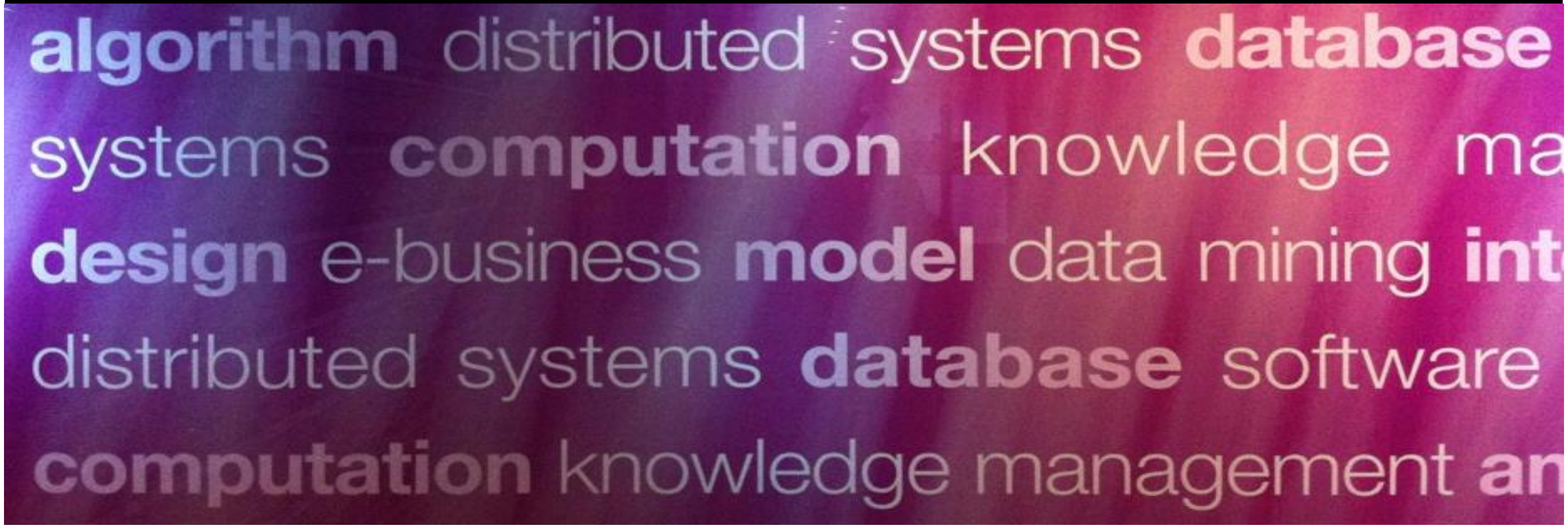
MONASH University

Information Technology

FIT3143 - LECTURE WEEK 12

REVISION & FINAL EXAM DETAILS


algorithm distributed systems database
systems computation knowledge ma
design e-business model data mining int
distributed systems database software
computation knowledge management an

Topic Overview

- Brief review of topics covered from Week 1 to Week 11 of the semester
- Final assessment (or final exam) details.

Note: These slides contain embedded weblinks.

Brief review of topics covered from Week 1 to Week 11 of the semester

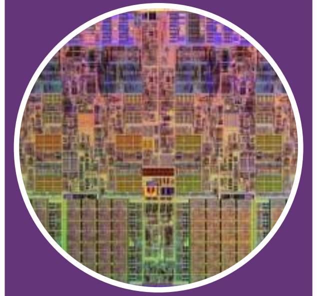
FIT3143 Components

Shared Memory Parallelism

- Weeks 2-3

Distributed Memory Parallelism

- Weeks 4-9



Instruction Level Parallelism with SSE

- Weeks 10

Parallel computing on GPUs

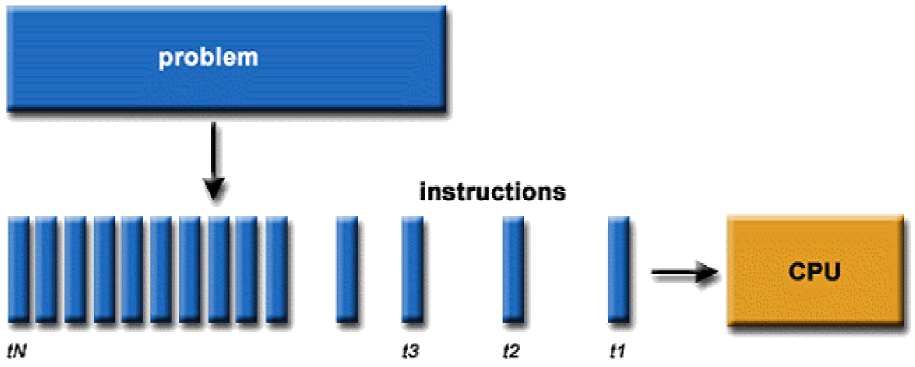
- Week 11

Algorithms

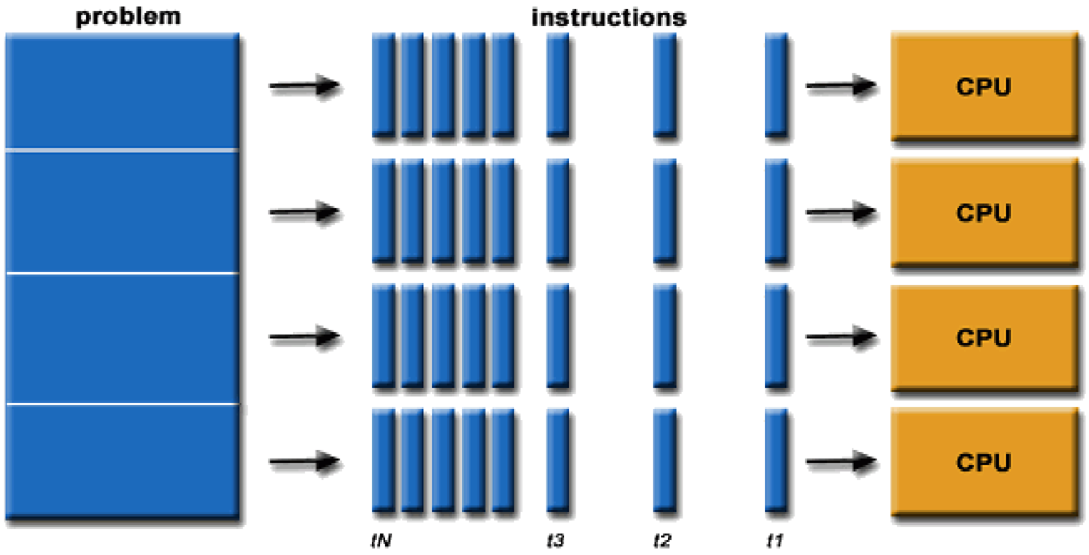
Week 1 – Introduction to Parallel Computing

1. Parallel computing concept and applications
2. Parallel computing models
3. Distributed computing
4. Parallel computing performance

