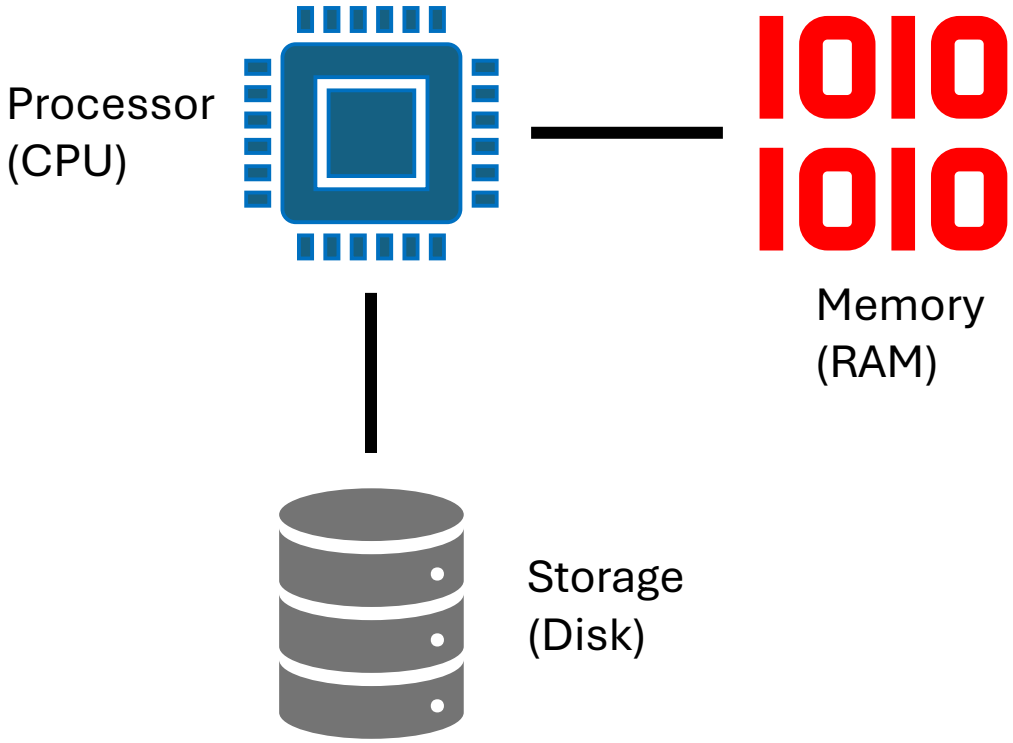
Parallelism –
Essential Concepts for HPC

