सुस्वागतम्
நல்வரவு
ಸುಸ್ವಾಗತ
సుస్వాగతం
സുസ്വാഗതം
સુસ્વાગતમ
ସୁସ୍ୱାଗତମ
ਸੁਆਗਤਮ
সুস্বাগতম
خوش آمدید

Ministry of Electronics and Information Technology
Government of India

Introduction to Parallel Programming

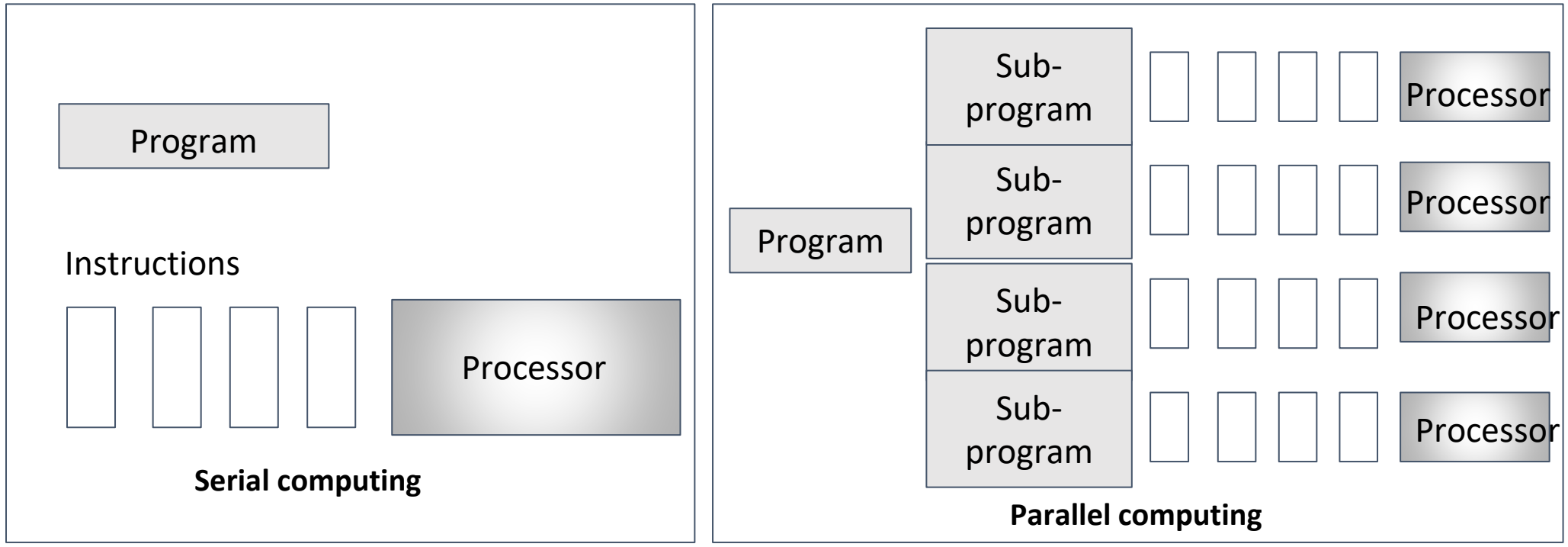
Sowmya shree
HPC-Tech
CDAC Pune

Content

- Introduction to parallel programming
  - What is Parallel Programming?
  - Need of Parallel programming.
  - Why Parallel programming?
- Introduction to parallel hardware.
- General Parallel Computing Terminology
- Process and Threads.
- Demo

Introduction to parallel programming

What is parallel programming?



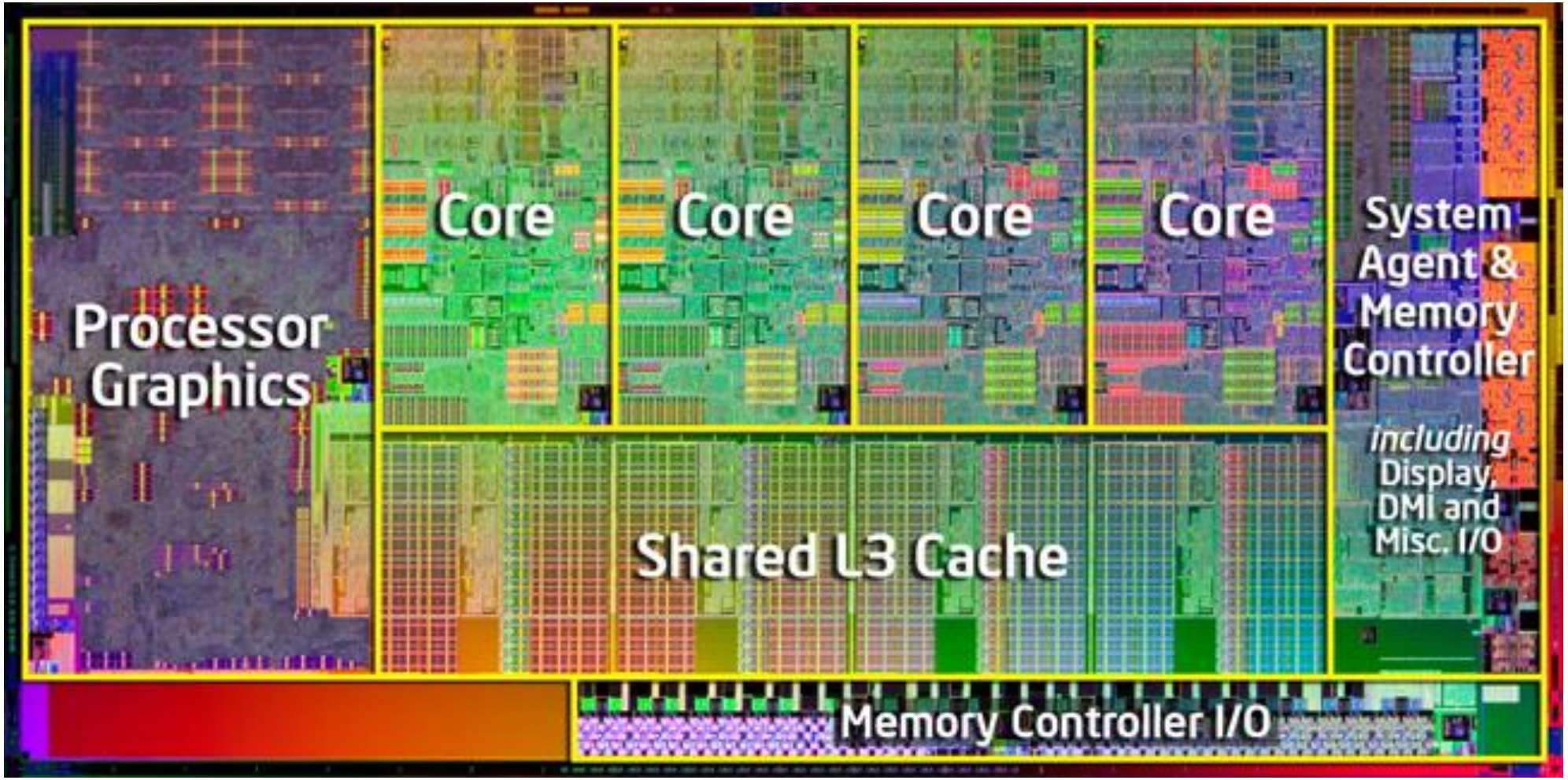
- Need of parallel programming.
  - The exponential growth of processing and network speeds means that parallel architecture isn’t just a good idea; it’s necessary.
  - Dual-core, quad-core, 8-core, and even 56-core chips are all examples of parallel computing. So, while parallel computers aren’t new, here’s the rub: new technologies are cranking out ever-faster networks, and computer performance has grown 250,000 times in 20 years.
  - For instance, in just the healthcare sector, AI tools will be rifling through the heart rates of a hundred million patients, looking for the telltale signs of A-fib or V-tach and saving lives. They won’t be able to make it work if they have to plod along performing one operation at a time

