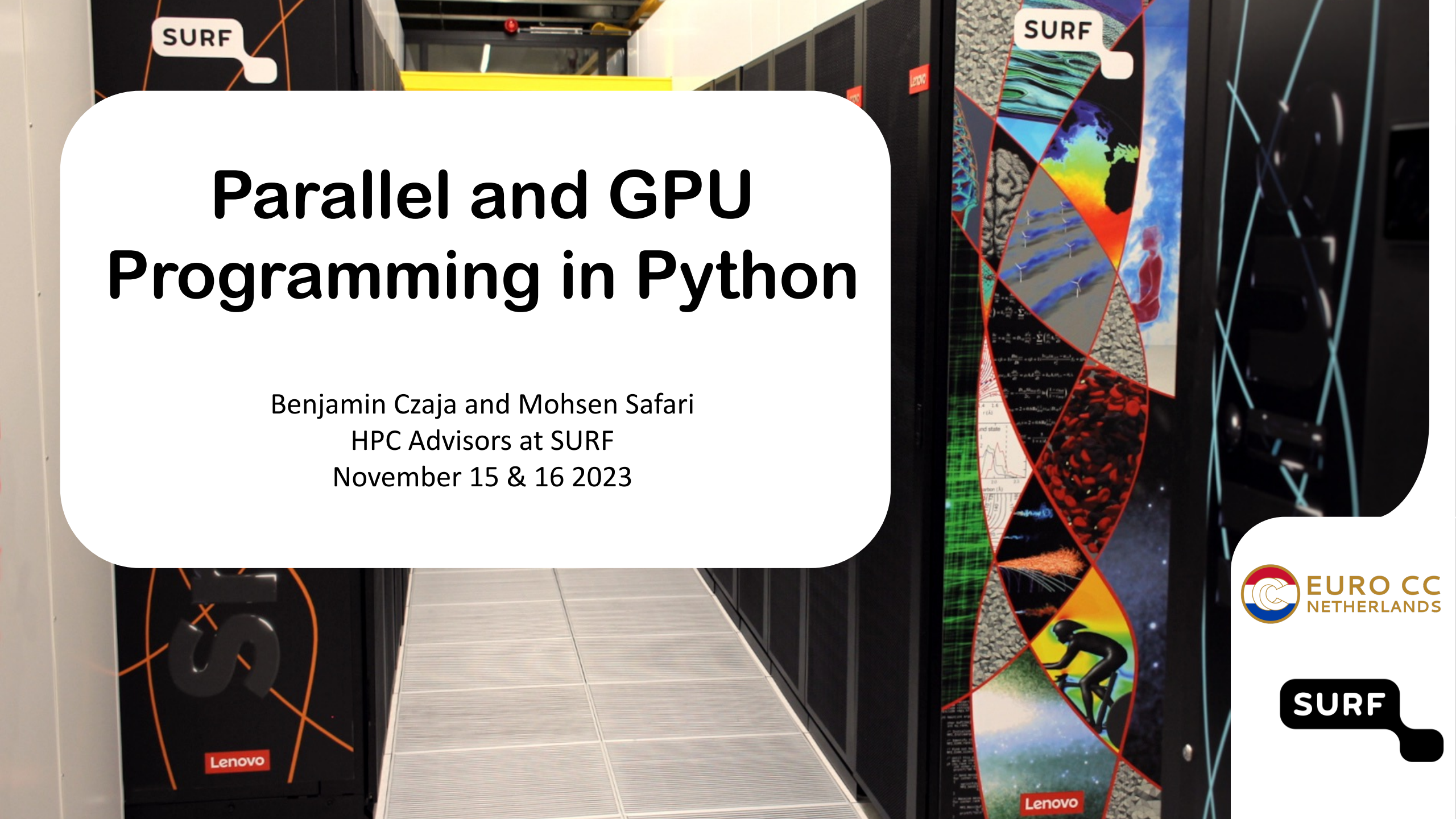


Parallel and GPU Programming in Python

Benjamin Czaja and Mohsen Safari
HPC Advisors at SURF
November 15 & 16 2023

EURO CC NETHERLANDS

SURF

Course objective

- Where?
- What?

Course objective

- Where?
  - Multi-core processors (CPUs)
  - Many-core GPUs
- What?
  - Python

Content Outline (Day 1)

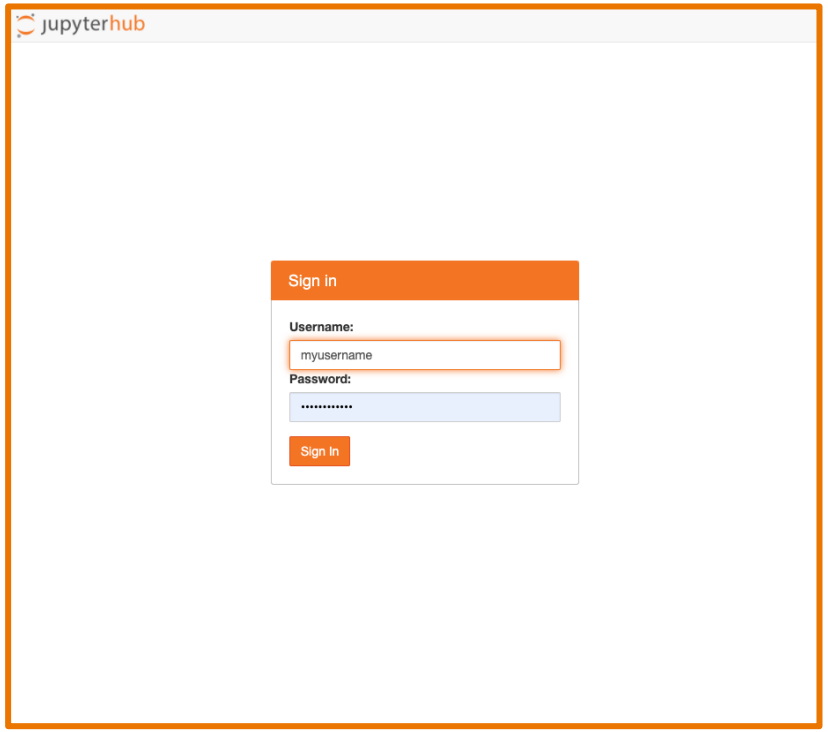
- Introduction to Compute architectures
  - Moore's Law
  - Multicore processors
- Introduction to Python
  - Global Interpreter lock (GIL)
- Parallelism in Python
  - Threads vs Processes
  - Concurrent.futures
- HPC with Python
  - Profiling
  - Numba
- Hands-on and Q&A!!!

Parallel and GPU programming in Python
Day 1

Start Time	End Time	Subject
10:00	11:00	Introduction to Python and Parallel Computing
11:00	11:15	Coffee Break
11:15	12:30	Hands-on: Introduction to Python and Parallel Programming
12:30	13:00	Lunch
13:00	13:45	Hands-on: Threads vs Processes
13:45	14:00	Coffee Break
14:00	15:00	Hands-on: Profiling/Numba

Parallel and GPU programming in Python

- Set up of the training
  - Account on the system
  - Access to the Jupyter Hub
  - Training material and hands-on (on the cluster!)
  - https://github.com/sara-nl/Parallel-and-GPU-programming-in-Python

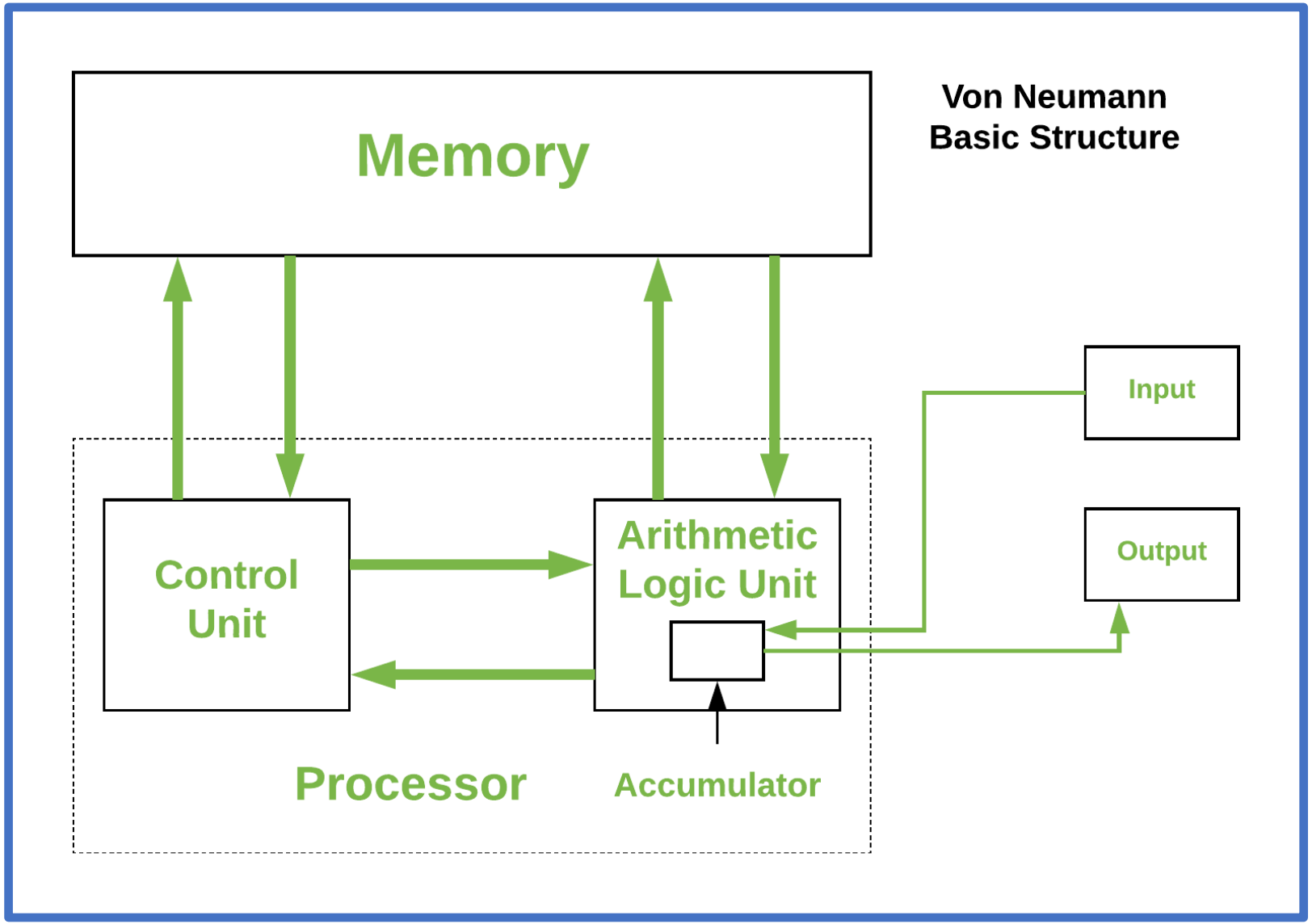


- Jupyter-hub

  URL: https://jupyter.lisa.surfsara.nl/jhlsrf018

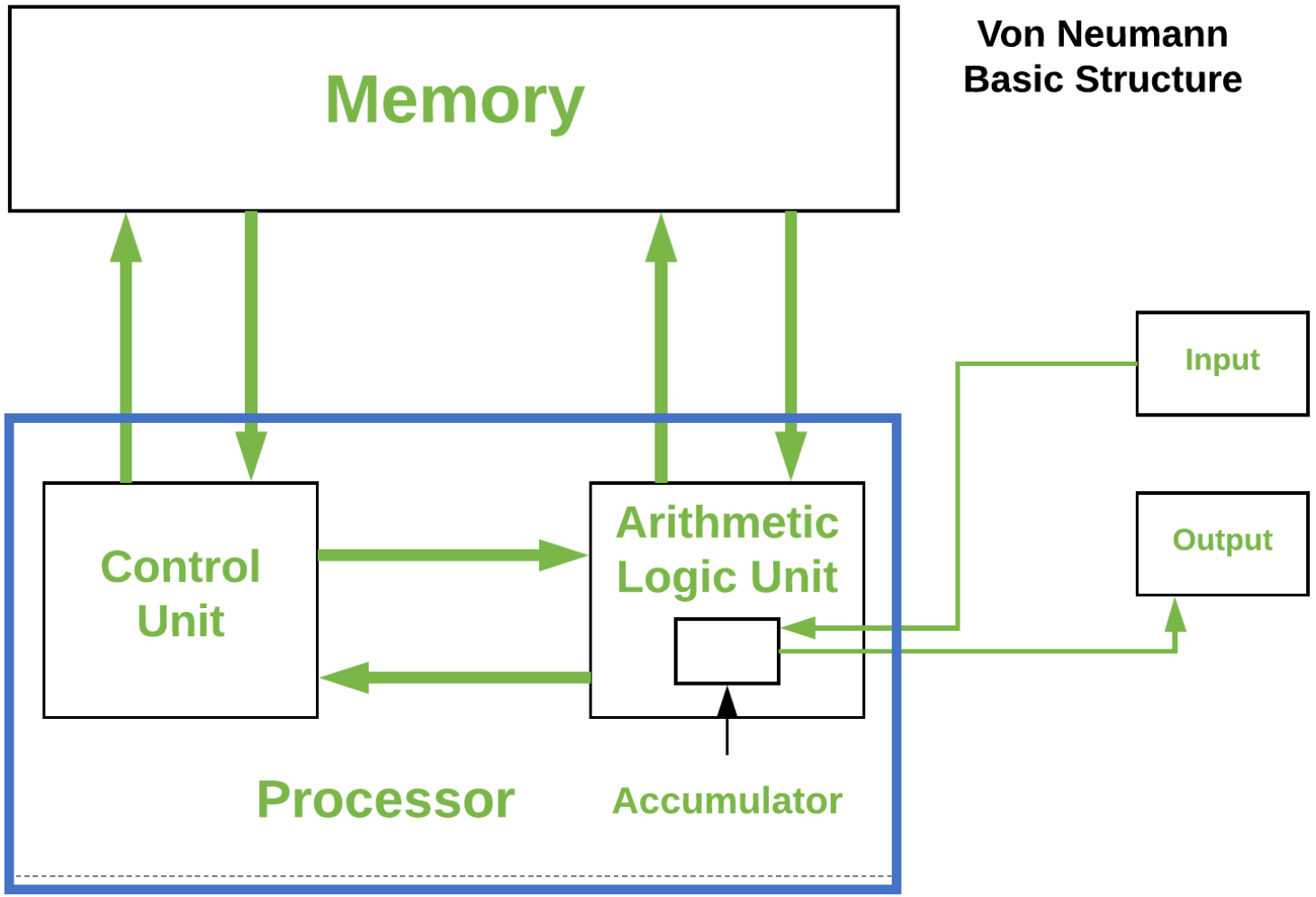
Introduction to the “where”…. CPU architecture

The Basics: The architecture of a computer



1. Central Processing Unit (CPU)
2. Main Memory Unit
3. Input/Output Device

The Basics: What is a Processor?



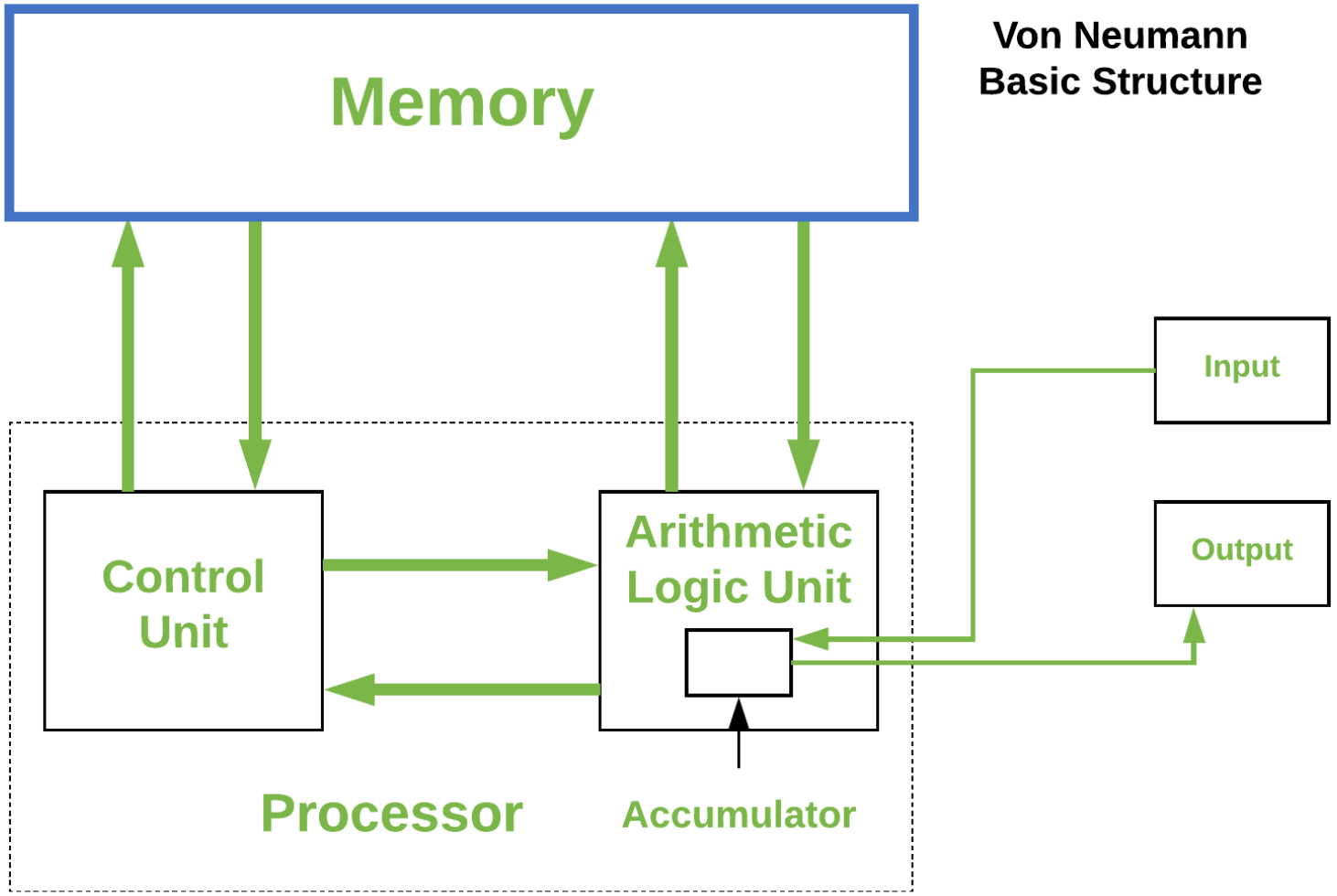
Central Processing Unit (CPU)

1. **Control Unit**. Handles all processor control signals. It directs all input and output flow, fetches code for instructions, and controls how data moves around the system.