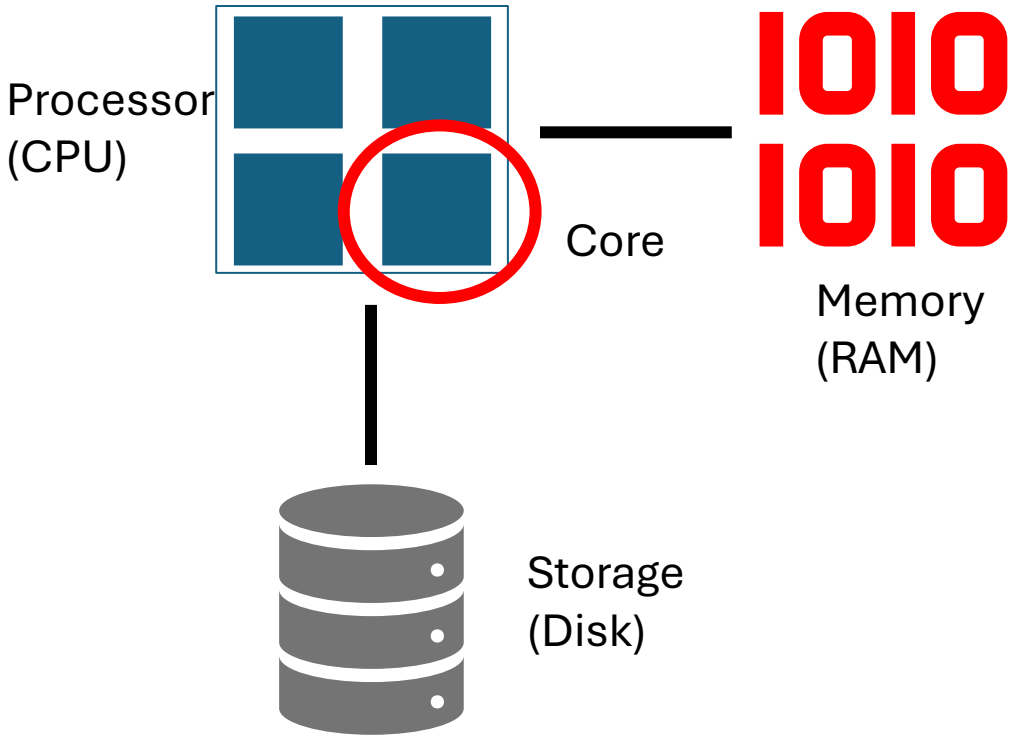
Dr. Lena Kanellou

kanellou@ics.forth.gr

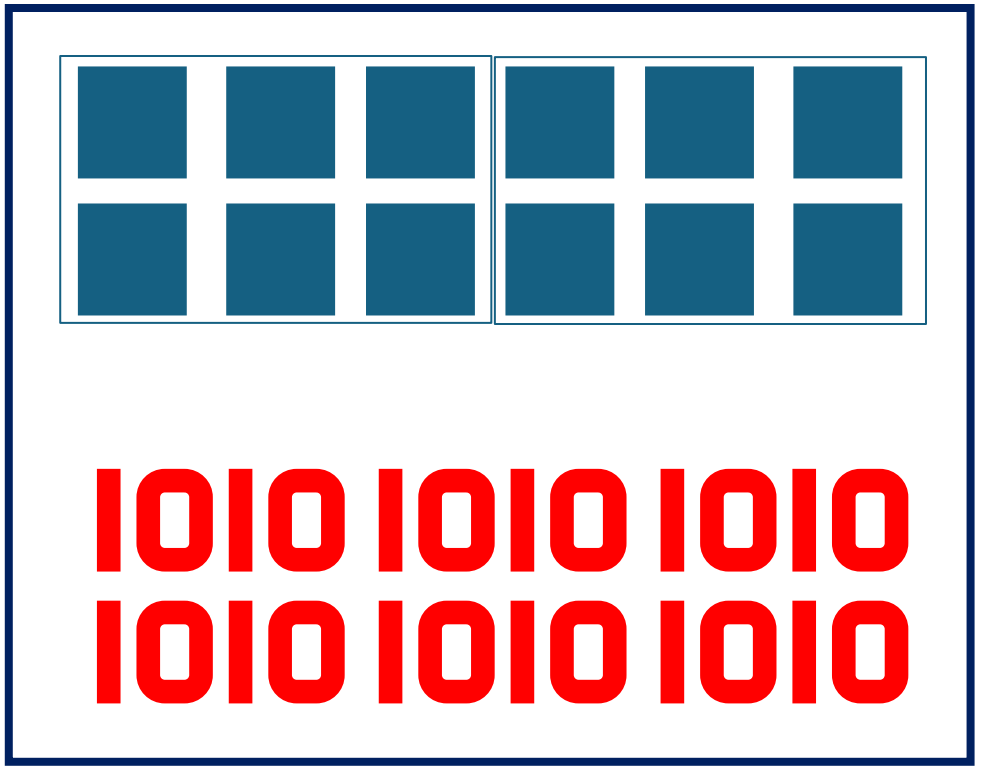
A typical HPC cluster anatomy



A typical HPC cluster anatomy

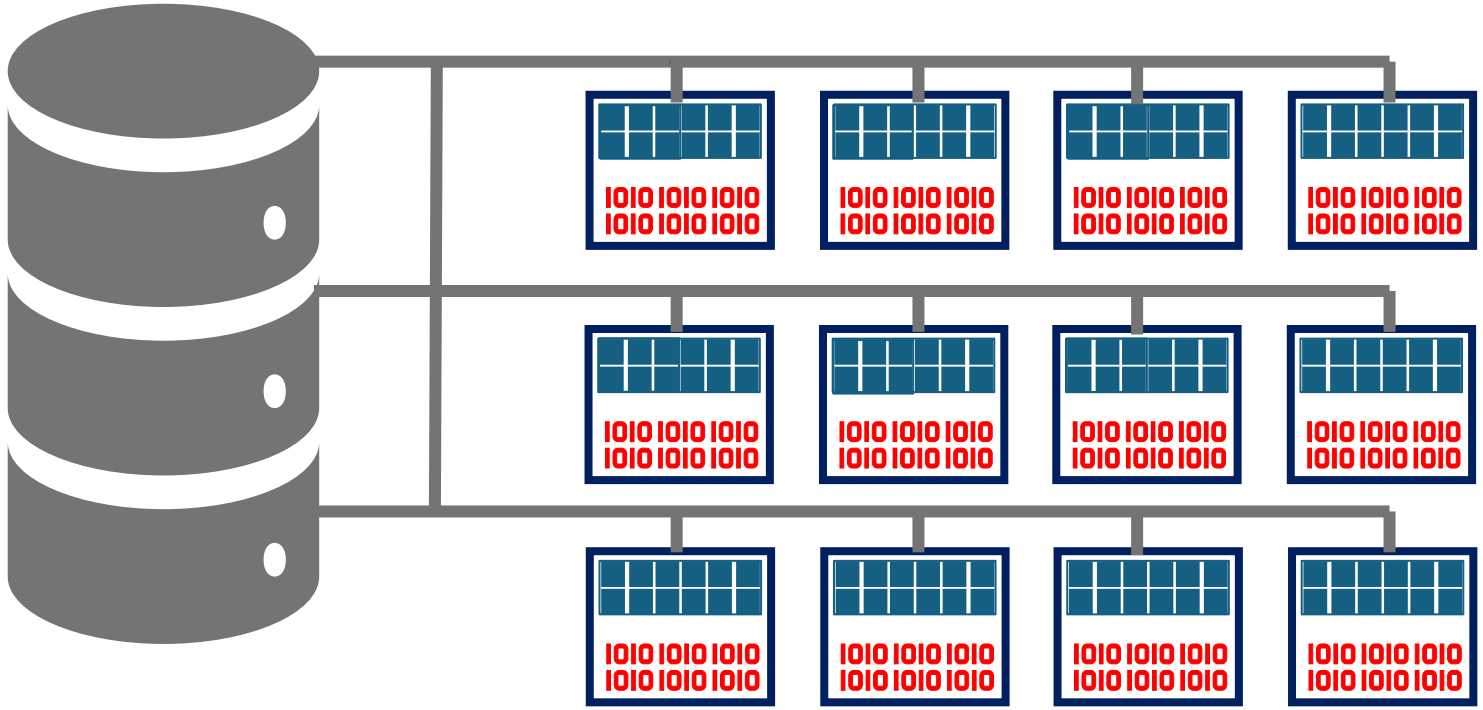


A typical HPC cluster anatomy

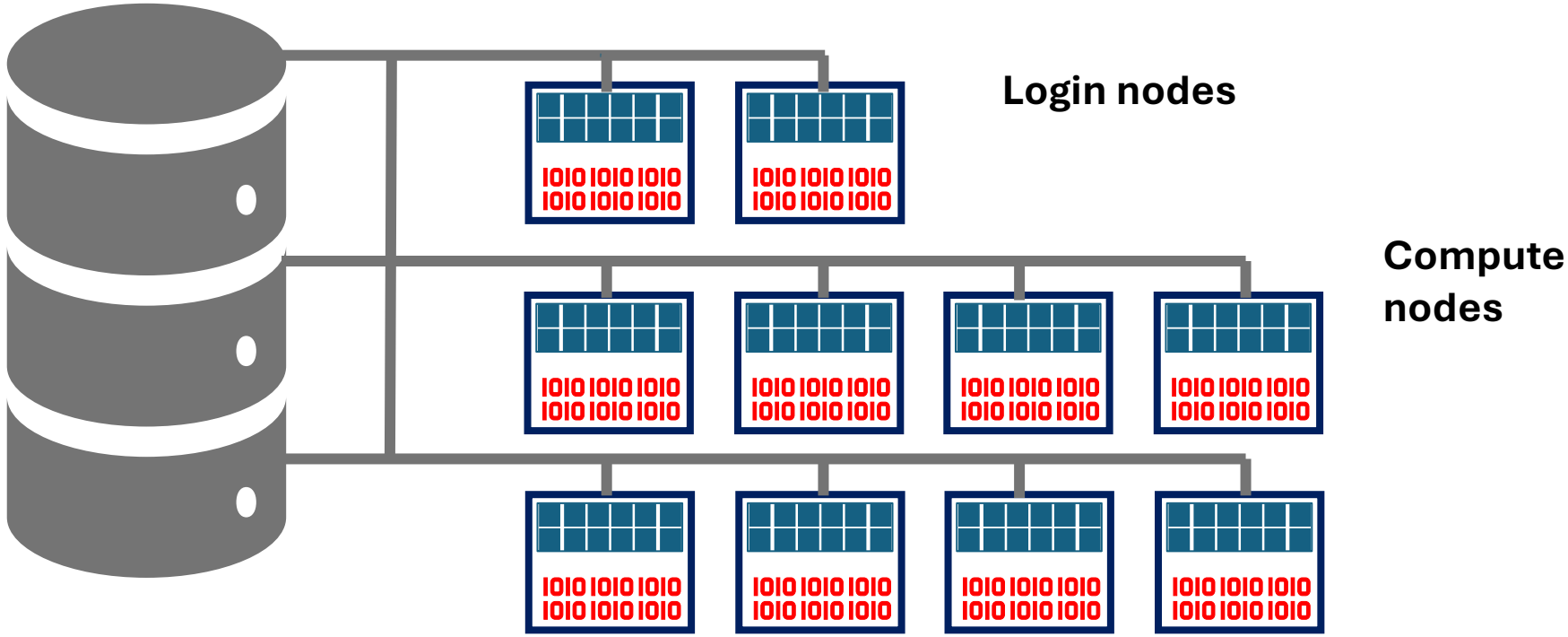


Node

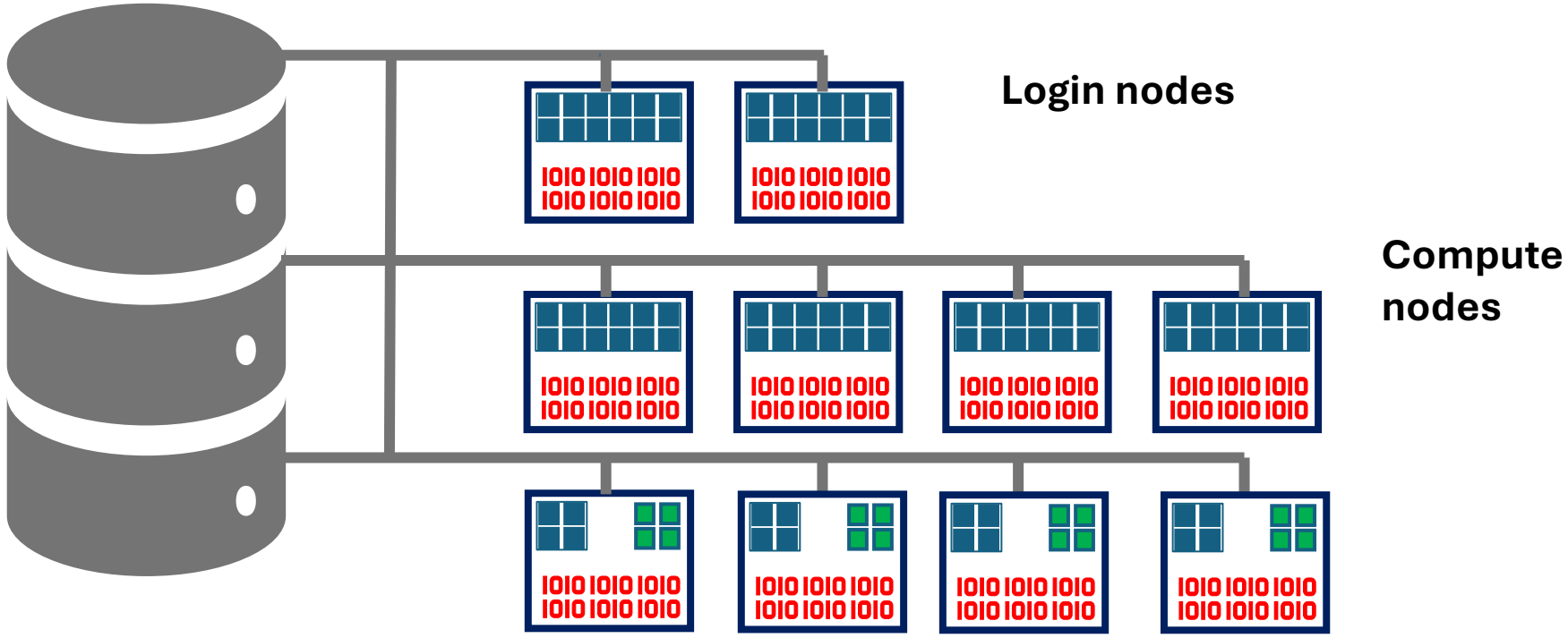