Introduction to parallel programming

Why parallel programming?

- PARALLEL COMPUTING MODELS THE REAL WORLD.
- SAVES TIME
- SAVES MONEY
- SOLVE MORE COMPLEX OR LARGER PROBLEMS
- MAKE BETTER USE OF UNDERLYING PARALLEL HARDWARE

Introduction to parallel hardware.



General Parallel Computing Terminology

- Node

  A standalone "computer in a box." Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.

- CPU

  Contemporary CPUs consist of one or more cores.

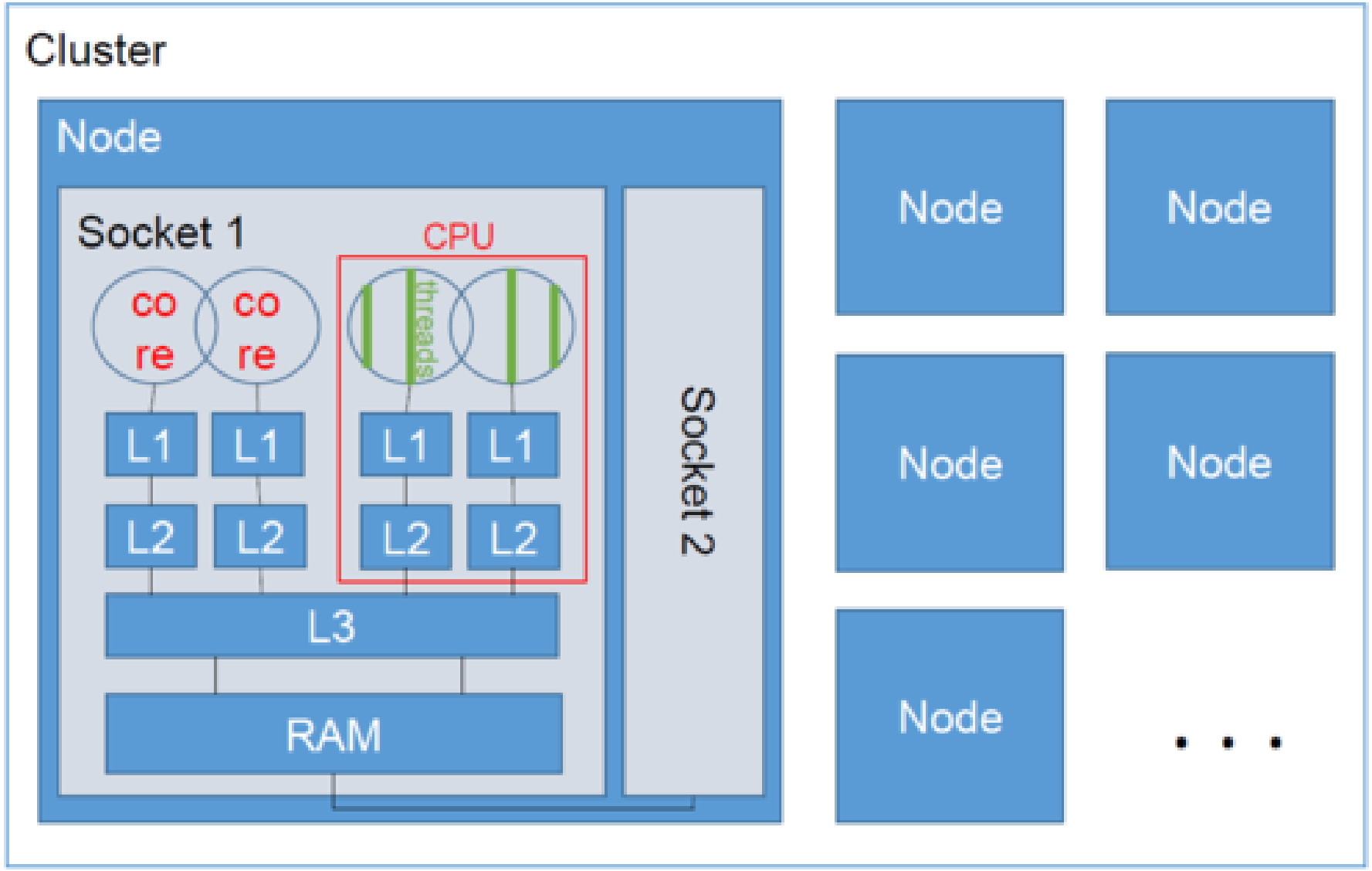
- Process

  A process is an instance of a program that is being executed or processed

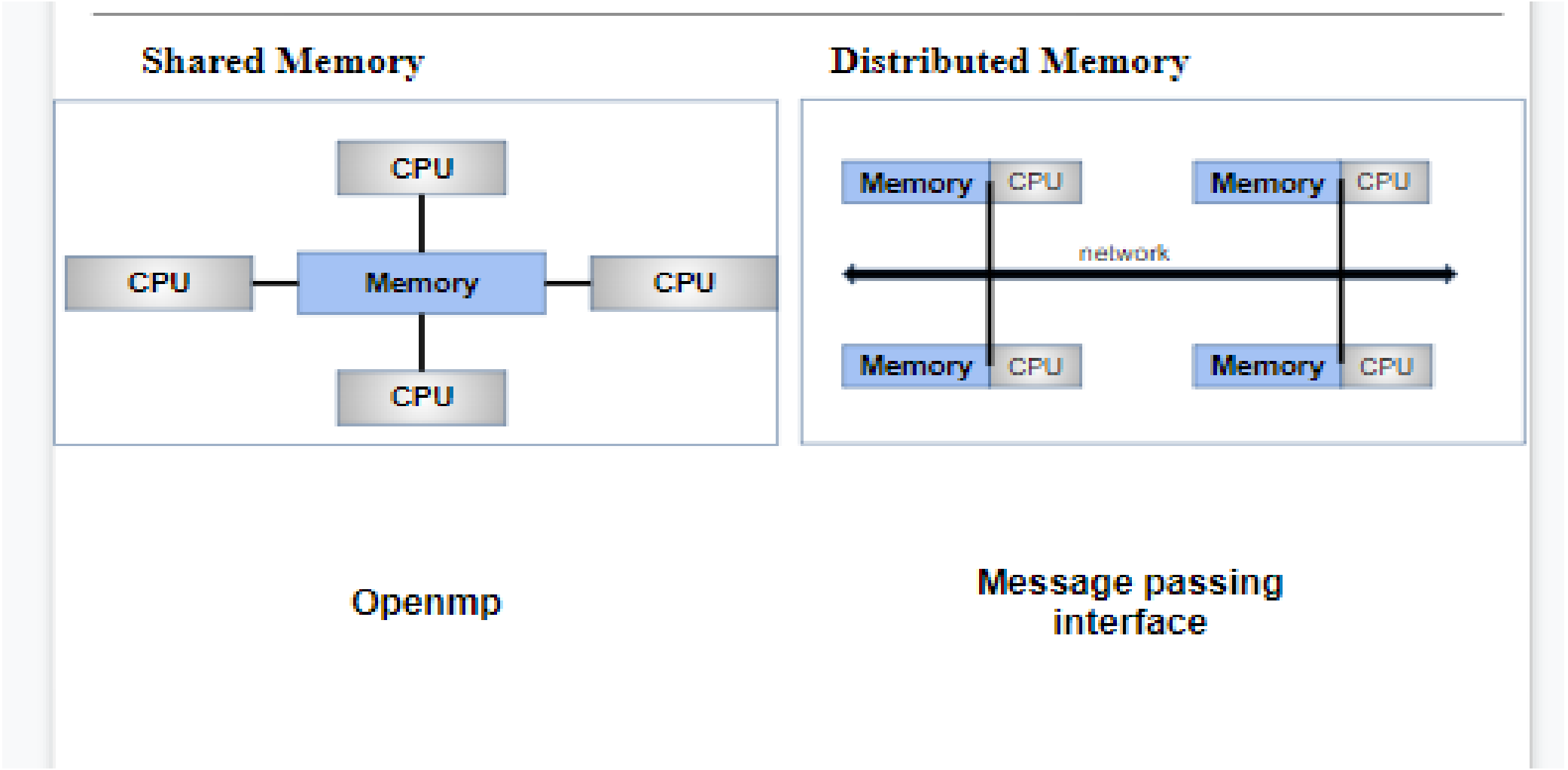
- Thread

  Thread is a segment of a process or a lightweight process that is managed by the scheduler independently

