The 17th Annual International Symposium on High Performance Computing Systems and Applications

17ième symposium international sur les applications et systèmes de calcul de haute performance

Introduction to Parallel Computing

Presented by

Carol Gauthier

Centre de Calcul Scientifique (CCS)
Université de Sherbrooke

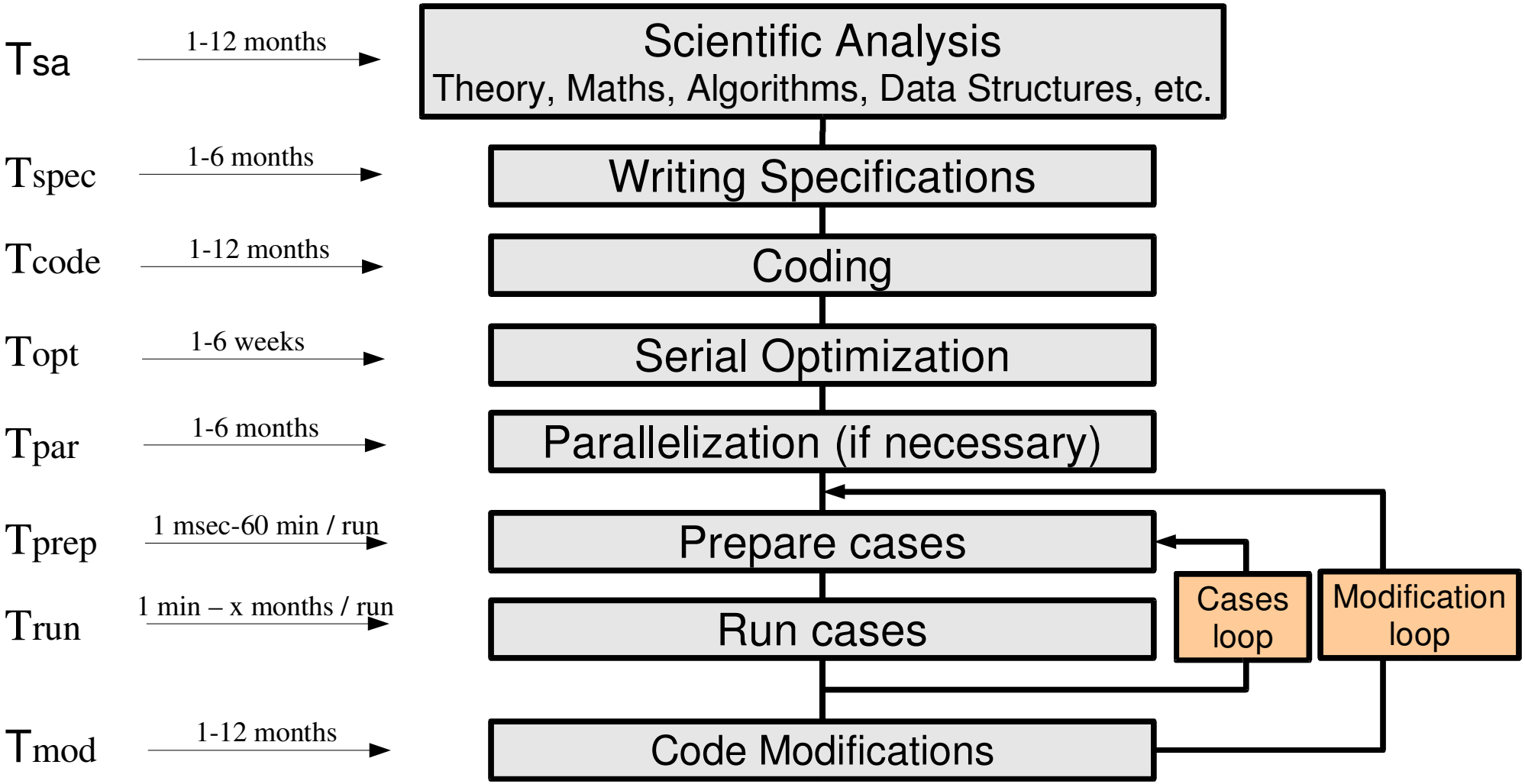
Réseau québécois de calcul de haute performance (RQCHP)

Summary of the presentation

1) Overall Performance Strategy
2) Goals of Parallel Computing
3) Parallel Computer Architectures
4) Parallelization Strategy
5) Shared Memory Parallel Programming
6) Distributed Memory Parallel Programming

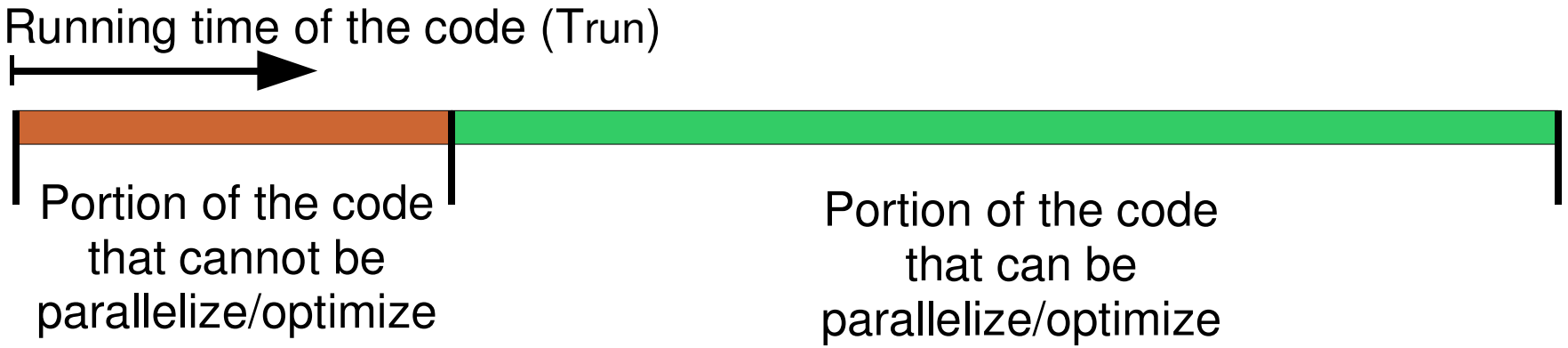
Overall Performance Strategy

Scientific Application Developpment Steps

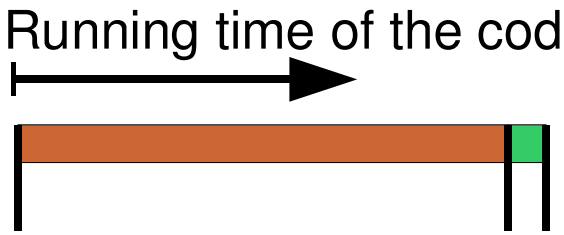


Overall Performance Strategy

Amdahl's Law



Even after an excellent parallelisation or optimisation...

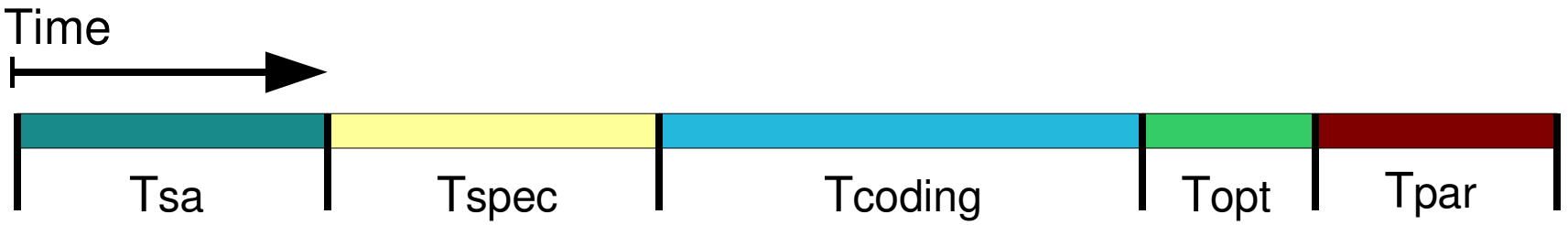


.... this code will never run more than 3 times faster !!!

