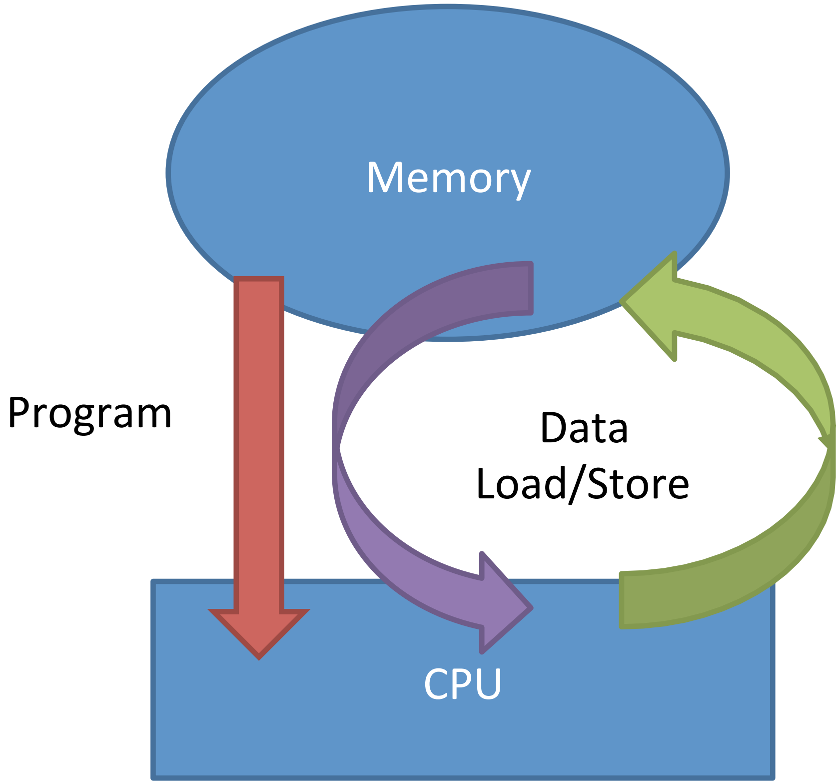
Thinking in Parallel

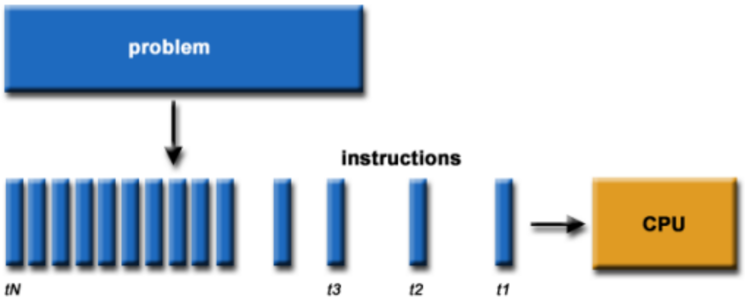
Ivan Girotto – igirotto@ictp.it

International Centre for Theoretical Physics (ICTP)