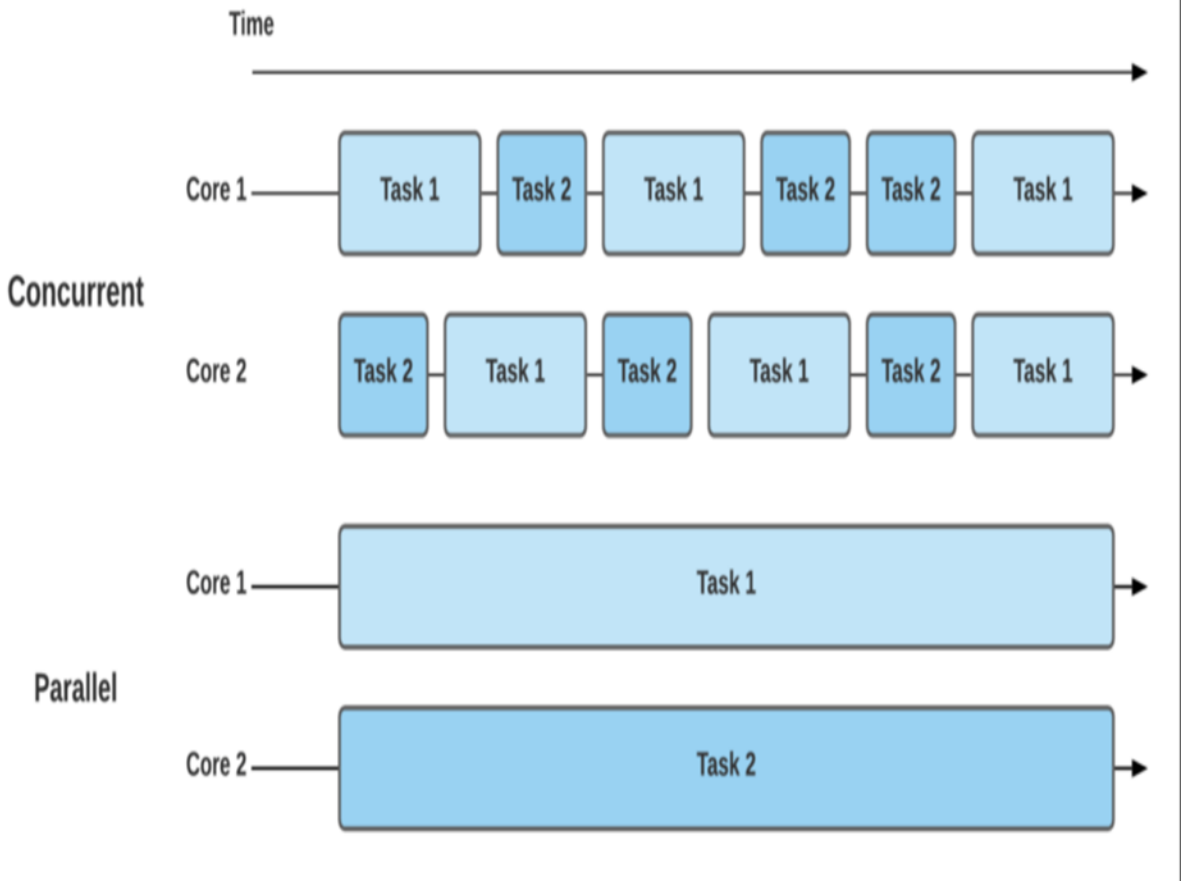
General Parallel Computing Terminology



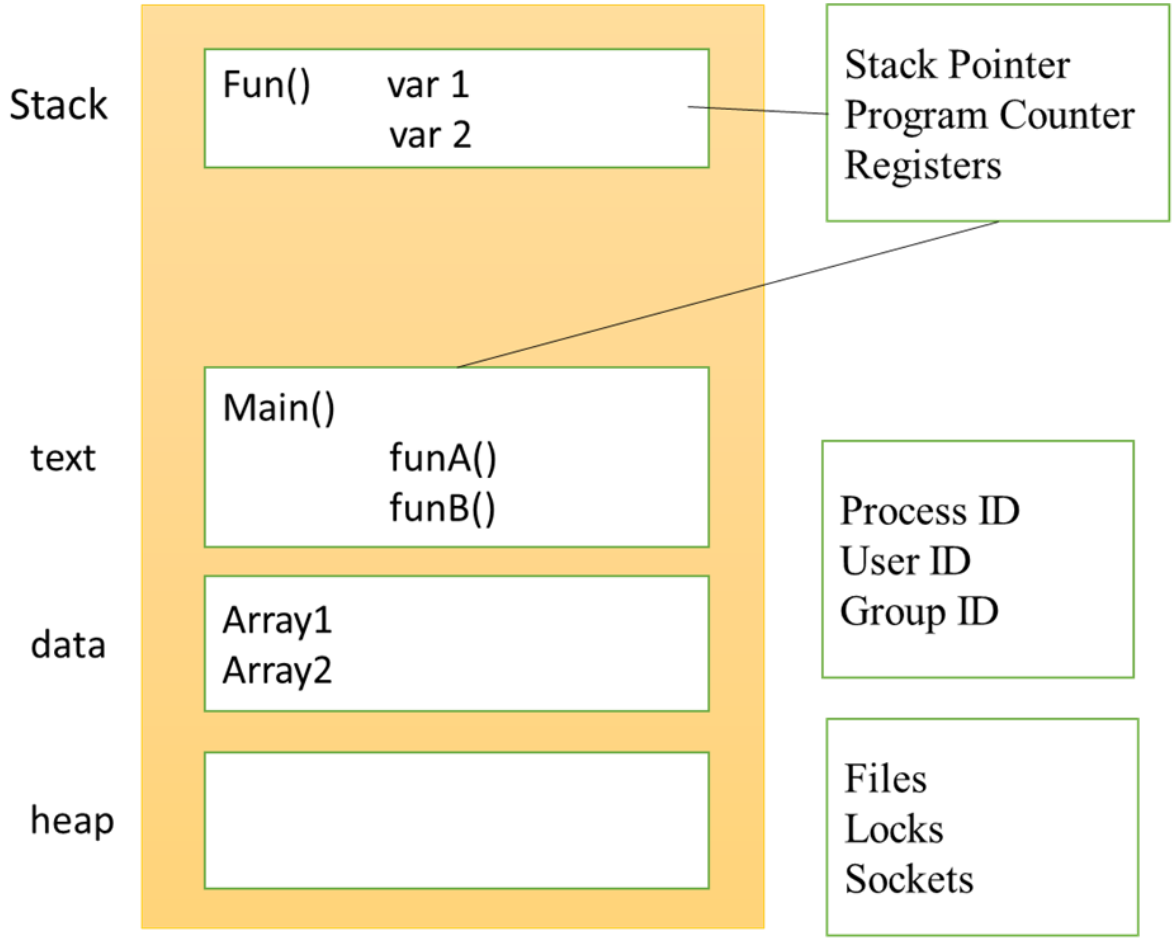
General Parallel Computing Terminology



- Concurrency and Parallelism
  - Concurrency: A condition of a system in which multiple tasks are logically active at one time