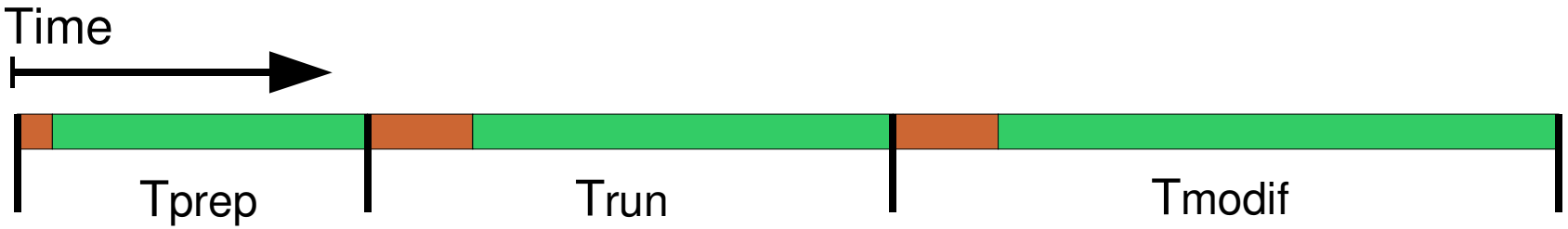
Overall Performance Strategy

Overview of Development and Production Time

Development



Production



Goals of Parallel Computing

Larger and Longer Cases

Solve larger cases
Paralelisation will allow to alleviate the memory limitations of a single processor (or a single node or workstation).

Solve each problem case faster
Parallelisation will allow to solve each case of your problem within a reasonnable amount of time.

"If each case of your problem is solved within the memory of a node, and within a reasonnable amount of time using a single processor, there is no need to parallelize!"

