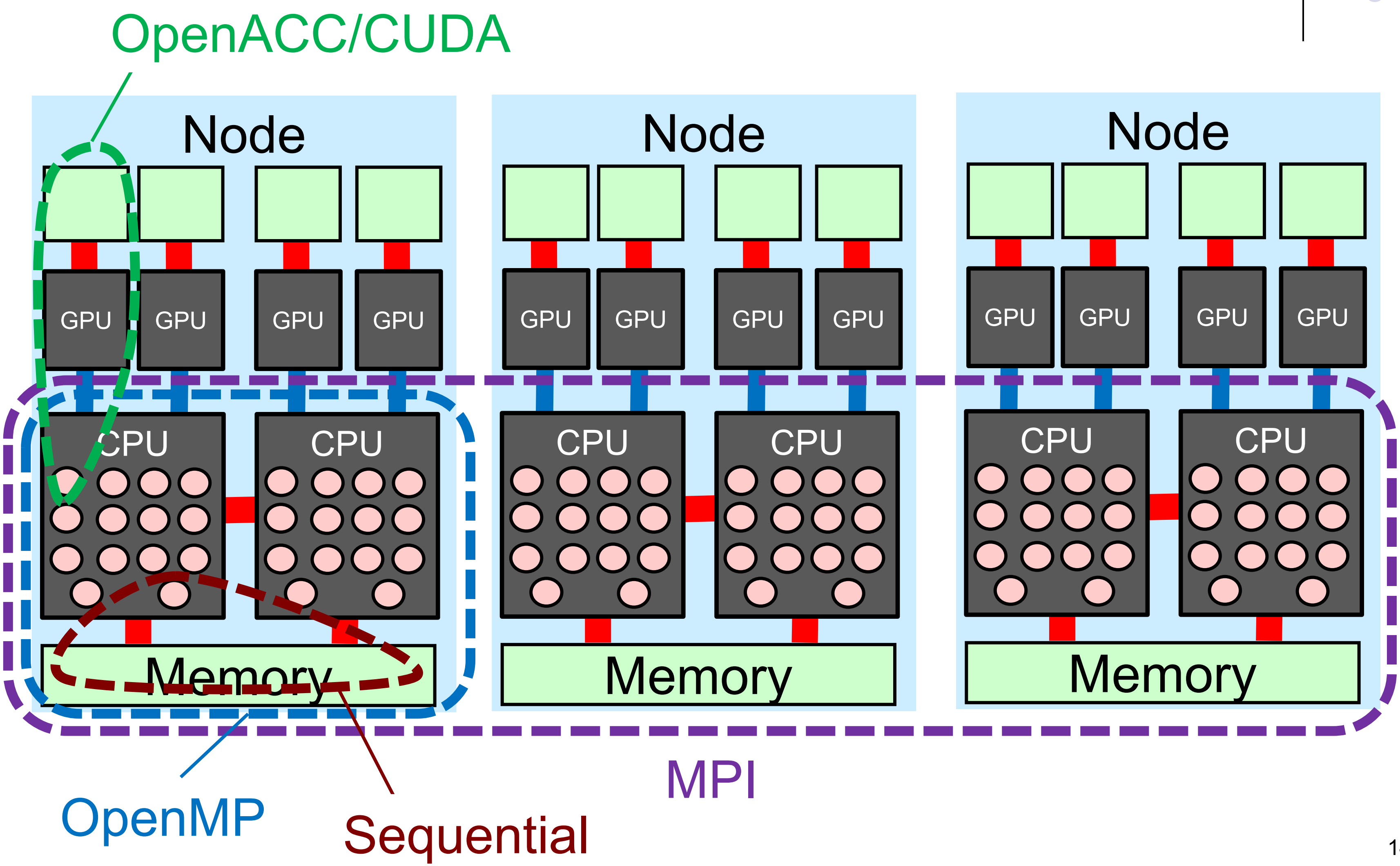
- MPI
- socket
- Hadoop, Spark…

Programming Models on Architecture



- Shared memory model (Part 1) can use only cores in a single node (up to 28 cores on TSUBAME3)
- Distributed memory model (Part 3) supports large scale parallelism (~15,000 cores on TSUBAME3)