  - Parallelism: A condition of a system in which multiple tasks are actually active at one time



- How program execute in memory?

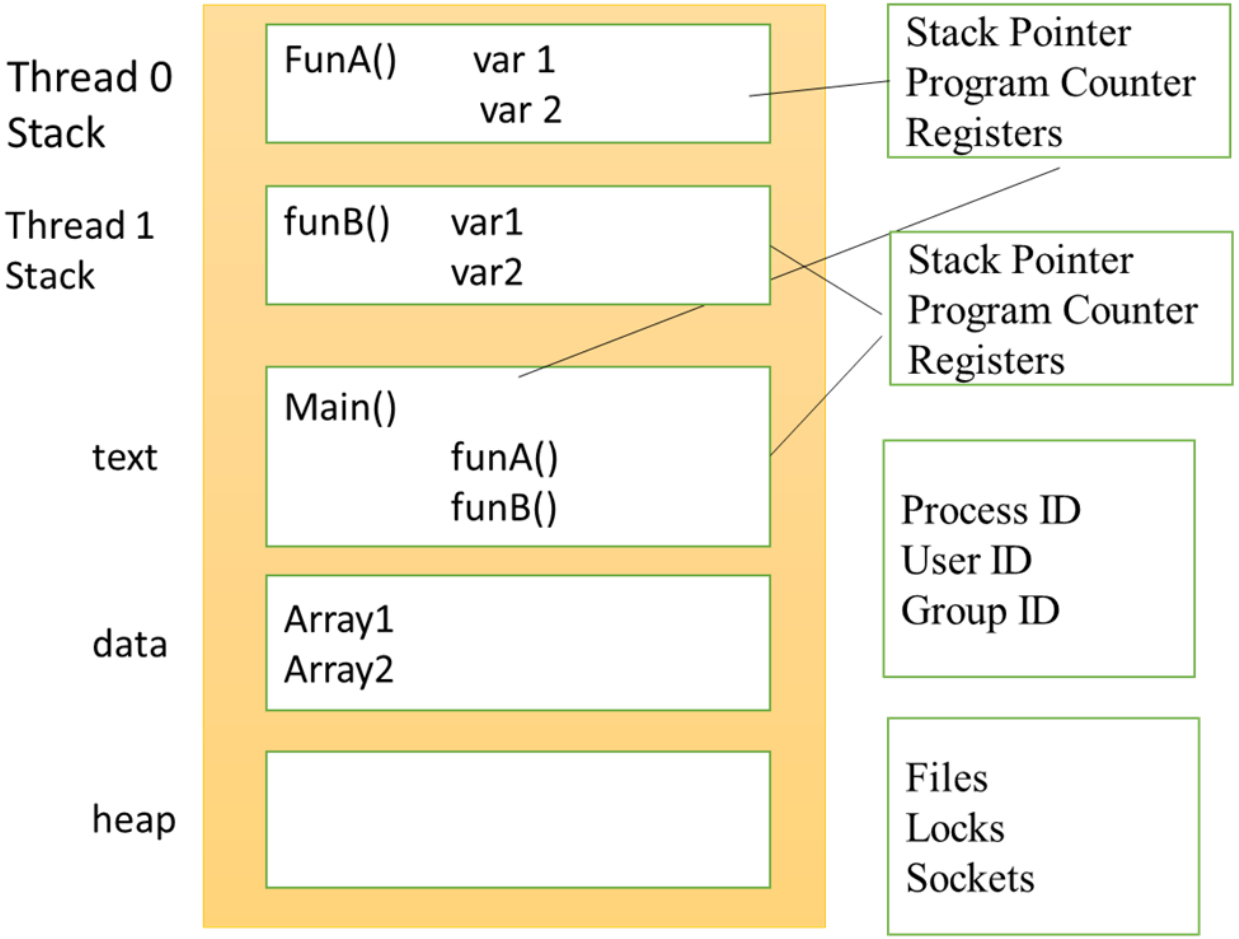


Process:

- An instance of a program execution.

- The execution context of a running program … i.e. the resources associated with a program's execution.

- How program execute in shared memory?