Parallel Computer Architectures

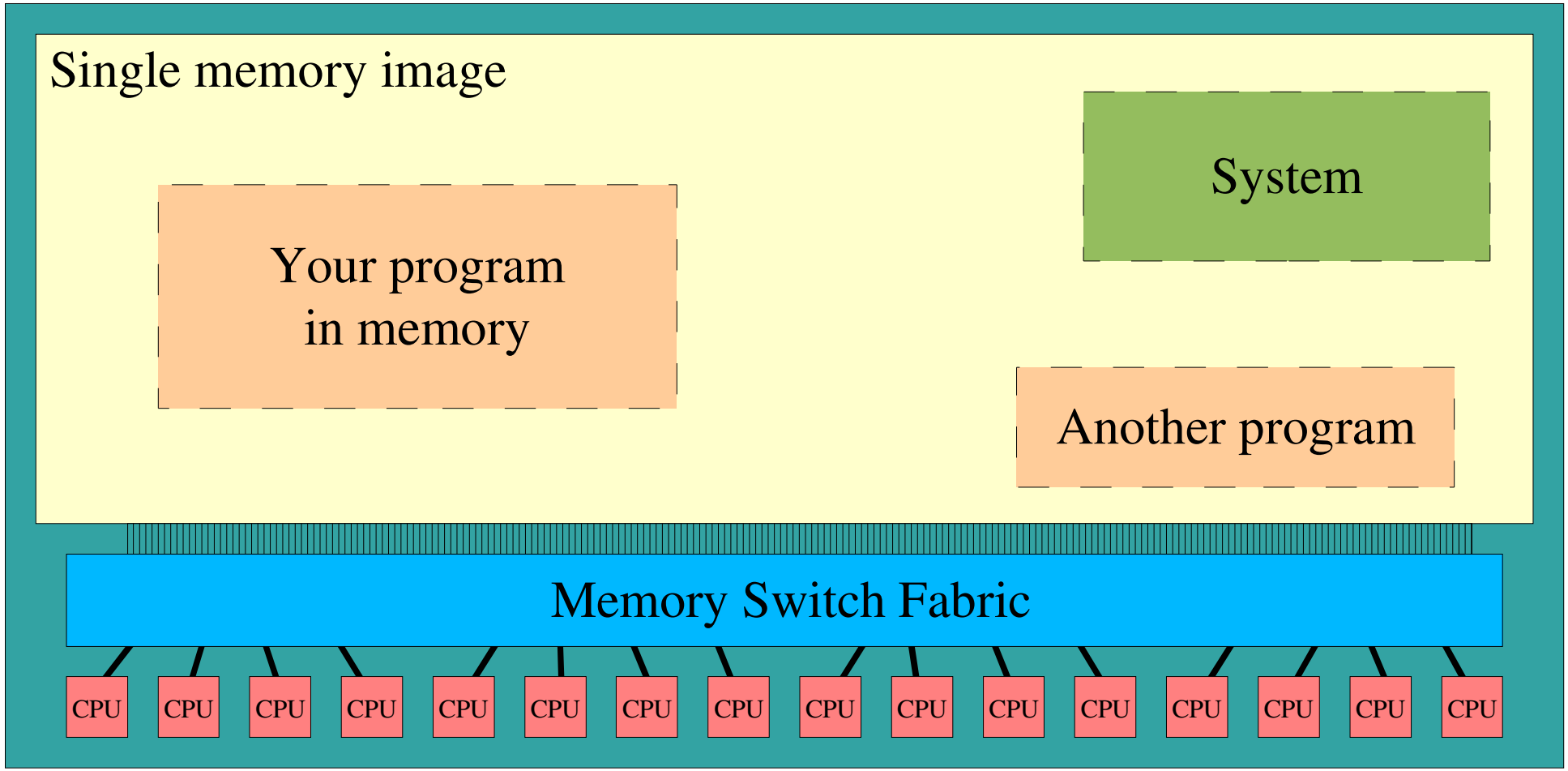
1) Shared Memory Architectures

2) Distributed Memory Architectures: Clusters

3) Hybrid Architectures

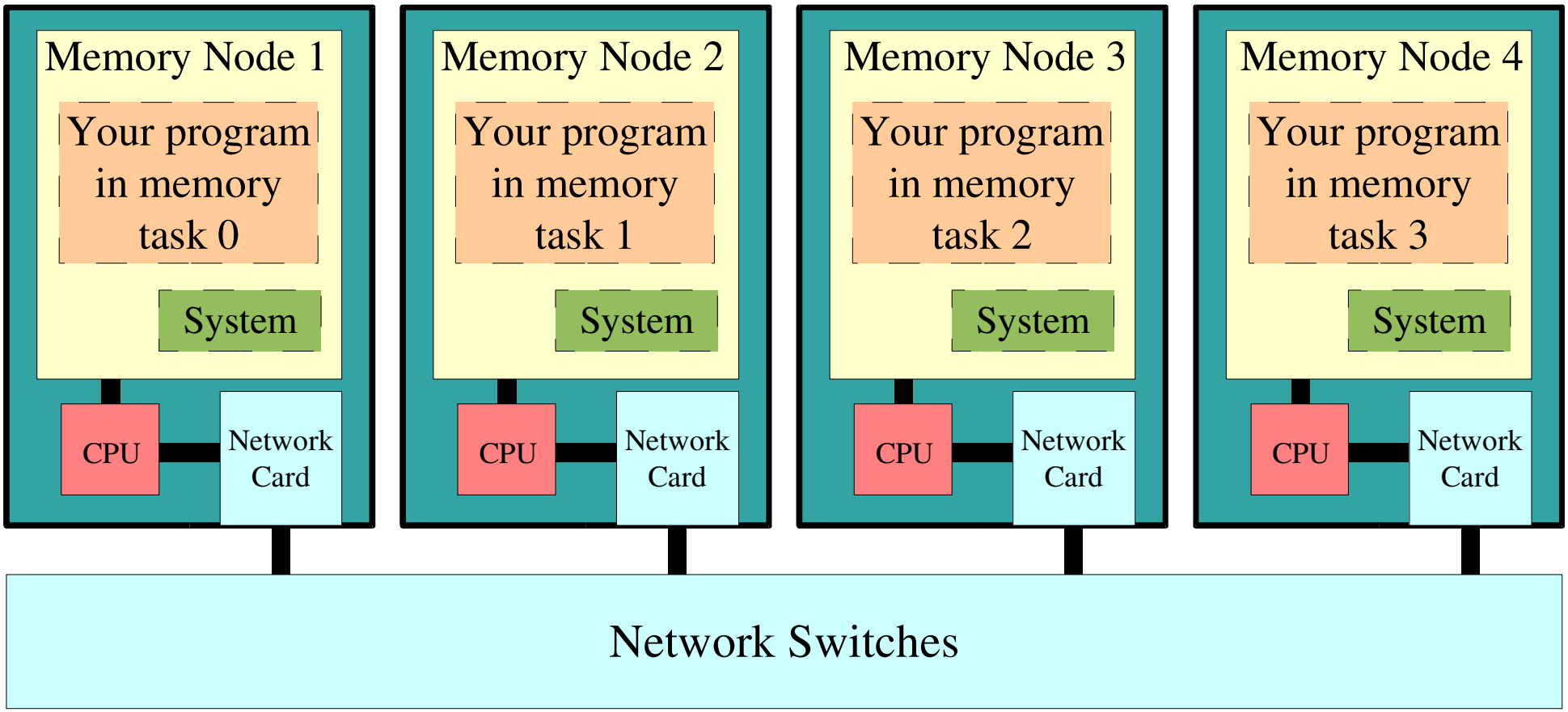
Parallel Computer Architectures

Shared Memory Architectures



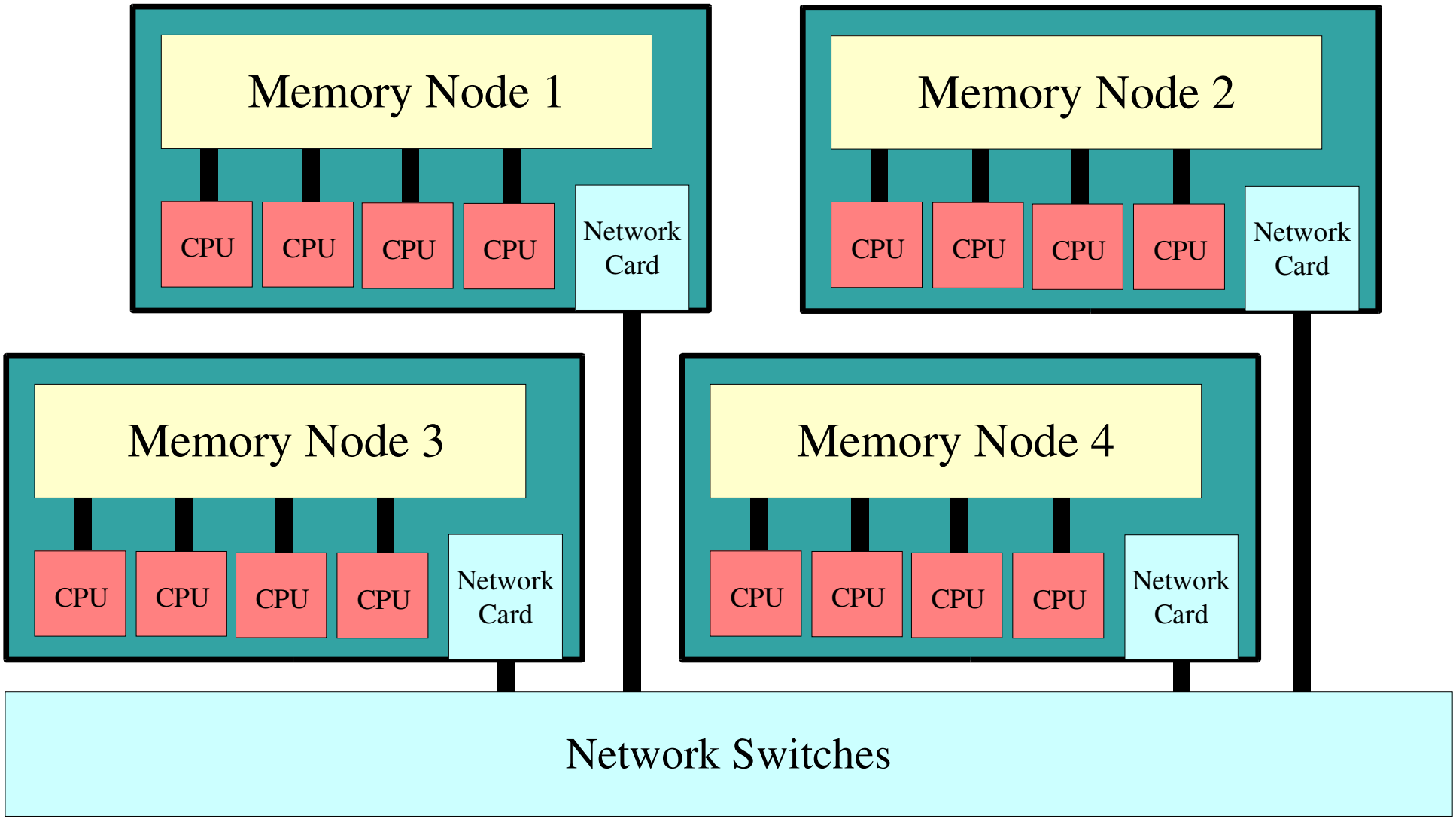
Parallel Computer Architectures

Distributed Memory Architectures: Clusters

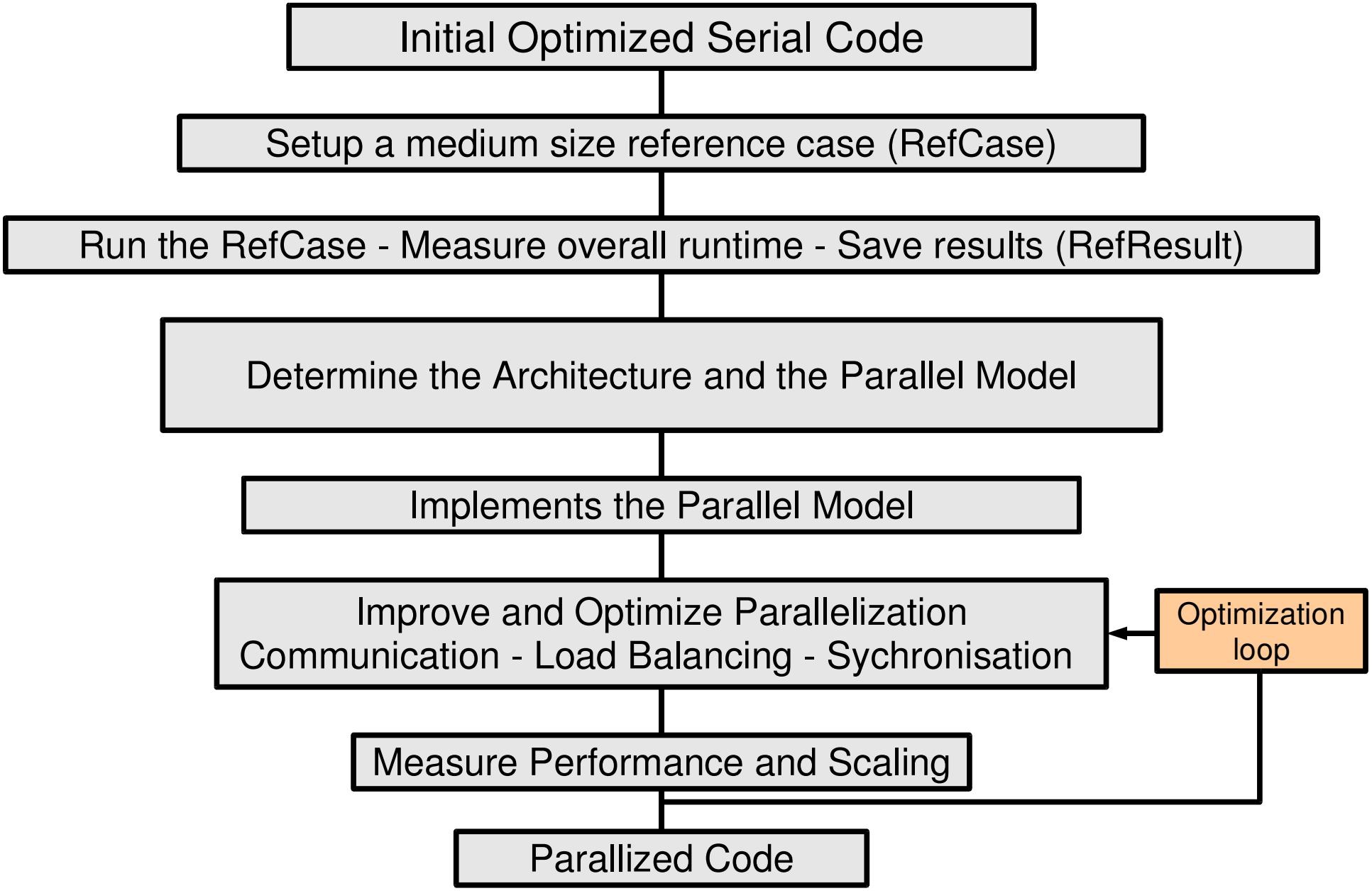


Parallel Computer Architectures

Hybrid Architectures



Parallelization Strategy



Shared Memory Parallel Programming

1) Shared Memory Programming Interfaces
2) OpenMP: Accessing Multiple Threads
3) OpenMP: Directives
4) OpenMP: Subroutines and Functions
5) OpenMP: Environment Variables
6) OpenMP: Examples
7) OpenMP: False Sharing!