2. **Arithmetic Logic Unit.** Handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic operations.

3. **Registers.** High-speed storage areas in the CPU.

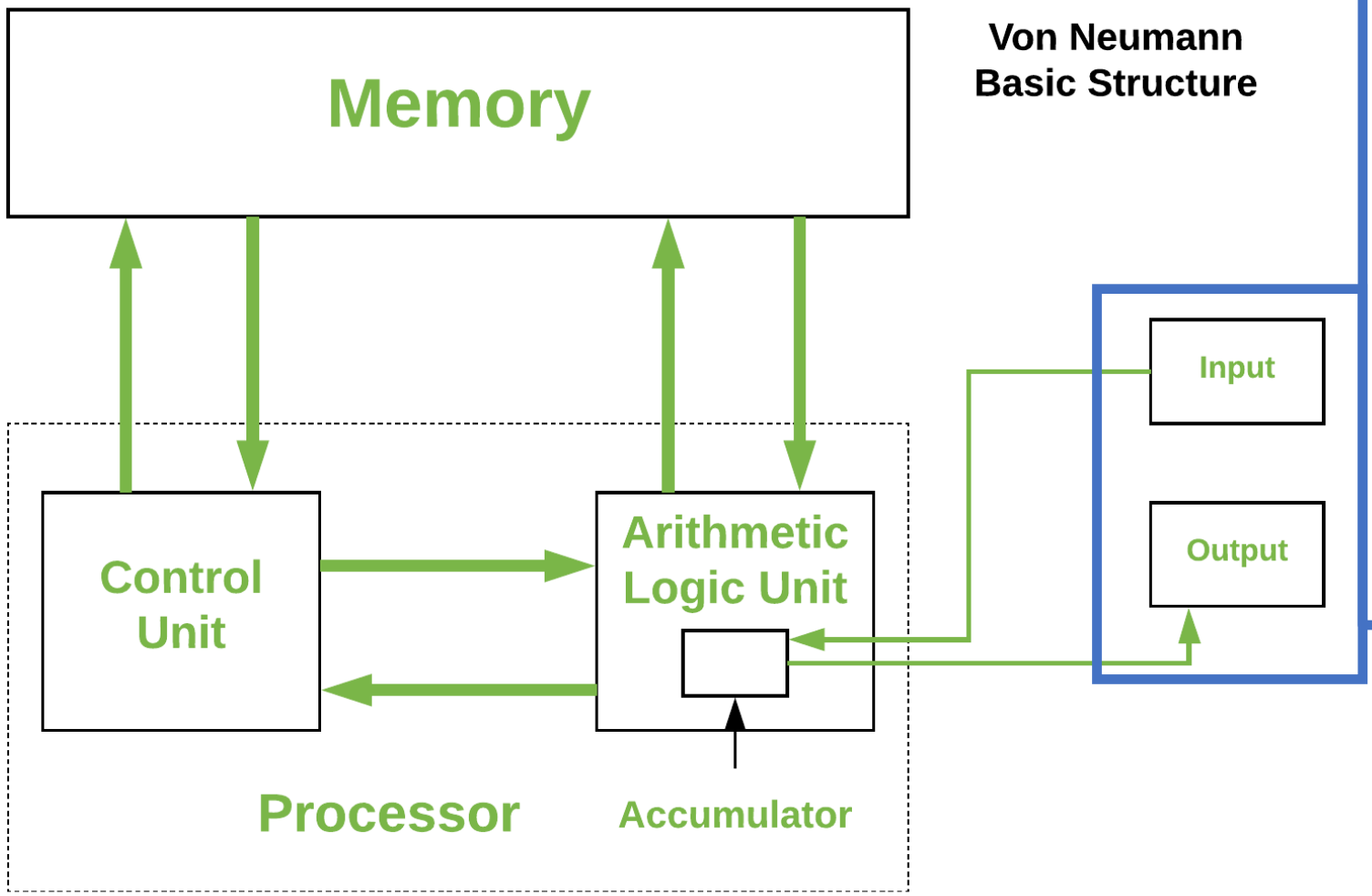
The Basics: What is Memory?



Memory

- Memory is the electronic holding place for the instructions and data a computer needs to reach quickly. It's where information is stored for immediate use.
- Primary Memory/random access memory (RAM)
- **Volatile**, meaning it isn't retained when the computer is turned off

The Basics: What is I/O?

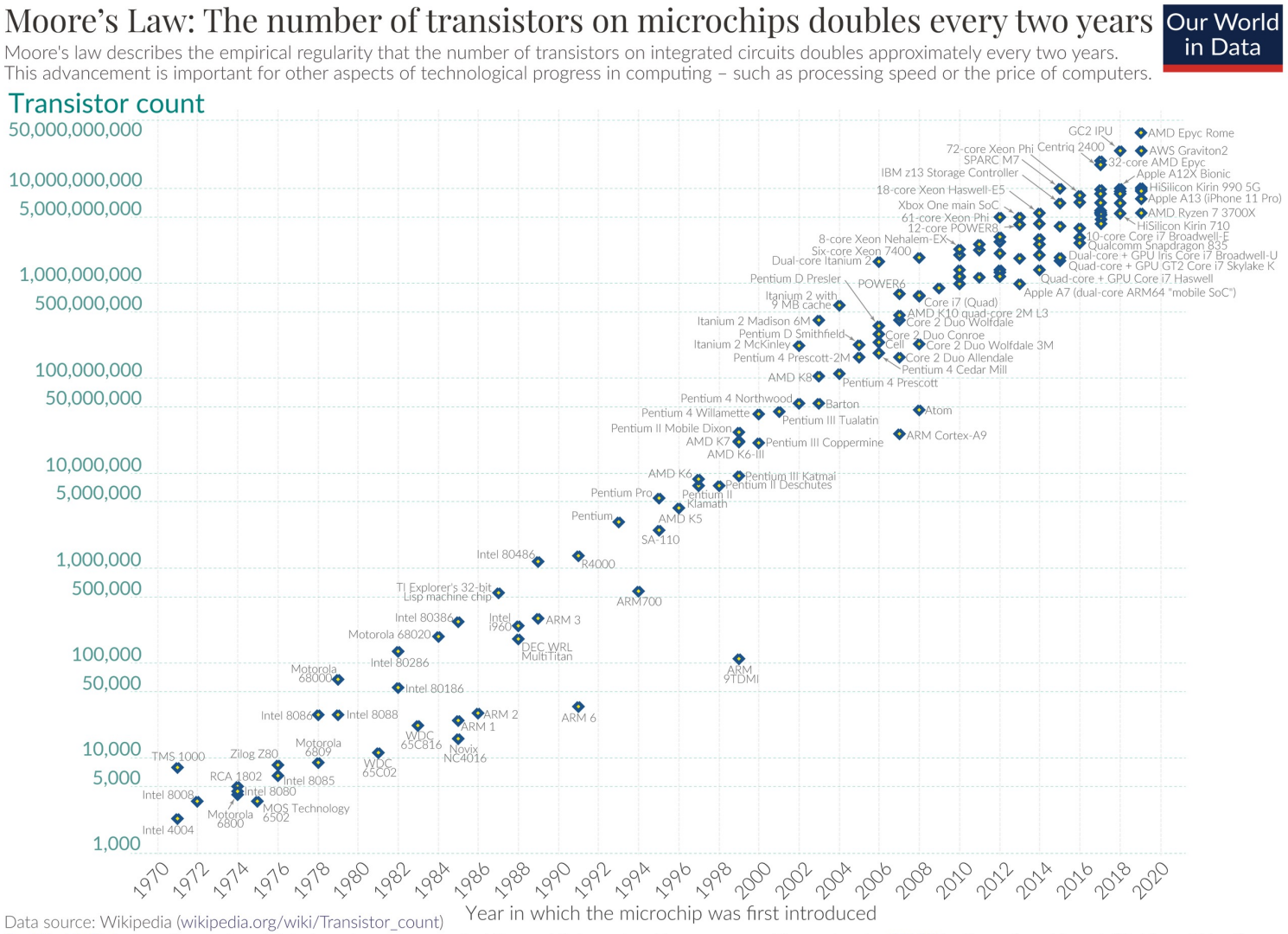


Von Neumann Basic Structure

Input/Output (I/O)

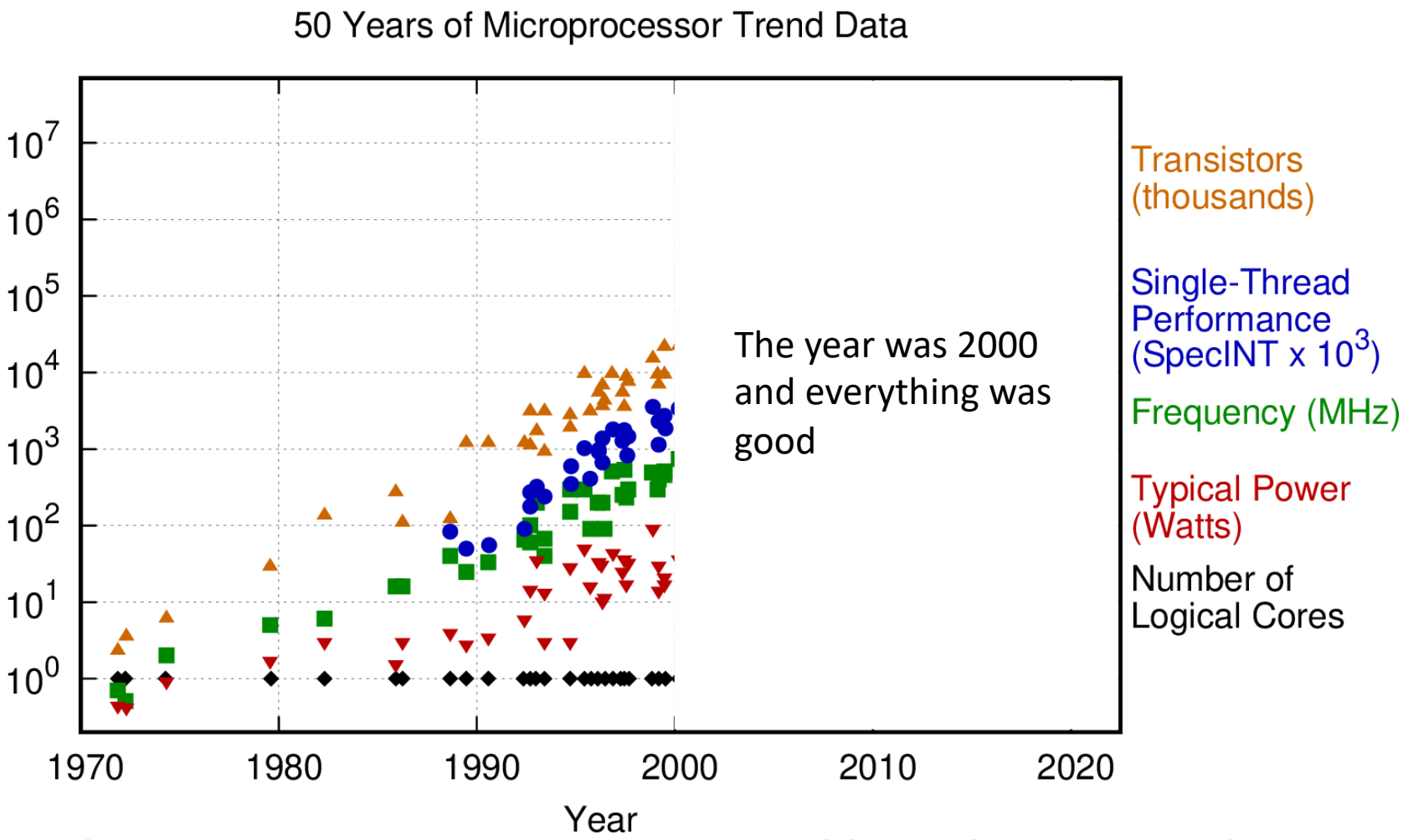
- Information processing system designed to send and receive data from a computer hardware component, device, or network.
  - Keyboard
  - Other computers in a network
  - Storage device (Hard drive)

Moore’s Law



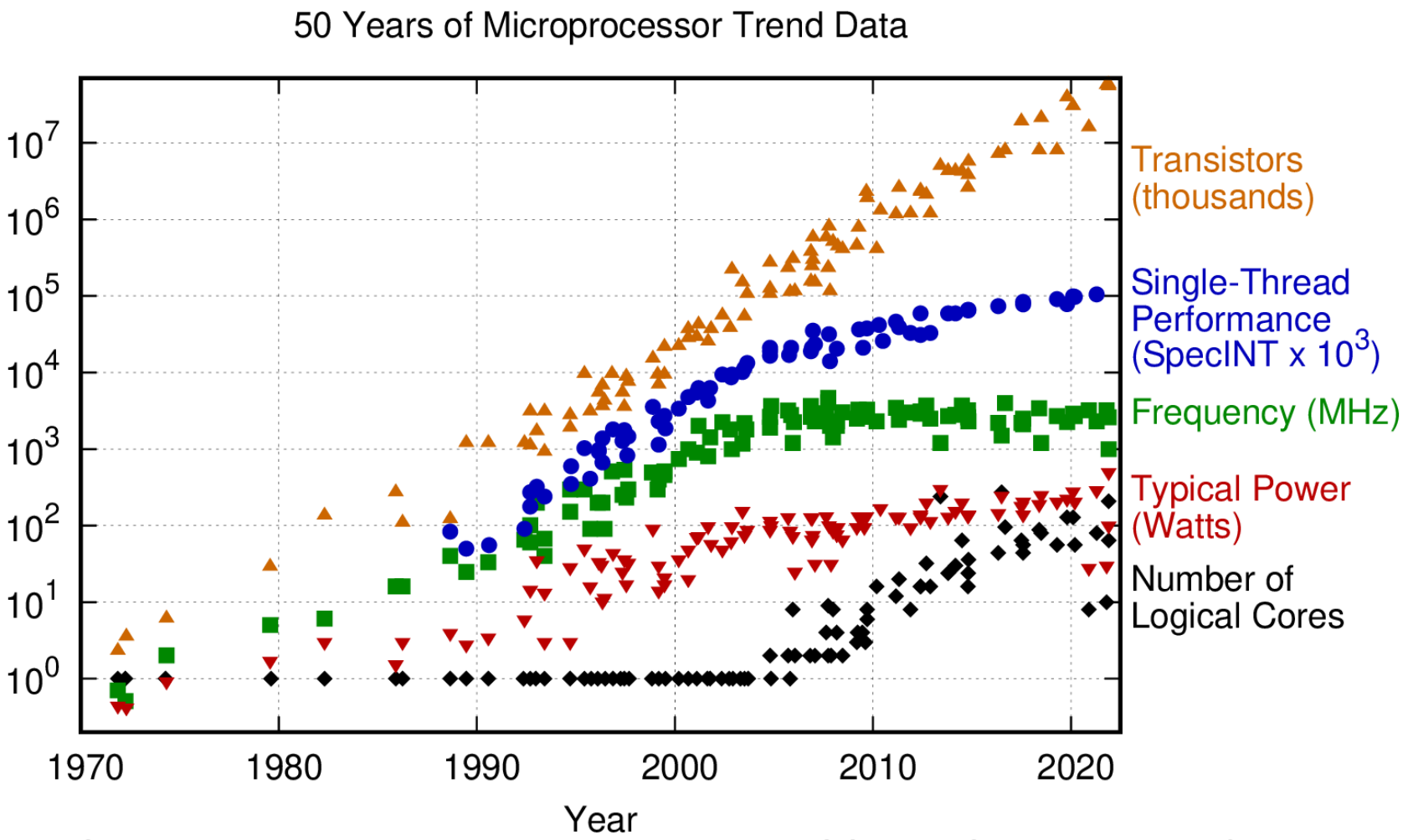
- In 1965, Gordon E. Moore co-founder of Intel postulated that the number of transistors that can be packed into a given unit of space will double about every two years.
    - Built into this is the assumption that processor performance increases every two years because of this, yet the cost will become less.