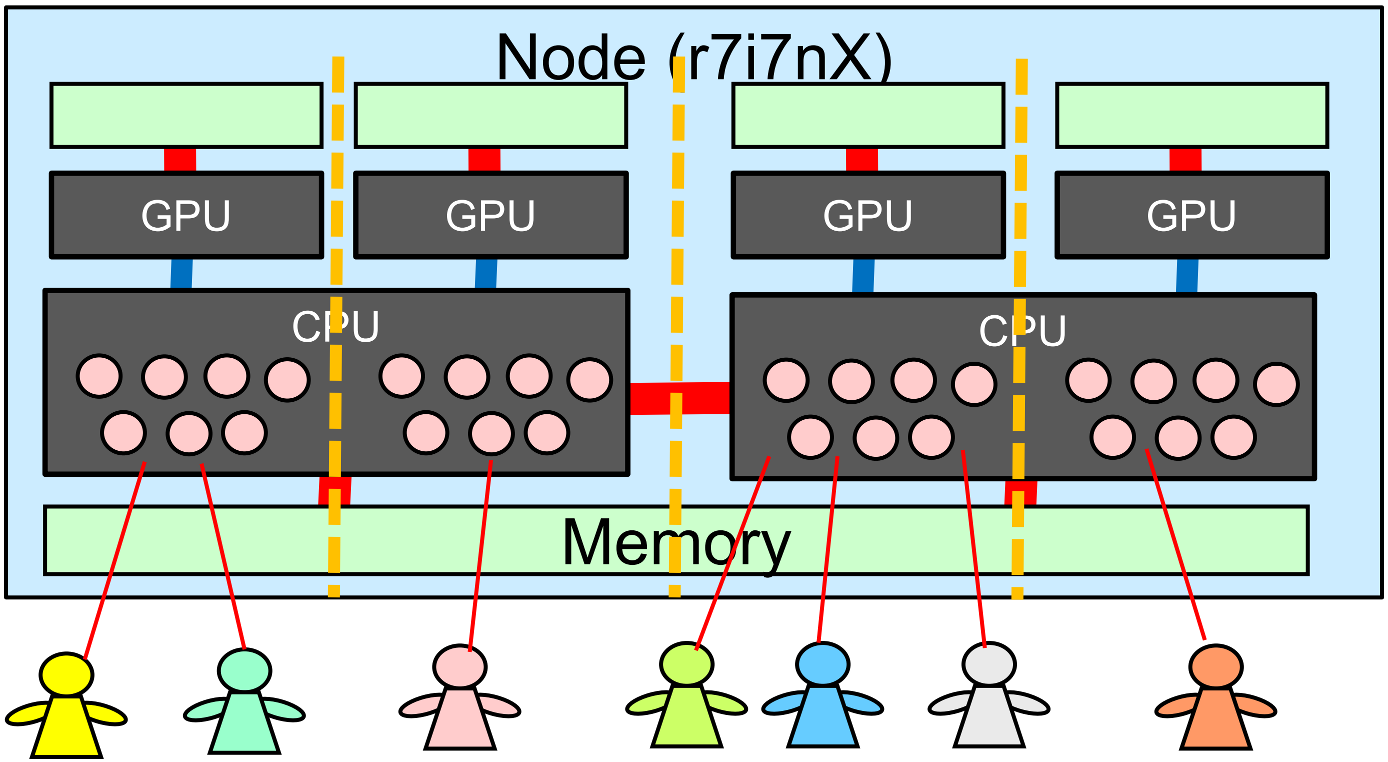
Parallel Programming Methods on TSUBAME



Standard route

Web-only route

TSUBAME Interactive Node



A node is partitioned into 4. Each user can use

- ¼ node = 7 CPU cores + 60GB memory + 1 GPU (3584cores+16GB mem)
- Only one partition simultaneously

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

Sample Programs in this Lecture

- Samples are at /gs/hs1/tga-ppcomp/23/ directory
  - You have to a member of tga-ppcomp group
    - If "ls /gs/hs1/tga-ppcomp/23" 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
  - Where is somewhere? If you are using web-only route, ~/t3workspace/ may be good
  - ~/t3workspace/ is automatically made when you use Jupyter lab first