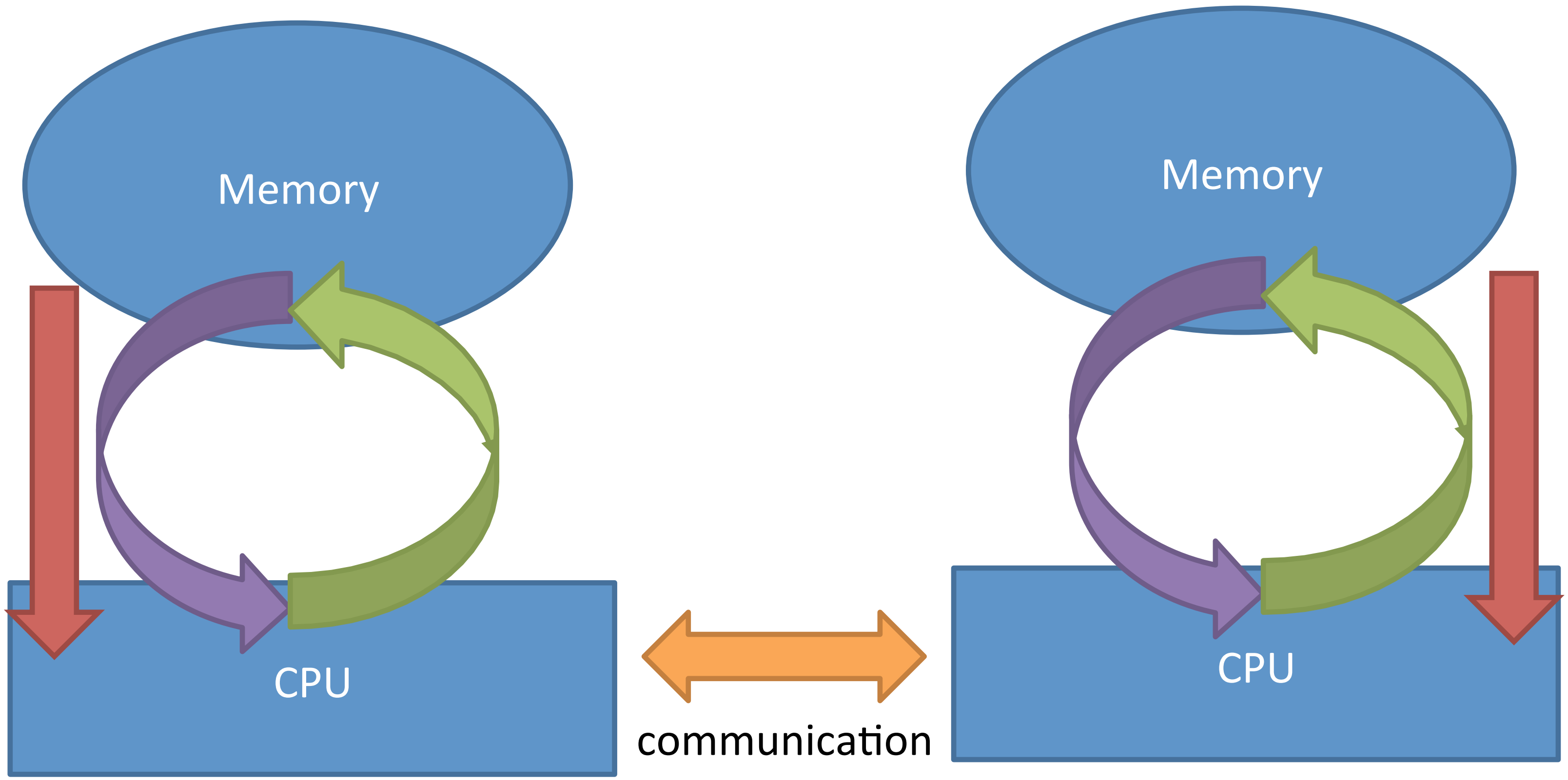
Serial Programming



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.