Moore's Law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

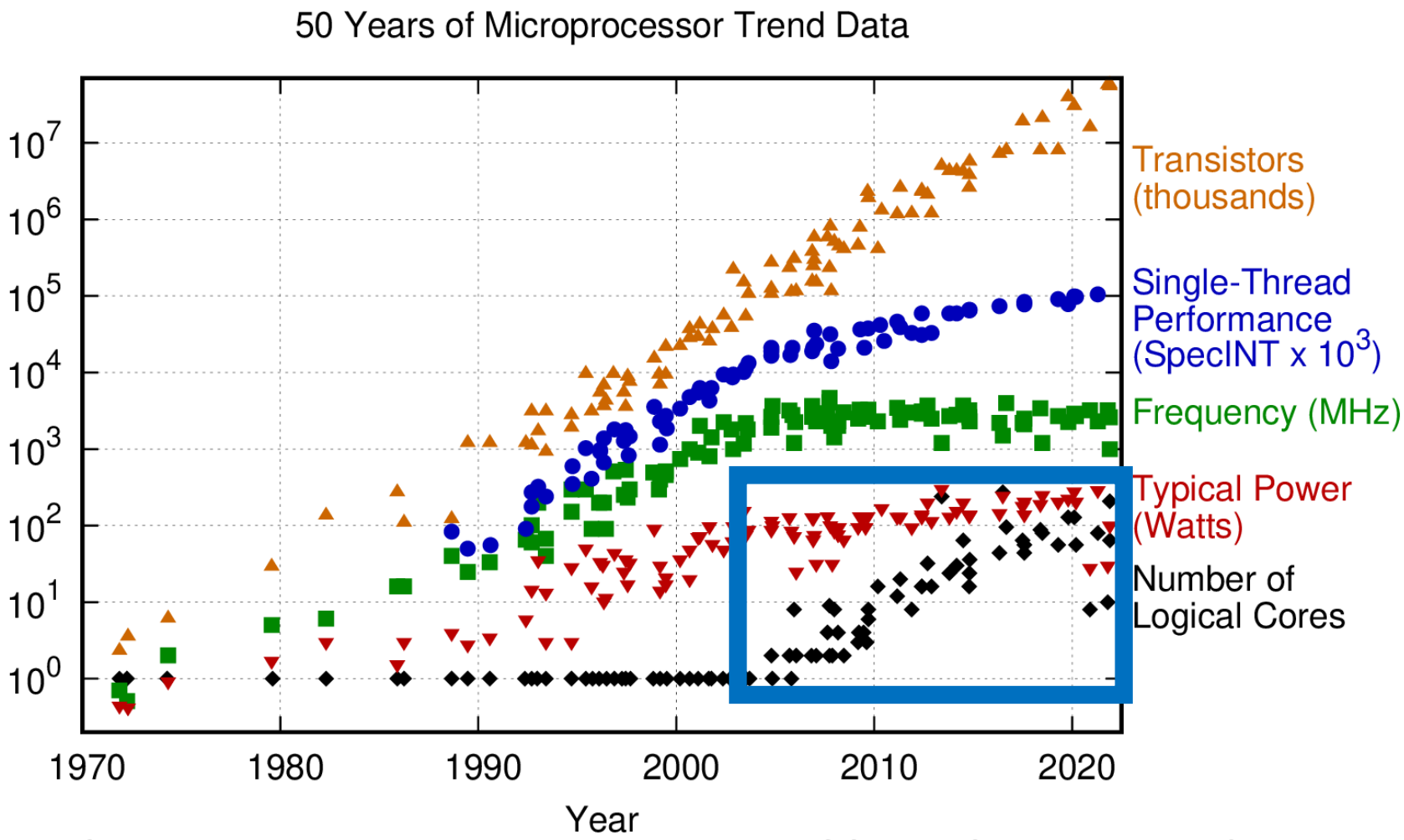
Moore's Law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

- Single-thread performance has flattened

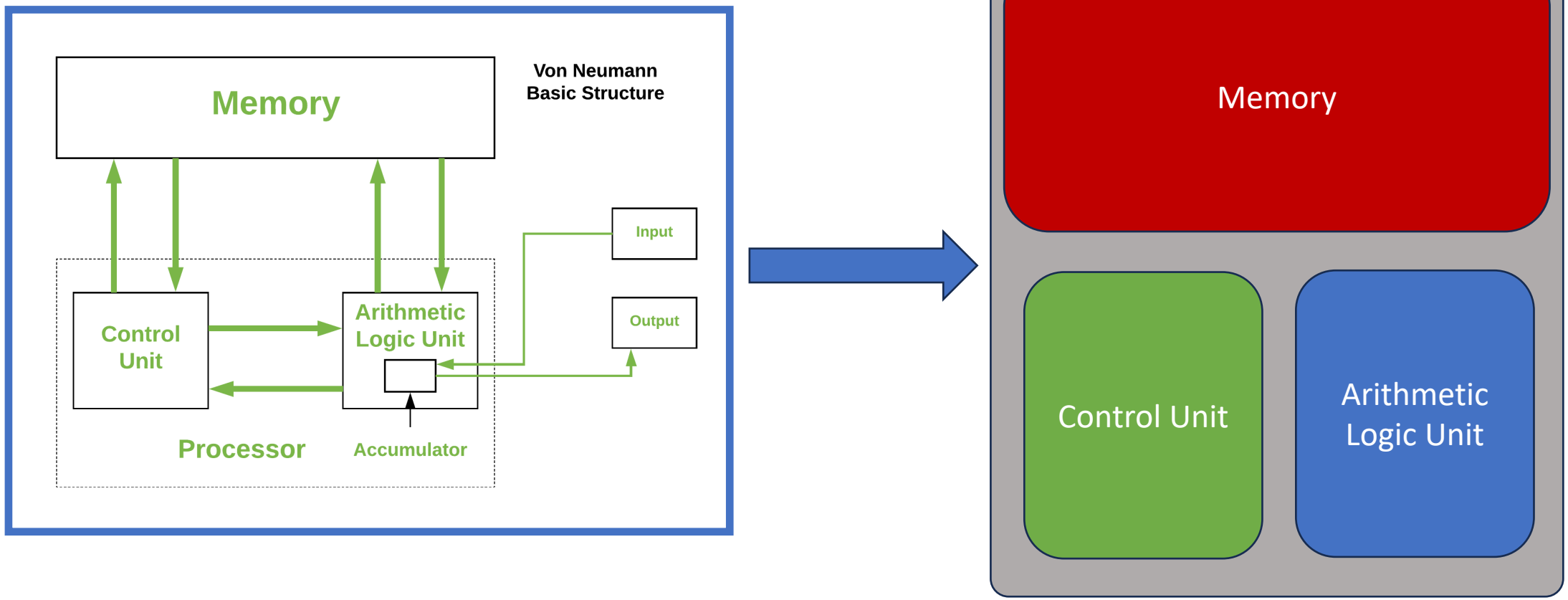
Moore's Law



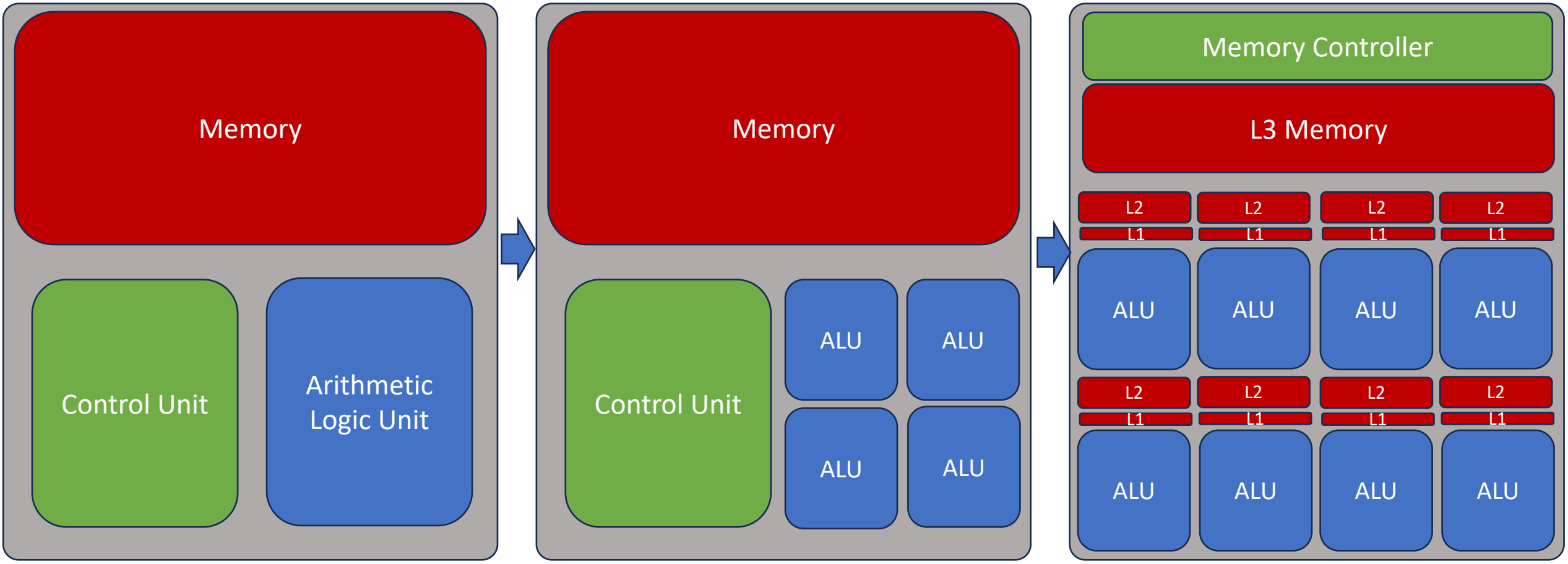
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

- Single-thread performance has flattened
- The number of logical cores has increased

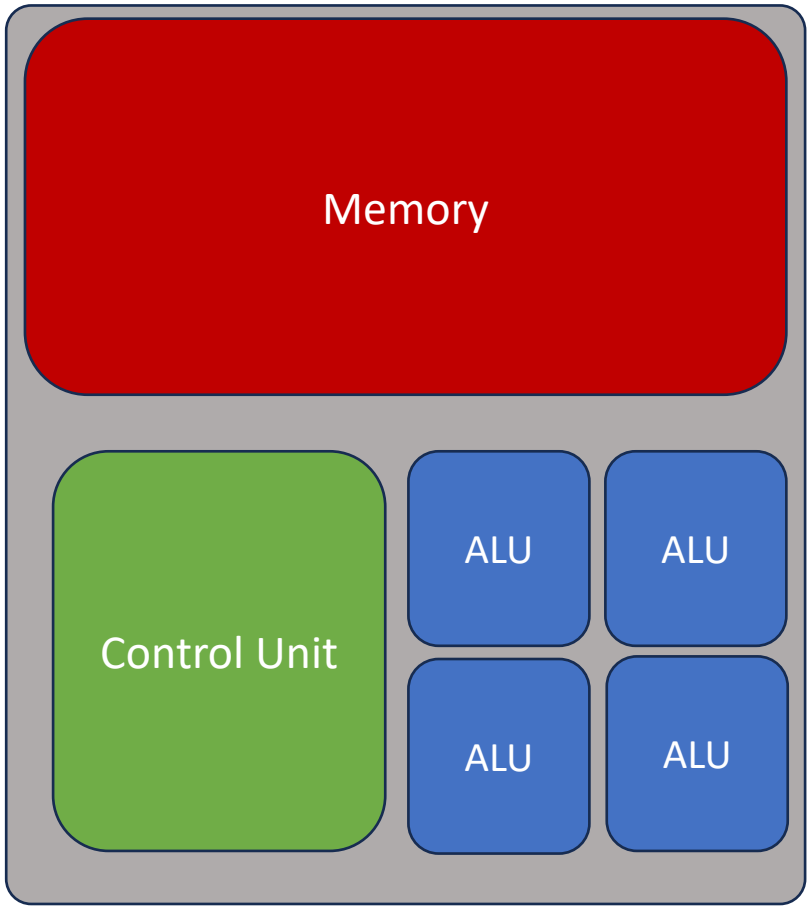
Re-ordering….. Single “core” Processor/CPU



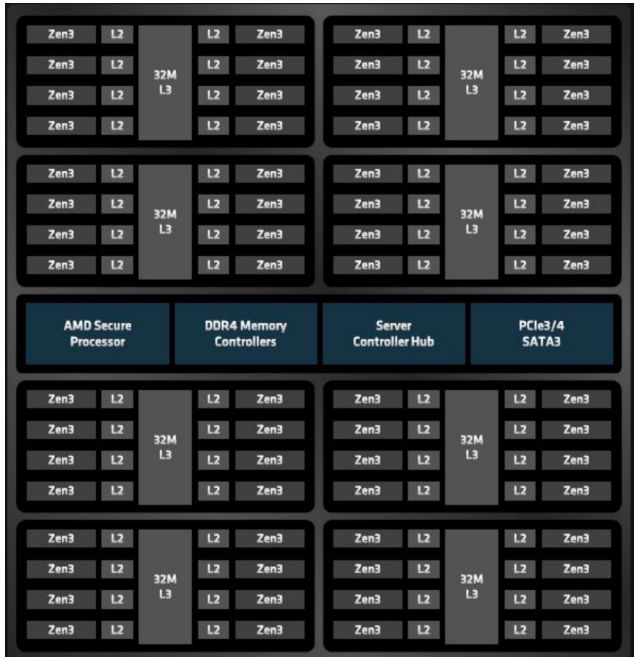
Single-core CPU to Multi-core CPU



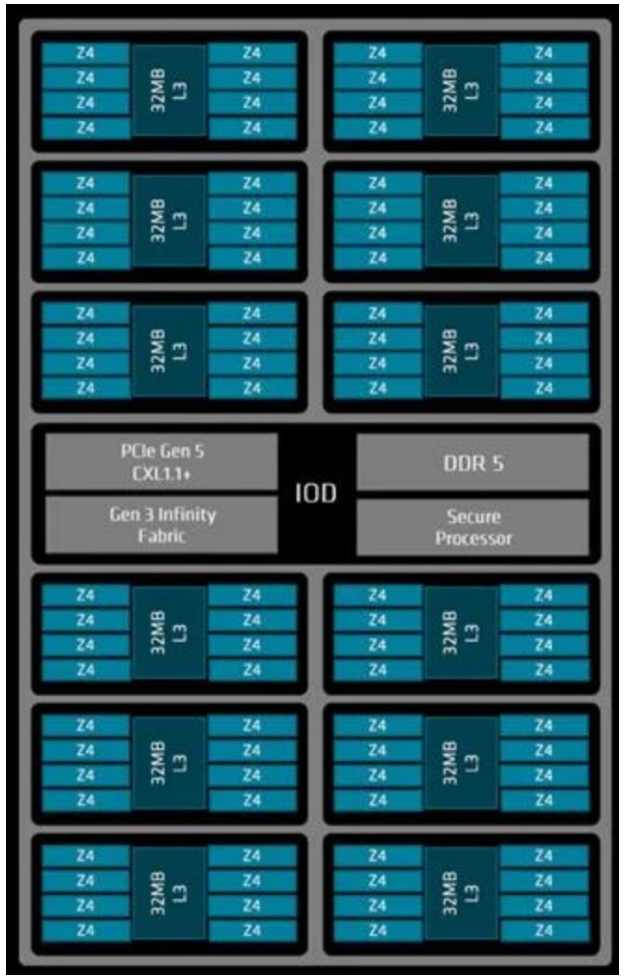
Multi-core
4 cores



AMD Rome (Zen2)
64 cores

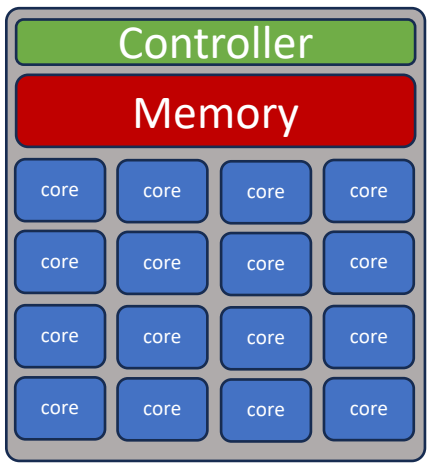


AMD Genoa (Zen4)
96 cores



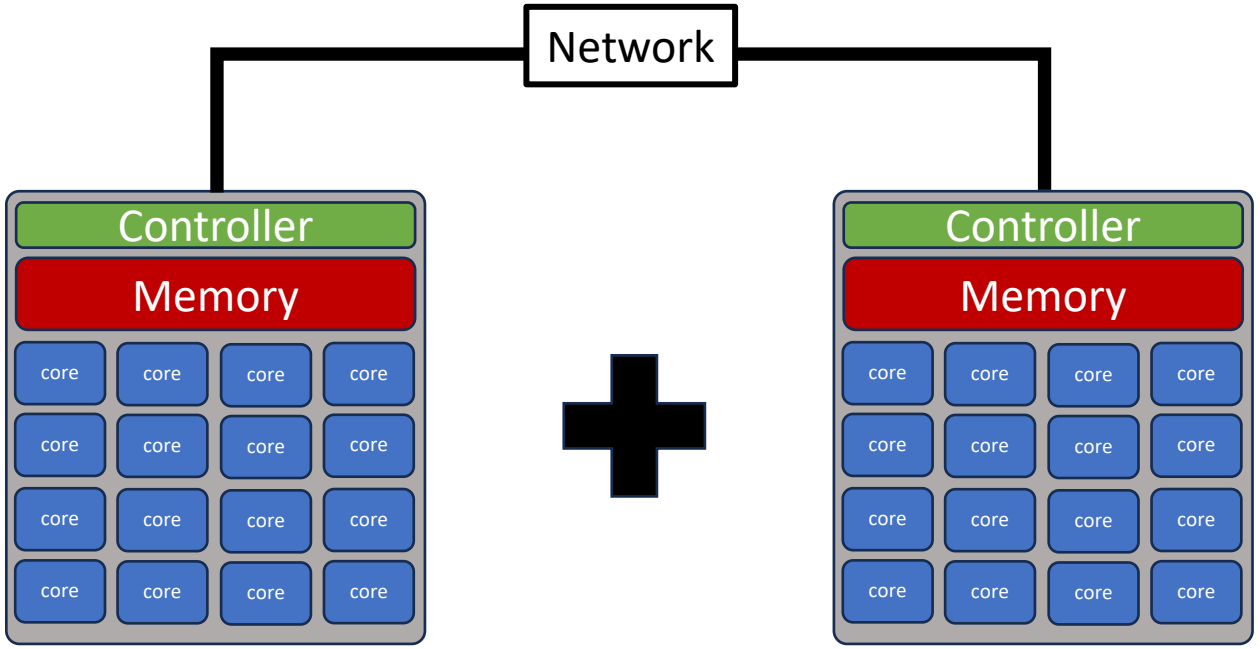
Lets Build a Compute Cluster

This is called “node”