Week 1 – Introduction to Parallel Computing



Serial Computing

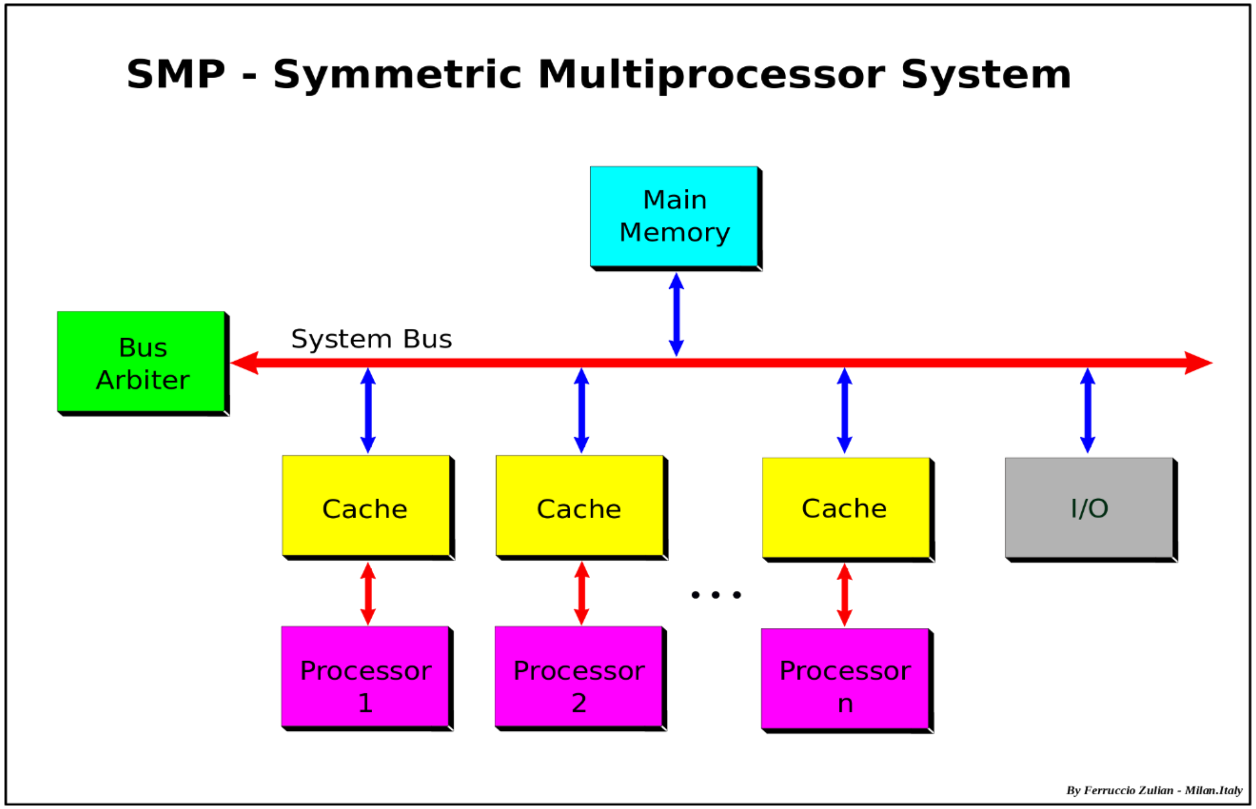


Parallel Computing

Adapted from https://computing.llnl.gov/tutorials/parallel_comp/

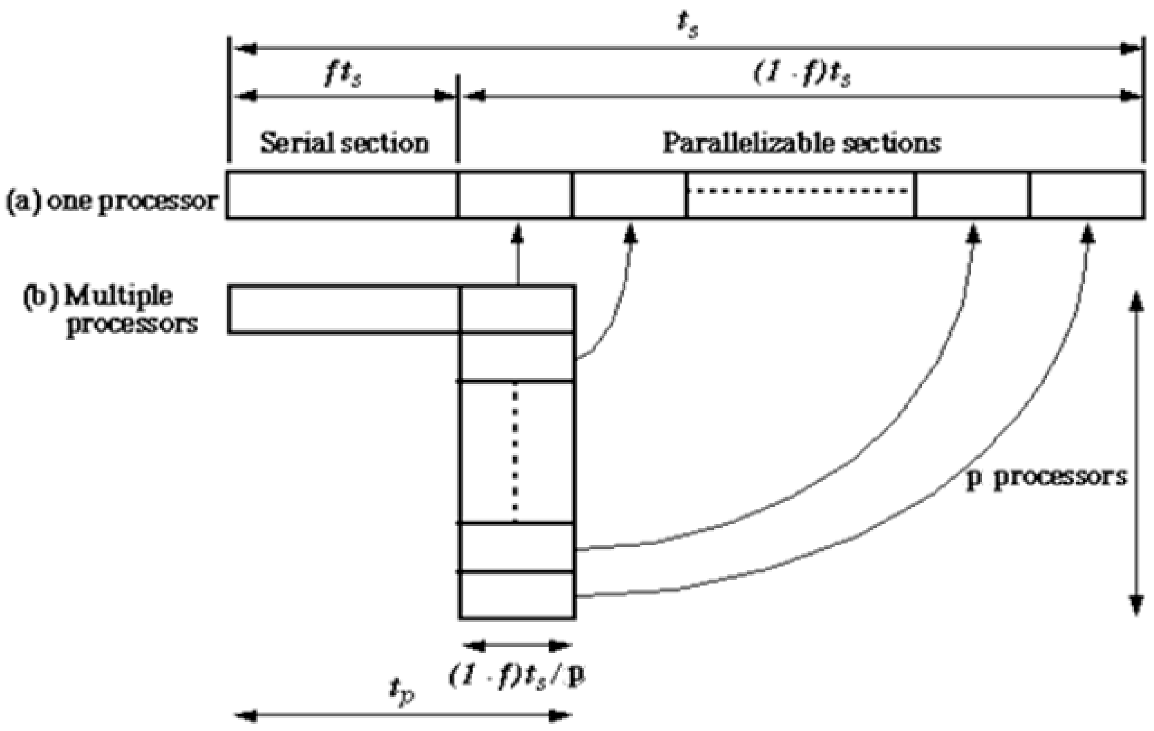
Week 1 – Introduction to Parallel Computing

Shared Memory Parallelism



Week 1 – Introduction to Parallel Computing

Parallelizing sequential problem-Amdahl's law



Alternative:

$$\frac{1}{r_s + \frac{r_p}{p}}$$

r_s is the serial ratio (non-parallelizable portion)

r_p is the parallel ratio (parallelizable portion)

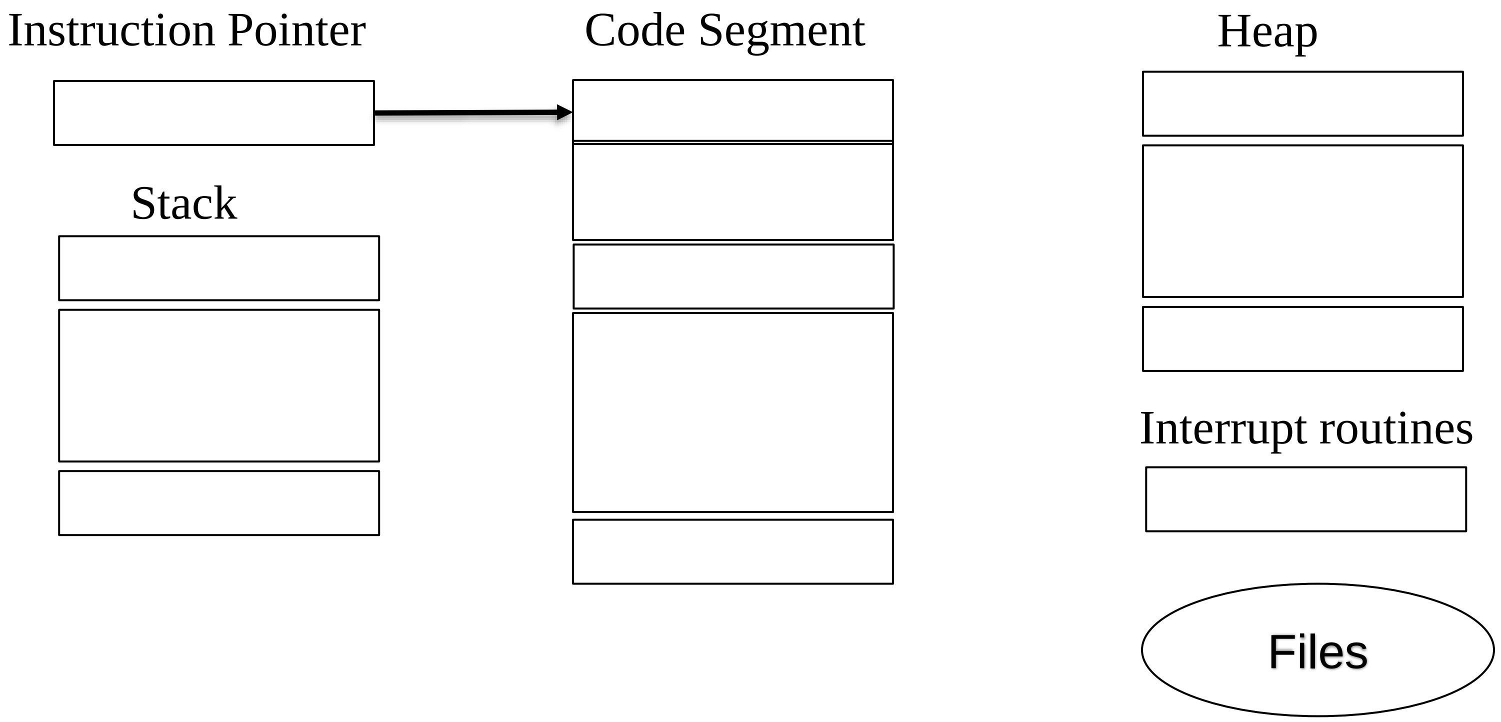
p is the number of processors

Week 2 – Parallel Computing on Shared Memory

1. Shared memory architecture and constructs for specifying parallelism.
2. POSIX for shared memory parallel programming.
3. OpenMP for shared memory parallel programming.

