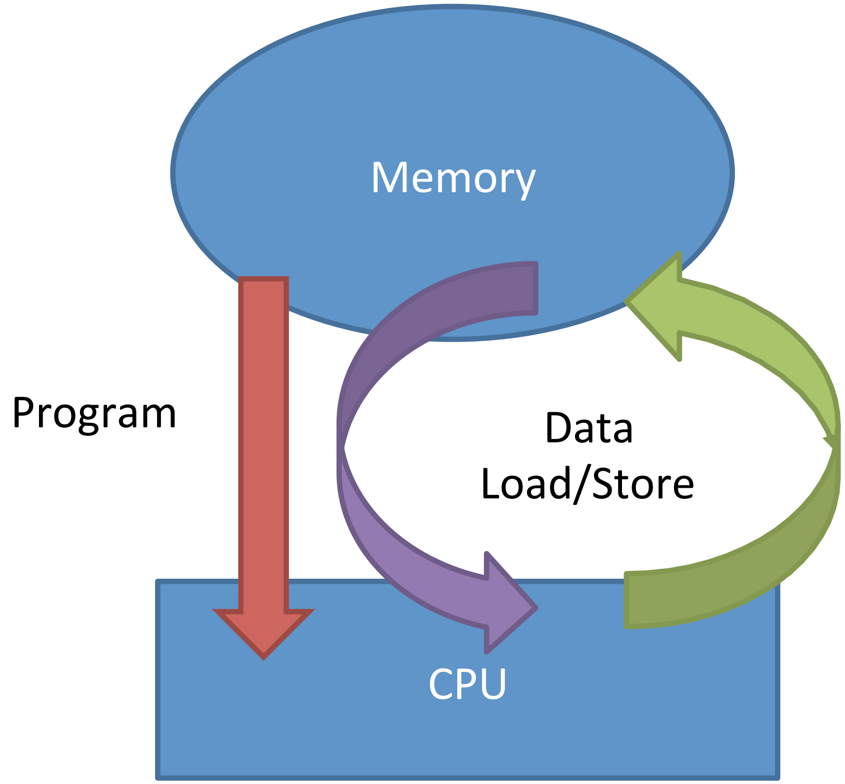
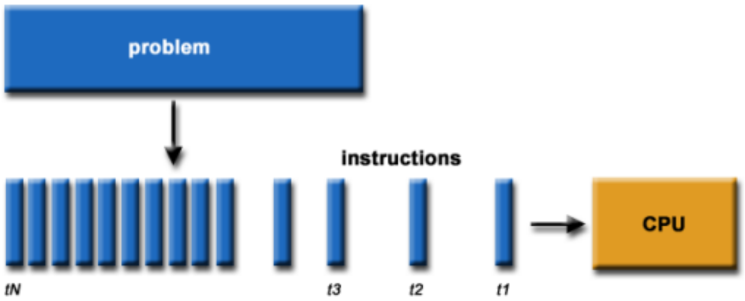
Thinking in Parallel

Ivan Girotto – igirotto@ictp.it
International Centre for Theoretical Physics (ICTP)