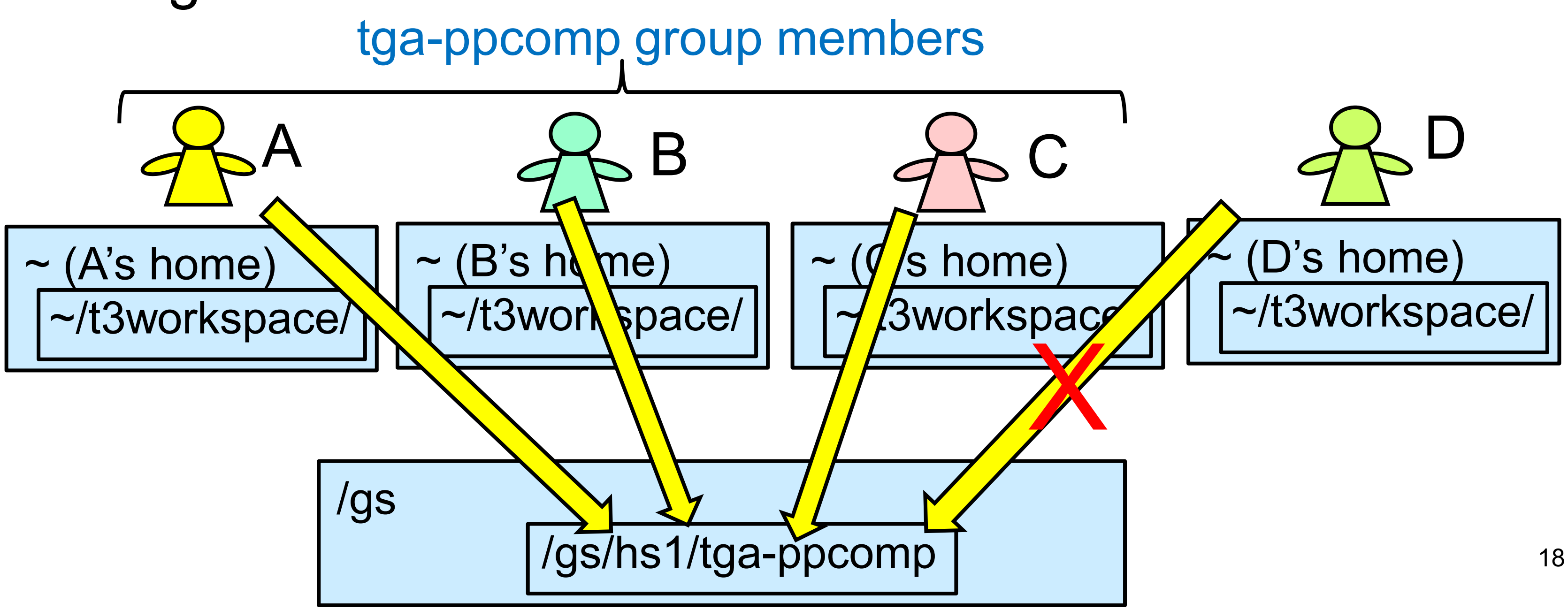
Copy "mm" sample to your home directory

[make sure that you are at an interactive node (r7i7nX)]
cd ~/t3workspace   *[In web-only route]*
cp -r /gs/hs1/tga-ppcomp/23/mm .