Week 2 – Parallel Computing on Shared Memory

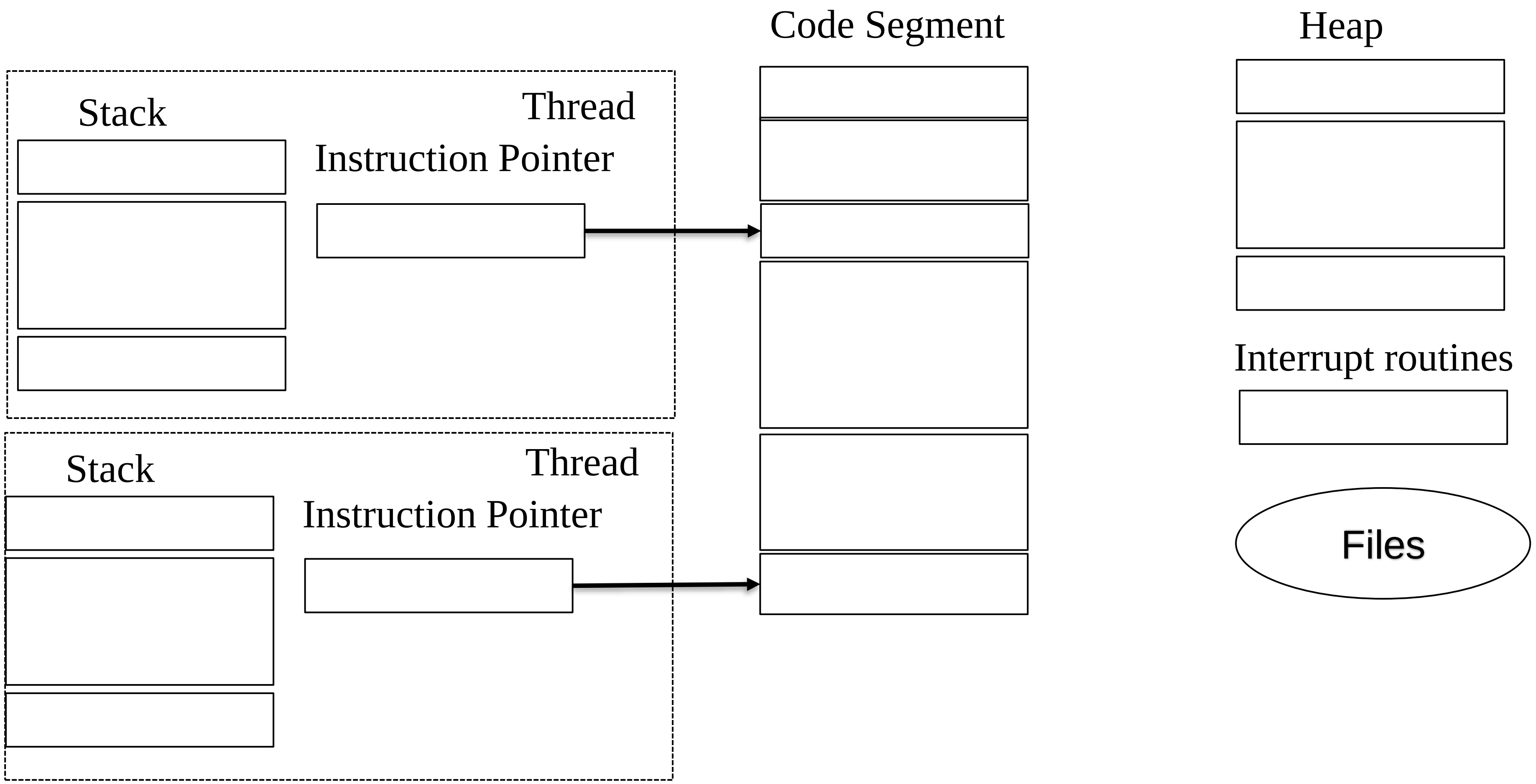
Process



An instruction pointer holds address of the next instruction to be executed. A stack is present for procedure calls, and a heap, system routines and files.

Week 2 – Parallel Computing on Shared Memory

Threads



Each thread has its own instruction pointer pointing to the next instruction of the thread to be executed. Each thread needs its own stack and also stores information regarding registers but shares the code and other parts. A process can have more than one thread.

POSIX Threads

What are Pthreads?

- POSIX.1c standard
- C language interface
- Threads exist within same process
- All threads are peers
    - No explicit parent-child model
    - Exception: “main thread” holds process information

Week 3 OpenMP

What Is OpenMP (or Open Multi-Processing)?

- ❑ Compiler directives for multithreaded programming
- ❑ Easy to create threaded Fortran and C/C++ codes
- ❑ Supports data parallelism model
- ❑ Incremental parallelism
  - ❑ Combines serial and parallel code in single source

- ❑ **Incremental parallelism is when you can modify only a portion of the code to test for better performance in parallel, then move on to another position in the code to test. Explicit threading models require a change to all parts of the code affected by the threading.**
- ❑ **Serial code can be “retrieved” by not compiling with the OpenMP options turned on. Assuming that there were no drastic changes required to get the code into a state that could be parallelized.**

http://www.openmp.org

OpenMP* Architecture

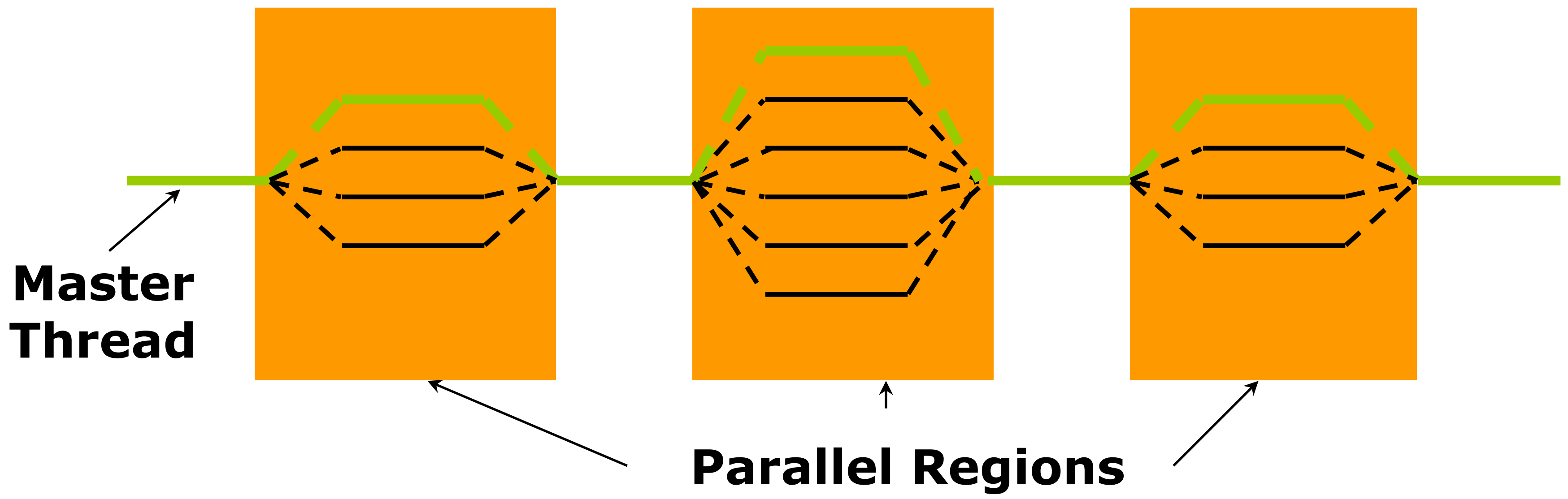
- Fork-join model // execution model of thread in OpenMP parallel regions
- Work-sharing constructs // for the pragmas and directives
- Data environment constructs // for the pragmas and directives
- Synchronization constructs // for the pragmas and directives
- Extensive Application Program Interface (API) for finer control
  // for the pragmas and directives

Programming Model

Fork-join parallelism:

- **Master thread** spawns a **team of threads** as needed
- **Parallelism is added incrementally: the sequential program evolves into a parallel program**

Applications starts execution in serial with Master Thread. At each parallel region encountered, threads are forked off, execute concurrently, and then join together at the end of the region.



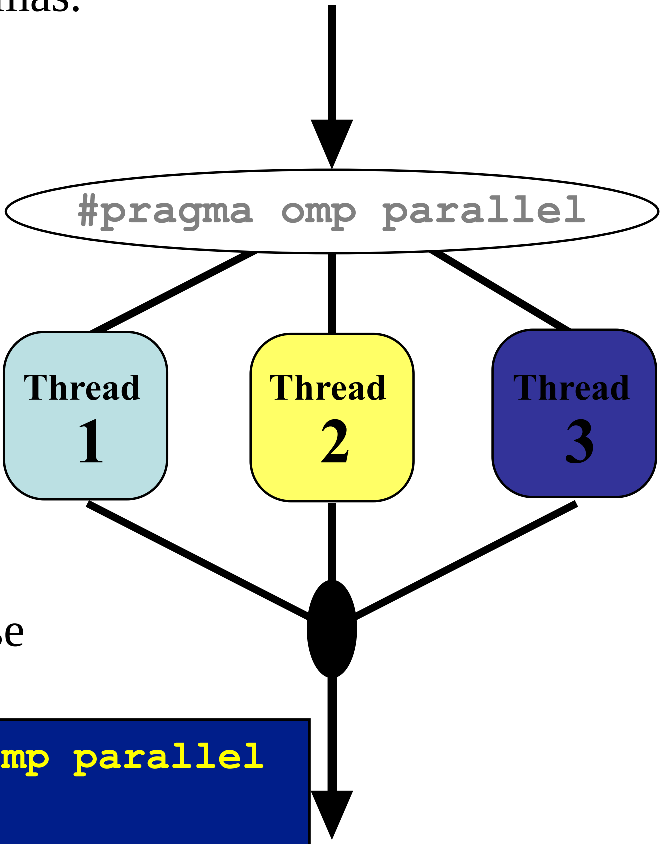
OpenMP* Pragma Syntax

Most constructs in OpenMP* are compiler directives or pragmas.
For C and C++, the pragmas take the form:

#pragma omp ***construct [clause [clause]…]***

Parallel Regions

- ❑ Defines parallel region over structured block of code
- ❑ The '**parallel**' pragma that defines a parallel region.
- ❑ Threads are created as '**parallel**' pragma is crossed
- ❑ Threads block at end of region
- ❑ Data is shared among threads unless specified otherwise