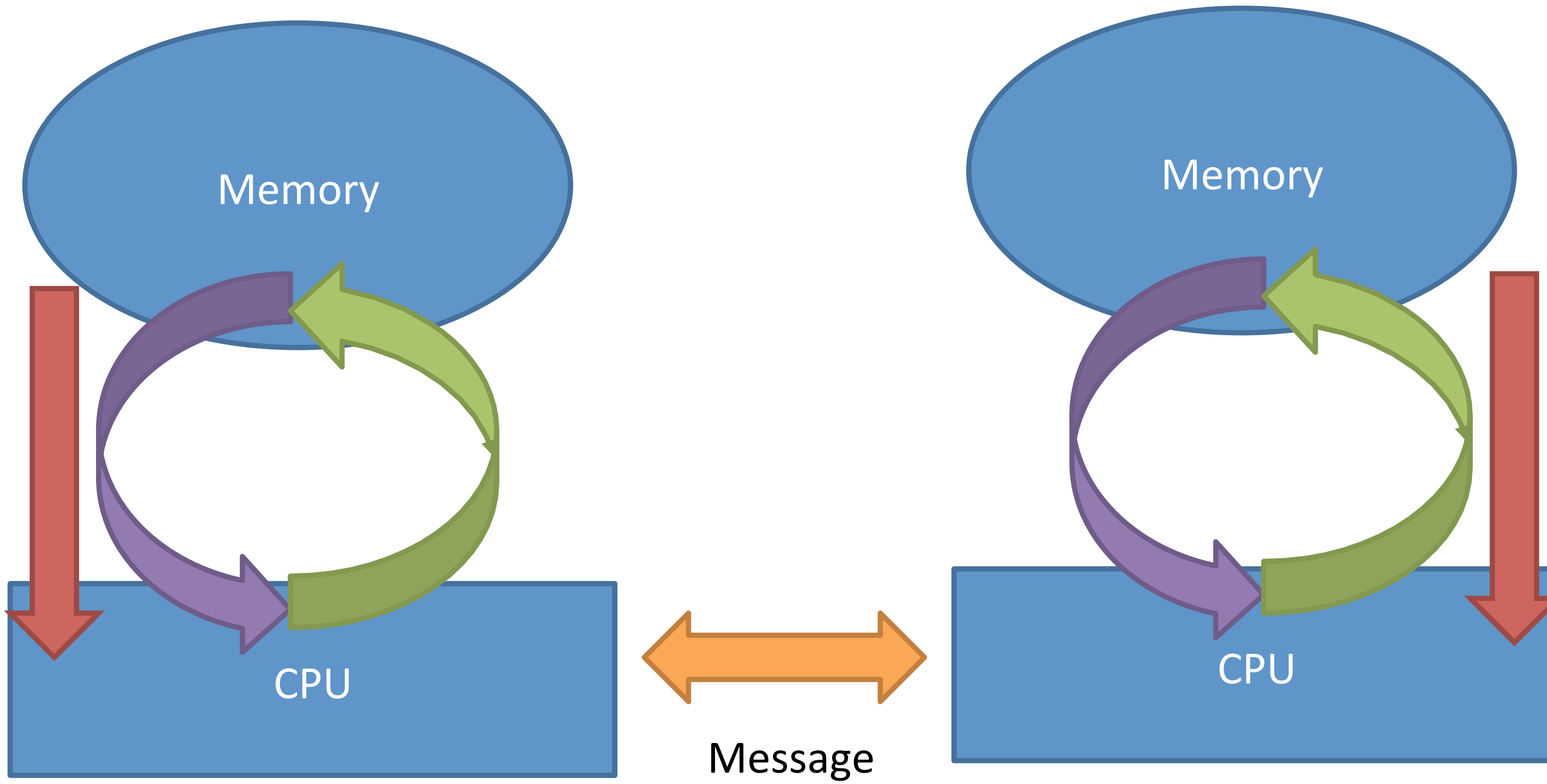
Serial Programming



A problem is broken into a discrete series of instructions.
Instructions are executed one after another.
Only one instruction may execute at any moment in time.