A typical HPC cluster anatomy



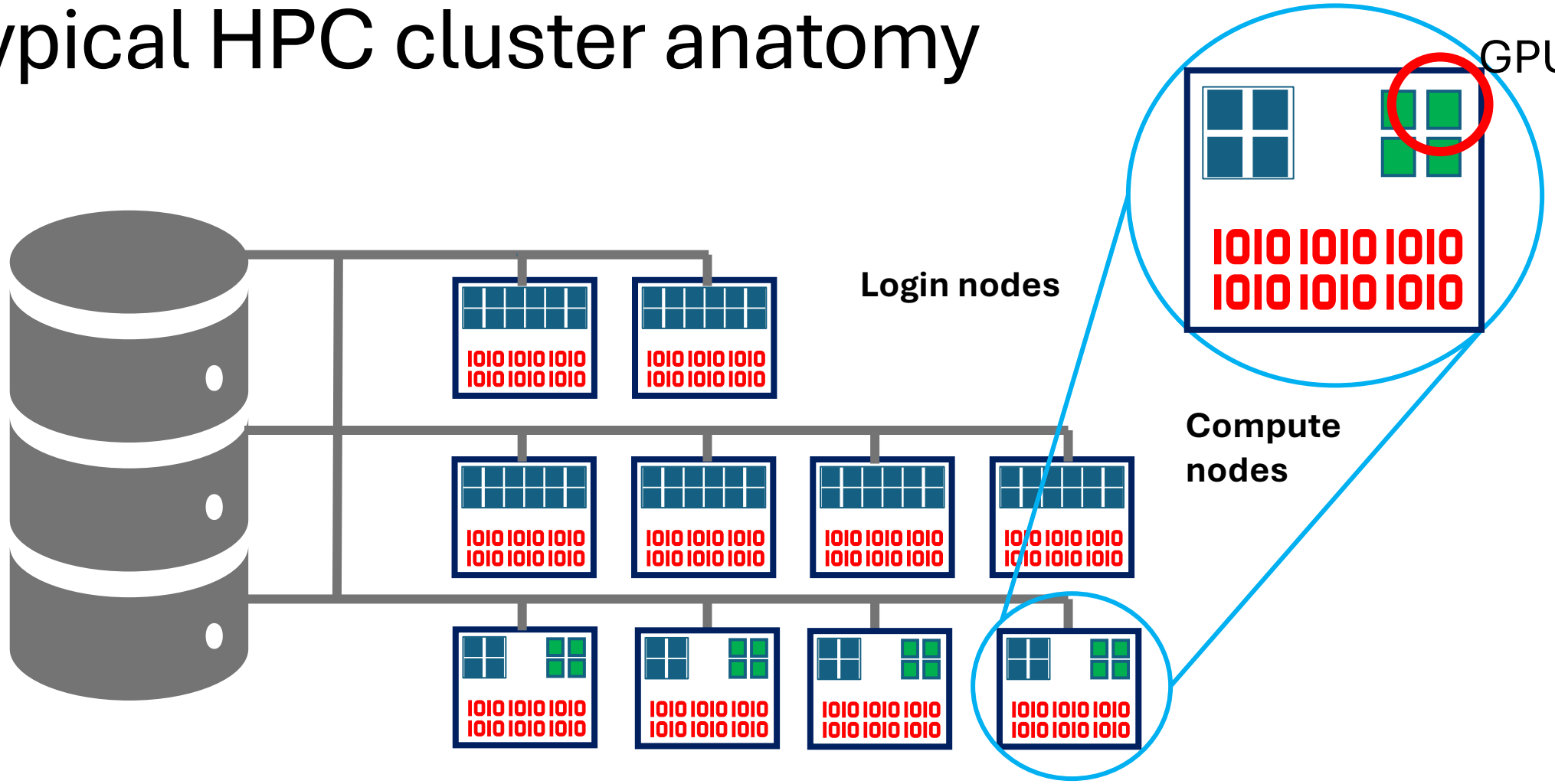
A typical HPC cluster anatomy



A typical HPC cluster anatomy



A typical HPC cluster anatomy



What is Parallelism in Computing?

But first...

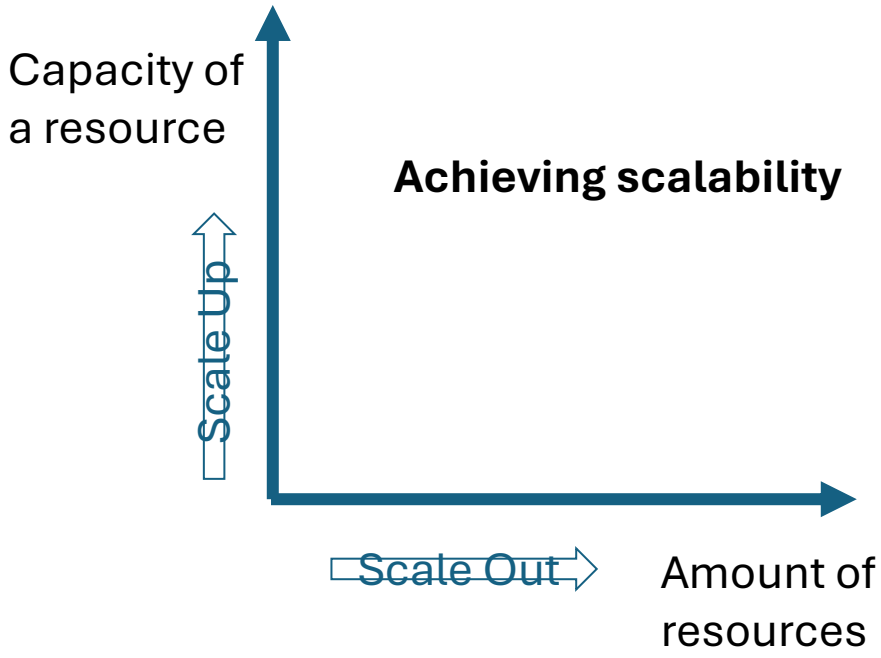

EURO
Greece
HPC Training Series
Course 16
Compute at Scale: Concepts
and Spotlight on AI, IoT, Simulations
| PRESENTATION LANGUAGE: GREEK |
SEPTEMBER 29, 2025 | 10:00 EET | ONLINE

Computing at scale

- Continue to function correctly as the load increases – be that load data, number of users, workload.
- Adapt to the increased load by adding more resources – be they computing units, storage, network bandwidth, etc
- Handle real-world situations

Scalability and scaling

Scalability: A *metric* that indicates the ability of a system to increase (or decrease) in performance and/or cost as a response to changes in processing demand.



Determining scalability

Strong scaling:
Increase the resources, while the problem size remains the same.

Weak scaling:
Increase the resources, and the problem size.

What is Parallelism in Computing?