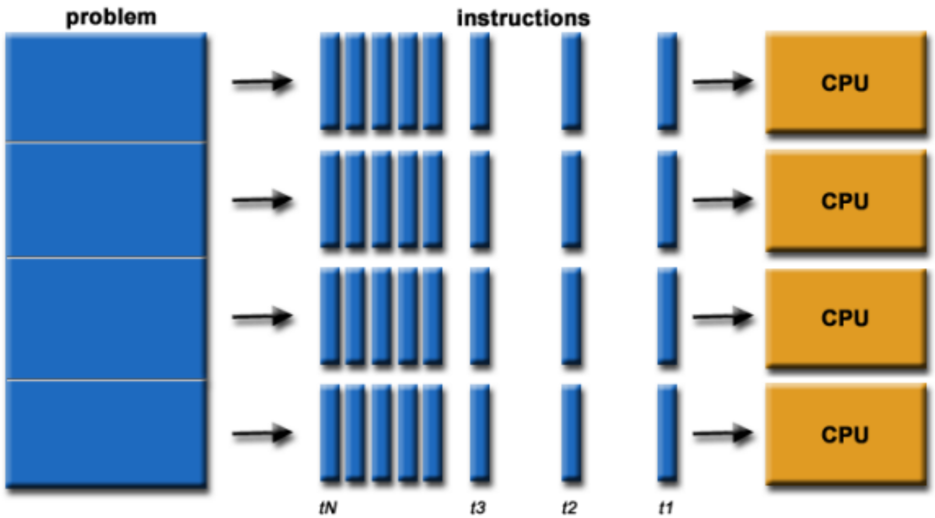


Parallel Programming



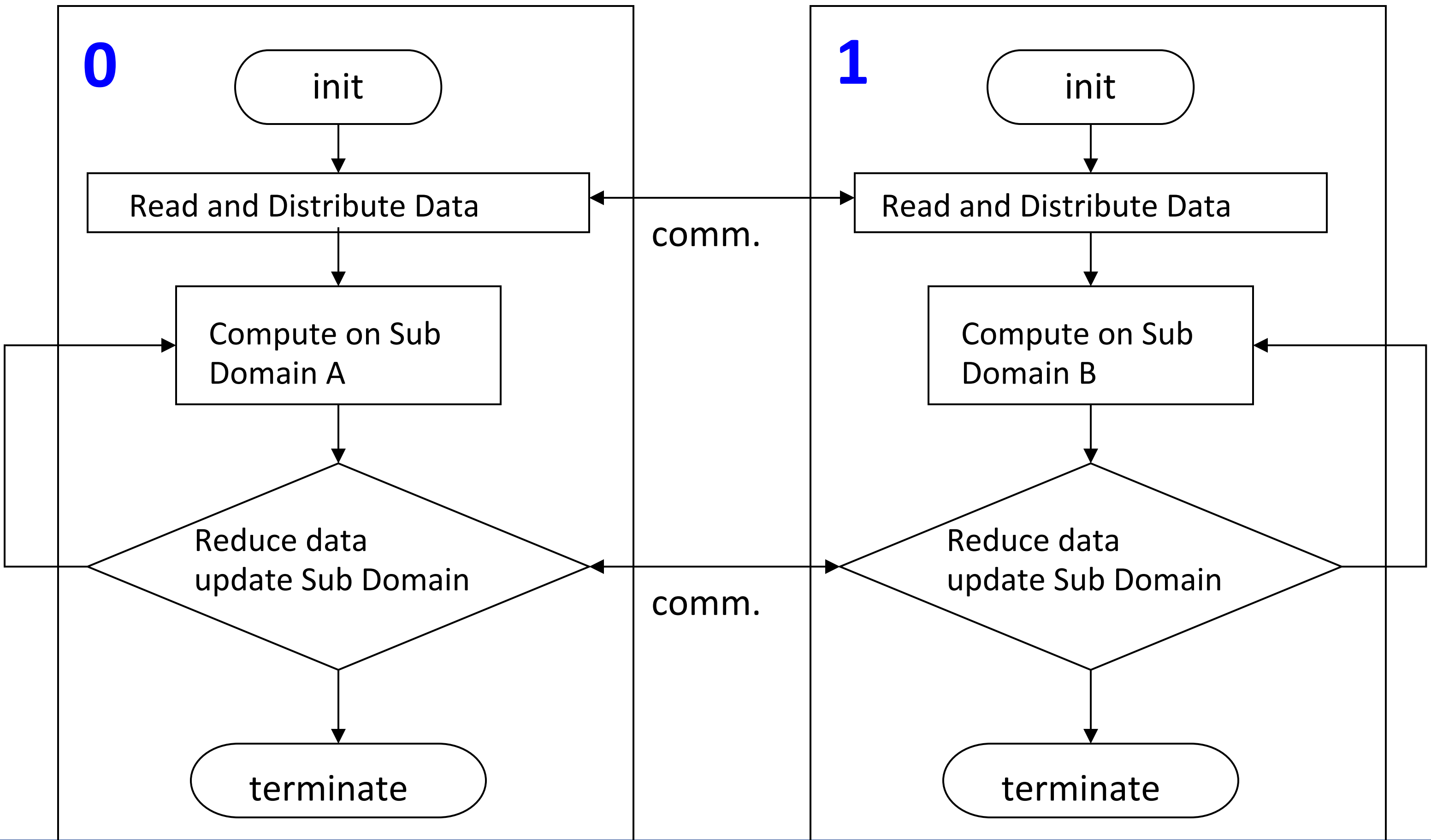
Concurrency

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

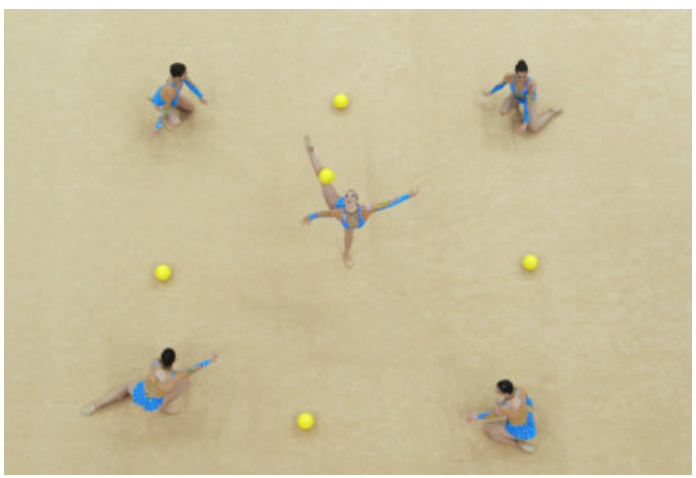
Type of Parallelism