Threads:

- Threads are "light weight processes"

- Threads share Process state among multiple threads … this greatly reduces the cost of switching context.

Demo Time

Hardware information

$lscpu or cat /proc/cpuinfo

```
(base) [cdacapp@login03 ~]$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    20
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
Stepping:              7
CPU MHz:               999.908
CPU max MHz:           3900.0000
CPU min MHz:           1000.0000
BogoMIPS:              5000.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              28160K
NUMA node0 CPU(s):     0-19
NUMA node1 CPU(s):     20-39
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts ac
pi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_goo
d nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sd
bg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
 lahf_lm abm 3dnowprefetch epb cat_l3 cdp_l3 invpcid_single intel_ppin intel_pt ssbd mba ibrs ibpb stibp ibrs
_enhanced tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm
cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd avx512bw avx512vl xsaveopt xsavec xge
tbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts pku ospke avx512_vnni md_clear
 spec_ctrl intel_stibp flush_l1d arch_capabilities
(base) [cdacapp@login03 ~]$
```

Process and Thread information

$top –u <username>

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
43243	cdacapp	20	0	167096	3204	1664	R	1.0	0.0	0:00.09	top
28610	cdacapp	20	0	6498508	296596	1560	S	0.0	0.1	0:09.44	julia
28611	cdacapp	20	0	6493920	300736	1564	S	0.0	0.1	0:08.59	julia
28612	cdacapp	20	0	6569328	327268	1592	S	0.0	0.1	0:08.79	julia
28613	cdacapp	20	0	6565220	330992	1568	S	0.0	0.1	0:09.04	julia
42747	cdacapp	20	0	167760	2724	1208	S	0.0	0.0	0:00.04	sshd
42748	cdacapp	20	0	121768	4188	1728	S	0.0	0.0	0:00.18	bash

An Operating system’s perspective

- Supports multi-core
- Perceives each core as a separate processor
- Scheduler maps threads/process to different cores

Multi-core CPU

- Multi-core processor are MIMD where different cores execute different threads (MI), operating on different part of memory (Multiple data).

- Multi-core is a shared memory multiprocessor (SMP) where all cores share memory

Thank you

Introduction to OPENMP

Open Multi Processing

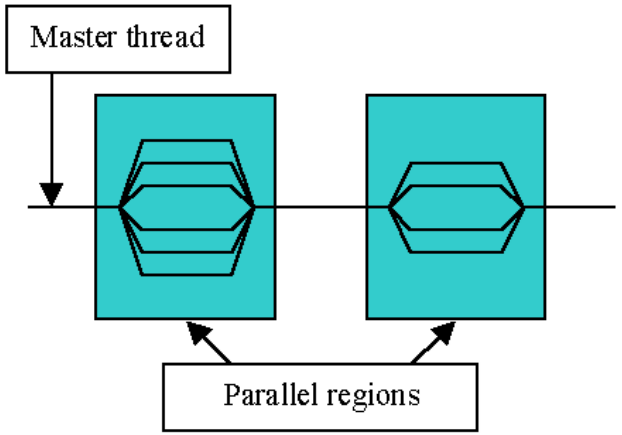
Content

- Introduction to openmp
  - What is OpenMP?
  - History of OpenMP
  - Why openmp
- OpenMP Programming Model
- OpenMP Stack
- Hello world in openmp
  - Basic Syntax
  - Hello world c program
  - Compile and execution

Introduction to openmp

What is OpenMP?

- An Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared memory parallelism
- Comprises three primary API components
  - Compiler Directives
  - Runtime Library Routines
  - Environment Variables
- Portable
  - The API is specified for C/C++ and Fortran
  - Has been implemented for most major platforms including Unix/ Linux platforms and Windows

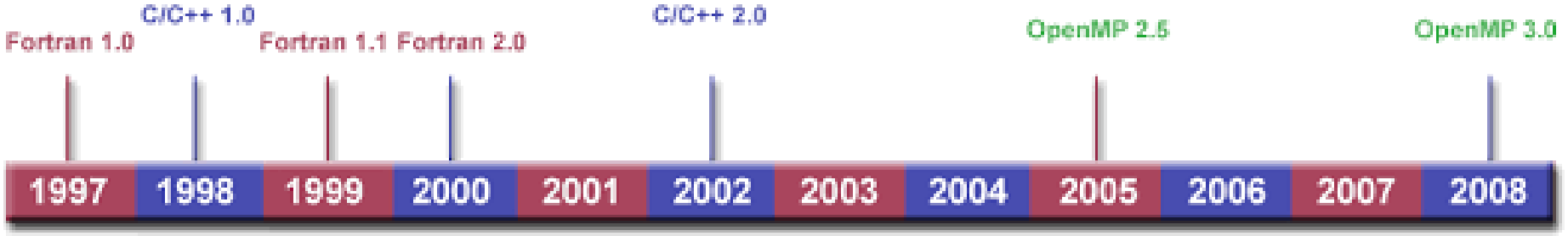


What is OpenMP? (cont.)

- Standardized
  - Jointly defined and endorsed by a group of major computer hardware and software vendors
  - Expected to become an ANSI standard later???
- What does OpenMP stand for?
  - Short version: Open Multi-Processing
  - Long version: Open specifications for Multi-Processing via collaborative work between interested parties from the hardware and software industry, government and academia

Introduction to openmp

History of OpenMP



Version 1.0 : OpenMP for Fortran 1.0, in October 1997

Version 1.0 : OpenMP for C/C++, in October 1998

Version 2.0 : OpenMP for Fortran 1.1, in 1999

Version 2.0 : OpenMP for C/C++, in 2000

Version 2.0 : OpenMP for C/C++, in 2002

Version 2.5 : OpenMP for C/C++/Fortran, in 2005

History of OpenMP(cont.)

- Up to version 2.0, OpenMP primarily specified ways to parallelize highly regular loops, as they occur in matrix-oriented numerical programming,

- In Version 3.0 Included features like concept of tasks and the task construct.