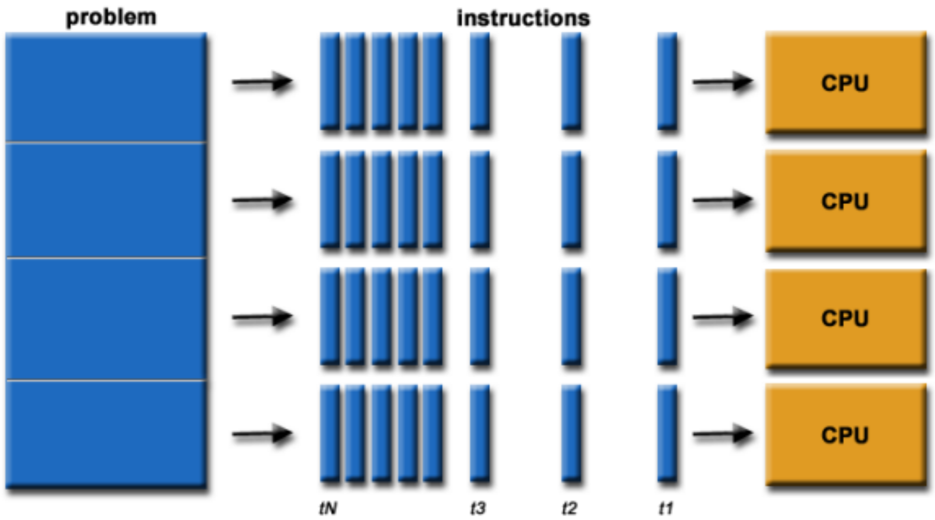


Parallel Programming



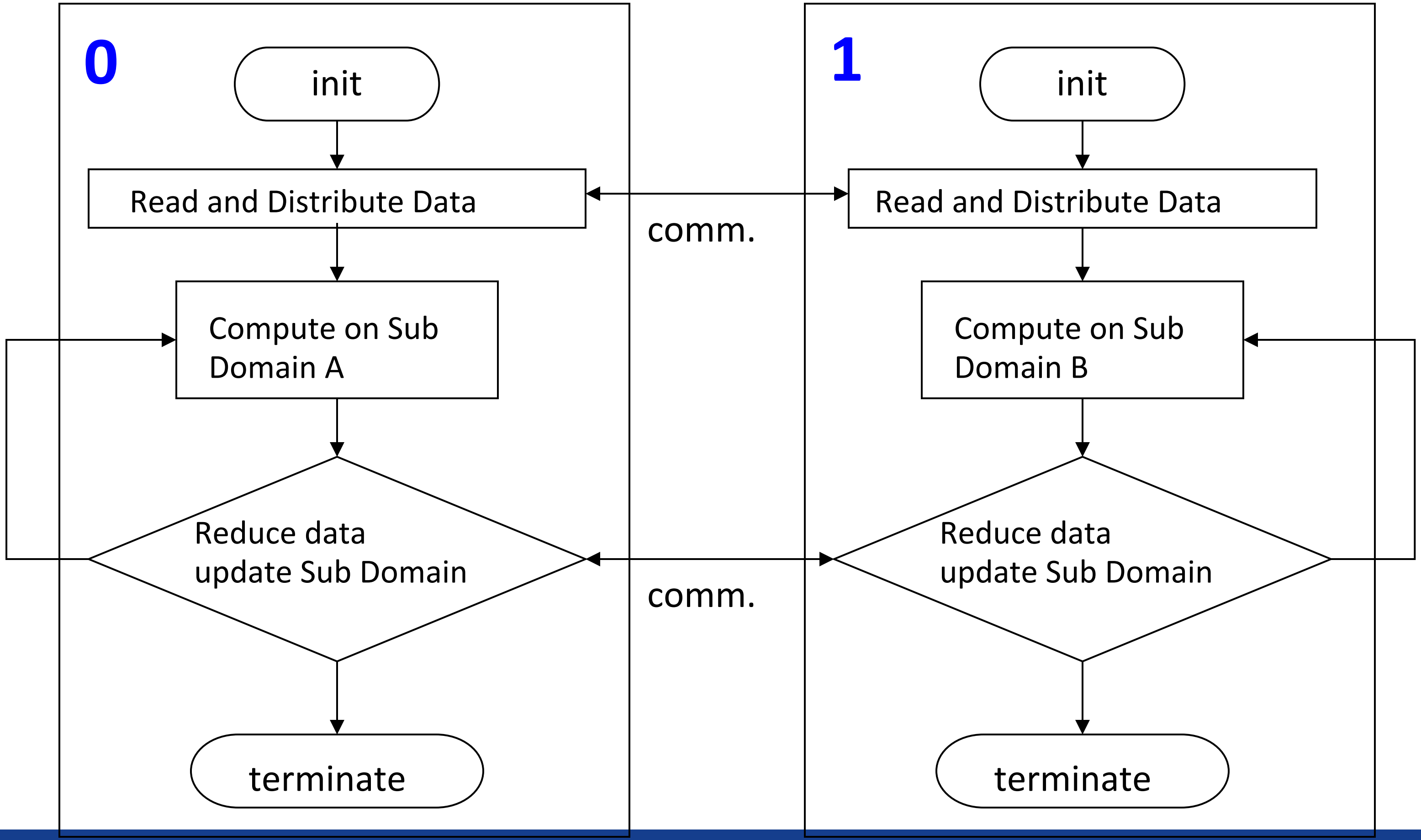
Concurrency

The first step in developing a parallel algorithm is to decompose the problem into tasks that can be executed concurrently



- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control / coordination mechanism is employed