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

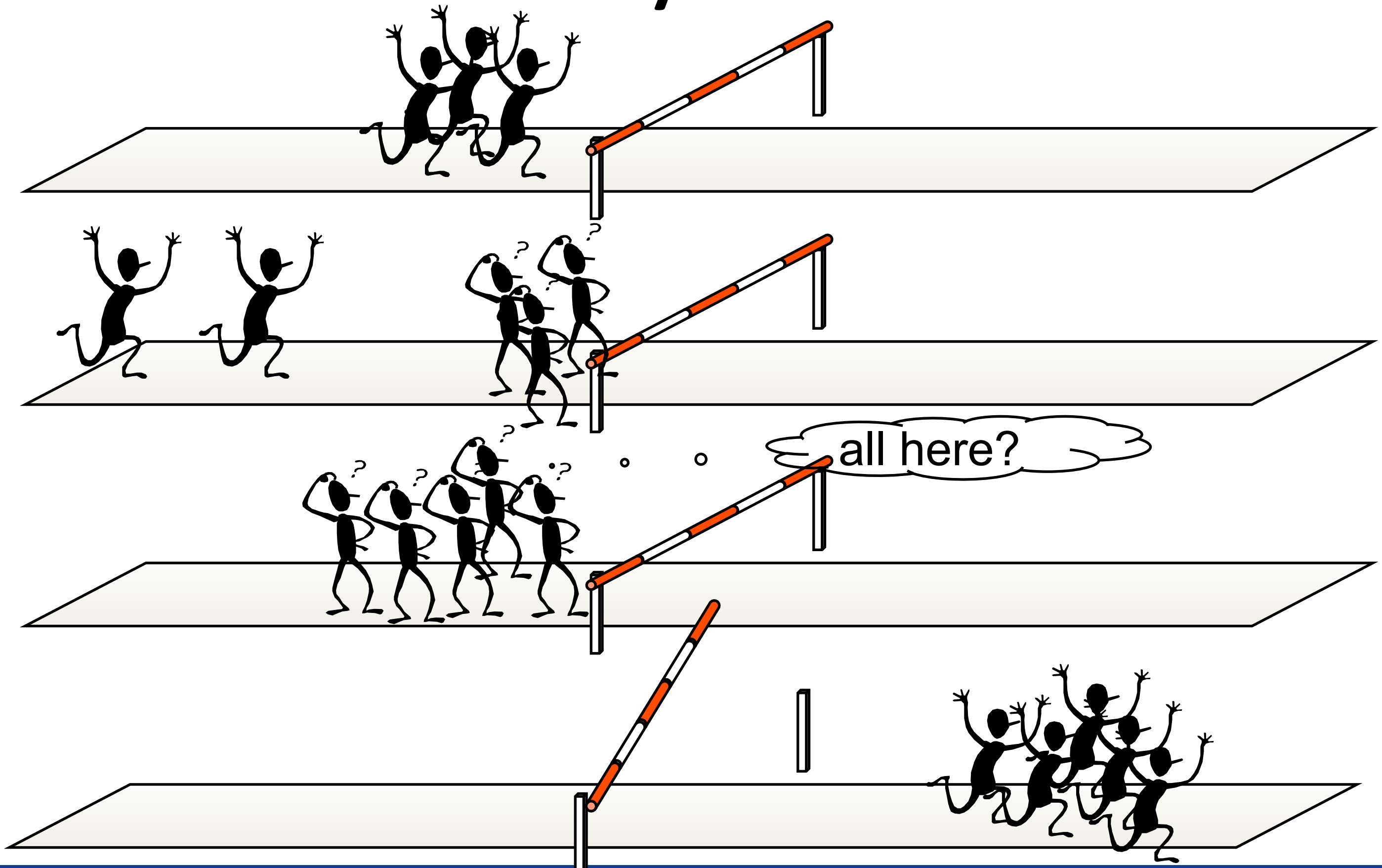
- **Data Parallelism**: different people are performing the same task, but on different equivalent and independent objects