Shared Memory Parallel Programming

Shared Memory Programming Interfaces

OpenMP

The standardized API for multithreaded parallel applications
(**http://www.openmp.org, http://www.compunity.org)**

A set of directives and functions
C$OMP PARALLEL, #pragma omp parallel ,etc...

Available for Fortran / C / C++

Simple to use compare to other API (POSIX Threads)

Shared Memory Parallel Programming

OpenMP: Accessing Multiple Threads

Serial program

> One instance of the program <

> One thread available <

OpenMP Parallel program

> One instance of the program <

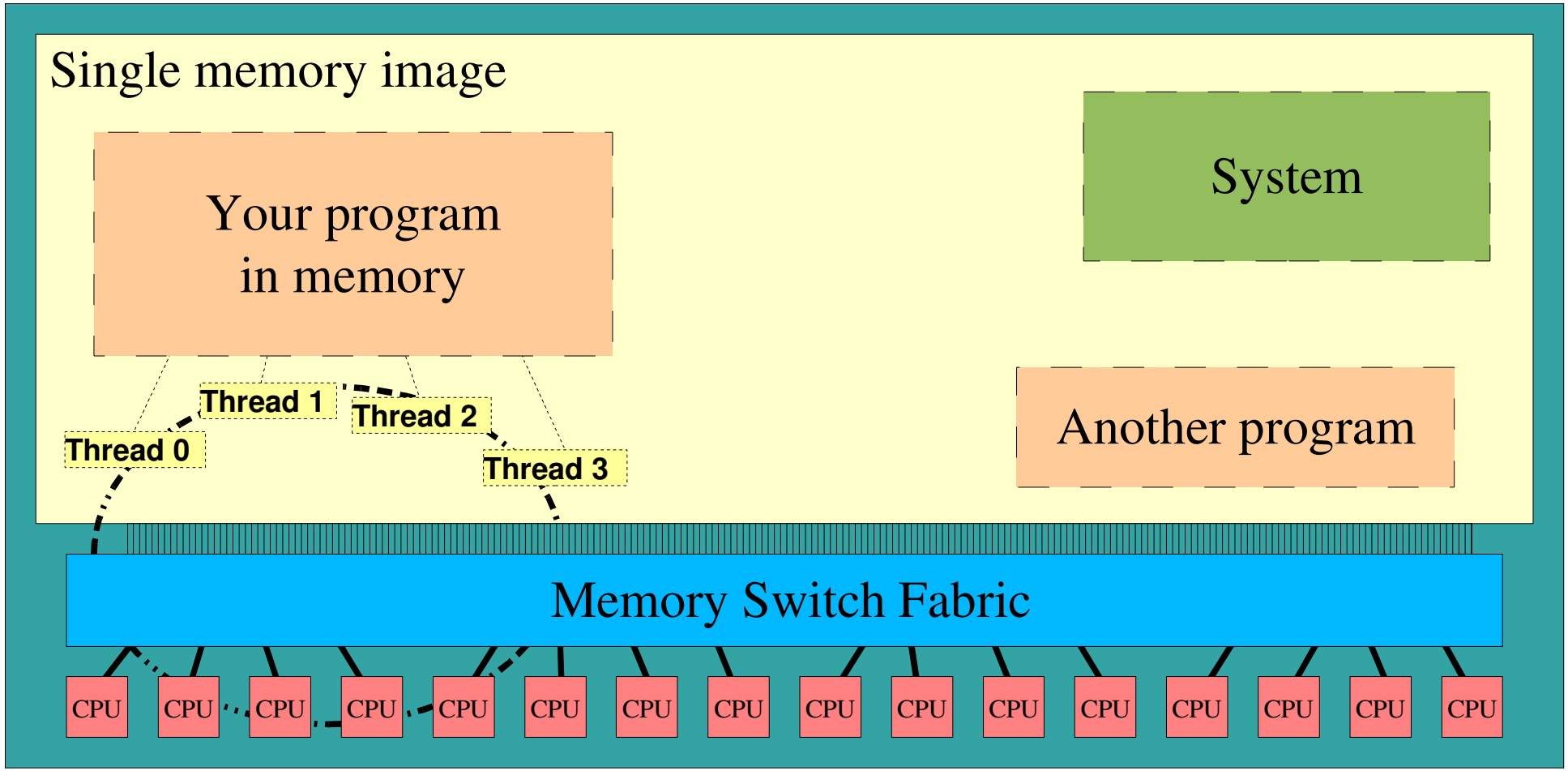
> Several threads available <

> A master thread spawns other threads <

A thread is a sequence of operations that is executed by the system. There are no priority or dependance among threads, so the order in which they will be executed is not known! Two threads belonging to a single instance of a program, could then be executed at the same time if several CPU are available to the system.

Parallel Computer Architectures

OpenMP: Accessing Multiple Threads



Shared Memory Parallel Programming

OpenMP: Directives

In Fortran (fixed source form)

!$OMP *directive_name [clause [[,] clause] ....]*

or

C$OMP directive_name [clause [[,] clause] ....]

or

**$OMP directive_name [clause [[,] clause] ....]*

In Fortran (free source form)

!$OMP *directive_name [clause [[,] clause] ....]*

In C/C++

#pragma omp *directive-name [clause[clause] ....]*

Shared Memory Parallel Programming

OpenMP: Directives

Fortran

```
!$OMP PARALLEL
   parallel code ...
!$OMP END PARALLEL

!$OMP DO
!$OMP END DO

!$OMP SECTIONS
 [!$OMP SECTION]
 [!$OMP SECTION]
!$OMP END SECTIONS

!$OMP SINGLE
!$OMP END SINGLE

!$OMP MASTER
!$OMP END MASTER

.... and more...
```

C/C++

```
#pragma omp parallel
   structured-block

#pragma omp for
   for-loop

#pragma omp sections
{
[#pragma opm section]
   structured-block
[#pragma opm section]
   structured-block
}

#pragma omp single
   structured-block

#pragma omp master
   structured-block

.... and more...
```

Shared Memory Parallel Programming

OpenMP: Uselful Subroutines and Functions

Fortran

Functions

```
omp_get_num_threads()

omp_get_max_threads()

omp_get_thread_num()

omp_get_num_procs()

omp_in_parallel()
```

Subroutines

```
call omp_set_num_threads(int)
```

C/C++

Functions

```
omp_get_num_threads()

omp_get_max_threads()

omp_get_thread_num()

omp_get_num_procs()

omp_in_parallel()

omp_set_num_threads(int)
```

Shared Memory Parallel Programming

OpenMP: Environment Variables

OMP_NUM_THREADS
Number of threads during the run

OMP_SCHEDULE
Type of scheduling in DO directives (static, dynamic, guided)

OMP_DYNAMIC
Enable/disable dynamic adjustments of the number of threads

OMP_NESTED
Enable/disable nested parallelism

Shared Memory Parallel Programming

OpenMP : An example Using !$OMP PARALLEL and !$OMP DO

```
... lines of codes

!$OMP PARALLEL
... lines of codes

!$OMP DO
do k=1,10000
 do j=1,10000
  do i=1,5000
   a(i,j,k)=c(i)*k - b(j)
  end do
 end do
end do
!$OMP END DO

... lines of codes

!$OMP PARALLEL

... lines of codes
```

One master thread (serial code)

Entering parallel region

Each thread is working with different k values

Exiting parallel region

One master thread (serial code)

Shared Memory Parallel Programming

OpenMP : An example Using !$OMP PARALLEL DO

```
... lines of codes


!$OMP PARALLEL DO
do k=1,10000

 do j=1,10000
  do i=1,5000
   a(i,j,k)=c(i)*k - b(j)
  end do
 end do

end do
!$OMP PARALLEL DO

... lines of codes
```

One master thread (serial code)

Entering parallel region

Each thread is working with different k values

Exciting parallel region

One master thread (serial code)