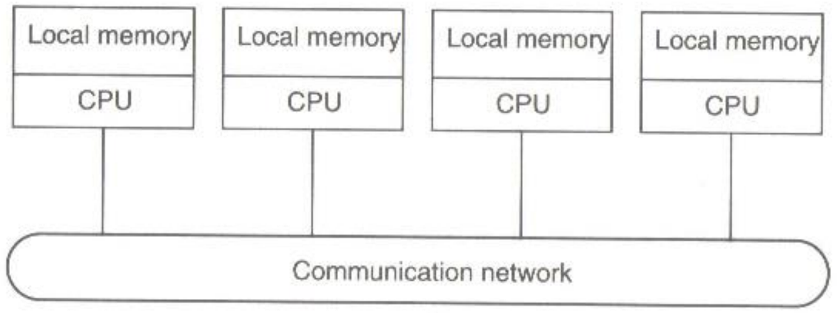
- ❑ **Pragma will operate over a single statement or block of statements enclosed within curly braces.**
- ❑ **Variables accessed within the parallel region are all *shared* by default.**

C/C++ :

```
        #pragma omp parallel
            {
                    block
            }
```

Week 4 - Introduction to Parallel Computing on Distributed Memory

- Loosely coupled systems
  - Processors do not share memory
  - Each processor has its own local memory



Parallel vs. Distributed computing

S.NO	PARALLEL COMPUTING	DISTRIBUTED COMPUTING
1.	Many operations are performed simultaneously	System components are located at different locations
2.	Single computer is required	Uses multiple computers
3.	Multiple processors perform multiple operations	Multiple computers perform multiple operations
4.	It may have shared or distributed memory	It have only distributed memory
5.	Processors communicate with each other through bus	Computer communicate with each other through message passing.
6.	Improves the system performance	Improves system scalability, fault tolerance and resource sharing capabilities

Source: https://www.geeksforgeeks.org/difference-between-parallel-computing-and-distributed-computing/

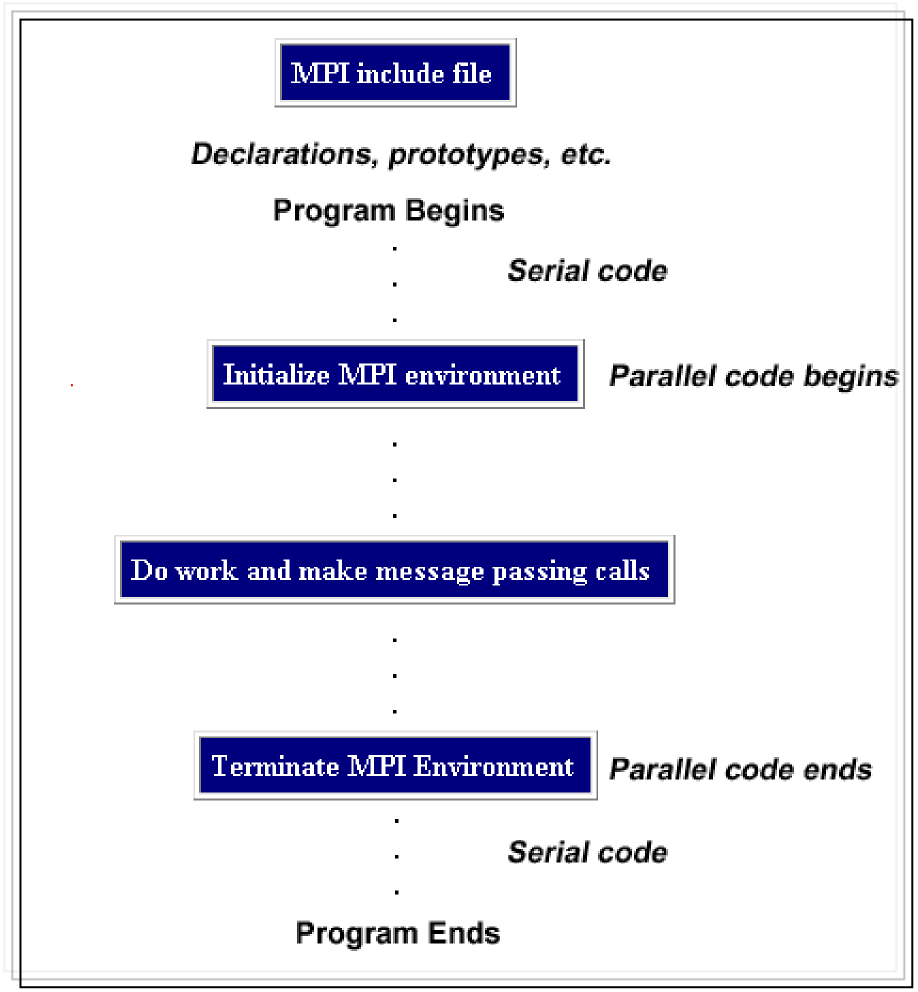
Week 4 – IPC & RPC

1. **What is the purpose of IPC?**
   - **information sharing among two or more processes**
2. **Differences between Synchronous and Asynchronous Communications?**
   - **When both the send and receive primitives of a communication between two processes use blocking semantics, the communication is said to be Synchronous; otherwise it is asynchronous**
3. List the Types of Failure in IPC
   - Loss of request message, Loss of response message, Unsuccessful execution of the request
4. How to implement Idempotency?
   - Adding sequence number with the request message and Introduction of ‘Reply cache’

Week 4 – IPC & RPC

5. **Three main types of Group Communications?**
   - **One to many, Many to one, Many to many**
6. One of the greatest challenges in Many to Many?
   - Ordered Delivery
7. **Name an all propose IPC protocol?**
   - **Remote Procedure Call (RPC)**
8. Name a few ways to optimize RPC?
   - Concurrent Access to Multiple Servers, Serving Multiple Requests Concurrently, Reducing Call Workload per Server
9. **Three different techniques for implementing Concurrent Access to Multiple Servers?**
   - **Threads, Early Reply, Call Buffering**

Week 5 – Parallel Computing In Distributed Memory – Message Passing Library


MPI include file
Declarations, prototypes, etc.
Program Begins
Serial code
Initialize MPI environment
Parallel code begins
Do work and make message passing calls
Terminate MPI Environment
Parallel code ends
Serial code
Program Ends

Week 6 – Synchronization, Mutex, Deadlocks

1. Real time Clock Synchronization Methods
2. Logical Clock Synchronization Techniques
3. Mutual Exclusion Approaches
4. Deadlock detection and handling

Week 6 – Synchronization, Mutex, Deadlocks

- Three Real Clock Synchronisation Methods?
  - **Cristian's Method**
  - **Berkeley Algorithm**
  - **Averaging Algorithm**
- Two Logical (Clock) Synchronisation Techniques
  - **Lamport**
  - **Vector Clock**
- Three Mutual Exclusion Approaches?
  - Centralised
  - Distributed
  - GME
- Main Deadlock Modeling Areas?
  - Necessary condition for occurrence
  - Deadlock Detection
  - Deadlock Handling

Week 7 – Faults & Distributed Consensus

- Classification of failures
  - Crash Failure
  - Omission Failure
  - Transient Failure
  - Byzantine Failure
  - Software Failure
  - Temporal Failure
  - Security Failure

- Fault-Tolerant System
  - Masking tolerance
  - Non-masking tolerance
  - Fail-safe tolerance
  - Graceful degradation