What is Parallelism in Computing?

What is Parallelism in Computing?

- Parallelism in computing is the ability to perform multiple operations *at the same time*.
  - Hardware level: The machine contains several instances of the same resource.
  - Software level: The application consists of several (identical) parts.
  - Algorithm level: The algorithm accounts for the splitting of the work into several (identical) subtask.

Why do we care about Parallelism at Scale?

- Scalability is the ability of a system to maintain efficiency as you add more resources.
- Thus, parallelism in a system / algorithm / design, implies the capacity for scalability.
  - Splitting a given workload allows us to take advantage of all available resources.
  - It also allows us to add more resources to share the workload.

Parallelism – Some essential terminology

Processor	An independent execution unit, either conceptual or physical, capable of running tasks concurrently with others (also: worker)
Serial run	Running an application on a single processor.
Problem size	The amount of data or amount of memory or even the amount of calculation repetitions that are necessary to solve a particular problem.
Speed-up	The ratio of the time of the serial run to the time of the parallel run for a given problem and a given problem size.
Task	A unit of work that can be executed independently.

Limits of parallelism: Amdahl's law

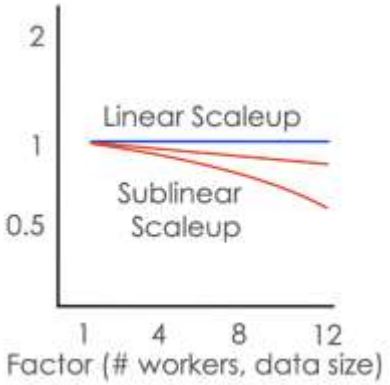
- Speed-up = $\frac{t_1}{t_N}$, where t_1 is the time a problem takes on one processor, as a serial run, and t_N is the time it takes when divided over N processors.
- However, not all parts of a problem can be parallelized!
  - Let F_p be the time required to execute the part of a problem that can be parallelized and F_s be the time required for the part that cannot. Then:
    $t_N = F_s + \frac{F_p}{N}$
- Amdahl's law was initially formulated with a fixed problem size in mind → measures **strong scaling** .

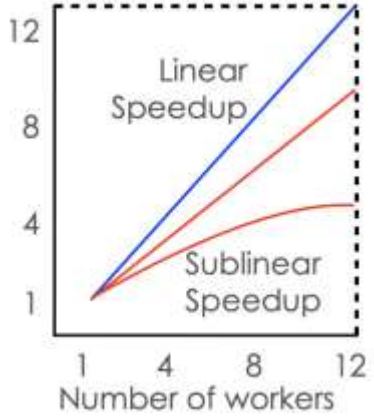
Gustafson's law

- When the problem size increases, as well as the number of processors, Amdahl's law is no longer accurate.
  - Gustafson found that the parallel part of a program scales with the problem size.
  - the amount of work done in parallel is linearly proportional to the number of processors
  - Scaled speed-up =$s + p \times N$, where s is the time required for the serial part of the problem and p is the time required for the parallel part of the problem. Thus, $s + p = 1$. Then, the scaled speed-up is: $1 + (N - 1) \times p$.
- Gustafson's law expresses **weak scaling**.

To recap

- **Weak scaling**: problem size increases proportionally to the number of parallel processes.
  - Helps analyze how big of a problem we can solve in a given system.
- **Strong scaling**: problem size remains the same for an increasing number of processes
  - Helps analyze how fast we can solve a problem of a given size, in a given system.





How to parallelize? Flynn's taxonomy

Sequential execution of instructions on a single data stream. Obsolete.

Example: traditional, early CPUs, e.g. Intel x86

Multiple instructions applied to a single data stream. Rare in practice.

Example: redundancy-based fault-tolerant systems.