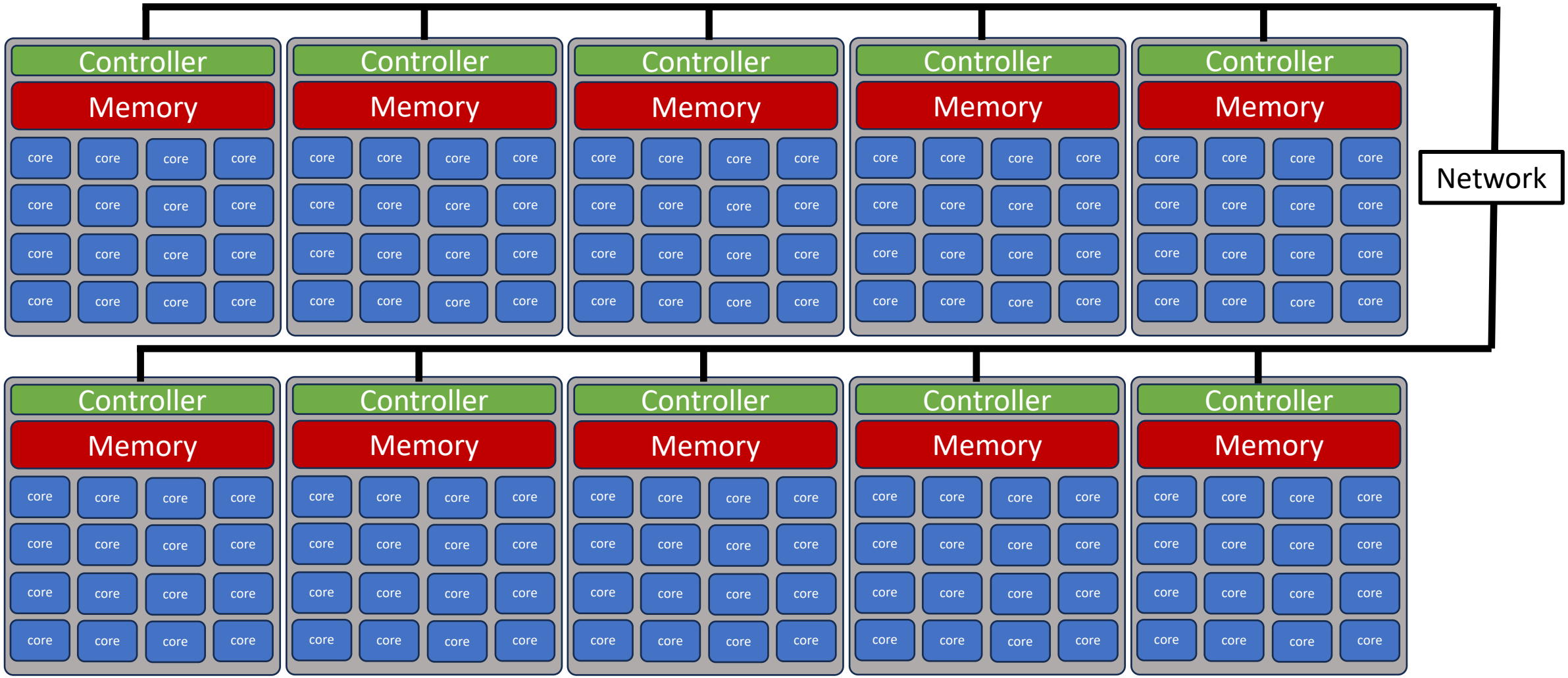
Lets take our theoretical 16 core CPU and connect it to more of the same CPUs

Lets Build a Compute Cluster



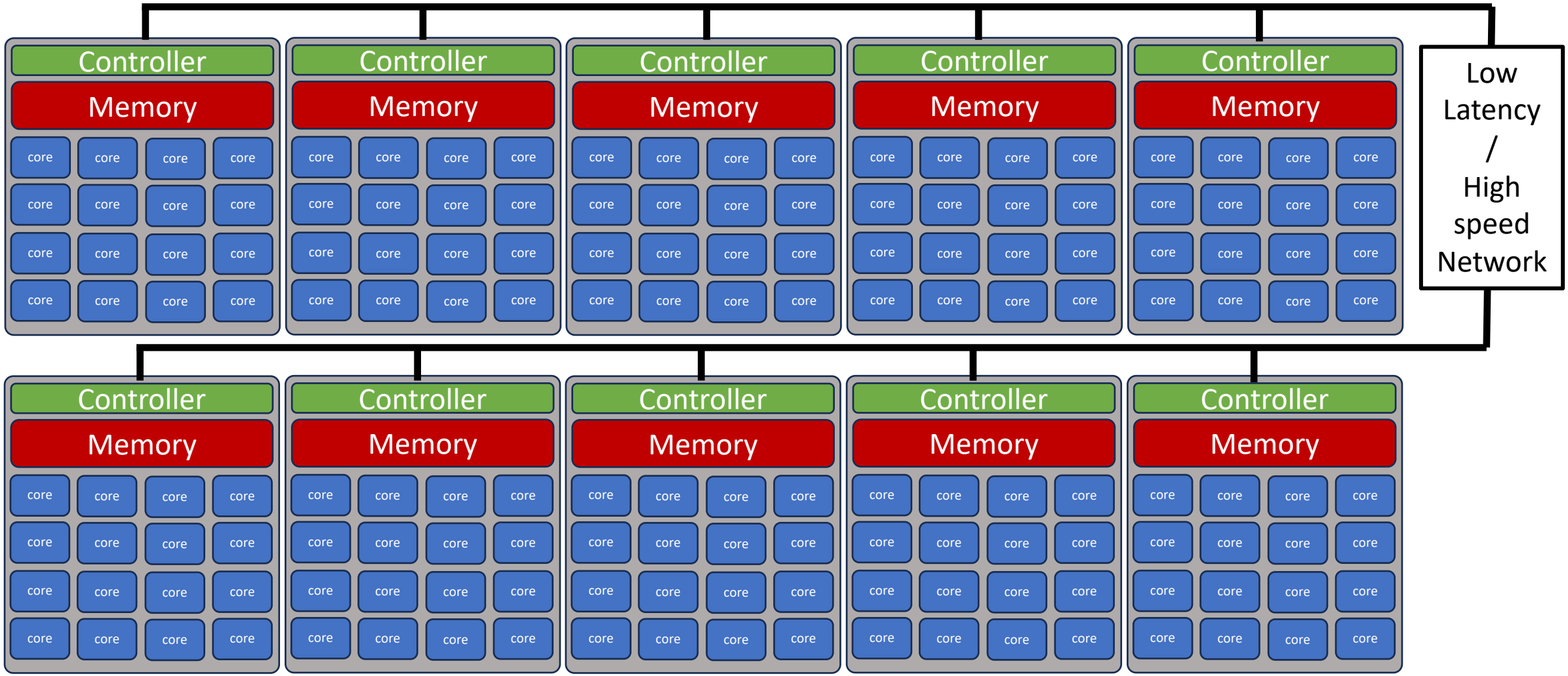
- We now have a 32 core machine!!
- Ok Ben's PowerPoint skills are limited …..

Lets Build a Compute Cluster



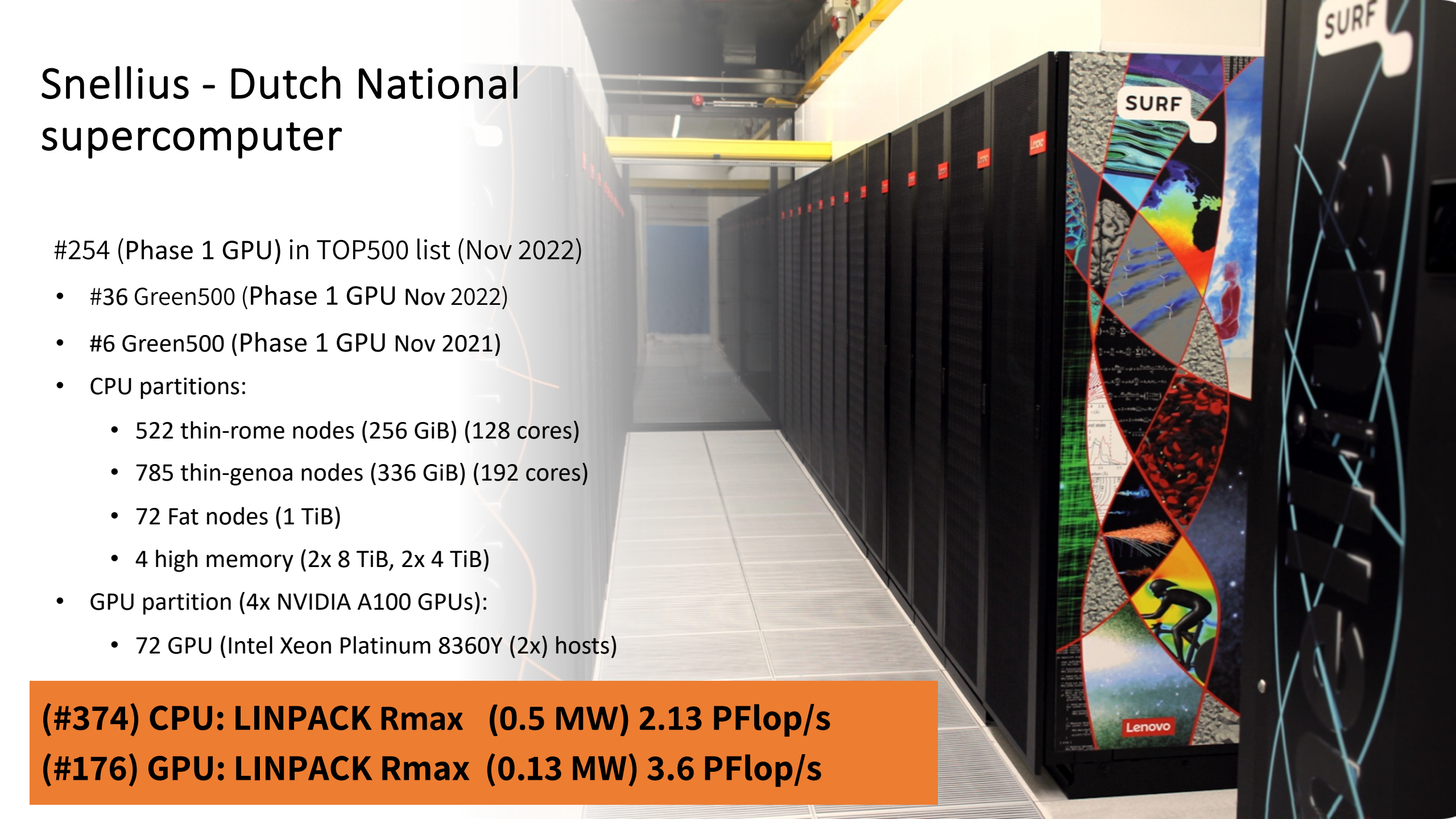
- We now have a 144 core machine!!
- Ok Ben's PowerPoint skills are limited …..

Lets Build a Supercomputer!!!!



- We now have a 144 core SUPERCOMPUTER!!
- Ok Ben's PowerPoint skills are limited …..

Snellius - Dutch National supercomputer



#254 (Phase 1 GPU) in TOP500 list (Nov 2022)

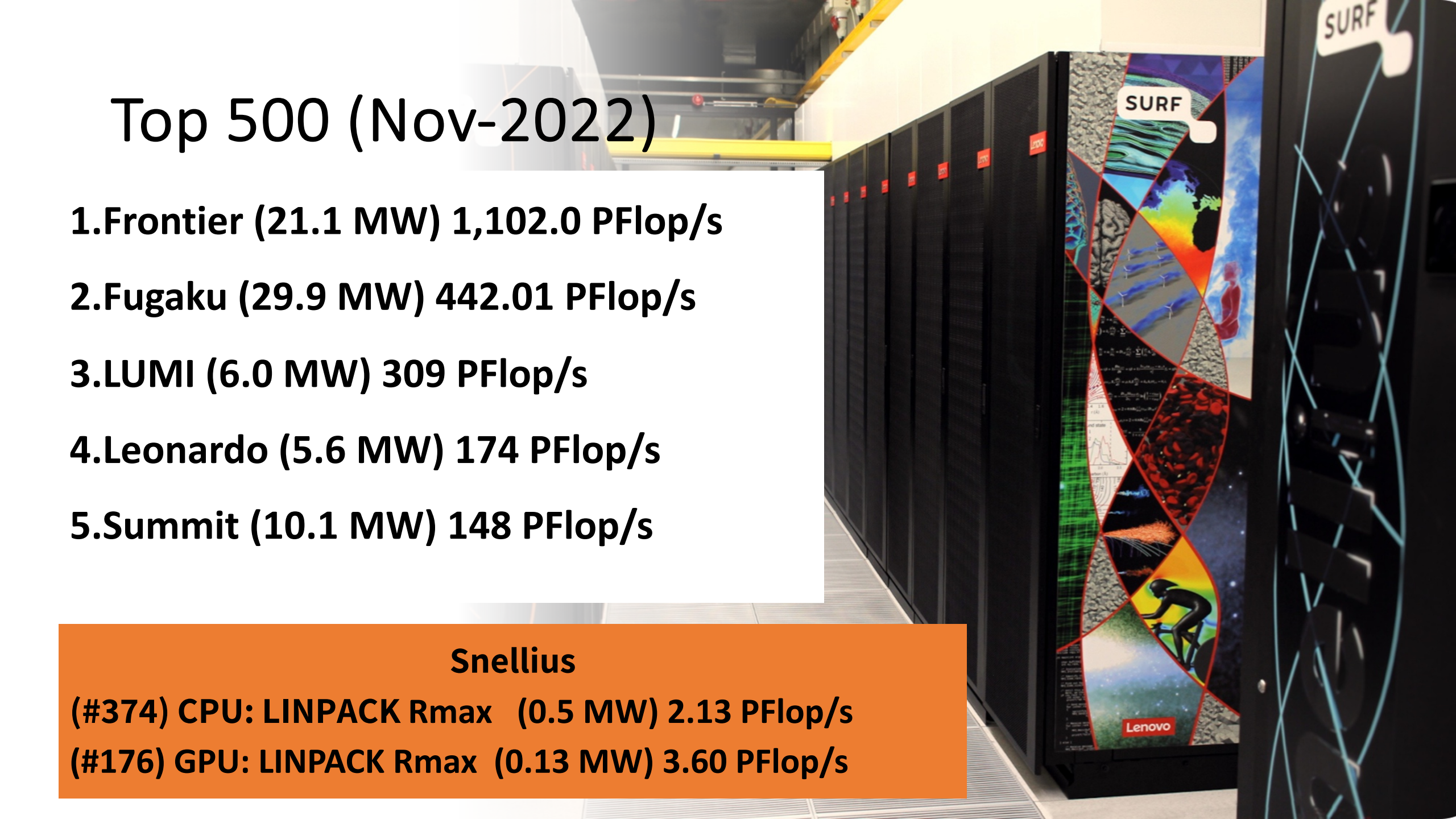
- #36 Green500 (Phase 1 GPU Nov 2022)
- #6 Green500 (Phase 1 GPU Nov 2021)
- CPU partitions:
  - 522 thin-rome nodes (256 GiB) (128 cores)
  - 785 thin-genoa nodes (336 GiB) (192 cores)
  - 72 Fat nodes (1 TiB)
  - 4 high memory (2x 8 TiB, 2x 4 TiB)
- GPU partition (4x NVIDIA A100 GPUs):
  - 72 GPU (Intel Xeon Platinum 8360Y (2x) hosts)

(#374) CPU: LINPACK Rmax (0.5 MW) 2.13 PFlop/s

(#176) GPU: LINPACK Rmax (0.13 MW) 3.6 PFlop/s

Top 500 (Nov-2022)

1. Frontier (21.1 MW) 1,102.0 PFlop/s
2. Fugaku (29.9 MW) 442.01 PFlop/s
3. LUMI (6.0 MW) 309 PFlop/s
4. Leonardo (5.6 MW) 174 PFlop/s
5. Summit (10.1 MW) 148 PFlop/s



Snellius

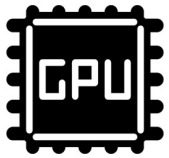
(#374) CPU: LINPACK Rmax (0.5 MW) 2.13 PFlop/s

(#176) GPU: LINPACK Rmax (0.13 MW) 3.60 PFlop/s

Working with a Supercomputer

User Experience

- Multiuser system
- Unix OS
- Optimized software



Compute power

- Many CPUs system
- Specialized Hardware
- Low-latency/High bandwidth Connections

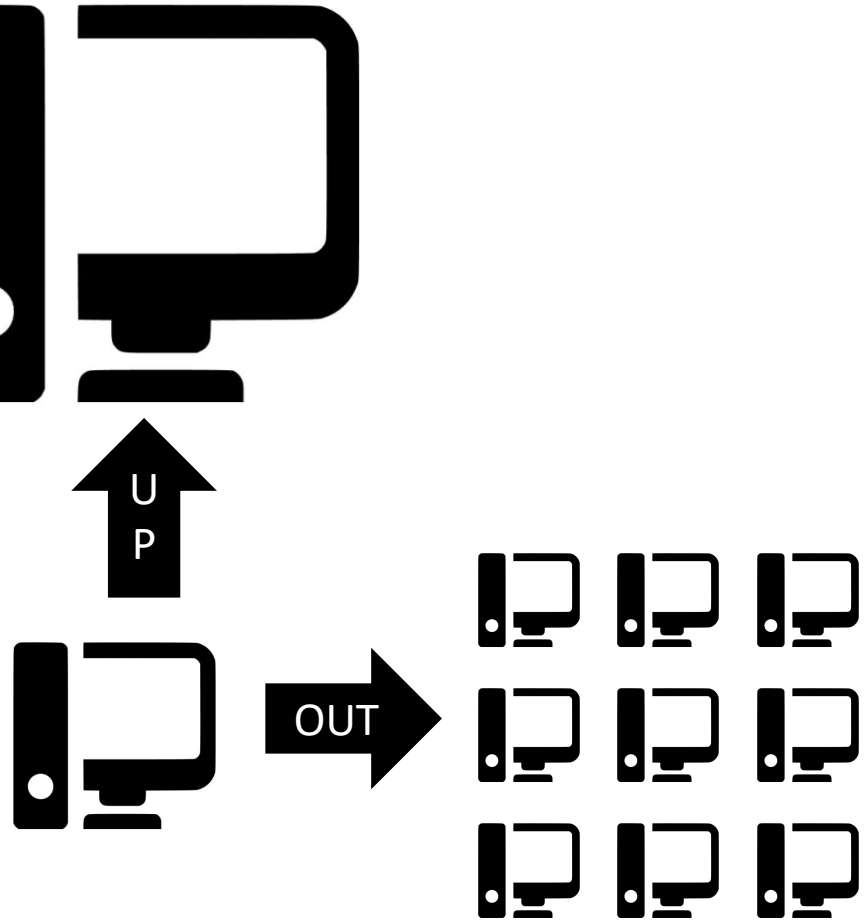
Storage

- Efficient I/O
- Large Memories

Working with a Supercomputer

Why, or more, when you need a Supercomputer?