Shared Memory Parallel Programming

OpenMP : An example that does not work !

```
... lines of codes


!$OMP PARALLEL DO
do k=1,10000

 do j=1,10000
  do i=1,5000
   a(i,j,k)=a(i,j,k+1) - b(j)
  end do
 end do

end do
!$OMP PARALLEL DO

... lines of codes
```

One master thread (serial code)

Entering parallel region

Problems !! A thread might use data modified by another thread

Exciting parallel region

One master thread (serial code)

Shared Memory Parallel Programming

OpenMP: An Example Using Private and Firstprivate Clauses

```
!$OMP PARALLEL DO PRIVATE(tmp1,tmp2)
!$OMP& FIRSTPRIVATE(b)
do k=1,10000

 tmp1=2.34*k*k
 do j=1,10000
  b(j)=b(j)+2
  tmp2 = tmp1 + b(j)
  do i=1,5000
   a(i,j,k)=a(i,j,k) - tmp2
  end do
 end do

end do
!$OMP PARALLEL DO
```

Each thread has a copy of tmp1, tmp2 and b

Each thread calculate tmp1 independently

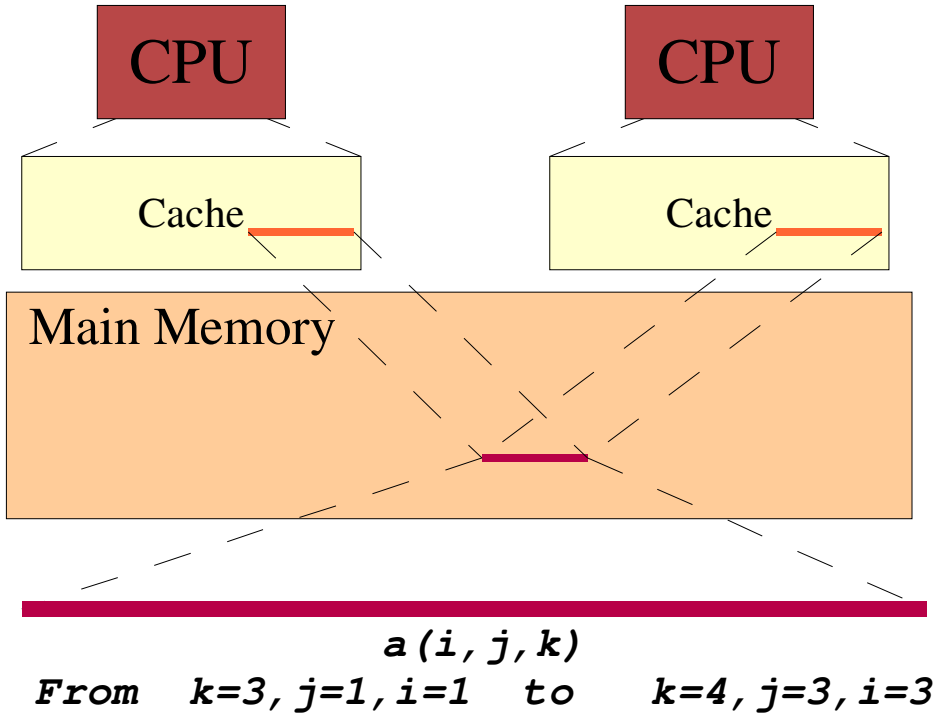
The initial values of b in each thread, is equal to the values of b in the master thread prior to entering the parallel region.

Shared Memory Parallel Programming

OpenMP: False Sharing !

False sharing is related to how the system manages the cache in a shared memory environment. To maintain the caches coherency, the system might have to flush a cache line of a CPU associated with data being requested by another CPU.

```
!$OMP PARALLEL DO
do k=1,10
 do j=1,5
  do i=1,5
   a(i,j,k)=c(i)*k - b(j)
  end do
 end do
end do
!$OMP PARALLEL DO
```



Introduction to Parallel Computing

Distributed Memory Parallel Programming

1) Distributed Memory Programming Interfaces
2) MPI : Several Instances of a Single Program
3) MPI : A Simple Program to get Started
4) MPI : Communication Models
5) MPI : Domain Decomposition
6) MPI : Commonly Used Calls
7) MPI : Data Types
8) MPI: A Simple Program to Get Started
9) MPI : Point-to-point Communications
10) MPI : Collective Communications
11) MPI : Using Communicators
12) MPI-2 Features

Distributed Memory Parallel Programming

Distributed Memory Programming Interfaces

Message Passing Interface (MPI)

- *Is a standard in development since 1994*
- *The most commonly used interface*
- *Several implementations available*

  MPICH, LAM MPI, Vendors specific
- *MPI-1 Standard first release: 1994*
- *MPI-2 Standard first release: 1996*
- *MPI Forum (http://www.mpi-forum.org/)*

Distributed Memory Parallel Programming

MPI : Several Instances of the Same Program

MPI Parallel program

> Several instances of the same program <

> Each instance represents an MPI Task <

> Usually One Task per CPU or Node <

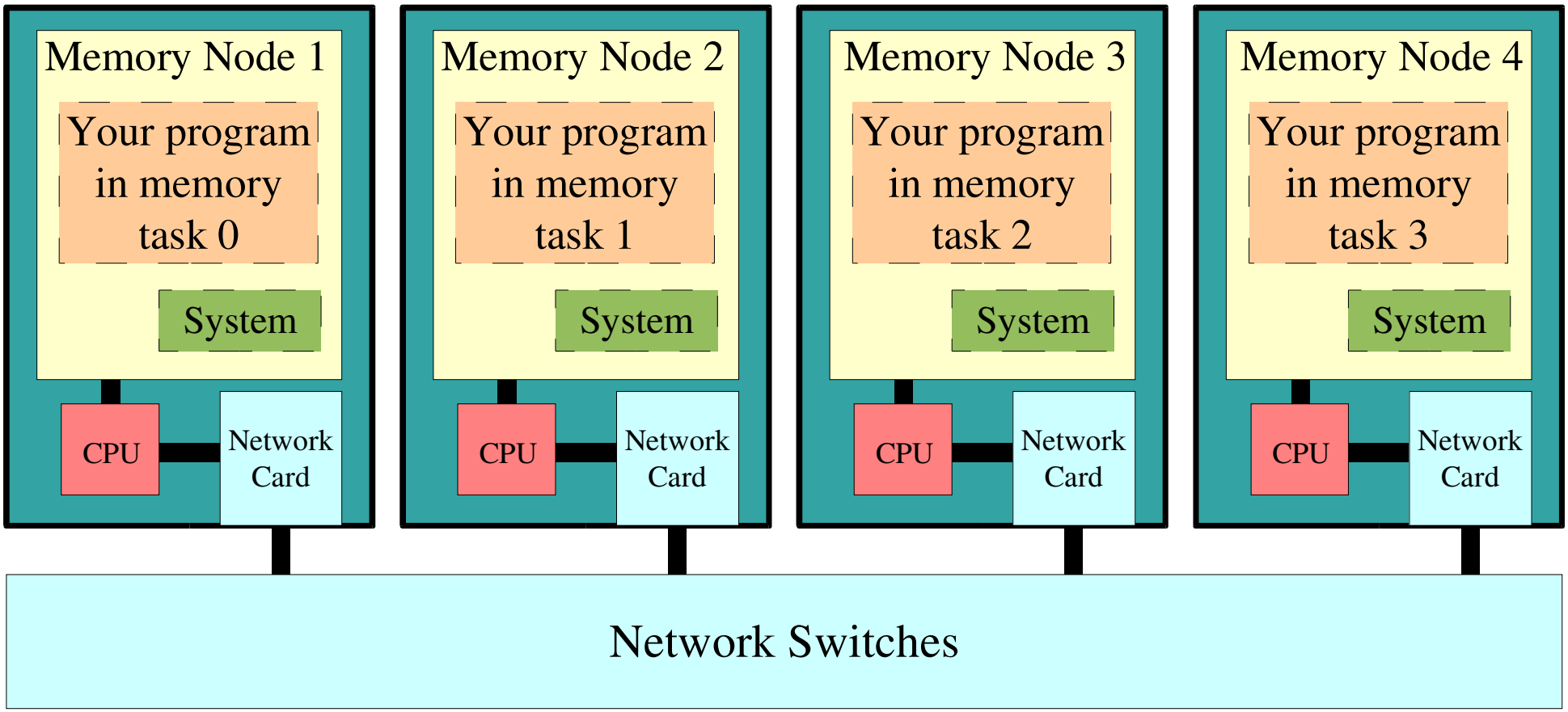
> A Task does not have direct access to other Tasks memory <