Process Interactions

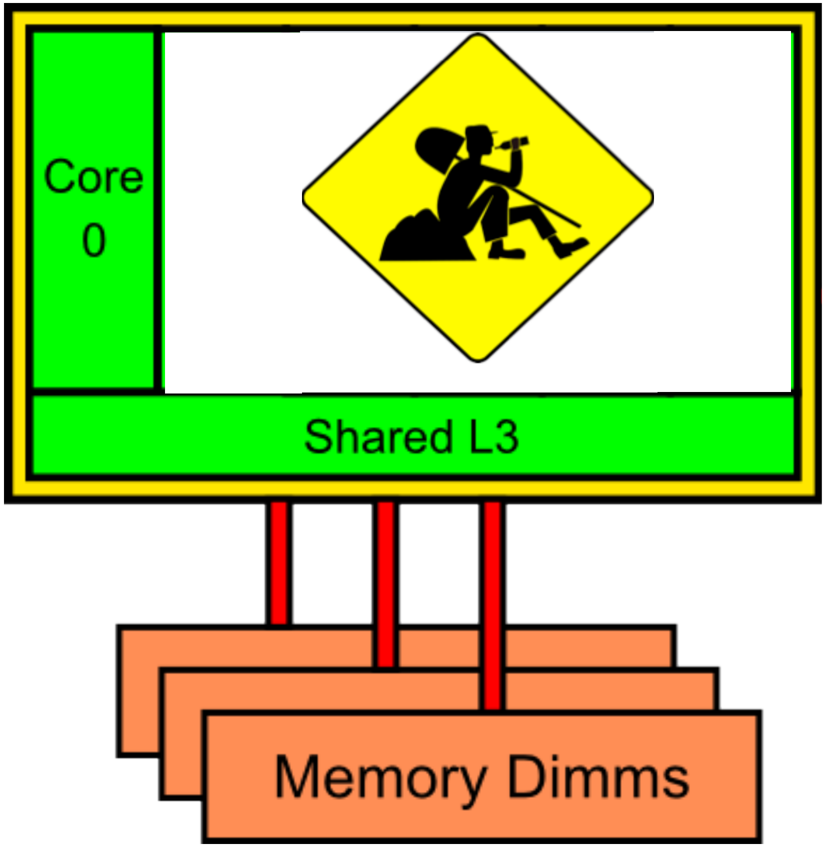
- The effective speed-up obtained by the parallelization depend by the amount of overhead we introduce making the algorithm parallel
- There are mainly two key sources of overhead:
    1. Time spent in inter-process interactions (**communication**)
    2. Time some process may spent being idle (**synchronization**)