- **Scale up**
  - Faster CPUs
  - Large memories
  - Specialized Hardware/Software
- **Scale out**
  - Large parallel applications
  - Many small- to medium- size jobs

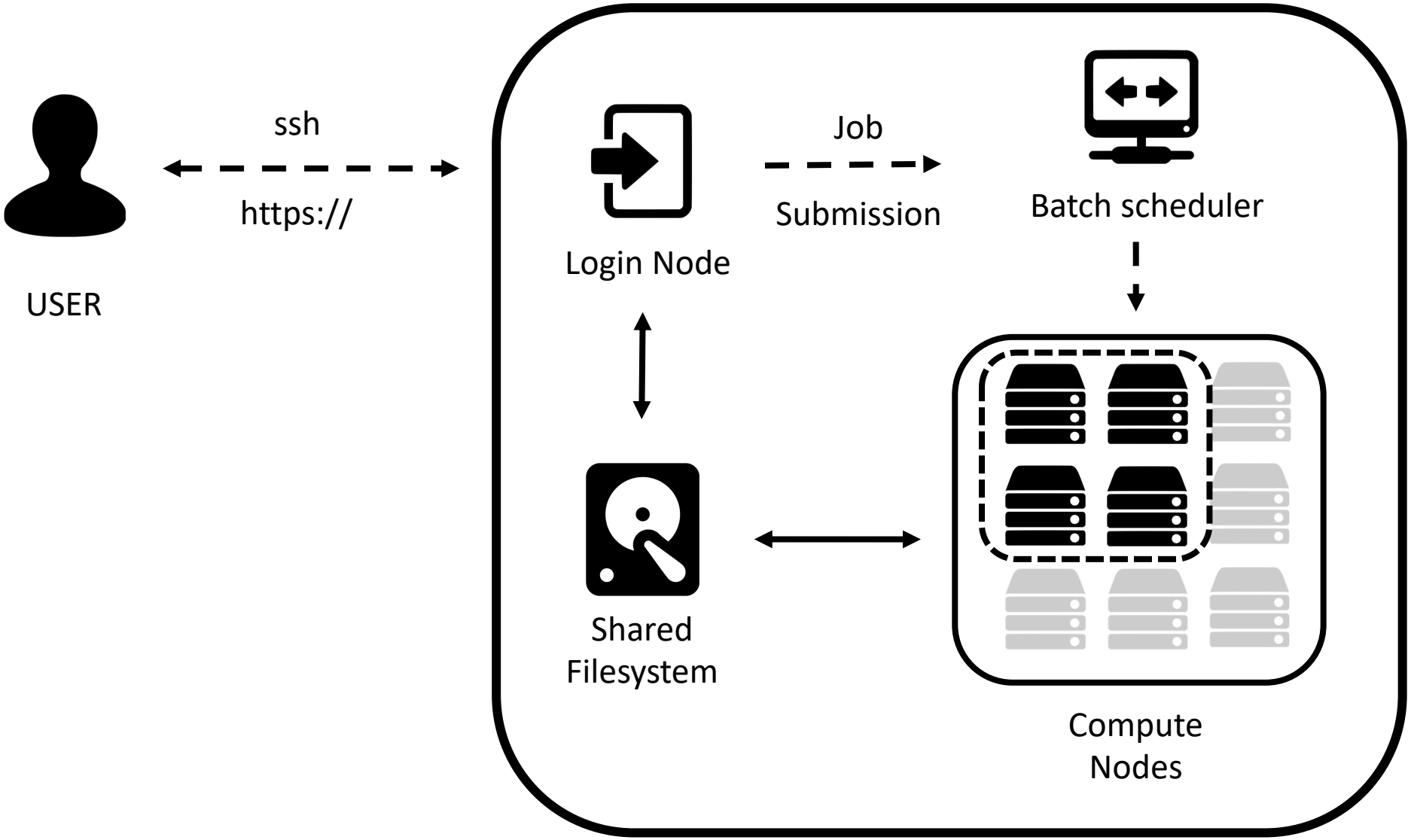


Working with a Supercomputer

Is NOT like this….

Working with a Supercomputer

Working with a Supercomputer



Introduction to …. Concepts of Parallel Programming

Introduction to parallel computing

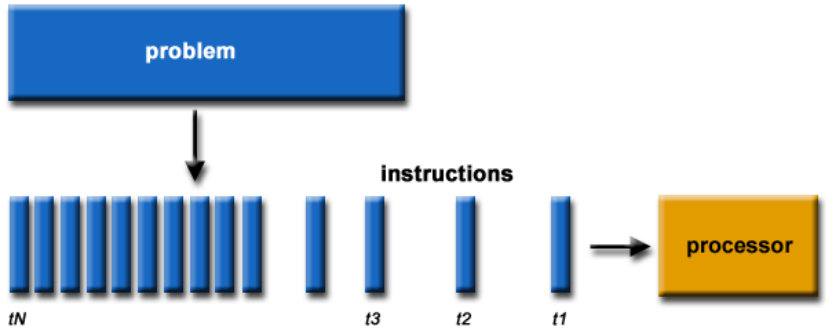
The goal is to understand:

- Merits and limits of parallel computing
- Parallel programming models (task / data parallelism)
- Differences between shared and distributed memory systems

Introduction to parallel computing

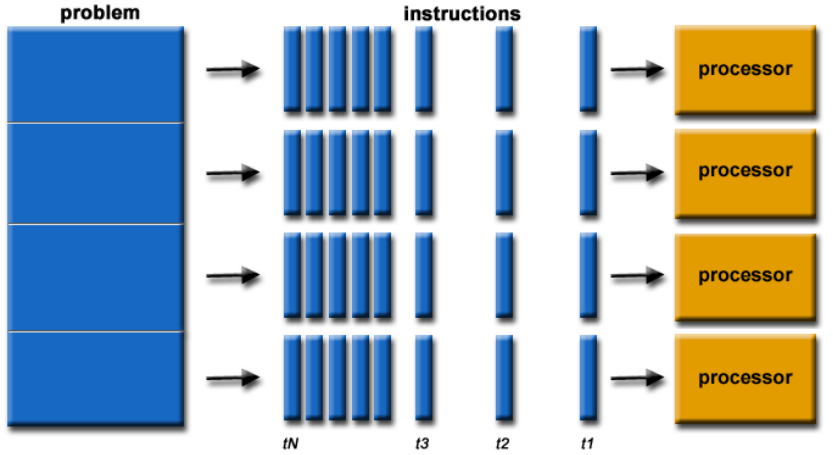
Serial computing

- A problem is broken into a discrete series of instructions, which are executed sequentially on a single processing unit (core).



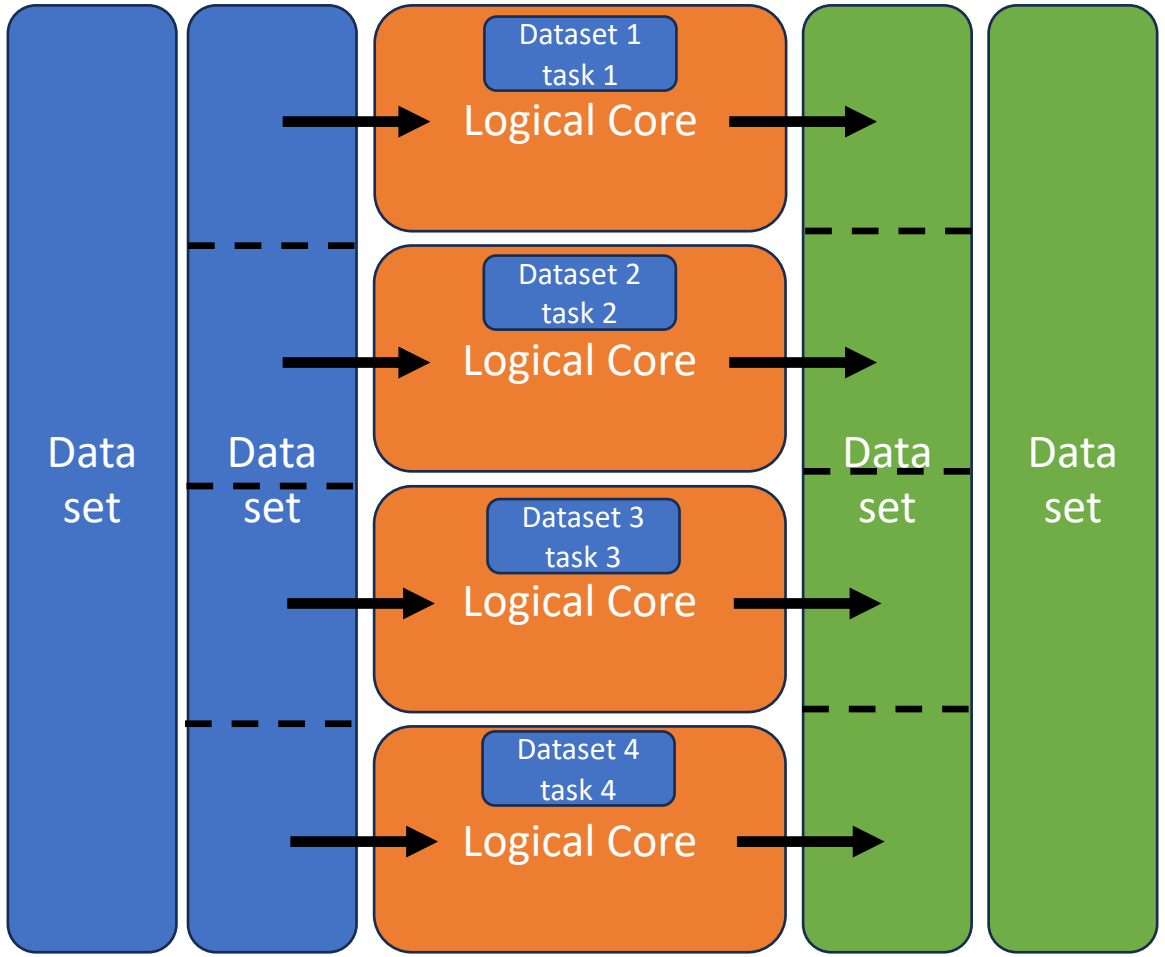
Parallel computing

- A problem is broken into discrete parts that can be solved using simultaneously multiple resources.