> Communication is done explicitly between Tasks <

> Each TASK has an ID <

Distributed Memory Parallel Programming

MPI : Several Instances of a Single Program



Distributed Memory Parallel Programming

MPI : Communication Models

Master-Slave Model

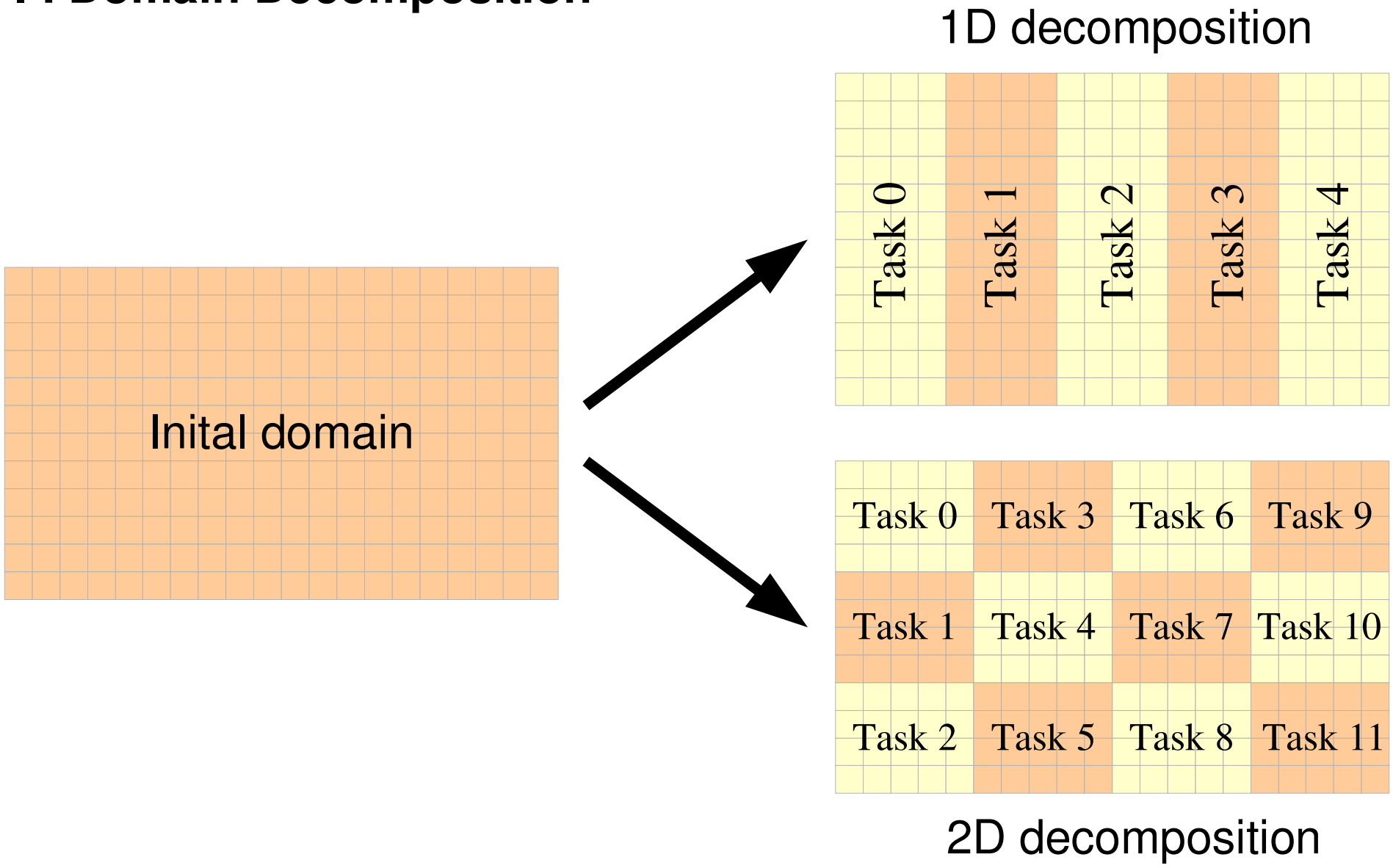
- Communications between Master task and slave tasks
- The Master dispatches jobs to slaves
- The Master gathers results
- Implies more point-to-point comunications than collective
- The Master task and a slave task might run on the same CPU
- Use this model when task jobs are of unpredictable length

Peer-to-peer Model

- No Master or Slave tasks
- Each task knows what it has to do
- Implies more collective communications
- Use this model when task jobs length are predictable

Distributed Memory Parallel Programming

MPI : Domain Decomposition



Distributed Memory Parallel Programming

MPI : Commonly Used Calls

MPI_INIT

MPI_FINILIZE

MPI_COMM_SIZE

MPI_COMM_RANK

MPI_BCAST

MPI_SEND, MPI_ISEND

MPI_RECV, MPI_IRECV

MPI_REDUCE, MPI_ALLREDUCE

MPI_GATHER, MPI_ALLGATHER

MPI_SCATTER

.....

Distributed Memory Parallel Programming

MPI : Data Types

Basic MPI datatypes for Fortran

MPI Datatype	Fortran
MPI_BYTE	
MPI_CHARACTER	CHARACTER
MPI_COMPLEX	COMPLEX*16
MPI_DOUBLE_PRECISION	REAL*8
MPI_INTEGER	INTEGER
MPI_LOGICAL	LOGICAL
MPI_REAL	REAL

Basic MPI datatypes for C

MPI Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	int
MPI_LONG	long
MPI_LONG_DOUBLE	long double
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Distributed Memory Parallel Programming

MPI : A Simple Program to Get Started

```
include “mpif.h”

program my_first_mpi_program

integer taskid,ntasks

call MPI_INIT(error)

call MPI_COMM_RANK(MPI_COMM_WORLD,ntasks)
call MPI_COMM_SIZE(MPI_COMM_WORLD,taskid)

write(6,*)'Hello from task ',taskid

call MPI_FINILIZE(error)

end
```