What is a Parallel Program

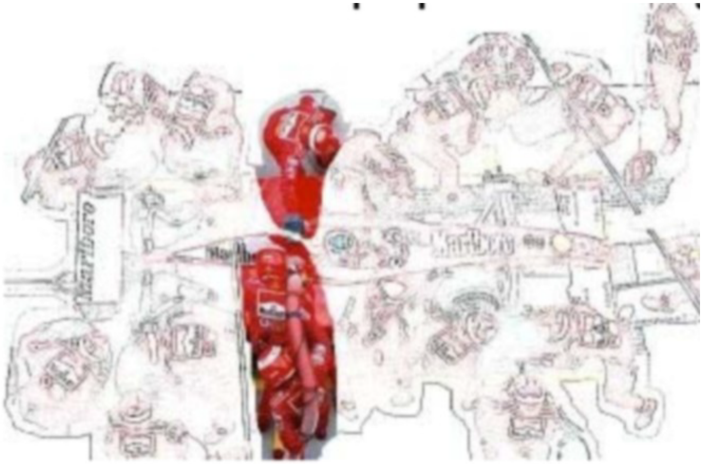


Fundamental Steps of Parallel Design

- Identify portions of the work that can be performed concurrently
- Mapping the concurrent pieces of work onto multiple processes running in parallel
- Distributing the input, output and intermediate data associated within the program
- Managing accesses to data shared by multiple processors
- Synchronizing the processors at various stages of the parallel program execution

Type of Parallelism

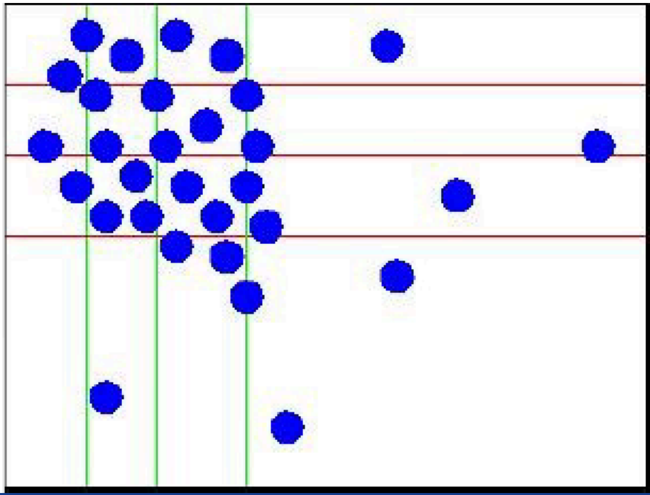
- **Functional (or task) parallelism**:
  different people are performing different task at the same time



- **Data Parallelism**:
  different people are performing the same task, but on different equivalent and independent objects

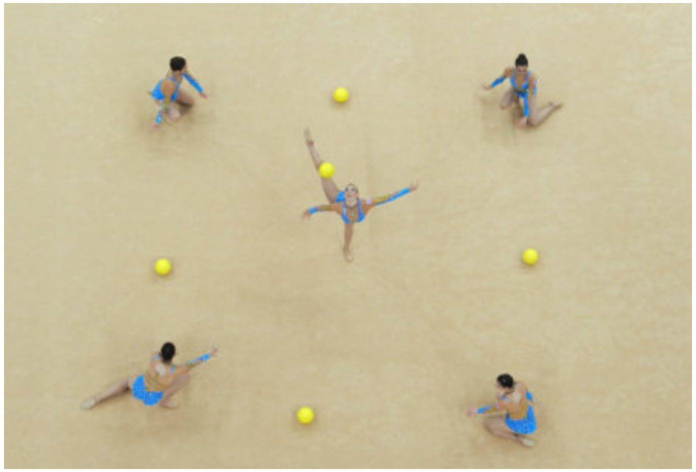
Limitations of Parallel Computing

- Fraction of serial code limits parallel speedup
- Degree to which tasks/data can be subdivided is limit to concurrency and parallel execution
- Load imbalance:
  - parallel tasks have a different amount of work
  - CPUs are partially idle
  - redistributing work helps but has limitations
  - communication and synchronization overhead