SISD: Single instruction, single data

SIMD: Single instruction, multiple data

MISD: Multiple instructions, single data

MIMD: Multiple instructions, multiple data

One instruction applied to multiple data elements simultaneously

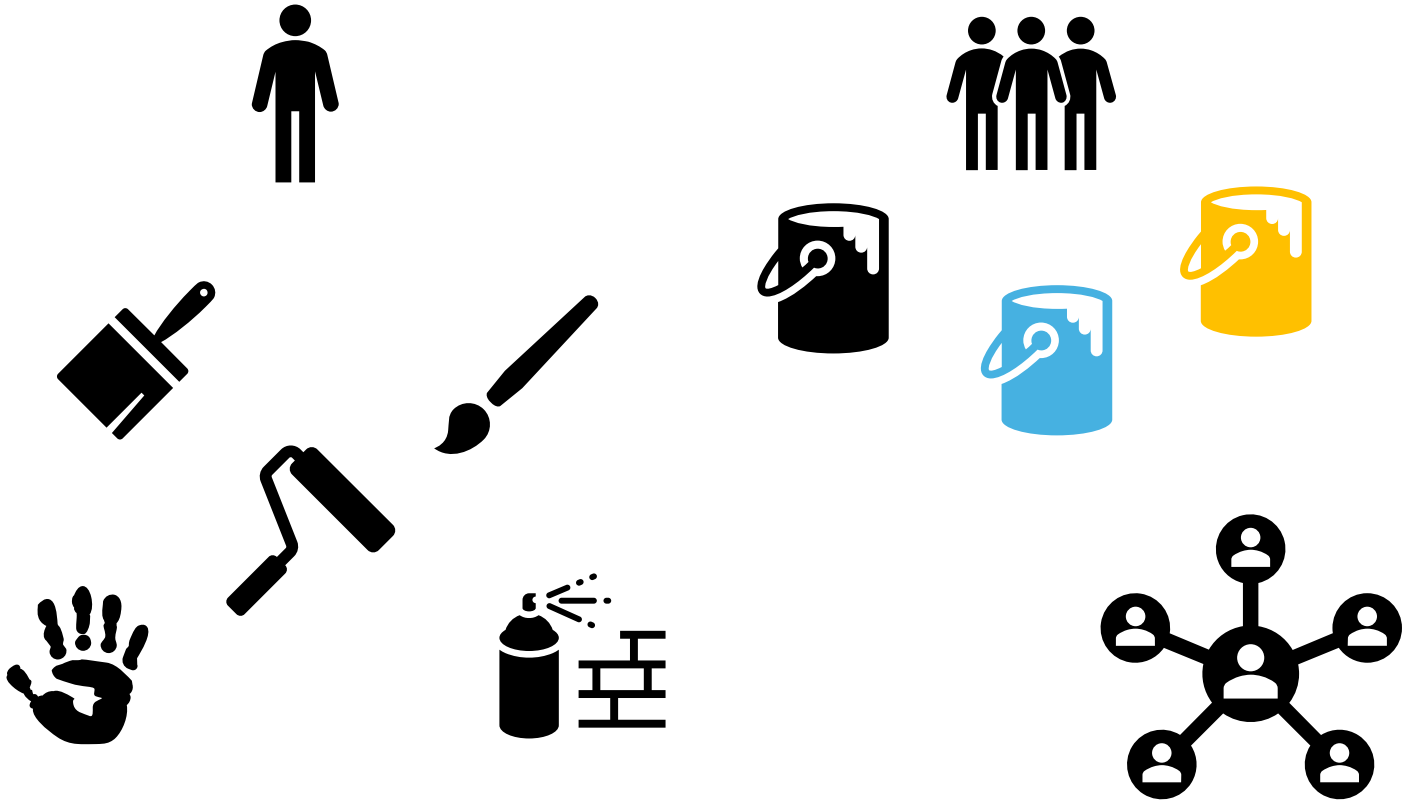
Examples: GPUs , vector processors,

Multiple processors execute different instructions on different data independently

Example: Multi-core processors (e.g. Core i7, AMD Ryzen), Symmetric Multiprocessing (SMP) systems, distributed systems

So, what is Parallelism in Computing?

So, what is Parallelism in Computing?



Parallel computing paradigms

- **Shared Memory parallelism**
  - multiple cores within a single node access and operate on a single, shared memory space.
  - can leverage multiple cores on a CPU or GPU (graphics processing unit).

Parallel computing paradigms

- **Distributed memory parallelism**
  - Tasks run on cores of separate nodes, each equipped with its own local memory.
  - Communication is achieved via exchange of messages on the interconnect between nodes.

Parallel, concurrent, distributed

- **Parallelism**:
  - use multiple cores or multiple processors where each performs a (identical) task independently.
  - May use a shared memory, may use individual memories.
  - Implies multiple instances of a computing resource.

Example: A GPU running multiple threads simultaneously.

Parallel, concurrent, distributed

- **Concurrency**:
  - perform multiple tasks with *overlapping durations*.
  - This may – or may not! – entail multiple processors!
    - Can even be achieved on a single-core processor by making the tasks time-share the processor.
      - Preemptive multitasking.
      - Process communication is achieved via pipes and signals.
  - Nowadays commonly implies a shared memory paradigm.

Example: A mobile phone web browser, handling page rendering, network requests, and user input.

Parallel, concurrent, distributed

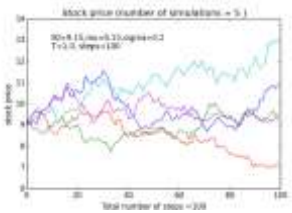
- **Distributed computation**:
  - Multiple processors communicate via messages.
  - Computation can be split in various ways:
    - Processors may each perform a (identical) task independently
    - Processors may co-operate or co-ordinate to perform one common application, split into different tasks that have interdependencies.
    - Processors may execute different applications that require synchronization for the access to shared resources.

Example: A Hadoop cluster of servers, each with its own memory, working together over a network to perform big data processing.