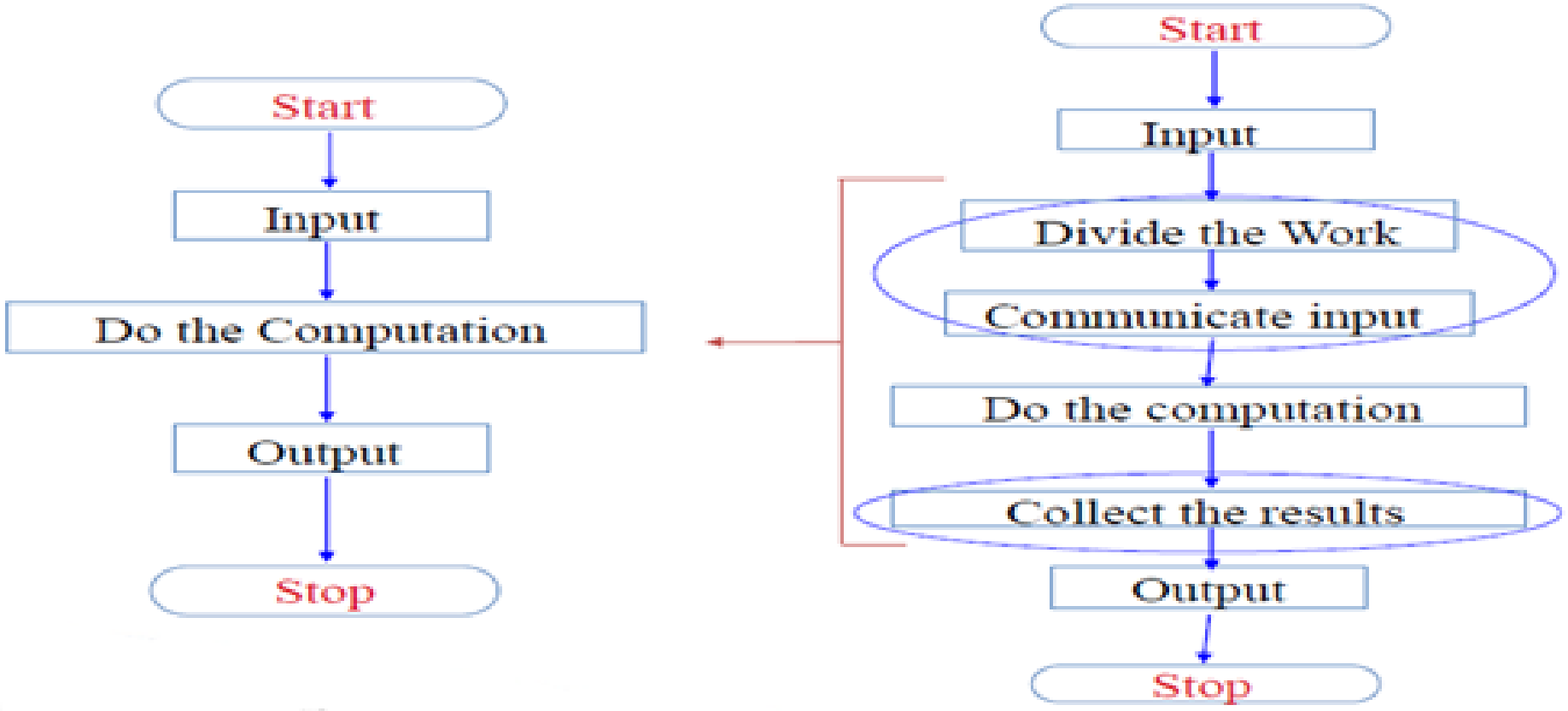
- In version 4.0 openmp improved following features:
- support for accelerators, atomics, error handling, thread affinity; tasking extensions; user defined reduction, SIMD support.

Introduction to openmp

Why Openmp?

- More efficient
- Hides the low-level details.
- OpenMP has directives that allow the programmer to:
  - specify the parallel region
  - specify whether the variables in the parallel section are private or shared
  - specify how/if the threads are synchronized
  - specify how to parallelize loops
  - specify how the works is divided between threads (scheduling)

Execution flow of Parallel Systems

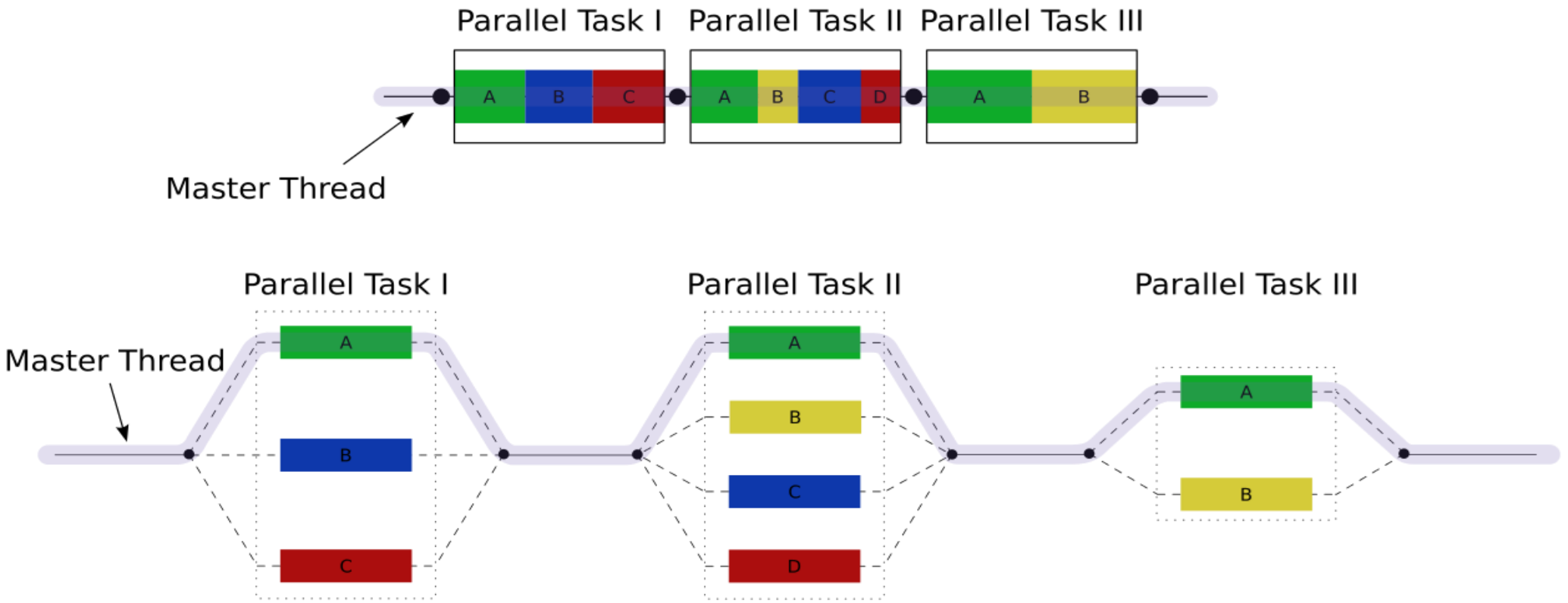


OpenMP Programming Model

- Shared memory, thread-based parallelism
  - OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm.
  - A shared memory process consists of multiple threads.
- Explicit Parallelism
  - OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization.
- OpenMP uses the fork-join model of parallel execution.
- Compiler directive based
  - Most OpenMP parallelism is specified through the use of compiler directives which are imbedded in C/C++ or Fortran source code.