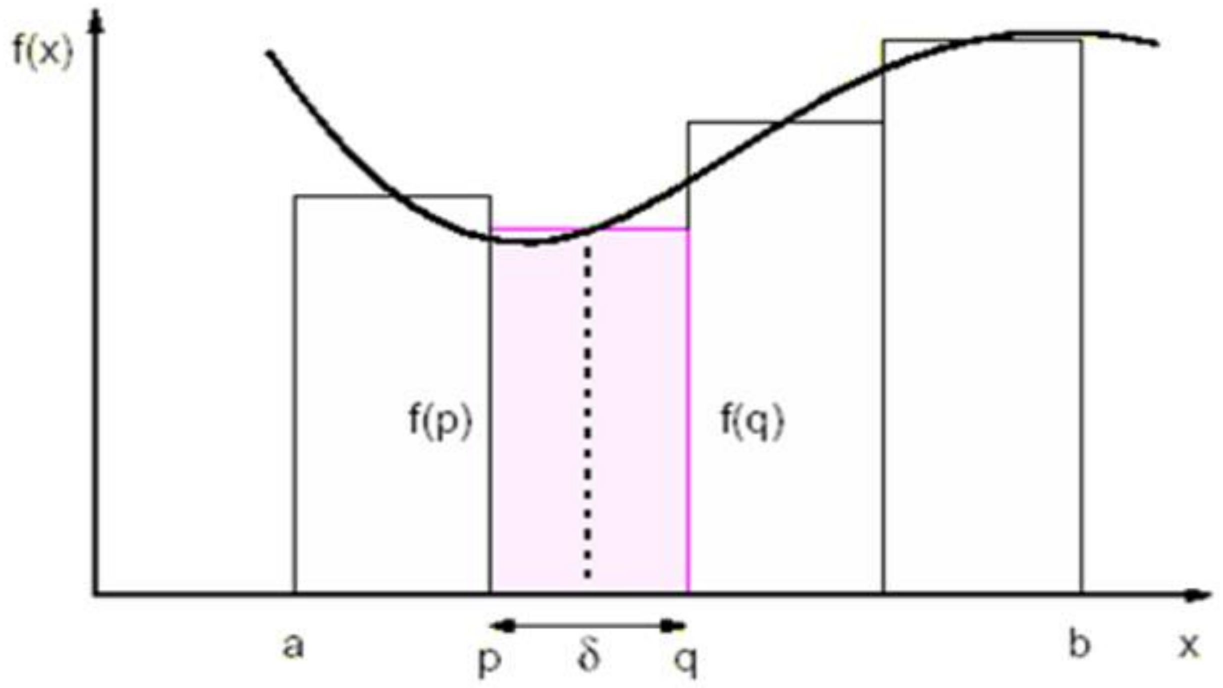
Week 7 – Numerical Integration

$$I = \int_a^b f(x)dx$$

- To integrate the function (i.e., to compute the ‘area under the curve’), we can divide the area into separate parts,each of which can be calculated by a separate process.
- Each region could be calculated using an approximation given by rectangle, where f(p) and f(q) are the heights of the two edges of the rectangular region and ∂ is the width (interval).
- The complete integration can be approximated by the summation of rectangular regions from a to b by aligning the rectangles so that the upper midpoint each rectangle intersect with the function. This has the advantage that errors on each side of midpoint end tend to cancel.

Week 7 – Numerical Integration (Rectangles)

- Each region under the curve is calculates using an approximation given by rectangles which are aligned.

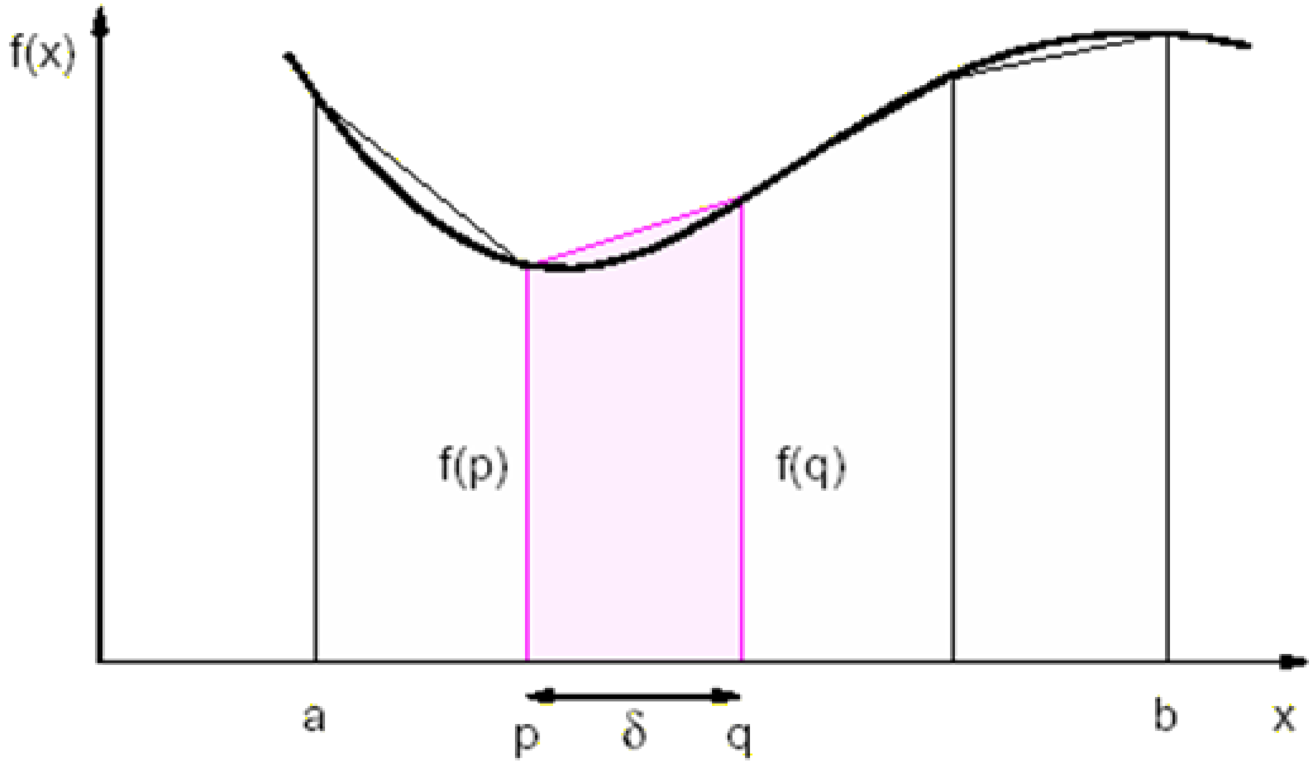


Week 7 – Numerical Integration (Trapezoid)

- Here we take the actual intersections of the lines with the function to create trapezoidal regions as shown in the fig.
- Each region is now calculated as 1/2(f(p) +f (q))∂
- Such approximate numerical methods for computing a definite integral using linear combination of values are called ***quadrature methods***

Week 7 – Numerical Integration (Trapezoid)

Area = ½ (f(p) +f (q)) ∂



Week 8 – Matrix Multiplication

Matrix multiplication:

$$C = A \cdot B$$

or

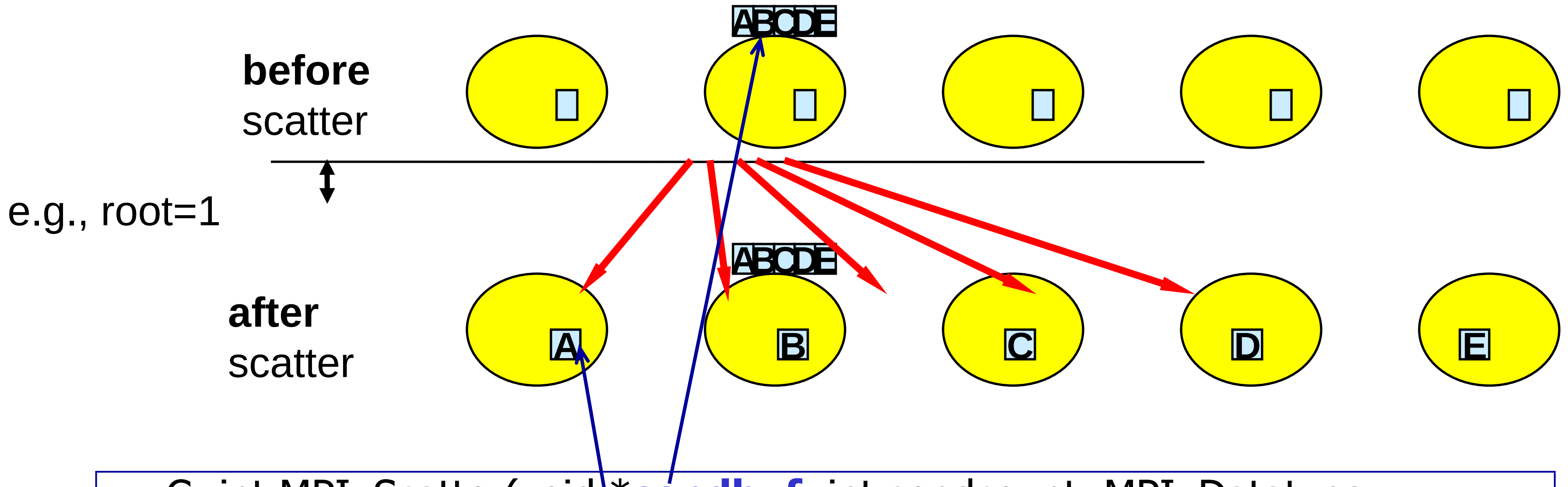
$$\begin{pmatrix} c_{0,0}, & c_{0,1}, & \dots, & c_{0,l-1} \\ & & \dots & \\ c_{m-1,0}, & c_{m-1,1}, & \dots, & c_{m-1,l-1} \end{pmatrix} = \begin{pmatrix} a_{0,0}, & a_{0,1}, & \dots, & a_{0,n-1} \\ & & \dots & \\ a_{m-1,0}, & a_{m-1,1}, & \dots, & a_{m-1,n-1} \end{pmatrix} \begin{pmatrix} b_{0,0}, & b_{0,1}, & \dots, & a_{0,l-1} \\ & & \dots & \\ b_{n-1,0}, & b_{n-1,1}, & \dots, & b_{n-1,l-1} \end{pmatrix}$$

The matrix multiplication problem can be reduced to the execution of m·l independent operations of matrix A rows and matrix B columns inner product calculation

$$c_{ij} = \left(a_i, b_j^T\right) = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, \quad 0 \le i < m, \; 0 \le j < l$$