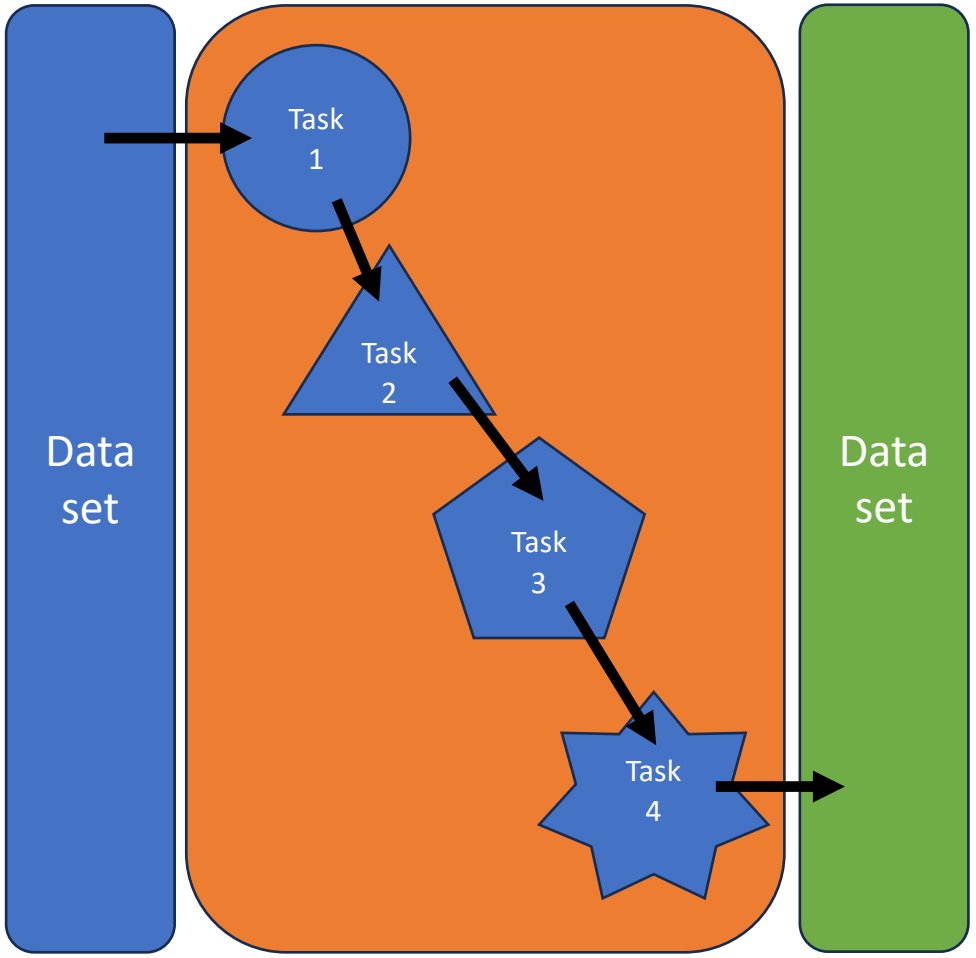


credits: https://computing.llnl.gov/tutorials/parallel_com

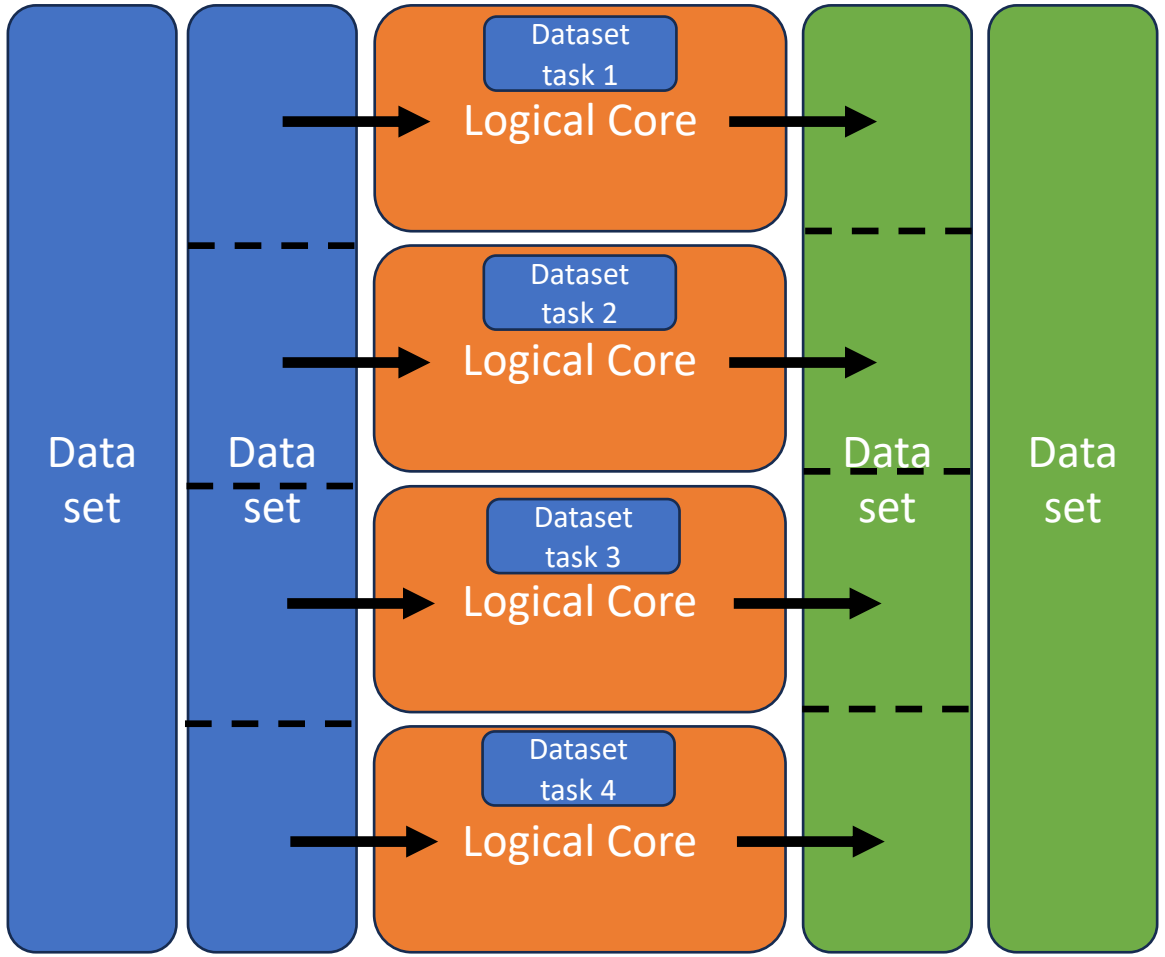
Data Parallel



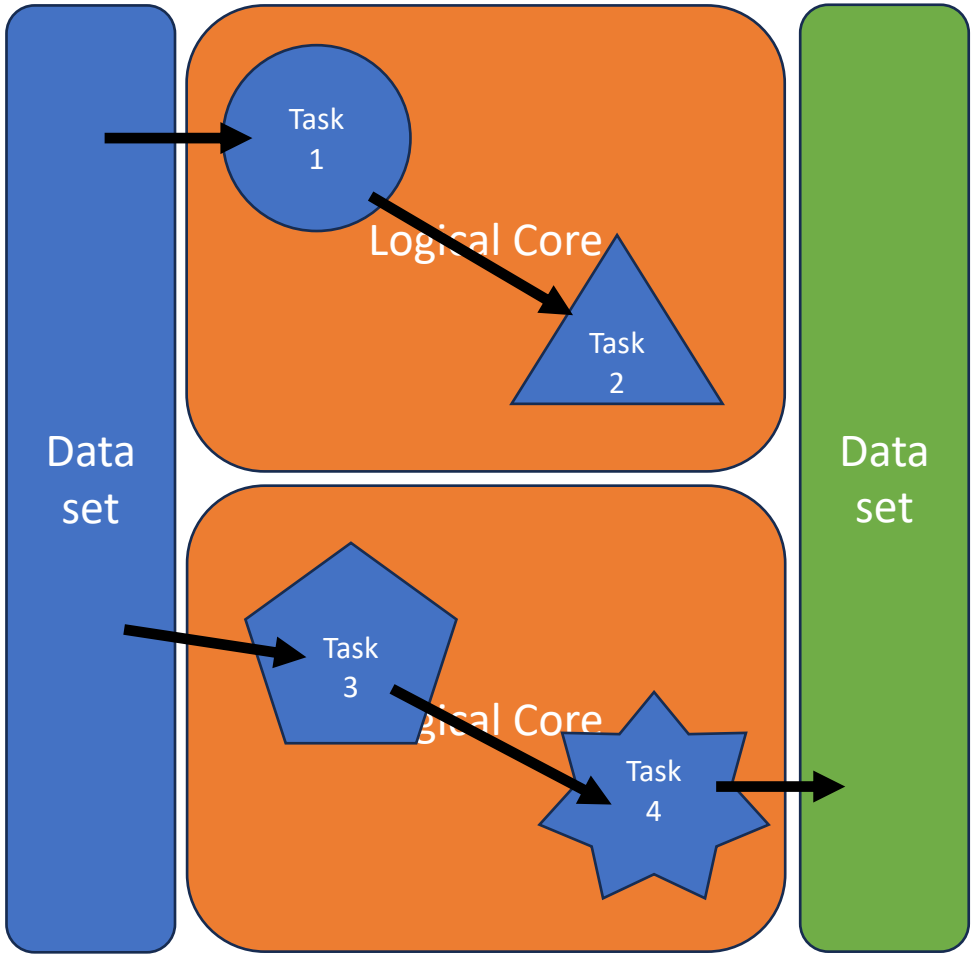
Task Parallel



Data Parallel



Task Parallel



Data Parallel vs Task Parallel

Data Parallel	Task Parallel
Same task are performed on different subsets of same data.	Different task are performed on the same or different data.
Synchronous computation is performed.	Asynchronous computation is performed.
As there is only one execution thread operating on all sets of data, so the speedup is more.	As each processor will execute a different thread or process on the same or different set of data, so speedup is less.
Amount of parallelization is proportional to the input size.	Amount of parallelization is proportional to the number of independent tasks is performed.
It is designed for optimum load balance on multiprocessor system.	Here, load balancing depends upon on the e availability of the hardware and scheduling algorithms like static and dynamic scheduling.

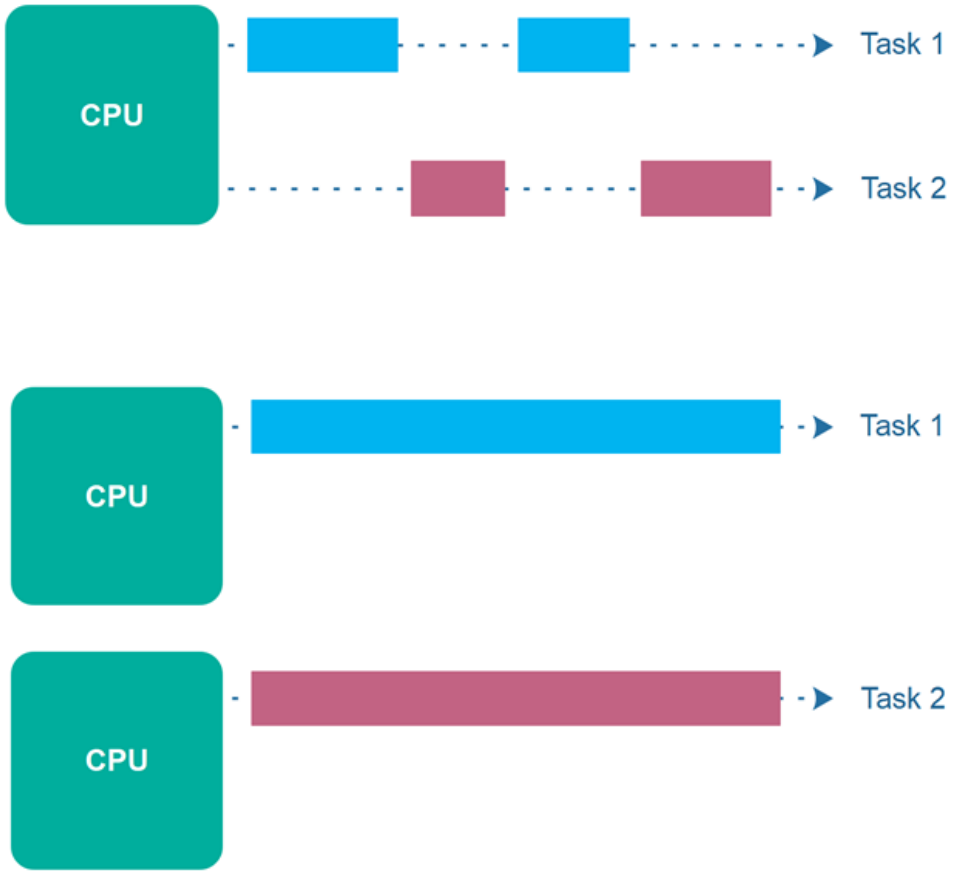
Data Parallel vs Task Parallel

Data Parallel	Task Parallel
Matrix-Vector Add Matrix Multiplication	Downloading multiple datasets from the internet (asynchronous I/O)

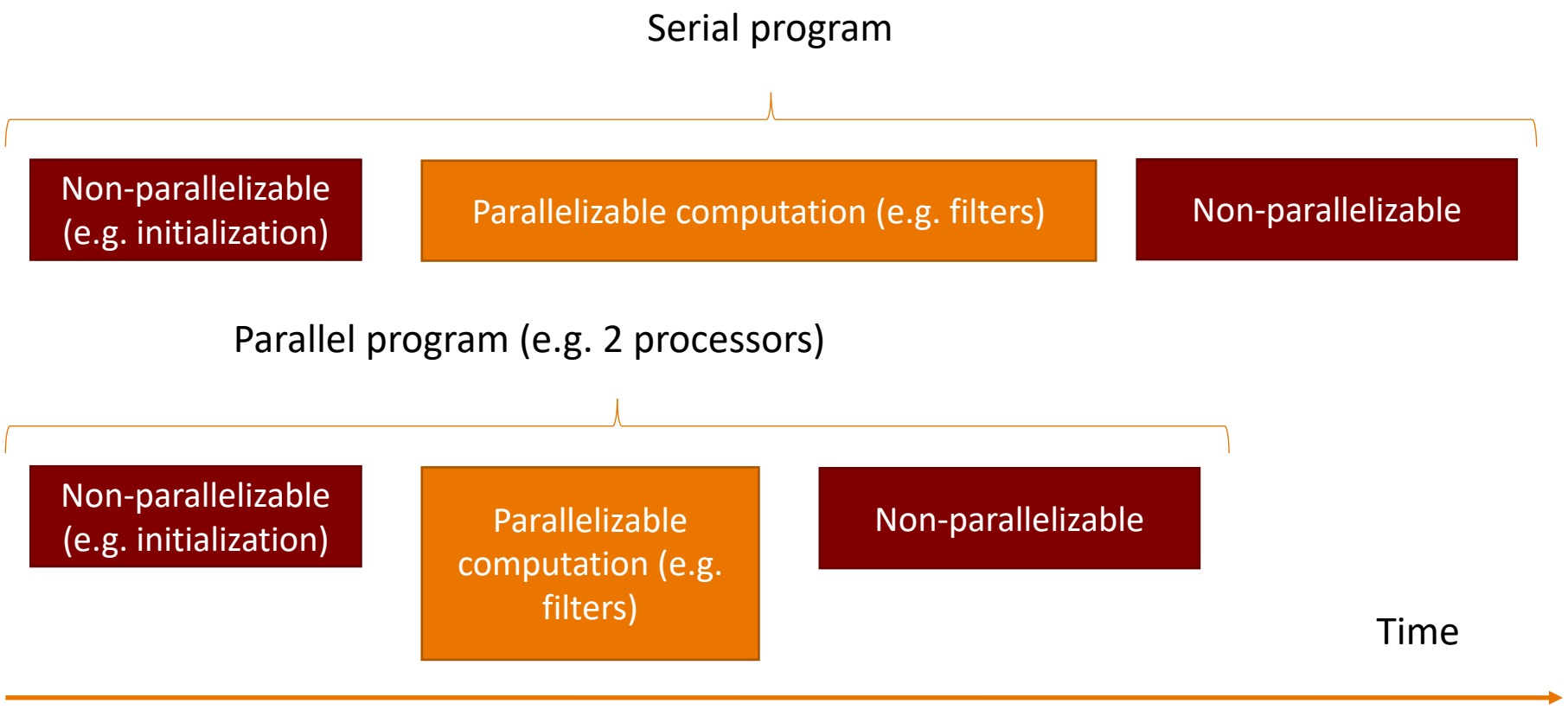
Concurrent vs. Parallel execution

Concurrency: is the execution of multiple tasks at the same time, regardless of the number of processors. This is the case, for example, when 2 tasks can start, run and complete in overlapping time. On task may not be finished before it begins the next. Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.

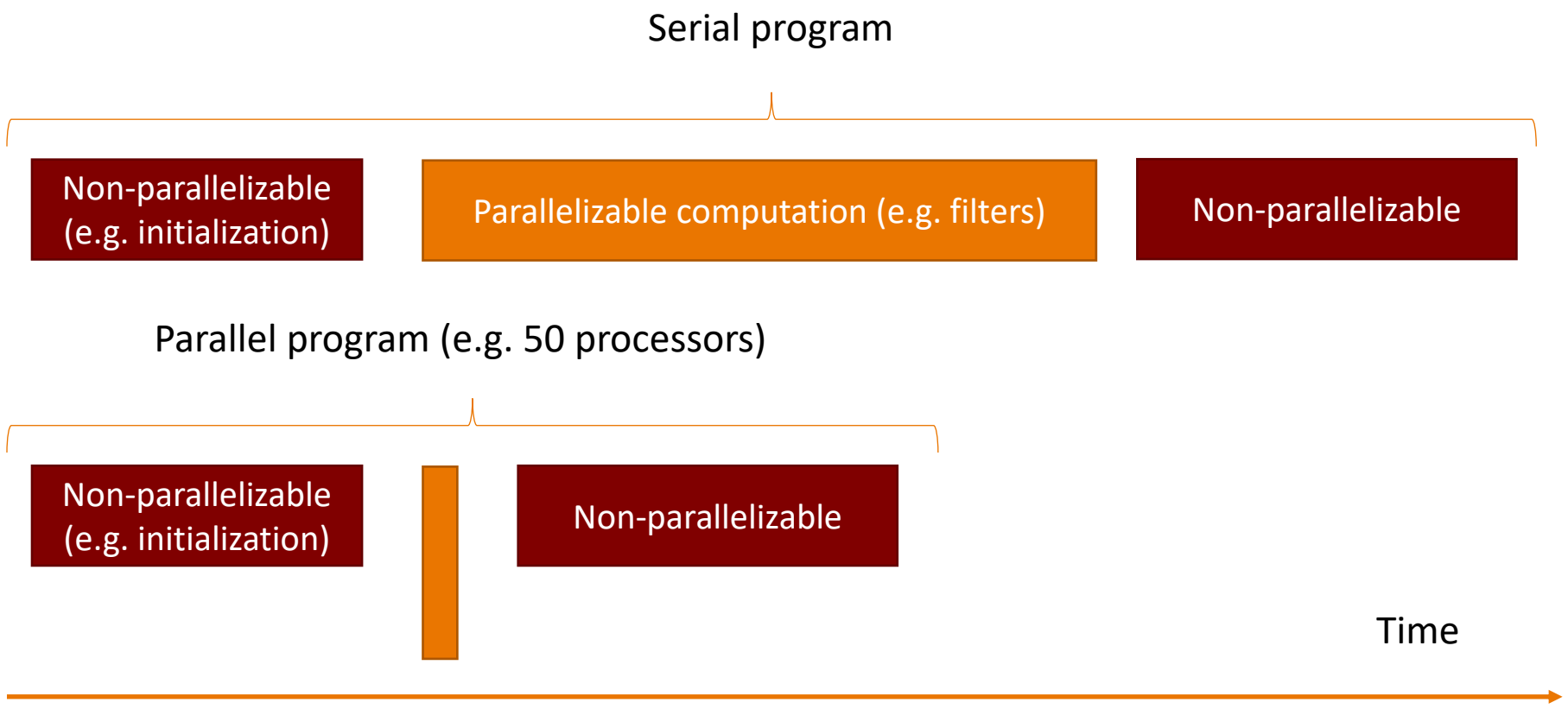
Parallelism: is the execution on multiple processors of two or more tasks that are running at the same time. Parallel programs use parallel hardware to speed-up computational time by splitting up a task into multiple, simple, and independent sub-task which can be performed simultaneously