don't forget space & dot

Disk Storage of TSUBAME

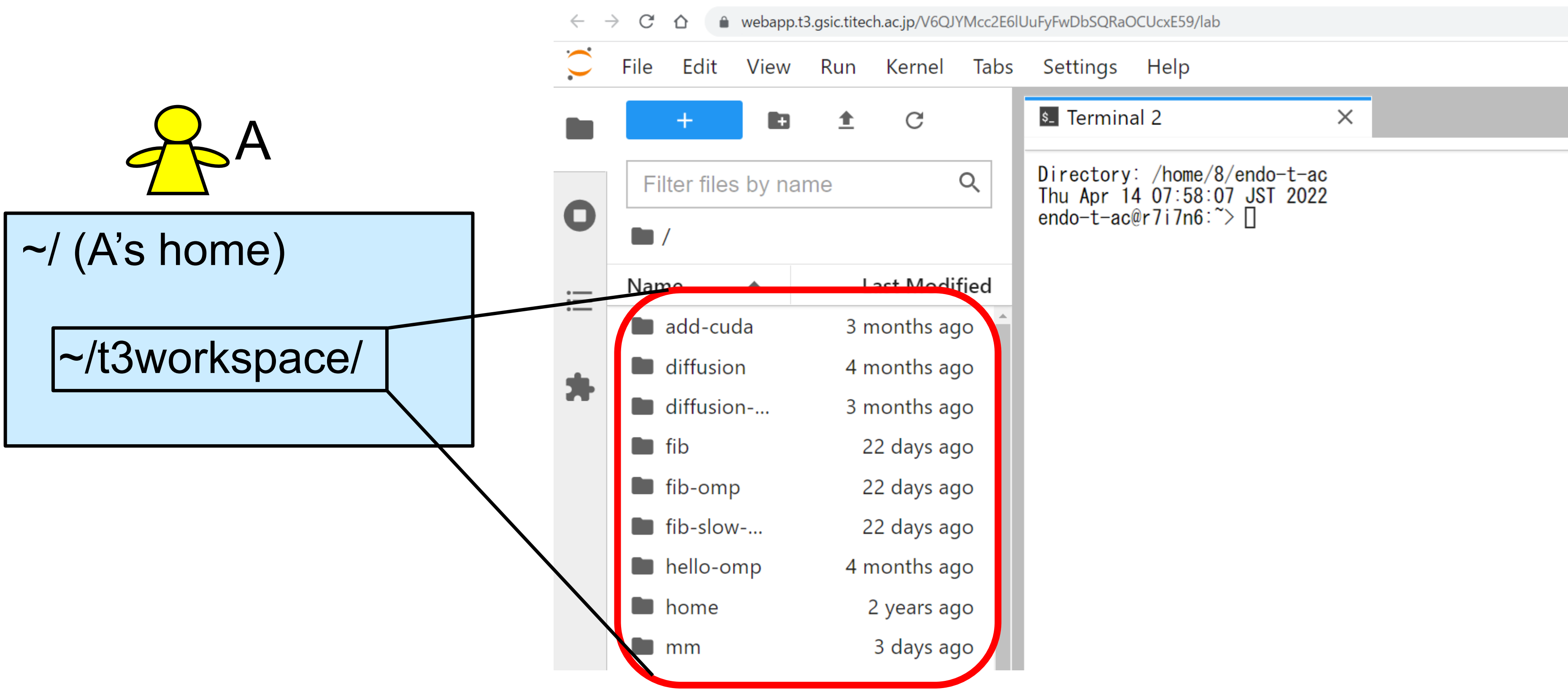
- ~/… (home directory) is assigned to each user
  - Actually, ~ is an alias to /home/??/22Mxxxxx (username)
- /gs/… are shared by multiple users
  - /gs/hs1/tga-ppcomp/… is shared by tga-ppcomp members
- Above storage can be accessed from all computing nodes and log-in nodes in TSUBAME



Web-only route

Notes in Web-Only Route

- The folder tree shows ~/t3workspace
  - Not the home directory (~/) itself



Make Copies of Other Samples

"pi" sample

```
cd ~/t3workspace    [In web-only route]
cp -r /gs/hs1/tga-ppcomp/23/pi .
```

"diffusion" sample

```
cd ~/t3workspace    [In web-only route]
cp -r /gs/hs1/tga-ppcomp/23/diffusion .
```

"fib" sample

```
cd ~/t3workspace    [In web-only route]
cp -r /gs/hs1/tga-ppcomp/23/fib .
```

"sort" sample

```
cd ~/t3workspace    [In web-only route]
cp -r /gs/hs1/tga-ppcomp/23/sort .
```

Executing Sample In Case of mm

[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 1000 1000 1000
[this is the execution of mm sample]