Data parallelism can be exploited to design parallel computations

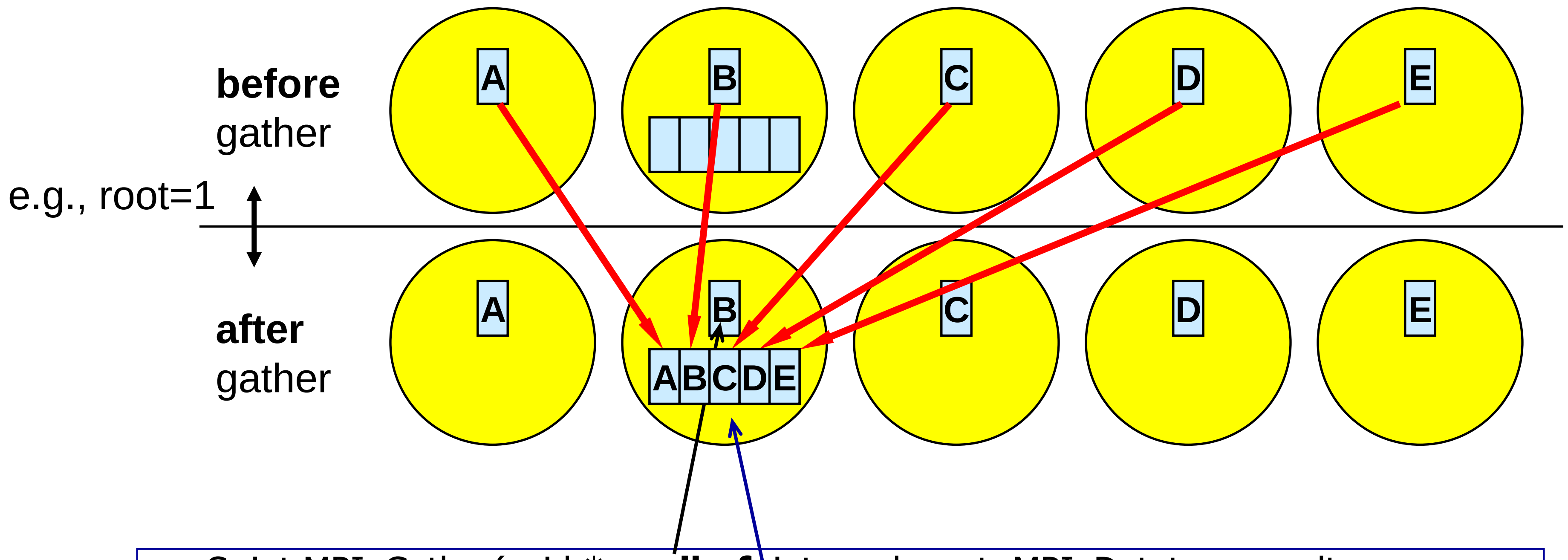
Week 9 - Scatter



- C: int MPI_Scatter(void ***sendbuf**, int sendcount, MPI_Datatype sendtype, void ****recvbuf***, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- C: int MPI_Scatterv(const void ***sendbuf**, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void ***recvbuf**, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Week 9 - Gather

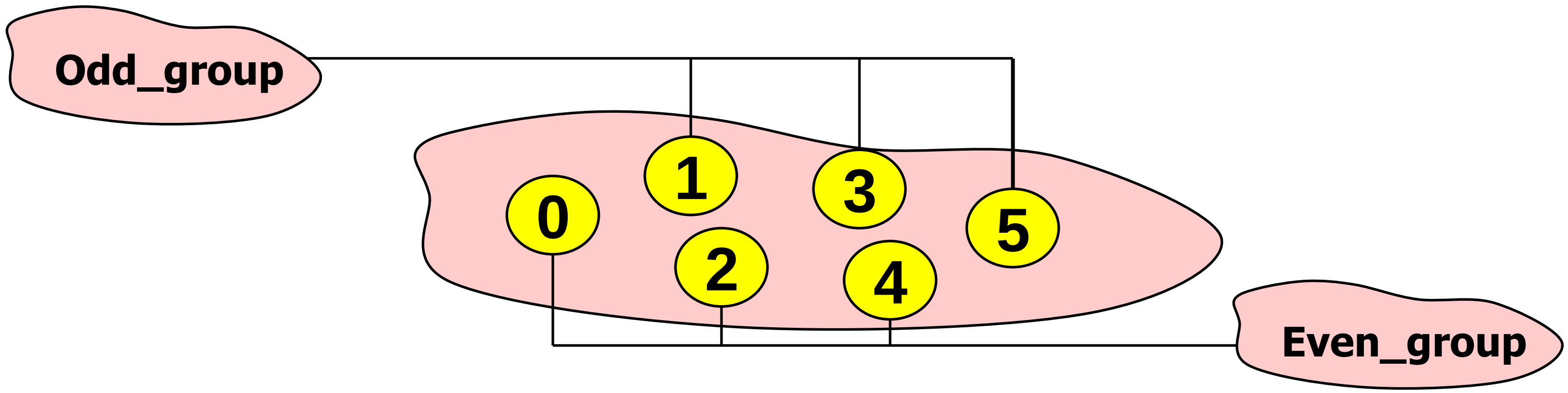


- C: int MPI_Gather(void ***sendbuf**, int sendcount, MPI_Datatype sendtype, void ****recvbuf***, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- C: int MPI_Gatherv(const void ***sendbuf**, int sendcount, MPI_Datatype sendtype, void ***recvbuf**, const int *recvcounts, const int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

Click here for sample C code implementation of MPI Scatter & Gather

Week 9 – Groups & Topologies



- Select processes ranks to create groups
- Associate to these groups *new* communicators
- Use these new communicators as usual
- MPI_Comm_group(comm, group) returns in *group* the group associated to the communicator *comm*

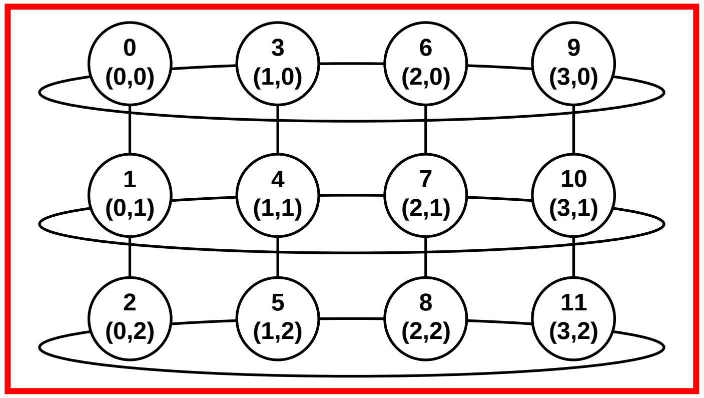
Week 9 – Topology Types

- Cartesian Topologies
  - each process is *connected* to its neighbor in a virtual grid,
  - boundaries can be cyclic, or not,
  - processes are identified by Cartesian coordinates,
  - of course,
    communication between any two processes is still allowed.
- Graph Topologies
  - general graphs,
  - not covered here.

Creating a Cartesian Virtual Topology

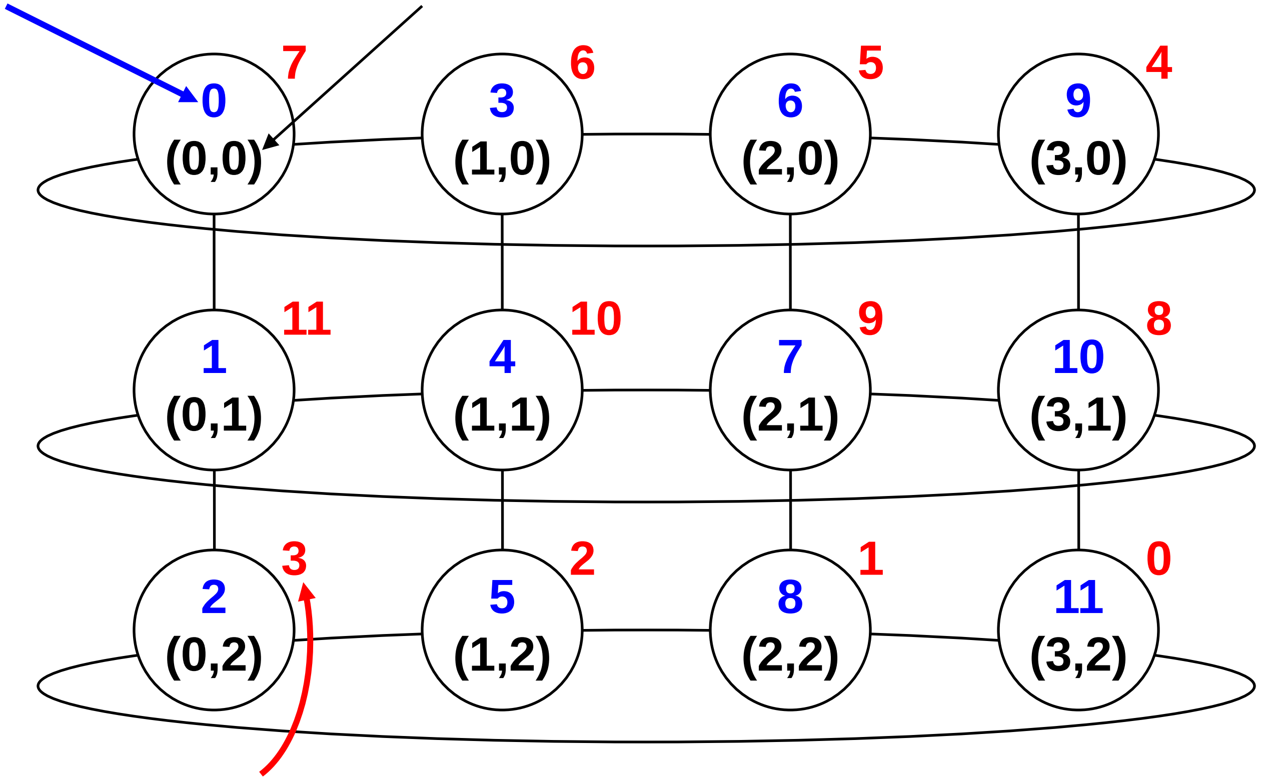
- int MPI_Cart_create(MPI_Comm comm_old, int ndims,
  int *dims, int *periods, int reorder,
  MPI_Comm *comm_cart)

comm_old	**=**	**MPI_COMM_WORLD**
ndims	**=**	**2**
dims	**=**	**(4, 3)**
periods	**=**	**(1/.true., 0/.false.)**
reorder	**=**	***see next slide***