Process Interactions

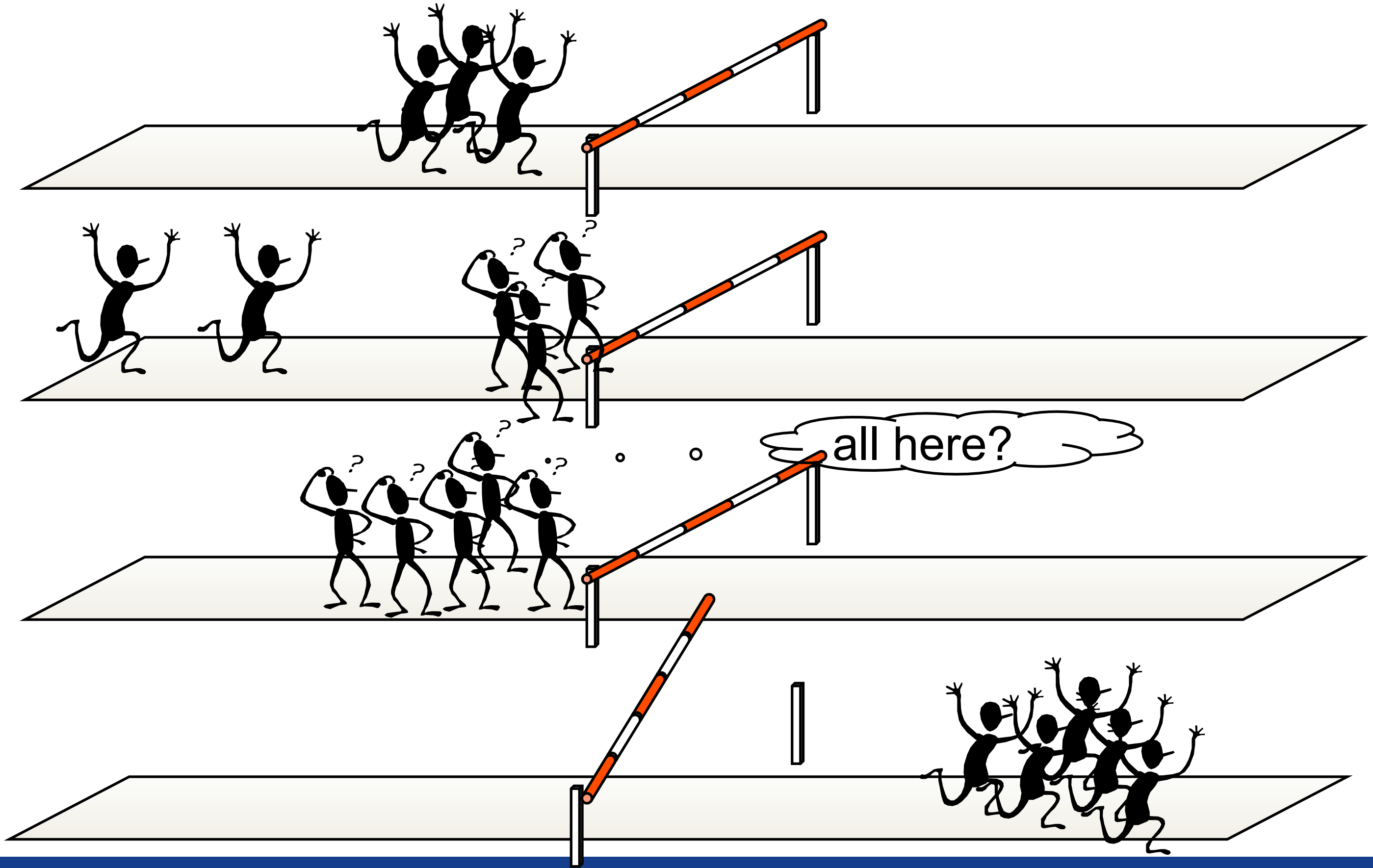
- The effective speed-up obtained by the parallelization depend by the amount of overhead we introduce making the algorithm parallel
- There are mainly two key sources of overhead:
    1. Time spent in inter-process interactions (**communication**)
    2. Time some process may spent being idle (**synchronization**)



Load Balancing

- Equally divide the work among the available resource: processors, memory, network bandwidth, I/O, ...
- This is usually a simple task for the problem decomposition model
- It is a difficult task for the functional decomposition model

Effect of Load Unbalancing



Minimizing Communication

- When possible reduce the communication events:
  - group lots of small communications into large one
  - eliminate synchronizations as much as possible. Each synchronization level off the performance to that of the slowest process