Introduction to parallel computing



Introduction to parallel computing

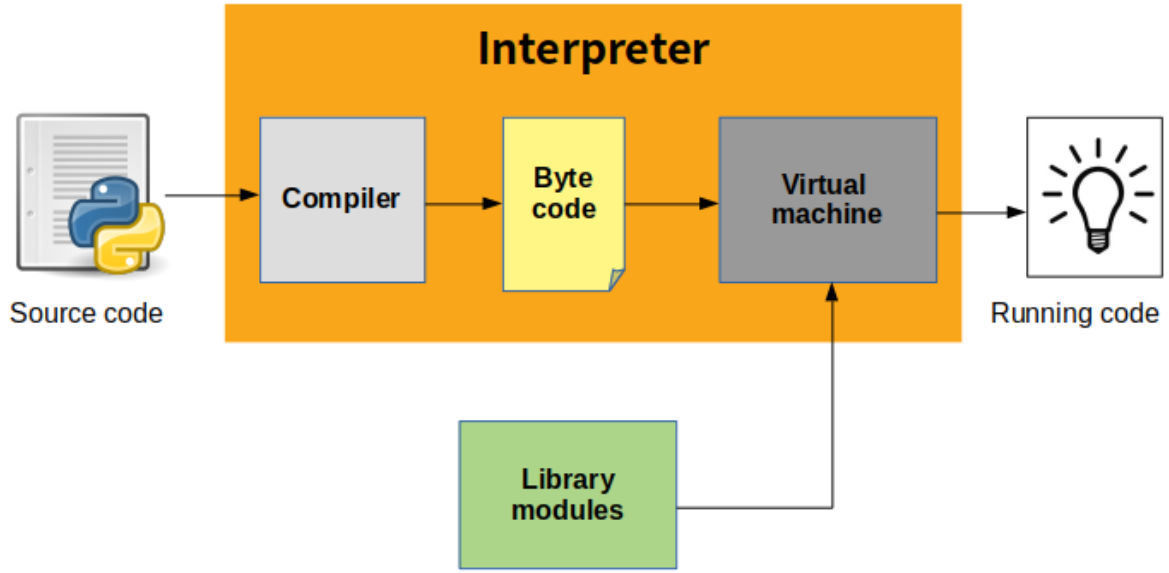


Introduction to the “what”…. Python

Parallel computing with Python

Programming with Python

https://www.python.org/

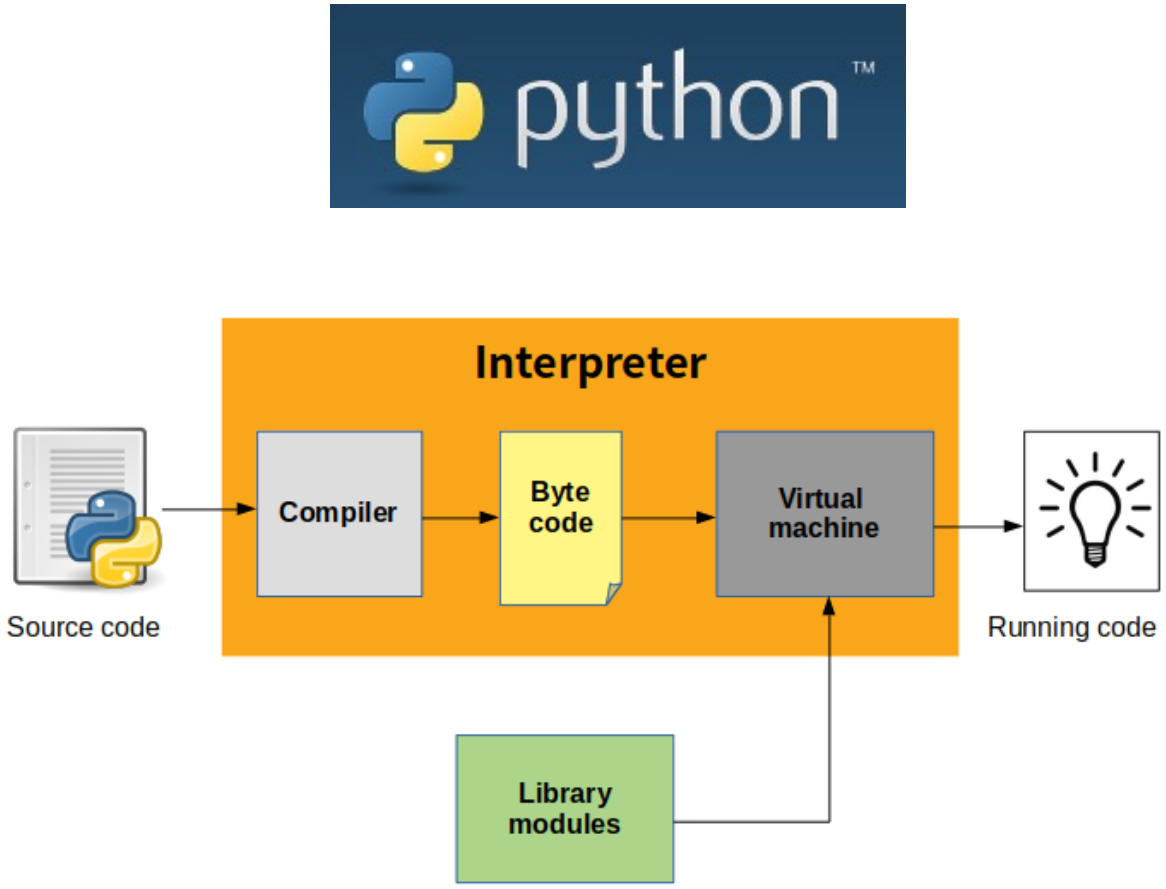


- Modern, interpreted programming language
- Object oriented
- Portable (Unix/Linux, Mac OS X, Windows)
- Open source
- Readable and simple syntax
- Many standard and third party libraries
- Can be used interactively

Parallel computing with Python

Programming with Python

- Byte code
  - Python source code is compiled into bytecode, the internal representation of a Python program in the interpreter.
  - The bytecode is also cached in .pyc and .pyo files so that executing the same file is faster the second time (recompilation from source to bytecode can be avoided).
  - This “intermediate language” is said to run on a virtual machine that executes the machine code corresponding to each bytecode.

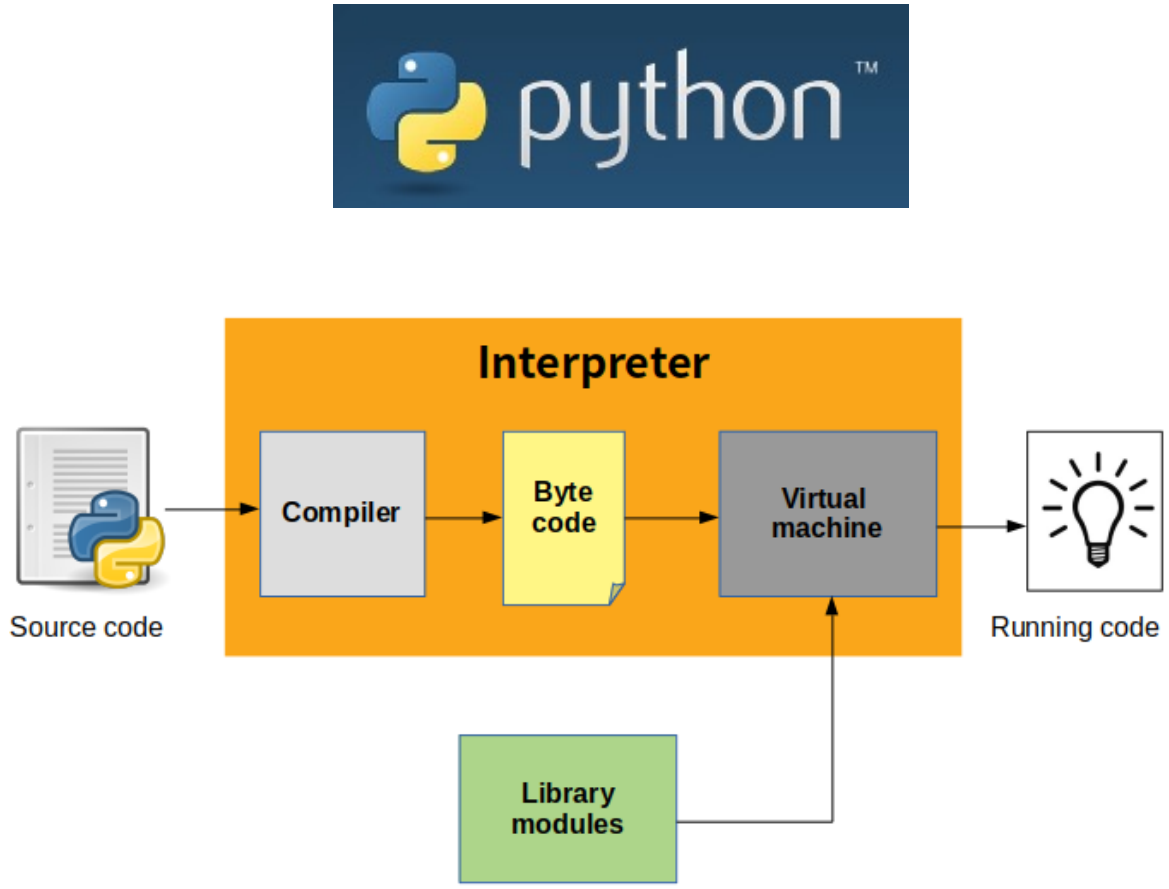


Parallel computing with Python

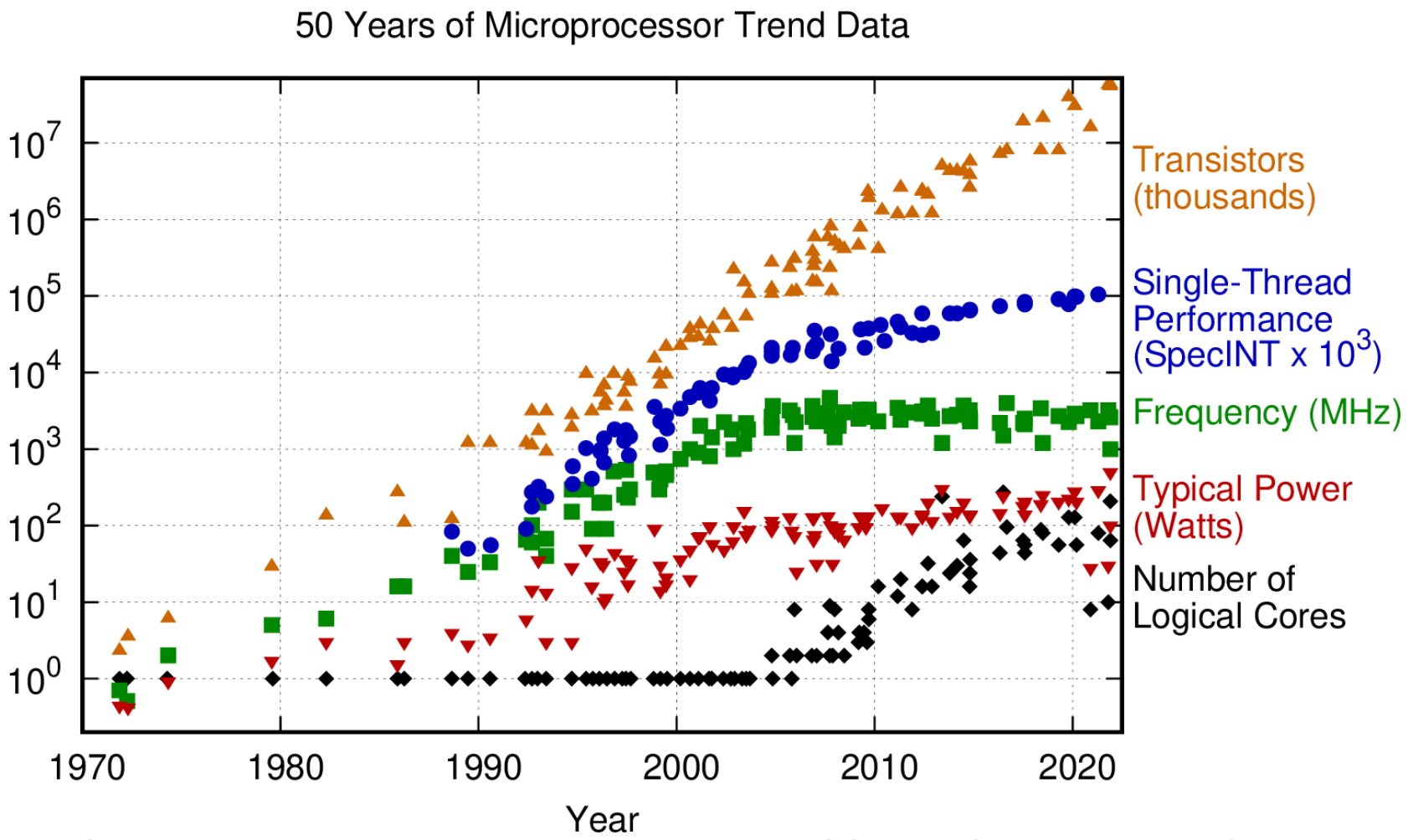
"Python was originally written without asynchronous or parallelism in mind"

- Guido van Rossum

(Lex Fridman Podcast 2022)



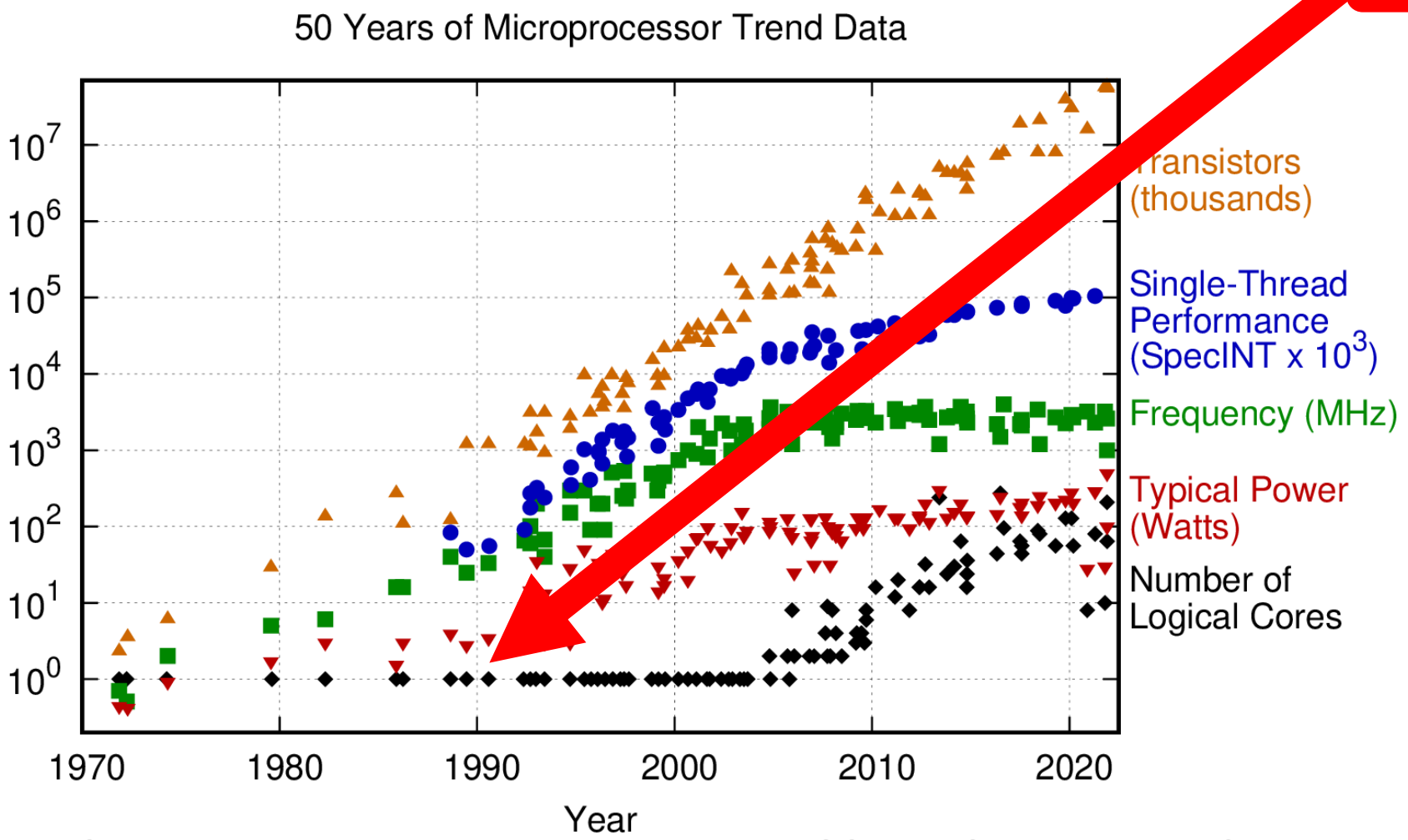
Threads/Parallelism/Many core CPUs



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

- **FORTRAN 90** (1991)
- **POSIX.1b** (1993)
  - Semaphore
  - Message passing
  - Shared memory
- **POSIX** (1995)
  - pthreads
- **OpenMP** (1997)

Threads/Parallelism/Many core CPUs



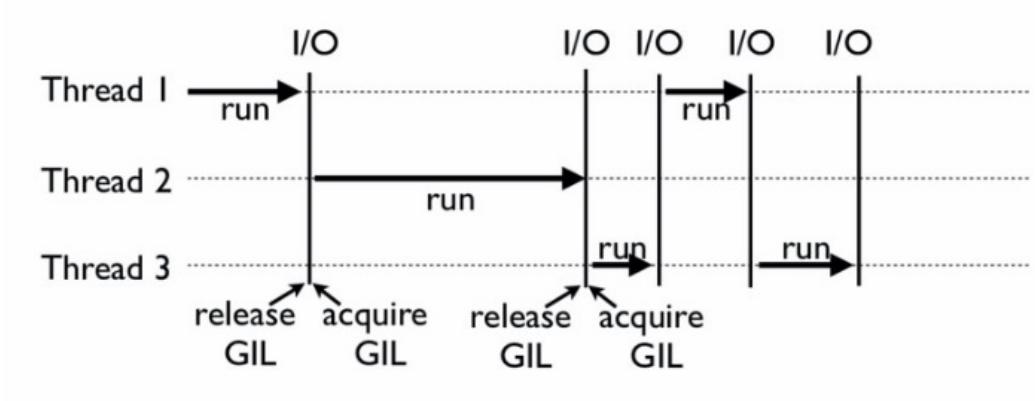
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

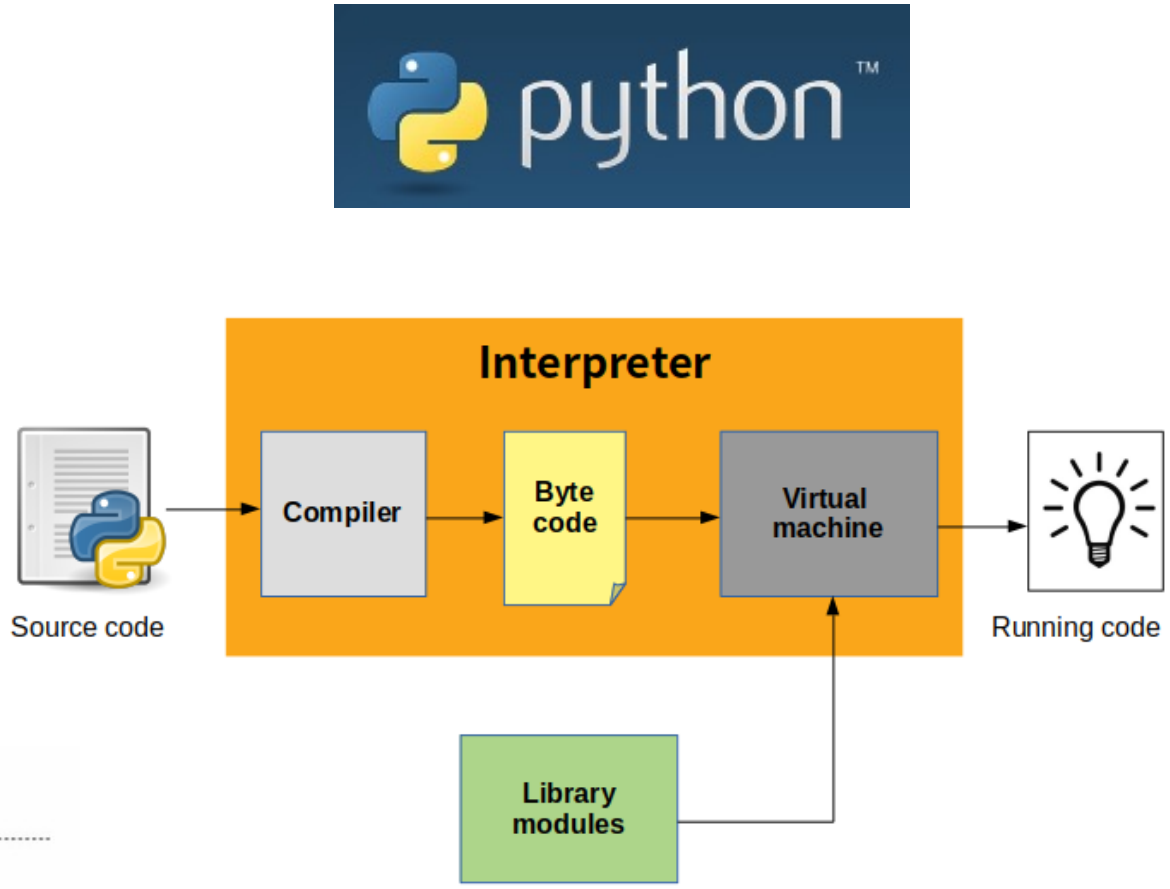
- **Python** (1991)
- **FORTRAN 90** (1991)
- **POSIX.1b** (1993)
  - Semaphore
  - Message passing
  - Shared memory
- **POSIX** (1995)
  - pthreads
- **OpenMP** (1997)