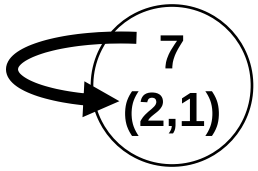


Example – A 2-dimensional Cylinder

- **Ranks** and **Cartesian process coordinates** in **comm_cart**
- Ranks in **comm** and **comm_cart** may differ, if **reorder = 1** or **.TRUE.**
- This reordering can allow MPI to optimize communications



Cartesian Mapping Functions

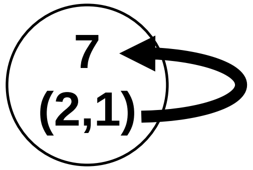


- Mapping
  ranks to
  process grid coordinates

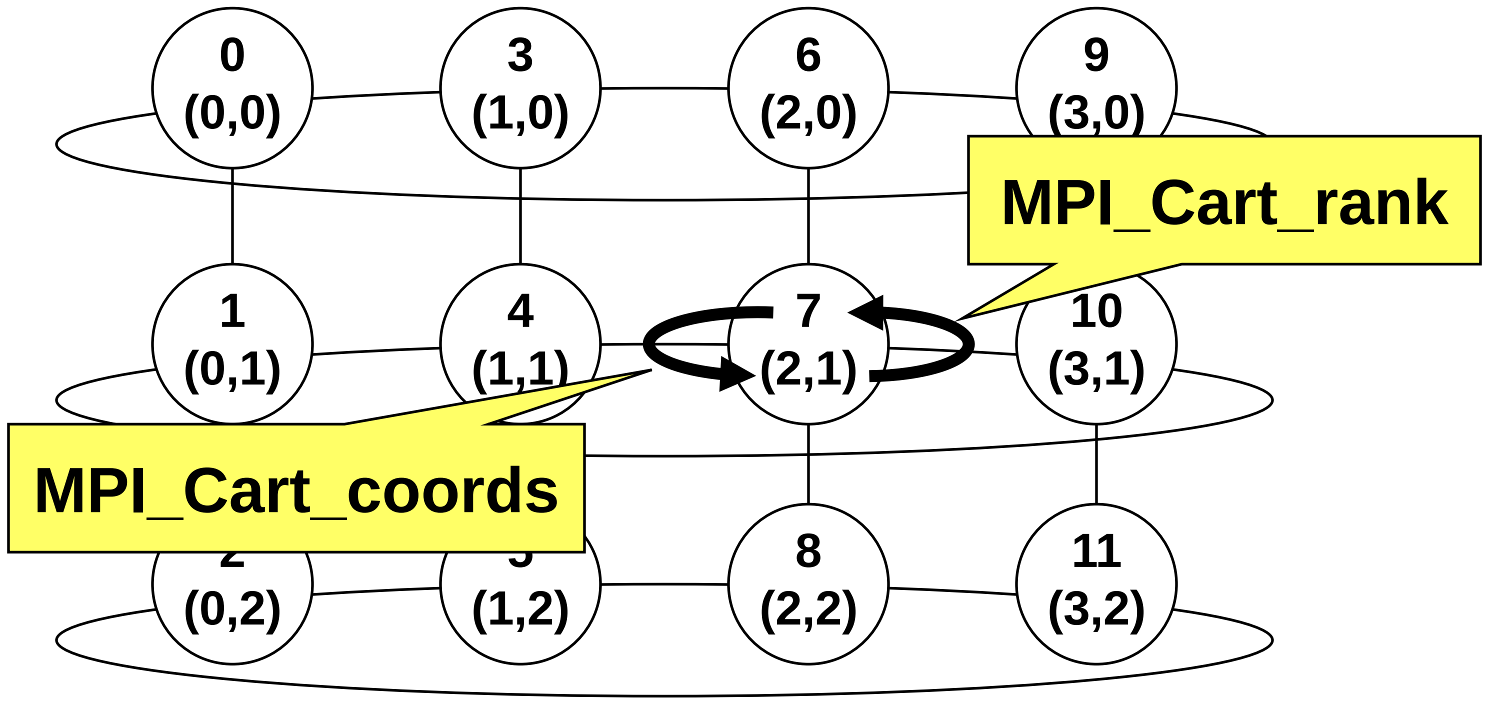
- int MPI_Cart_coords(MPI_Comm comm_cart,
  int rank, int maxdims, int **coords*)

Cartesian Mapping Functions

- Mapping process grid coordinates to ranks
- int MPI_Cart_rank(MPI_Comm comm_cart,
  int *coords, int **rank*)



Own coordinates

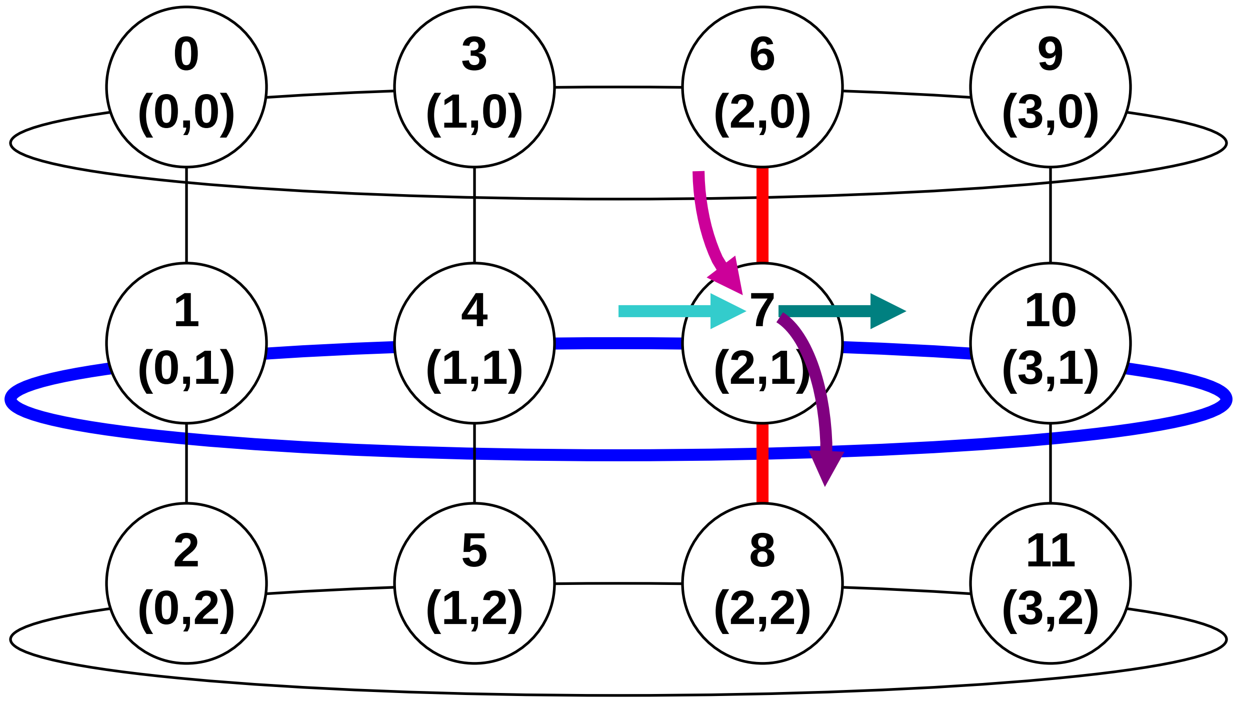


- Each process gets its own coordinates with
  MPI_Comm_rank(comm_cart, ***my_rank***, *ierror*)
  MPI_Cart_coords(comm_cart, **my_rank**, maxdims, ***my_coords***, *ierror*)

Cartesian Mapping Functions?