Overlap Communication and Computation

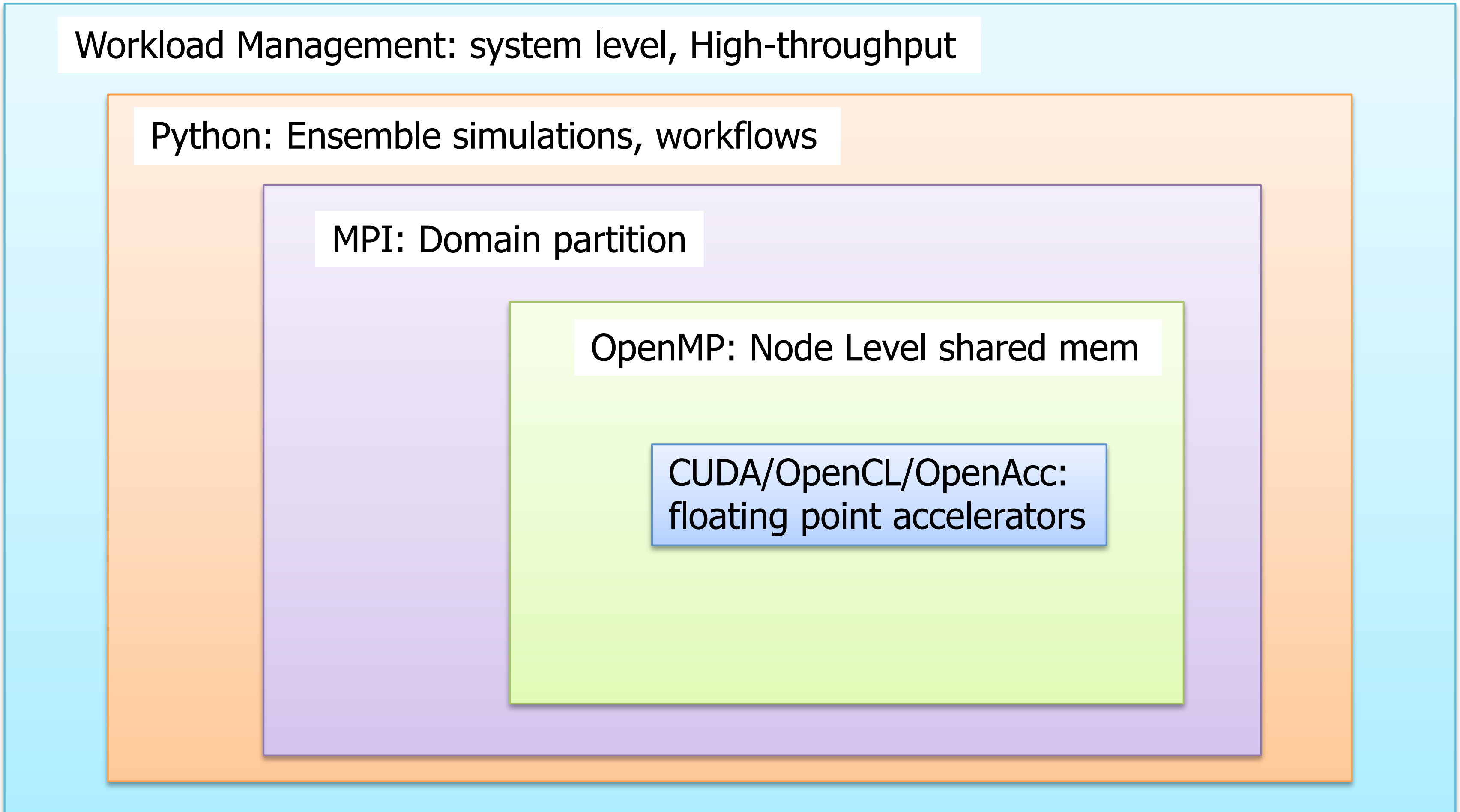
- When possible code your program in such a way that processes continue to do useful work while communicating
- This is usually a non trivial task and is afforded in the very last phase of parallelization
- If you succeed, you have done. Benefits are enormous