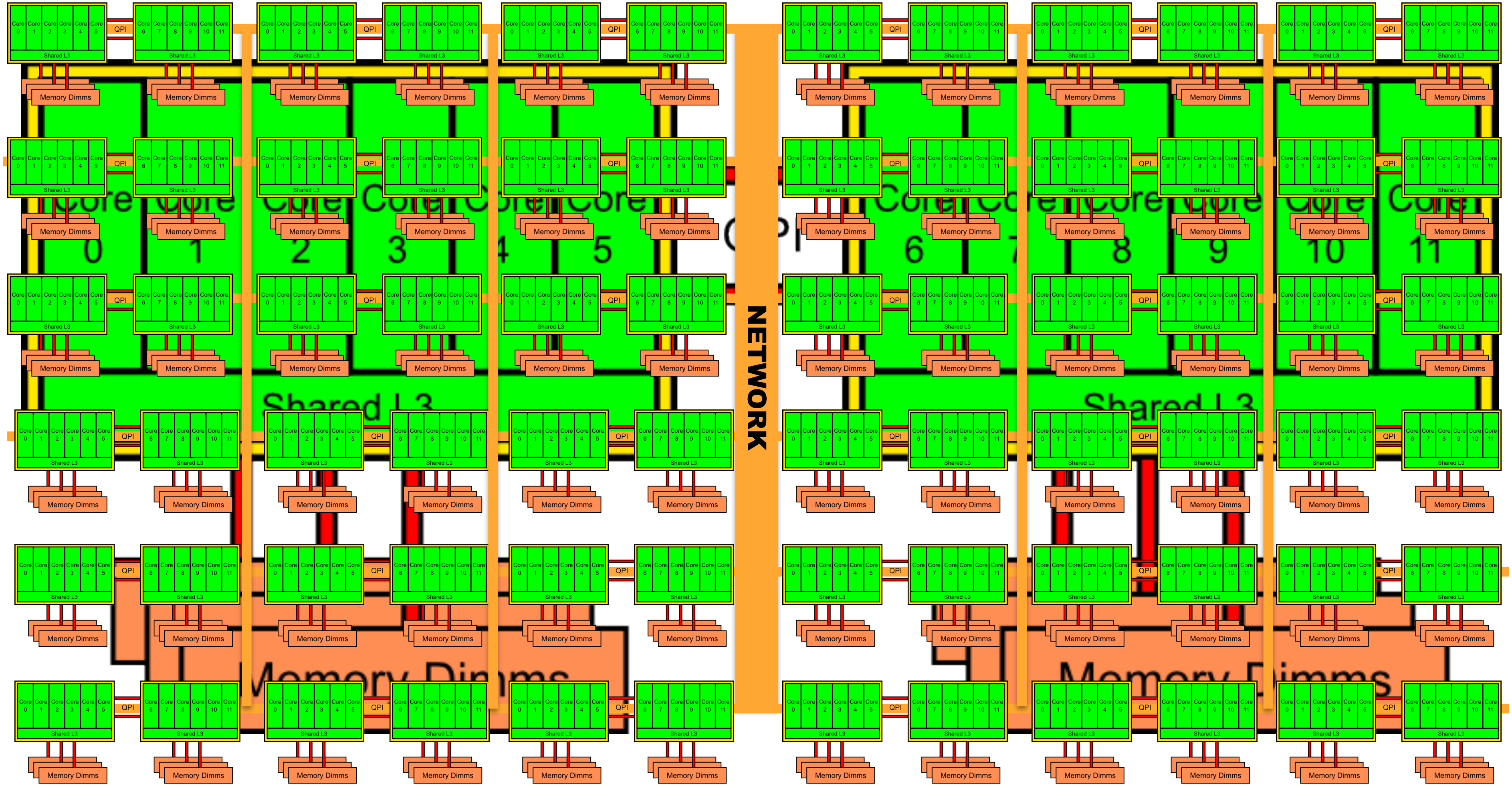


Barrier and Synchronization

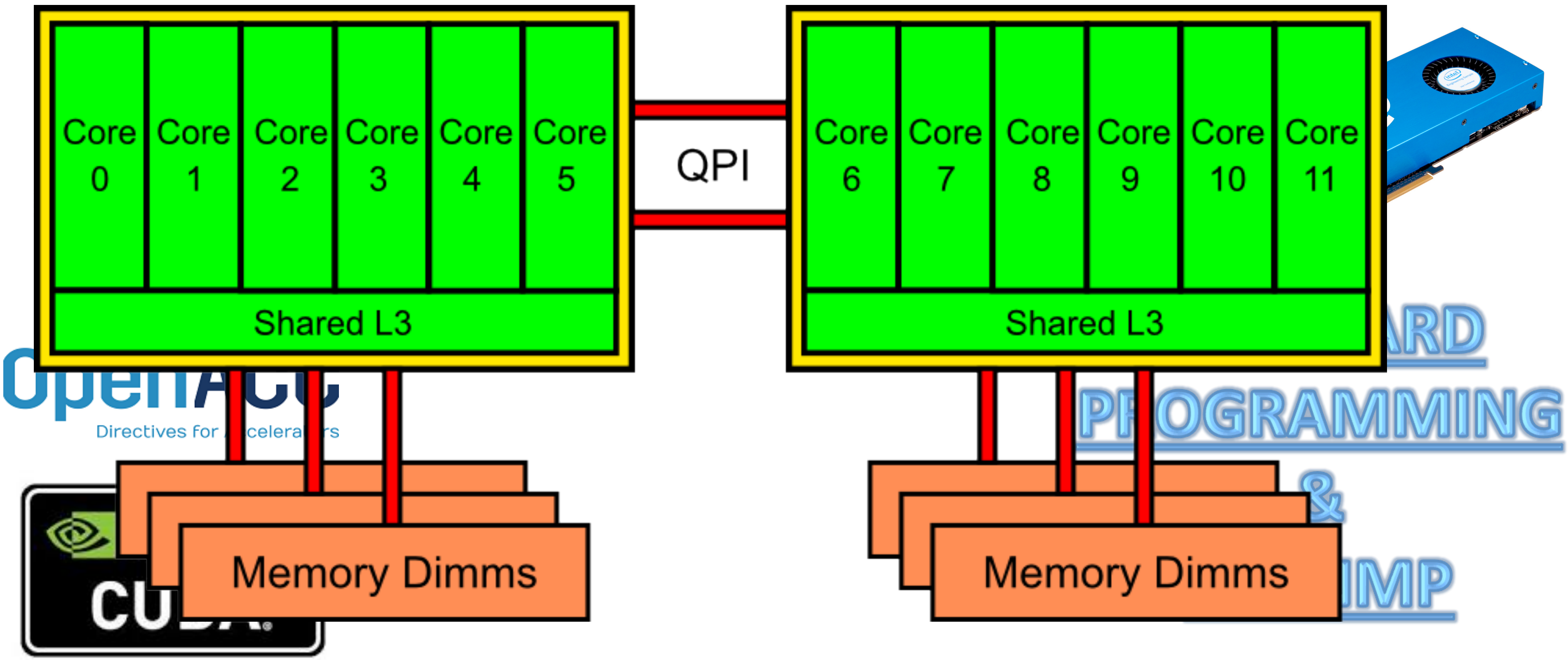


Scalable Programming




Core
QPI
Shared L3
Memory Dimms
NETWORK

Scalable Hybrid Programming

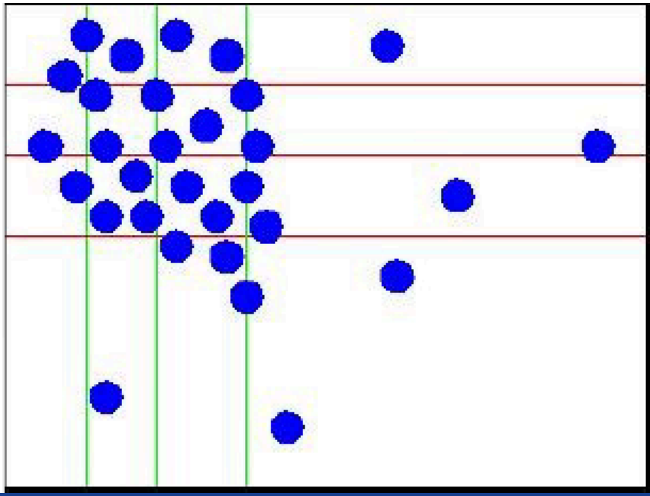


Granularity

- 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”

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



Shared Resources