MPI_INIT ***MPI_FINALIZE*** ***MPI_COMM_SIZE*** ***MPI_COMM_RANK***

Distributed Memory Parallel Programming

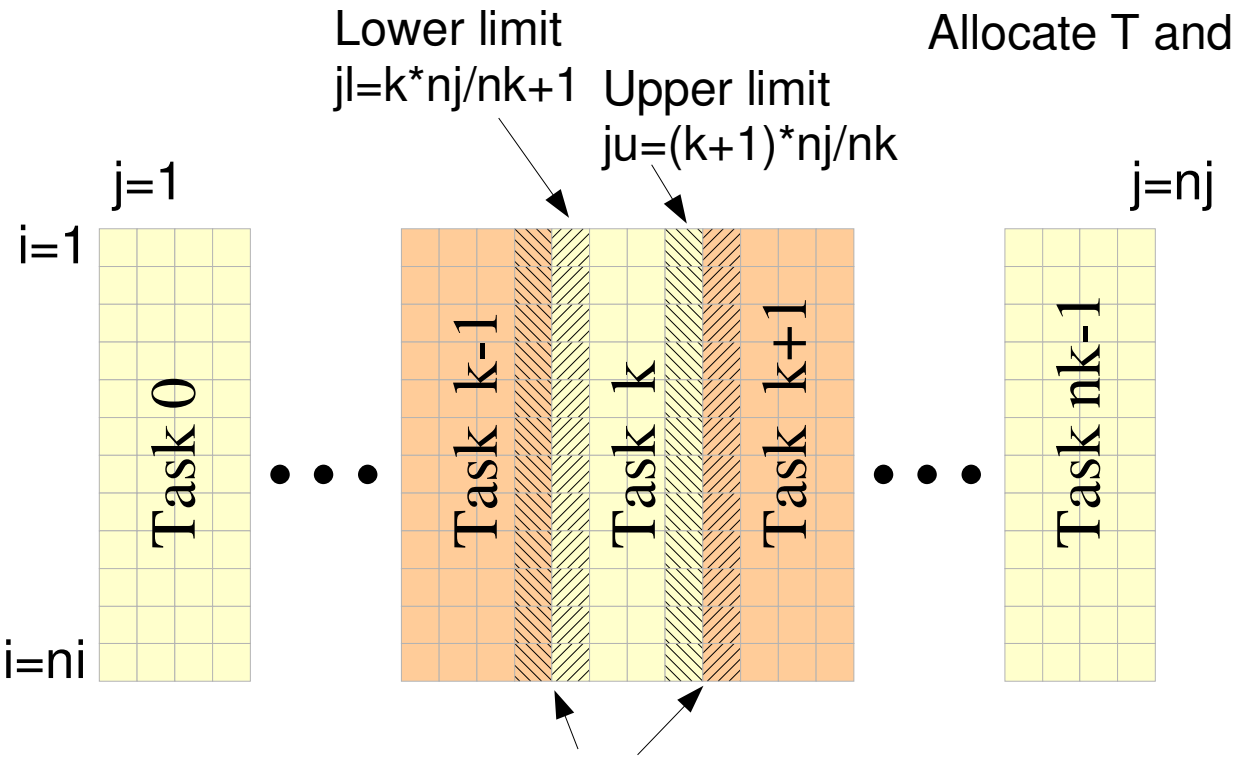
MPI Point-to-point Communications

Finite-difference temperature evolution

T(i,j) Temperatue at current time step
Tp(i,j) Temperature at previous time step

From Task k point of vue:
i=1 to ni
j=k*nj/nk+1 to (k+1)*nj/nk

Allocate T and Tp with these limits

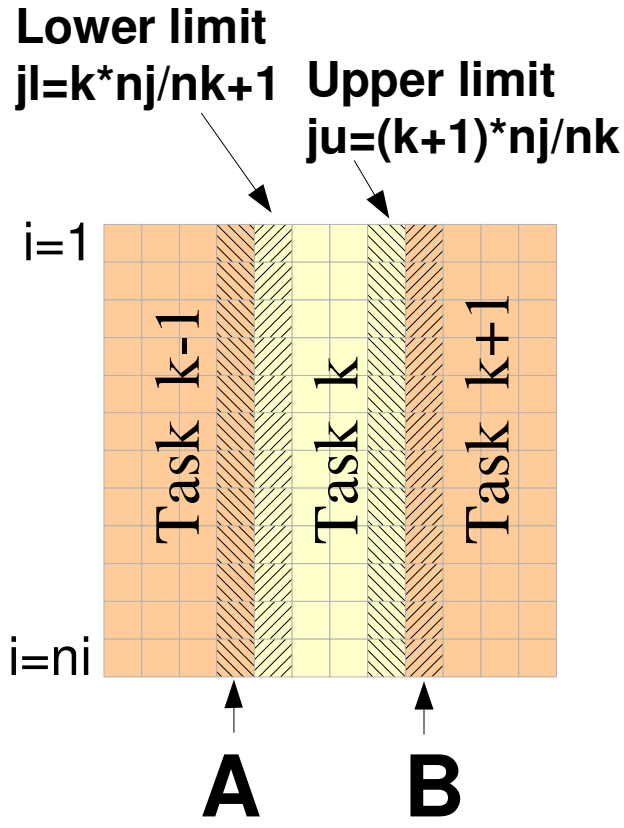


We will focus on task k exchanging data(Tp) with Task k-1 and k+1: Tp

Distributed Memory Parallel Programming

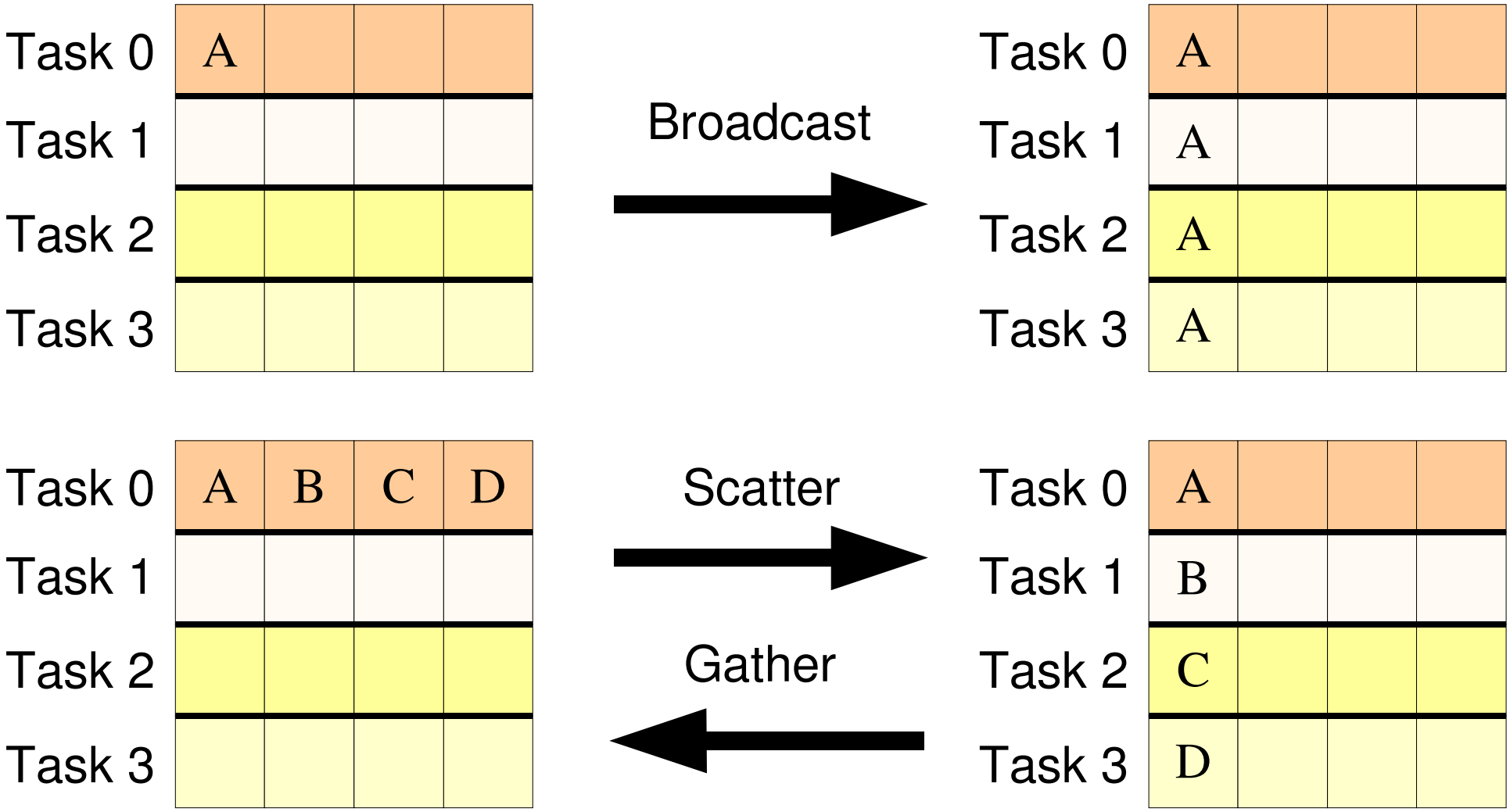
MPI Point-to-point Communications : MPI_ISEND MPI_IRECV

```
integer req(4),status(MPI_STATUS_SIZE,4)
call MPI_COMM_SIZE(MPI_COMM_WORLD,k)
call MPI_COMM_RANK(MPI_COMM_WORLD,nk)
jl=k*nj/nk+1
ju=(k+1)*nj/nk

allocate T(1:ni,jl:ju), Tp(1:ni,jl-1:ju+1)
...
call MPI_ISEND(Tp(1,ju),ni,MPI_REAL,k+1,0,
               MPI_COMM_WORLD,req(3),err)

call MPI_IRECV(Tp(1,jl-1),ni,MPI_REAL,k-1,0,
               MPI_COMM_WORLD,req(4),err)        } A

call MPI_ISEND(Tp(1,jl),ni,MPI_REAL,k-1,1,
               MPI_COMM_WORLD,req(1),err)

call MPI_IRECV(Tp(1,ju+1),ni,MPI_REAL,k+1,1,
               MPI_COMM_WORLD,req(2),err)        } B

call MPI_WAITALL(4,req,status,err)

    ... Calculate T(i,j) with Tp(i,j)
```