Granularity

- Granularity is determined by the decomposition level (number of task) on which we want divide the problem
- The degree to which task/data can be subdivided is limit to concurrency and parallel execution
- Parallelization has to become “topology aware”
  - coarse grain and fine grained parallelization has to be mapped to the topology to reduce memory and I/O contention
  - make your code modularized to enhance different levels of granularity and consequently to become more “platform adaptable”

Programming Parallel Paradigms

- Are the tools we use to express the parallelism for on a given architecture (see also SPMD, SIMD, etc...)
- They differ in how programmers can manage and define key features like:
  - parallel regions
  - concurrency
  - process communication
  - synchronism


Workload Management: system level, High-throughput
Python: Ensemble simulations, workflows
MPI: Domain partition
OpenMP: Node Level shared mem
CUDA/OpenCL/OpenAcc: floating point accelerators

Shared Resources

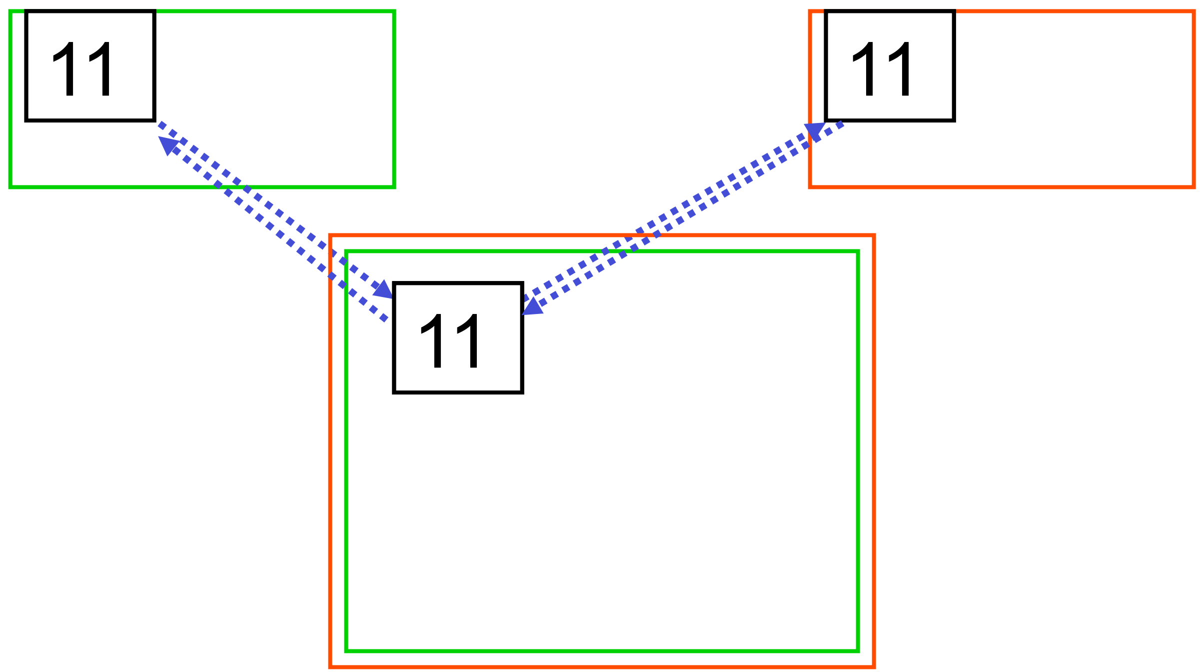
- In parallel programming, developers must manage exclusive access to shared resources
- Resources are in different forms:
  - concurrent read/write (including parallel write) to shared memory locations
  - concurrent read/write (including parallel write) to shared devices
  - a message that must be send and received

	Thread 1	Thread 2

Program	`load a`
	`add a 1`
	`store a`
		`load a`
		`add a 1`
		`store a`

Private data

Shared data



Fundamental Tools of Parallel Programming

MPI Program Design

- Multiple and separate processes (can be local and remote) concurrently that are coordinated and exchange data through "messages"

  => a "share nothing" parallelization
- Best for coarse grained parallelization
- Distribute large data sets; replicate small data
- Minimize communication or overlap communication and computing for efficiency

  => Amdahl's law: speedup is limited by the fraction of serial code plus communication