OpenMP Programming Model

Fork–join model:

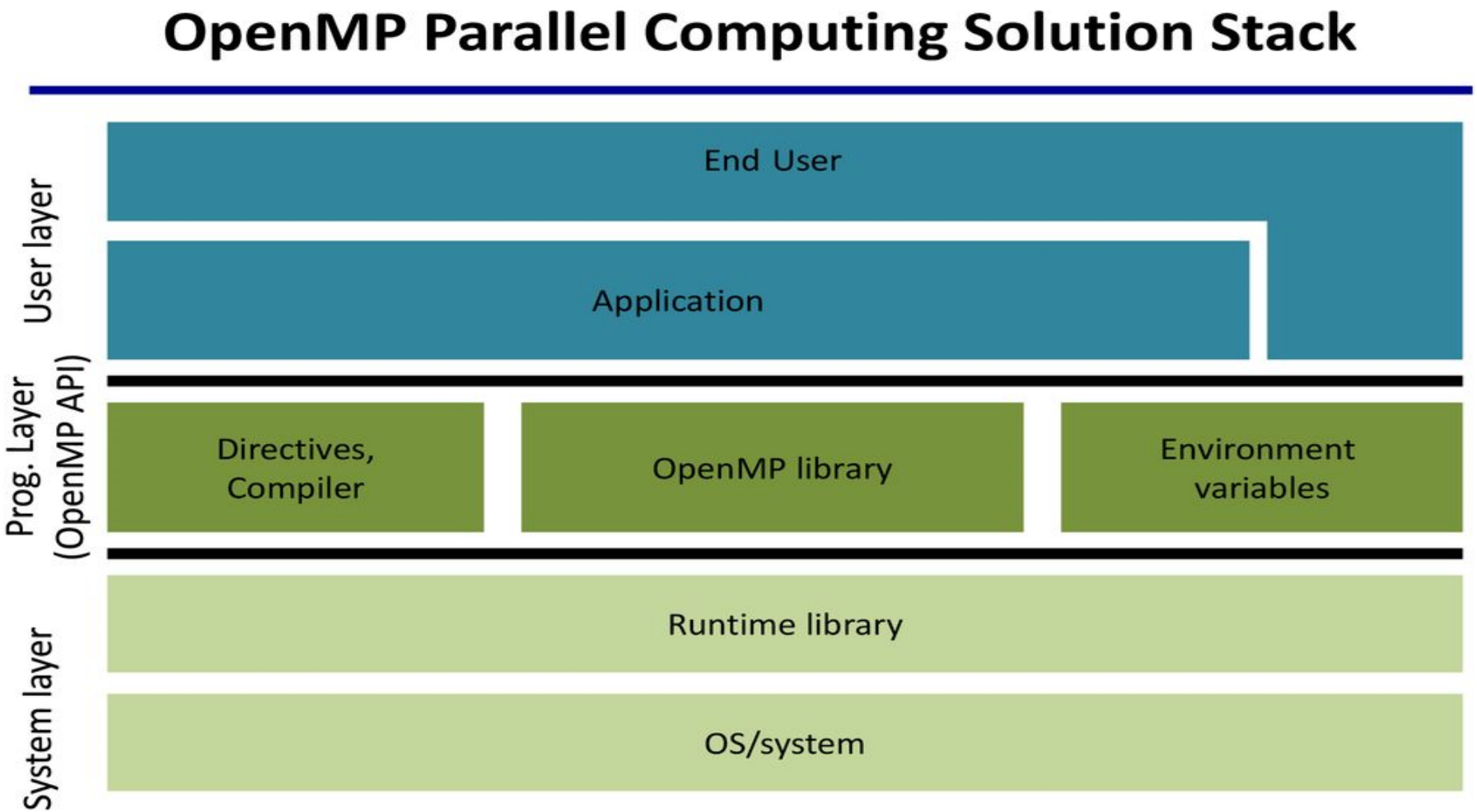


OpenMP Programming Model

Fork–join model: (cont.)

- All OpenMP programs begin as a single process: the master thread. The **master thread** executes sequentially until the first parallel region construct is encountered.
- **FORK**: the master thread then creates a **team** of parallel threads
- The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the team threads.
- **JOIN**: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread

OpenMP Stack



Hello world in openmp

Basic Syntax

- Function prototypes and types in the file:

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

- Most of the constructs in OpenMP are compiler directives.

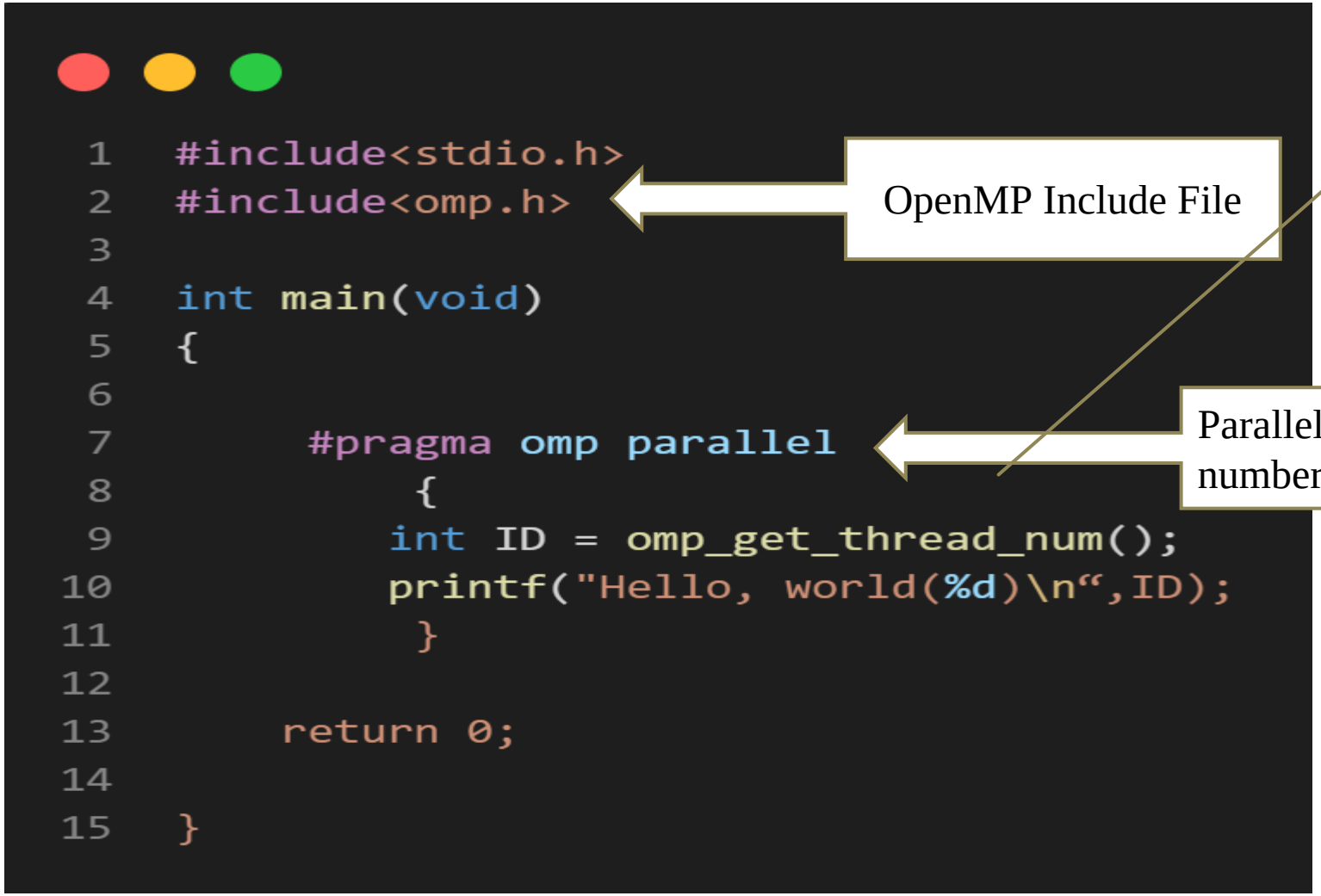
```
#pragma omp construct [clause [clause]…]
        {
        //..Do some work here
        }
        //end of parallel region/block
```