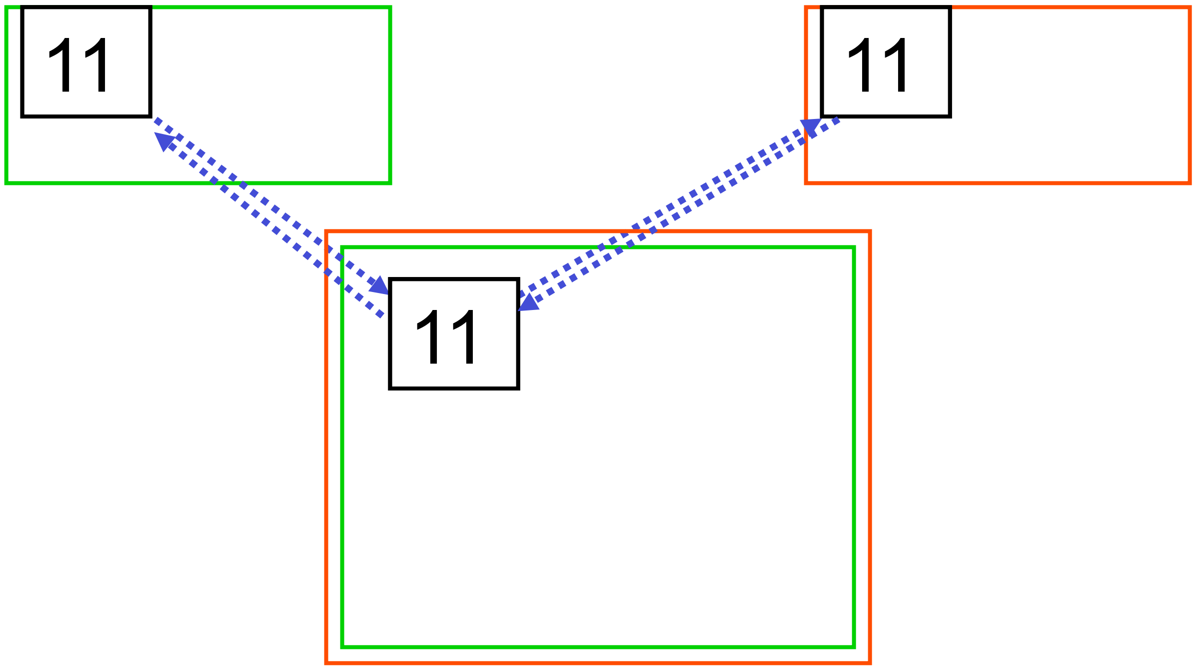
- In parallel programming, developers must manage exclusive access to shared resources
- Resources are in different forms:
  - concurrent read/write (including parallel write) to shared memory locations
  - concurrent read/write (including parallel write) to shared devices
  - a message that must be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