Phases of an MPI Program

1. Startup
   - Parse arguments (mpirun may add some!)
   - Identify parallel environment and rank of process
   - Read and distribute all data
2. Execution
   - Proceed to subroutine with parallel work (can be same of different for all parallel tasks)
3. Cleanup

CAUTION: this sequence may be run only once

```
program bcast

 implicit none

 include "mpif.h"

 integer :: myrank, ncpus, imesg, ierr
 integer, parameter :: comm = MPI_COMM_WORLD

 call MPI_INIT(ierr)
 call MPI_COMM_RANK(comm, myrank, ierr)
 call MPI_COMM_SIZE(comm, ncpus, ierr)

 imesg = myrank
 print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

 call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

 print *, "After Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

 call MPI_FINALIZE(ierr)

end program bcast
```

```fortran
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = ??
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
```

P_0

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = ??
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = ??
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```fortran
program bcast
  implicit none
  include "mpif.h"
  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD
  call MPI_INIT(ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = ??
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

 implicit none

 include "mpif.h"

 integer :: myrank, ncpus, imesg, ierr
 integer, parameter :: comm = MPI_COMM_WORLD

 call MPI_INIT(ierr)
 call MPI_COMM_RANK(comm, myrank, ierr)
 call MPI_COMM_SIZE(comm, ncpus, ierr)

 imesg = myrank
 print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 1
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 2
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 3
ierr = MPI_SUC...
comm = MPI_C...

call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

P_0	P_1	P_2	P_3
myrank = 0	myrank = 1	myrank = 2	myrank = 3
ncpus = 4	ncpus = 4	ncpus = 4	ncpus = 4
imesg = 0	imesg = 1	imesg = 2	imesg = 3
ierr = MPI_SUC...	ierr = MPI_SUC...	ierr = MPI_SUC...	ierr = MPI_SUC...
comm = MPI_C...	comm = MPI_C...	comm = MPI_C...	comm = MPI_C...

call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

P_0	P_1	P_2	P_3
myrank = 0	myrank = 1	myrank = 2	myrank = 3
ncpus = 4	ncpus = 4	ncpus = 4	ncpus = 4
imesg = 0	imesg = 0	imesg = 0	imesg = 0
ierr = MPI_SUC...	ierr = MPI_SUC...	ierr = MPI_SUC...	ierr = MPI_SUC...
comm = MPI_C...	comm = MPI_C...	comm = MPI_C...	comm = MPI_C...

```
program bcast

 implicit none

 include "mpif.h"

 integer :: myrank, ncpus, imesg, ierr
 integer, parameter :: comm = MPI_COMM_WORLD

 call MPI_INIT(ierr)
 call MPI_COMM_RANK(comm, myrank, ierr)
 call MPI_COMM_SIZE(comm, ncpus, ierr)

 imesg = myrank
 print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

 call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

 print *, "After Bcast operation I'm ", myrank, &
     " and my message content is ", imesg
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

  implicit none

  include "mpif.h"

  integer :: myrank, ncpus, imesg, ierr
  integer, parameter :: comm = MPI_COMM_WORLD

  call MPI_INIT(ierr)
  call MPI_COMM_RANK(comm, myrank, ierr)
  call MPI_COMM_SIZE(comm, ncpus, ierr)

  imesg = myrank
  print *, "Before Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

  print *, "After Bcast operation I'm ", myrank, &
      " and my message content is ", imesg

  call MPI_FINALIZE(ierr)
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

```
program bcast

 implicit none

 include "mpif.h"

 integer :: myrank, ncpus, imesg, ierr
 integer, parameter :: comm = MPI_COMM_WORLD

 call MPI_INIT(ierr)
 call MPI_COMM_RANK(comm, myrank, ierr)
 call MPI_COMM_SIZE(comm, ncpus, ierr)

 imesg = myrank
 print *, "Before Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

 call MPI_BCAST(imesg, 1, MPI_INTEGER, 0, comm, ierr)

 print *, "After Bcast operation I'm ", myrank, &
     " and my message content is ", imesg

 call MPI_FINALIZE(ierr)

end program bcast
```

P_0

myrank = 0
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_1

myrank = 1
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

P_2

myrank = 2
ncpus = 4
imesg = 0
ierr = MPI_SUCC
comm = MPI_C...

P_3

myrank = 3
ncpus = 4
imesg = 0
ierr = MPI_SUC...
comm = MPI_C...

Thanks for your attention!!