- Example:

```
#pragma omp parallel num_threads(4)
```

Hello world in openmp

OpenMP Hello World program using C



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

int main(void)
{

      #pragma omp parallel
          {
          int ID = omp_get_thread_num();
          printf("Hello, world(%d)\n“,ID);
           }

     return 0;

}
```

Your first openMP program

Check your system support: locate omp.h

Compilation: g++ -fopenmp hello.c

Execution: ./a.out

Flags:

GNU: -fopenmp for Linux, Solaris, AIX, MacOS, Windows.

IBM: -qsmp=omp for windows, AIX and Linux.

Sun: -xopwnmp for Solaris and Linux

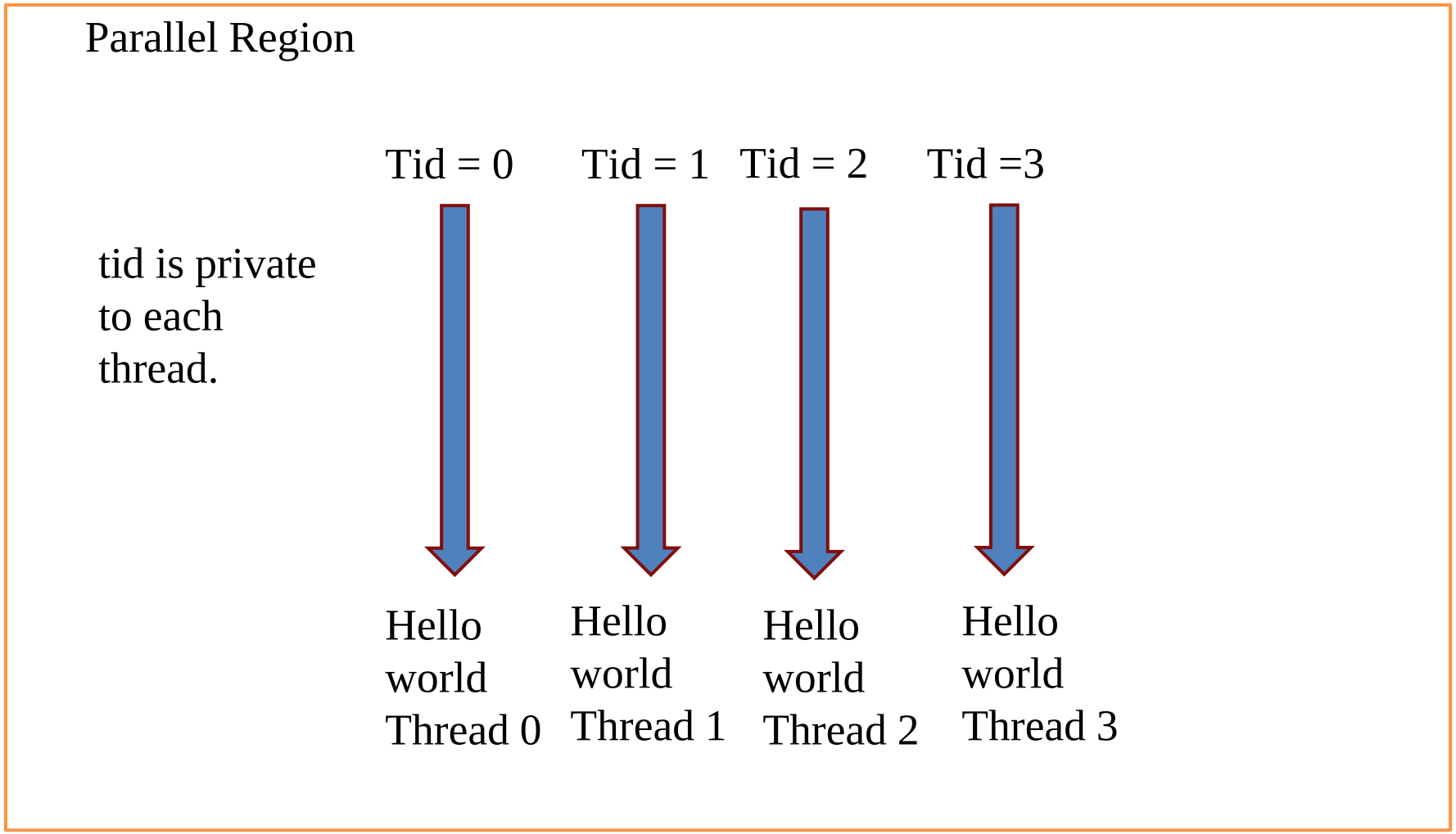
Intel: -qopenmp on Linux or Mac, or -Qopenmp on windows

PGI: -mp

Parallel Software Models

Programing Models

Number of threads 4



Compile and execution

Compile program in OpenMP

- Set number of threads:

  Using shell

  $ export OMP_NUM_THREADS=4

  Inside program before parallel region.

  omp_set_num_threads(4);

- For GNU C compiler:

  $ gcc -fopenmp HelloWorld.c -o Hello

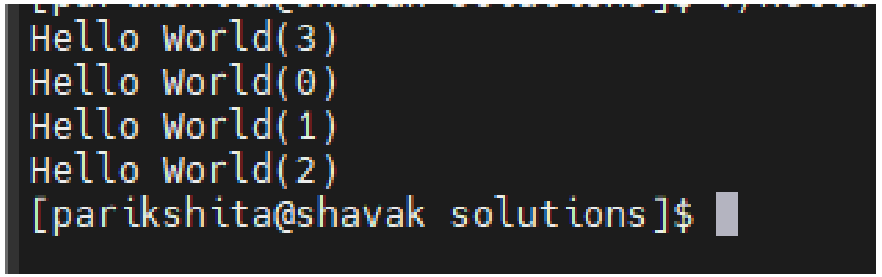
  $./Hello

- For Intel compiler:

  $ icc -qopenmp HelloWorld.c -o Hello

  $./Hello

Output of hello world program with 4 threads:



Thank you