Using Sample Programs (3) Executing Samples

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

- mm

```
./mm 1000 1000 1000
```

Options are matrix sizes *m,n,k*

- pi

```
./pi 10000000
```

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

```
./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

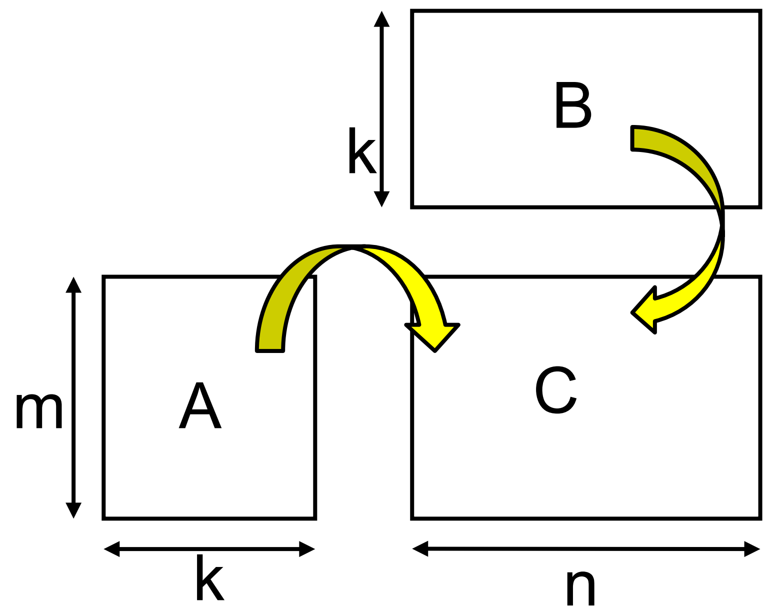
Make copy from /gs/hs1/tga-ppcomp/23/mm/

A: a ($m \times k$) matrix

B: a ($k \times n$) matrix

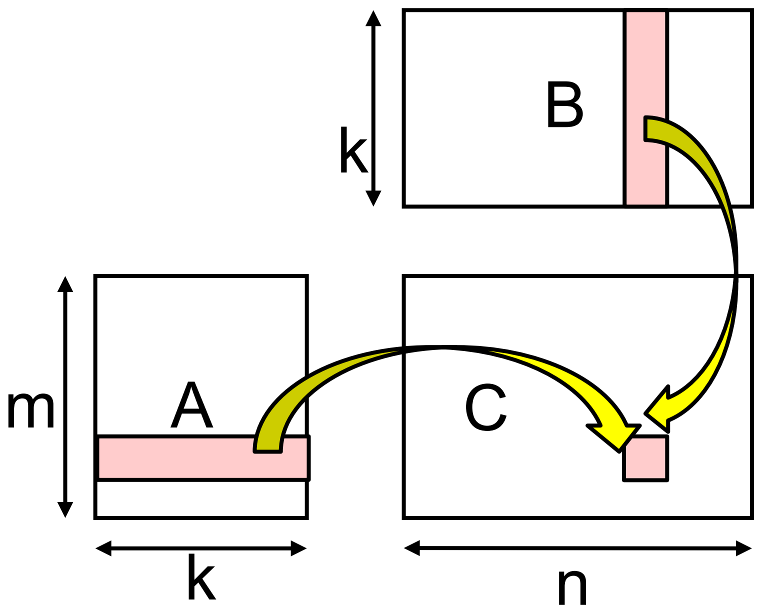
C: a ($m \times n$) matrix

$$C \leftarrow A \ B$$



- This sample supports variable matrix sizes
- Execution: ./mm [m] [n] [k]

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
      C[i,j] += A[i,l] * B[l,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

※ Exceptionally, C99 specification includes variable length arrays

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