	Thread 1	Thread 2

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

Private data

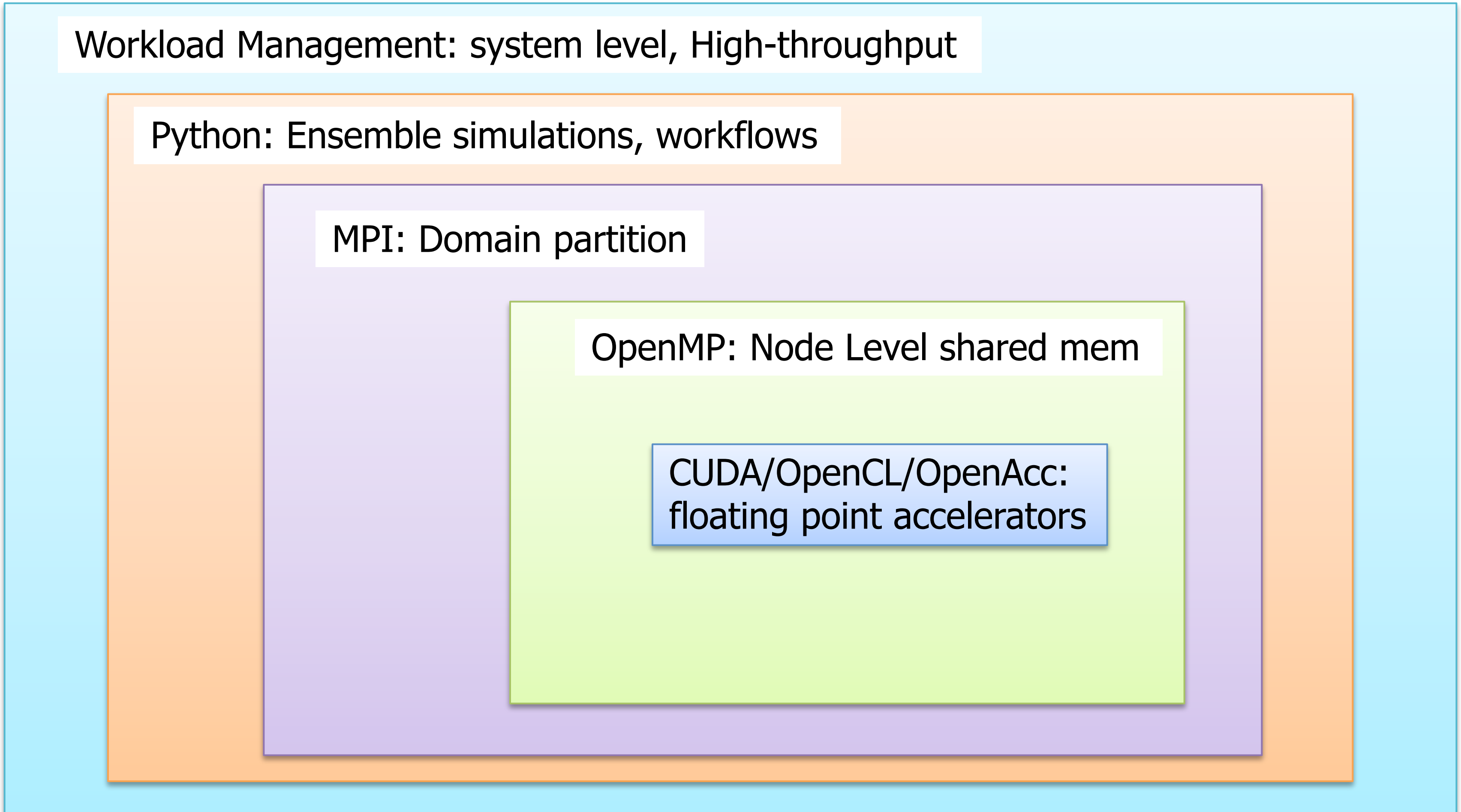
Shared data



Fundamental Tools of Parallel Programming

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

Thanks for your attention!!

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