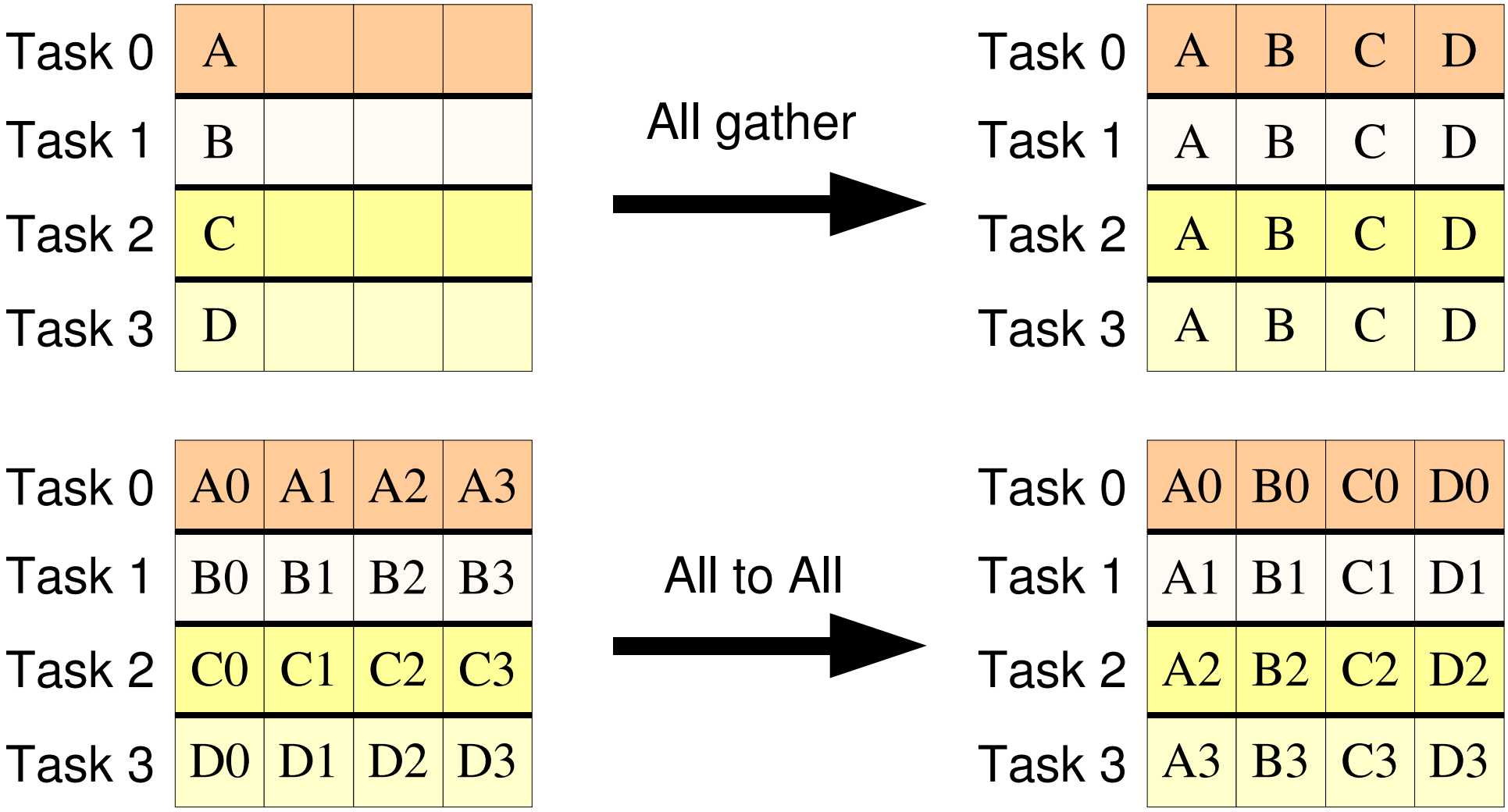
Distributed Memory Parallel Programming

MPI Collective Communications



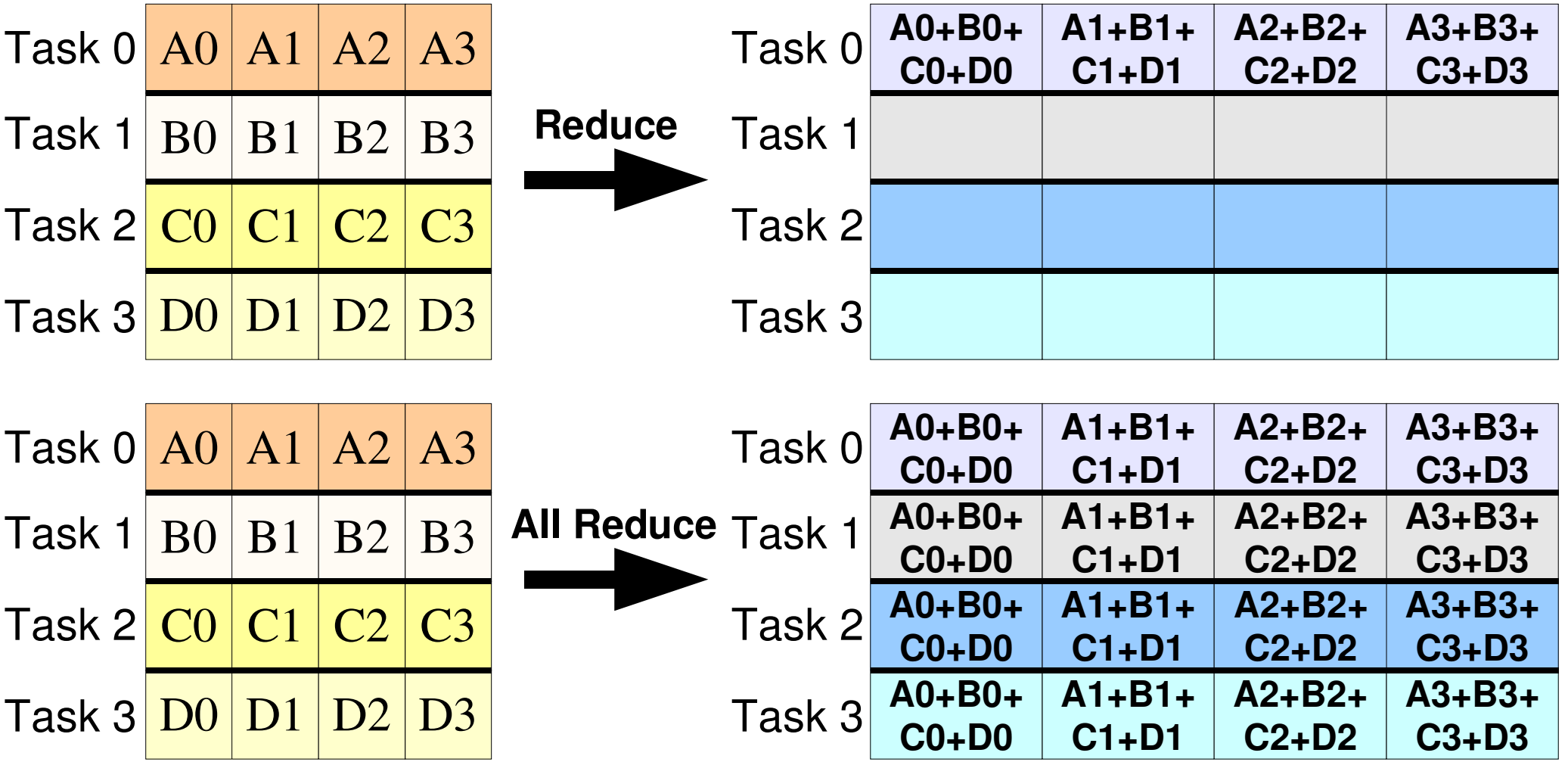
Distributed Memory Parallel Programming

MPI Collective Communications



Distributed Memory Parallel Programming

MPI Collective Communications : Reduce and Allreduce



Apply a reduction operation (+,x,logical,etc) to a vector over the set of tasks

Distributed Memory Parallel Programming

MPI : Using Communicators

- *Use more than global MPI_COMM_WORLD*
- *Defines groups of tasks*
- *Use Virtual Topologies*

Distributed Memory Parallel Programming

MPI-2 : New features

- *Parallel I/O*
- *Remote Memory Operations*
- *Dynamic Process Management*
- *C++ and Fortran 90 bindings*
- *Other improvements*

That's it!

Merci
Thank you

http://www.ccs.usherbrooke.ca