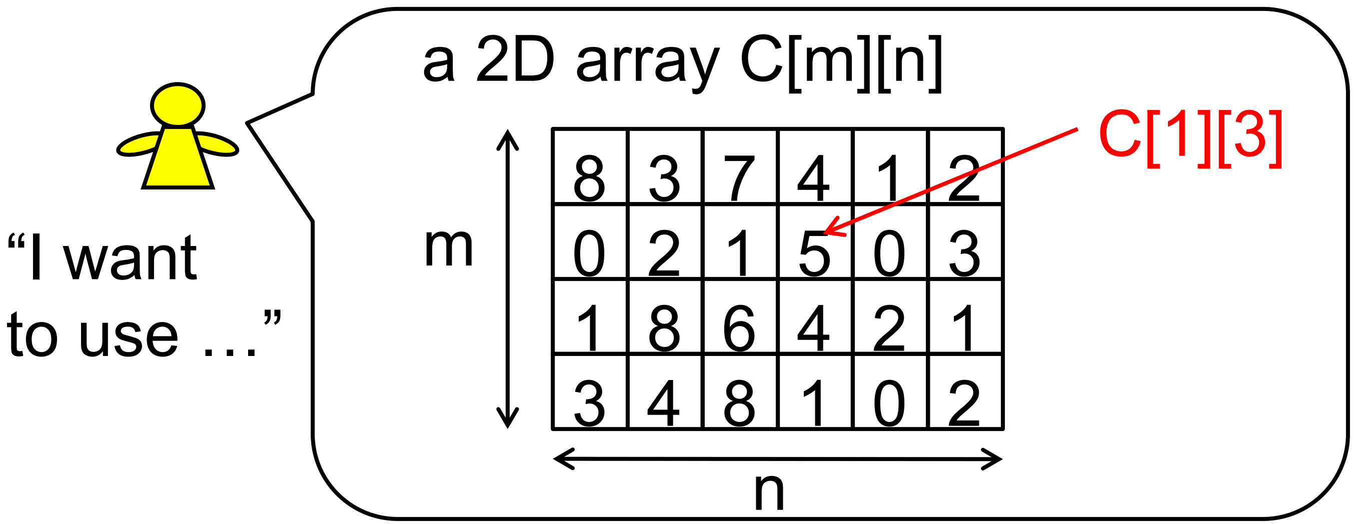
- We *malloc* m 1D arrays for every row (each has n length)
- We malloc 1D array of m length to store the above pointers

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

(mm sample uses this method)

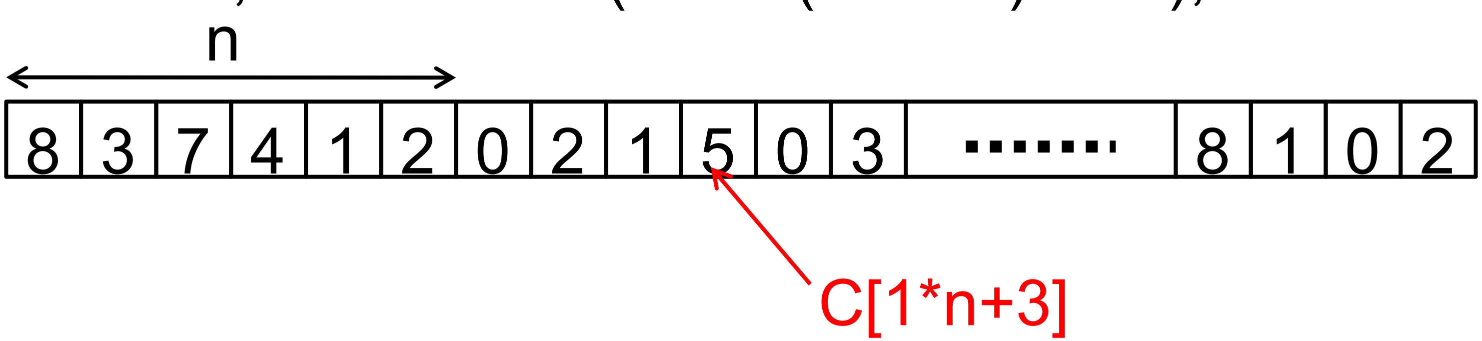
- To access an array element, we should use C[i*n+j] or C[i+j*m], instead of C[i][j]

Express a 2D array using a 1D array



Expressions in C language (Example)

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



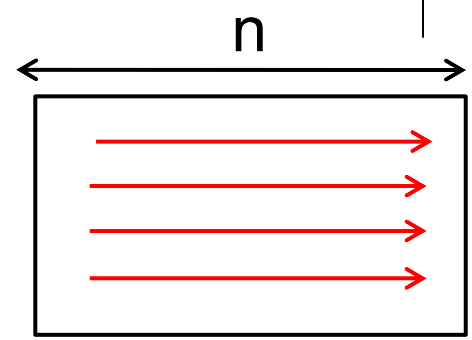
In this case, an element $C_{i,j}$ is C[i*n+j]

Two Data Formats

Row major format

- More natural for C programmers

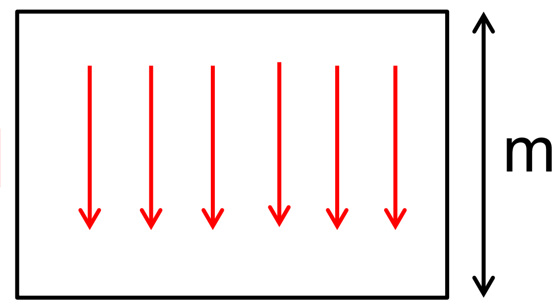
$C_{i,j} \Rightarrow C[i*n+j]$



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()

```
#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

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];
  }
```