- Computing ranks of neighboring processes
- int MPI_Cart_shift(MPI_Comm comm_cart, int direction, int disp,
  int **rank_prev*, int **rank_next*)
- Returns MPI_PROC_NULL if there is no neighbor.
- MPI_PROC_NULL can be used as source or destination rank in each communication

MPI_Cart_shift – Example



invisible input argument: **my_rank** in cart

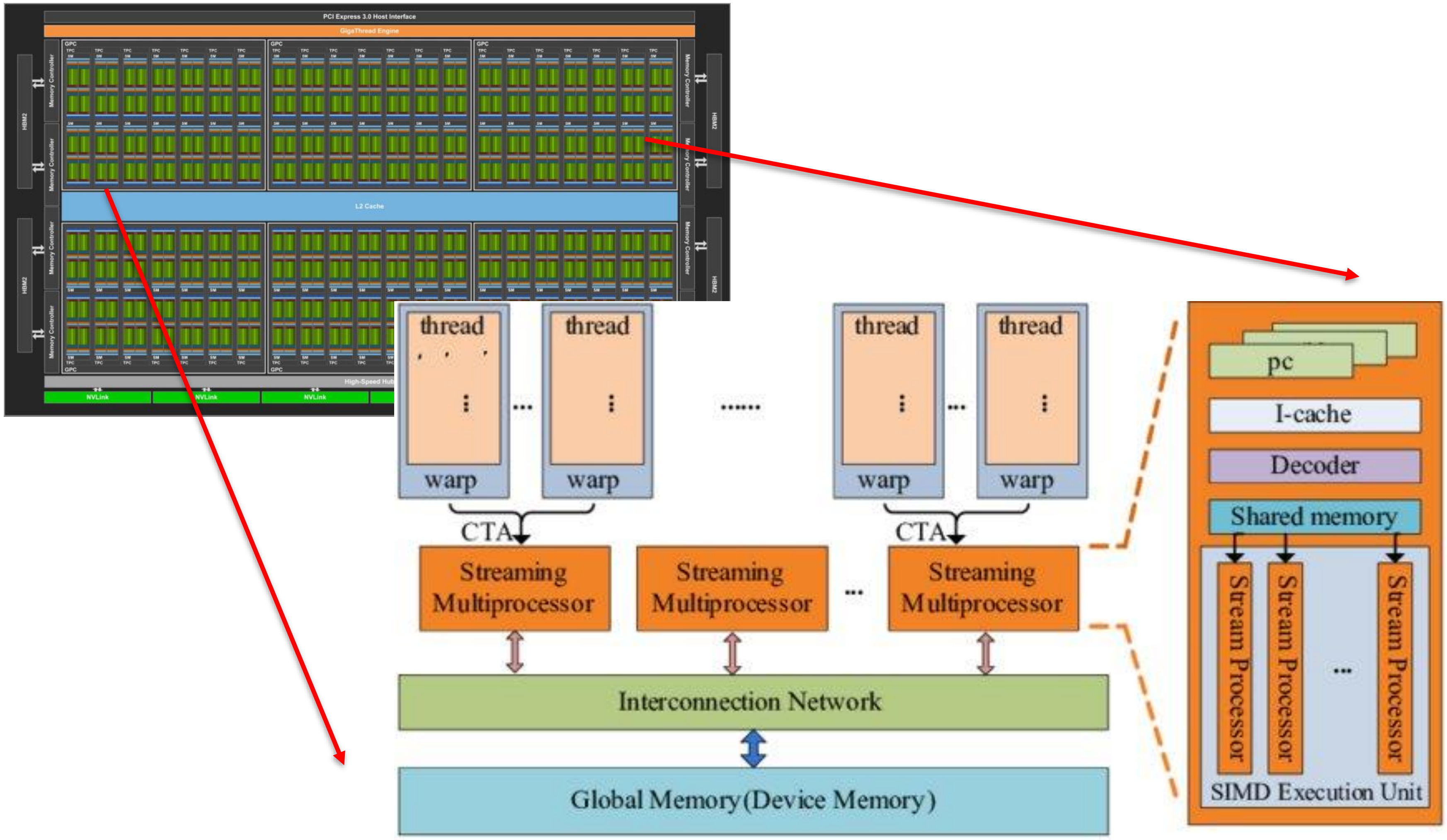
- MPI_Cart_shift(cart, direction, displace, *rank_prev*, *rank_next*, *ierror*)

example on	**0** or	**+1**	**4**	**10**
process rank=**7**	**1**	**+1**	**6**	**8**

Week 10 – Single Instruction Multiple Data (SIMD)

1997	1999	2000	2003	2007	2011
MMX	**SSE**	**SSE2**	**SSE3**	**SSE4**	**AVX**
- 8 64 bit registers - MM0 → MM7 - Shares register with X87 FPU - Only performs integer-based vector operations - Cannot perform integer and FPU operations simultaneously	- 8 128 bit registers, expandable to 16 registers for 64 bit architectures - XMM0 → XMM8, to XMM15 - Performs floating point based vector operations. - Integer based operations carried out using MMX	- Enabled both integer and floating based operations to be performed using the XMM registers	- Horizontal operations - Asymmetric processing	- Additional instructions for vector operations	- 16 256 bit registers - YMM0 → YMM15 - New instructions for vector operations.

Week 11 – Introduction GPGPUs



Final assessment (or final exam) details.

Final Exam Details

1. Date & time: Please refer to your exam timetable.
   - **Duration: 2 hours and 10 minutes**
   - **Session: Please check your exam timetable in Allocate+ for the start time based on your time zone.**
2. Format: eExam with online supervision
   - To find out more about these details, click here.
3. Topics covered in the exam
   - Lecture: Week 1 to Week 11
   - Labs: Week 1 to Week 11 (excluding Week 8)
   - Tutorials: Week 2 to Week 12
4. Questions:
   - Five (5) questions
   - Each question is worth 20 marks
   - Total exam marks: 100, which will be scaled to 50%
   - Answer all questions.
   - No Multiple Choice Questions.
   - No drawing required during the e-Exam.
   - Essay based questions which consist of theoretical and programming content.

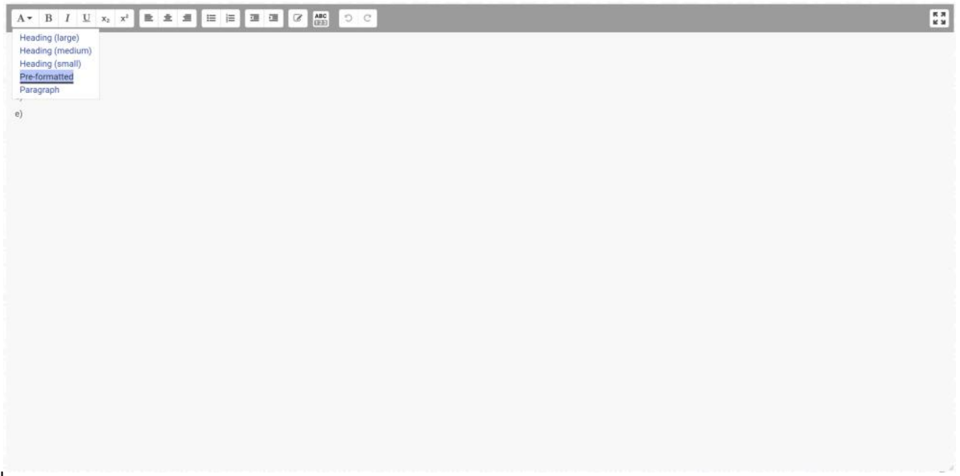
Authorised materials during the final exam

1. Calculators: Not permitted.
2. Working sheets: 10 blank sheets
3. For programming-based questions, you may copy the provided starter code snippet into the answer box in the eExam environment. Before you paste the code, click the 'Paragraph style' icon (the A symbol) and change it to 'Pre-formatted' (Refer to the figure below). This will preserve most of the formatting from the code.

   If you are facing hurdles typing your code in the eExam textbox environment, you may opt to copy and paste the starter code into a simple text editor, complete the code, and then paste the completed code back into the text box in the eExam environment. **However, you are only allowed to open one simple text editor application during the exam.** Allowed simple text editors include:

   - Apple MAC OS: TextEdit
   - Windows OS: Notepad or Notepad++

   You are strictly prohibited from pre-loading code files into the text editor. You are not required to compile your code during the exam. The use of VS Code is also prohibited.



Available function prototypes during the exam

1. The exam includes programming-based questions.
2. The training in C language during the semester would enable you to write a code to solve a problem in the exam.
3. A list of MPI function prototypes which are applicable to the exam questions will be provided during the exam. This should alleviate the hurdles of having to remember MPI functions with lengthy arguments. You may refer to the FIT3143 practice exam in Moodle for sample questions which contain function prototypes.
4. However, for OpenMP, you should remember the OpenMP directives which were covered in the lecture, tutorials and lab.

Contacting exam services

1. Please ensure that you refer the guidelines in sitting for the eExam

2. Please adhere to the eExam rules.

3. For special consideration and alternative arrangements, please refer to details here.

4. To get help and support, please refer to details here.

Final Exam Preparation

1. Refer to the Study Resources page in Moodle.
2. Revise the lecture materials.
3. Practice the lab & tutorial questions.
4. Review your second assignment.
5. Consult the teaching team during the pre-exam Consultations (Swot Vac week).
6. Attempt the practice exam questions in Moodle.

As a kind reminder, please assist us to complete SETU for FIT3143
Link to Monash SETU: https://monash.bluera.com/monash/

You all have been wonderful students. Thank you for enrolling to this unit and we hope you have learned well. We wish you all the best with your studies and work.

Thank you everyone.
FIT3143 Teaching Team, S2/2022