Types of parallelizable problems in HPC

Embarassingly parallel

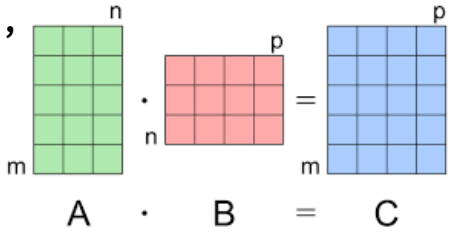
The problem consists of a collection of tasks which are entirely independent and do not need to communicate with each other. No inter-task communication or synchronization is required after tasks are launched.



Examples: Monte Carlo simulation, 3D video rendering on GPU, parameter sweeps

Data parallel

The problem consists of repeatedly applying the same operation (or sequence of operations) on large amounts of data that are split across workers. After each cycle, a synchronization phase takes place among workers, in which possibly some further operation, e.g., summation, reduction, etc, is applied on the produced data.

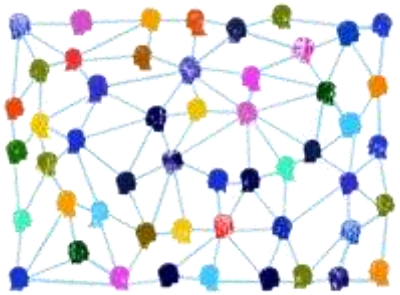


Examples: Training deep learning models, matrix multiplication.

Types of parallelizable problems in HPC

Task parallel

Different tasks execute in parallel on a set of data, instead of splitting it up. There may be dependencies among the tasks, which can be represented as a Directed Acyclic Graph.



Examples: Scientific simulation pipelines, sparse matrix factorization, parallel graph algorithms.

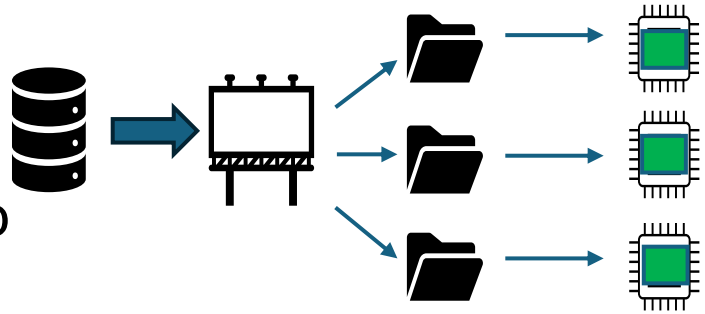
Pipeline Parallel

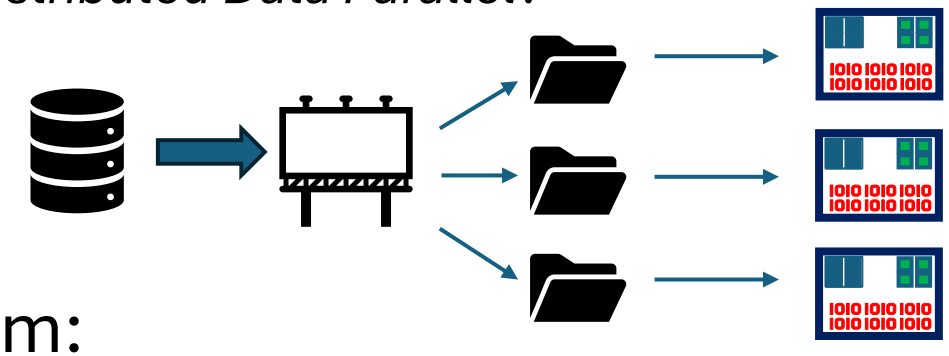
The computation required to solve the problem can be broken into stages and each stage can run concurrently on different data – like in an assembly line.

Examples: Streaming data processing, training transformers layer by layer across GPUs.

Examples of parallelism in ML training

- Data parallelism:
  - Split data into sections, replicate the model into multiple GPUs, assign a data portion to a GPU
    - *What about Distributed Data Parallel?*



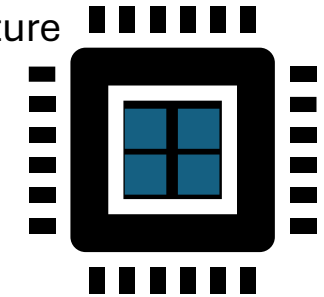


- Model parallelism:
  - Split the model into partitions, assign a partition to each GPU, manage data flow so as to appropriately process the relevant data in each GPU

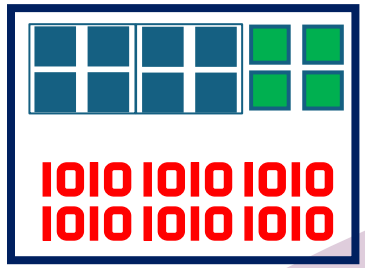
Takeaway: Parallelism everywhere

Pipelining on the core level

Multi-core architecture



Multi-processor node



Multi-node HPC infrastructure

Thank you!

Dr. Lena Kanellou

kanellou@ics.forth.gr