Parallel computing with Python

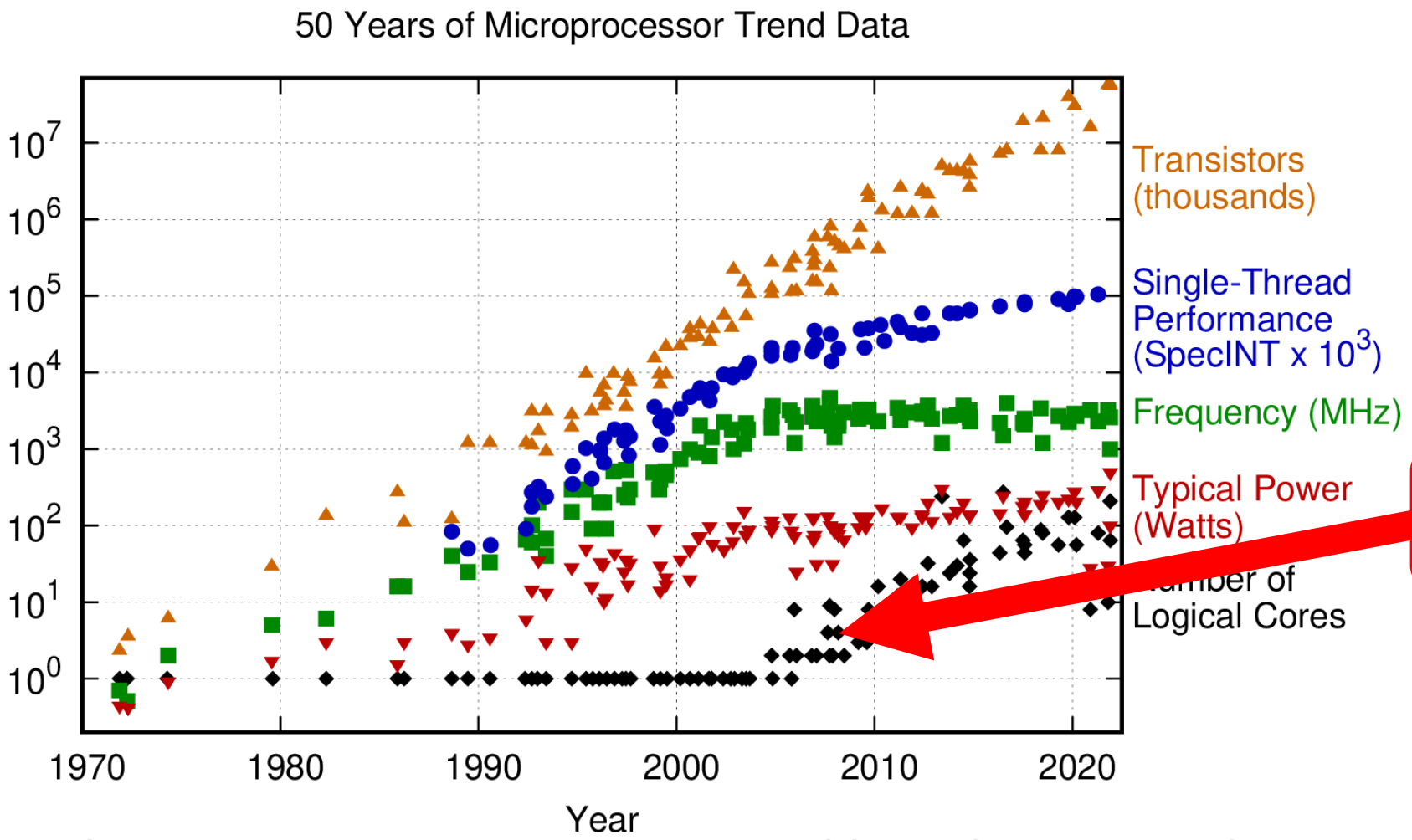
Global Interpreter lock

- Python's solution to make the interpreter thread safe. So only one OS thread can operate on the interpreter at a time.
  - **(mid 90's) multi-core CPUs were non-existent to rare**





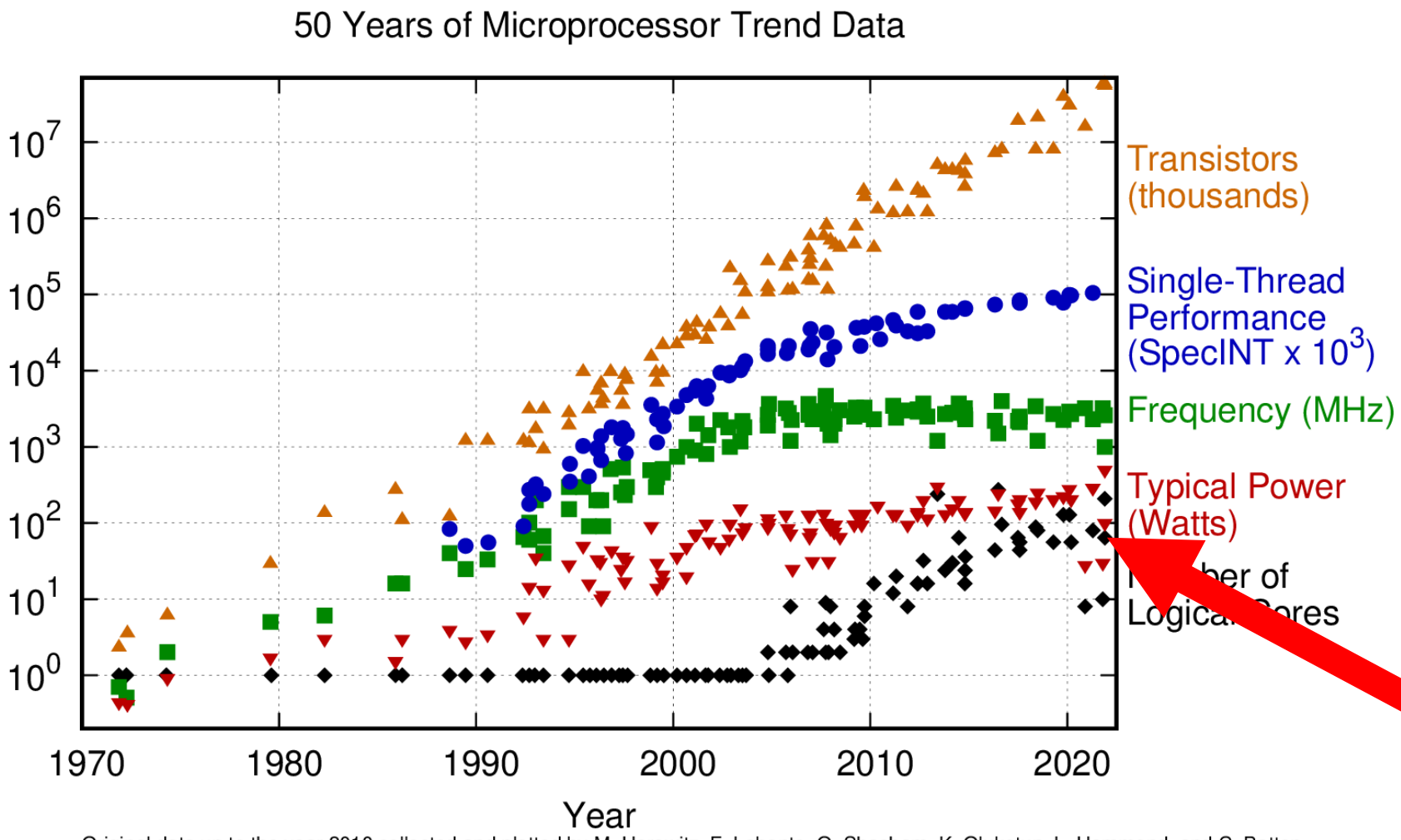
Threads/Parallelism/Many core CPUs



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

- **Python** (1991)
- **FORTRAN 90** (1991)
- **POSIX.1b** (1993)
  - Semaphore
  - Message passing
  - Shared memory
- **MPI** (1994 or 92)
- **OpenMP** (1997)
- **Multiprocessing Python** (2008)

Threads/Parallelism/Many core CPUs



- **Python** (1991)
- **FORTRAN 90** (1991)
- **POSIX.1b** (1993)
  - Semaphore
  - Message passing
  - Shared memory
- **MPI** (1994 or 92)
- **OpenMP** (1997)
- **Multiprocessing Python** (2008)
- **PEP 703** (2023)
  - Making the Global Interpreter Lock Optional in CPython

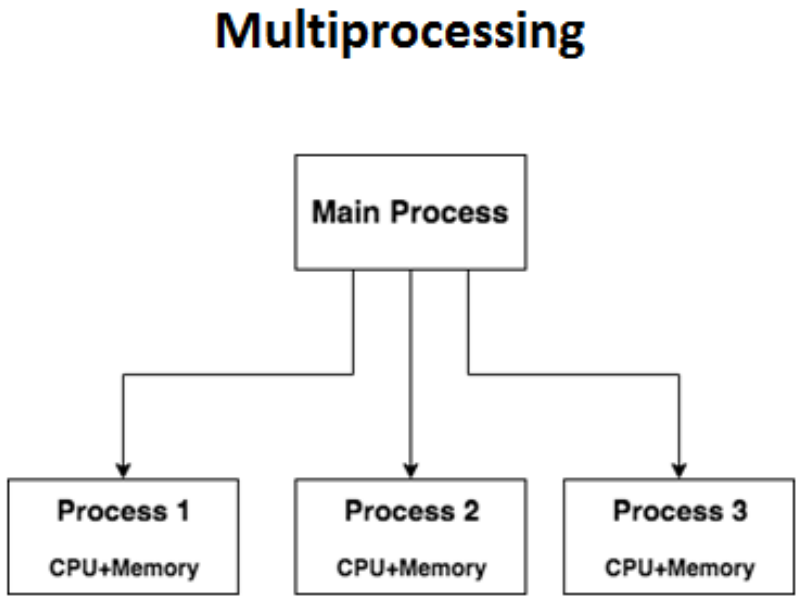
Parallel computing with Python

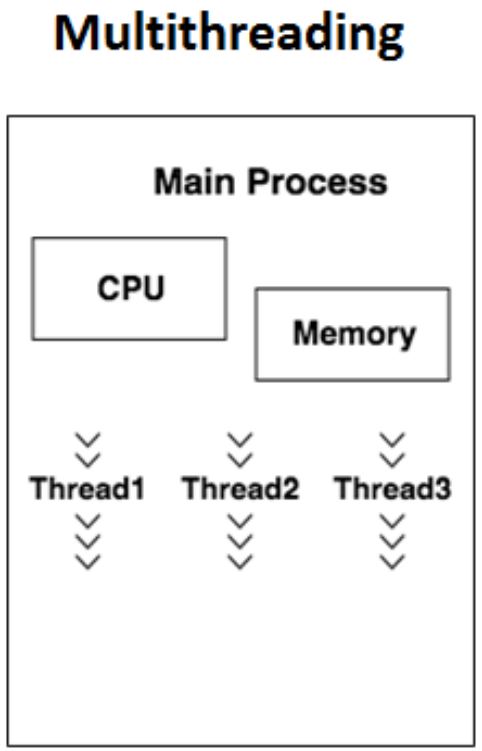
Python is a powerful scientific programming language

- Python is considered an easy to use, but “slow” programming language but there are several external libraries (python packages) and tools we can use to improve performances:
  - **Numpy and Scipy packages**
    - Mathematical and scientific libraries collections with optimized algorithms
  - **Subprocess and Multiprocessing packages**
    - Spawn and control processes and threads
  - **mpi4py, pycuda**
    - Interfaces to external MPI and CUDA API
  - **Numba**
    - Translates Python functions to optimized machine code at runtime

Parallel programming for CPU architectures in Python

Processes vs. Threads





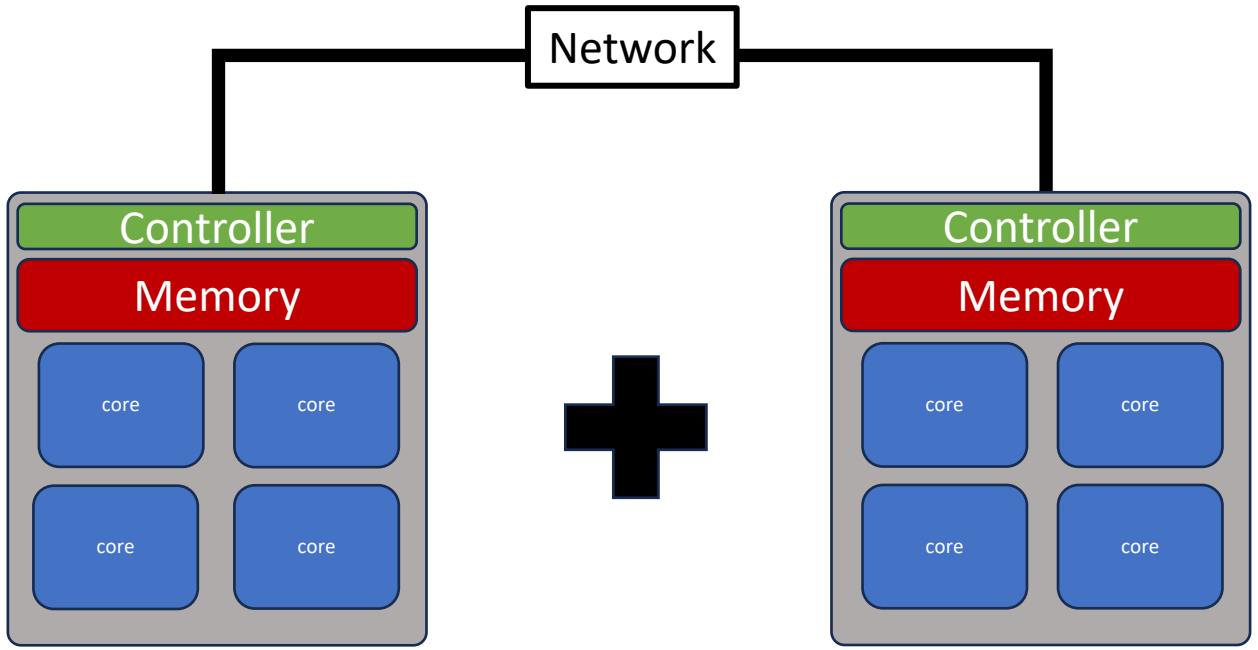
Parallel programming for CPU architectures in Python

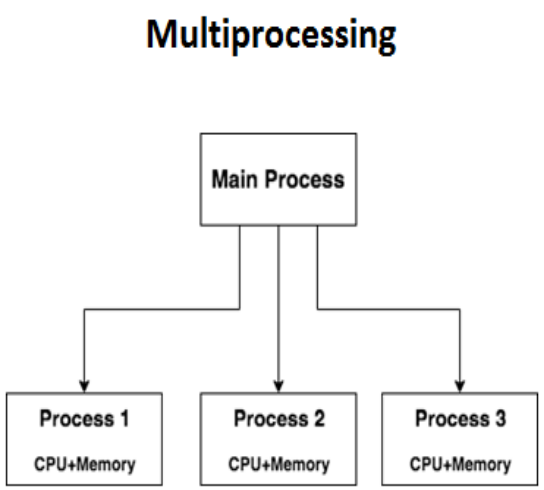
Processes vs. Threads

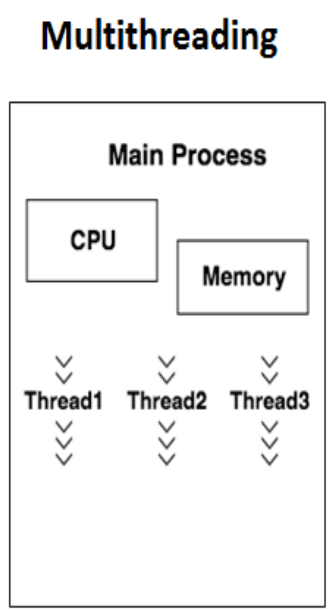
Processes are instances of a program (e.g: python interpreter, browser, etc.). They are independent execution units that have their own memory access space (view of the memory). Each process cannot access directly the shared data in other processes and need explicit communication protocols to exchange data between processes. Also, switching from one process to another requires some time (relatively) for saving and loading registers, memory maps, and other resources.

Threads are threads are separate points of execution within a single program, and can be executed either synchronously or asynchronously. A single process can contain multiple threads. Each thread share the same memory space, i.e. the memory space of the parent process. This would mean the code to be executed as well as all the variables declared in the program would be shared by all threads.

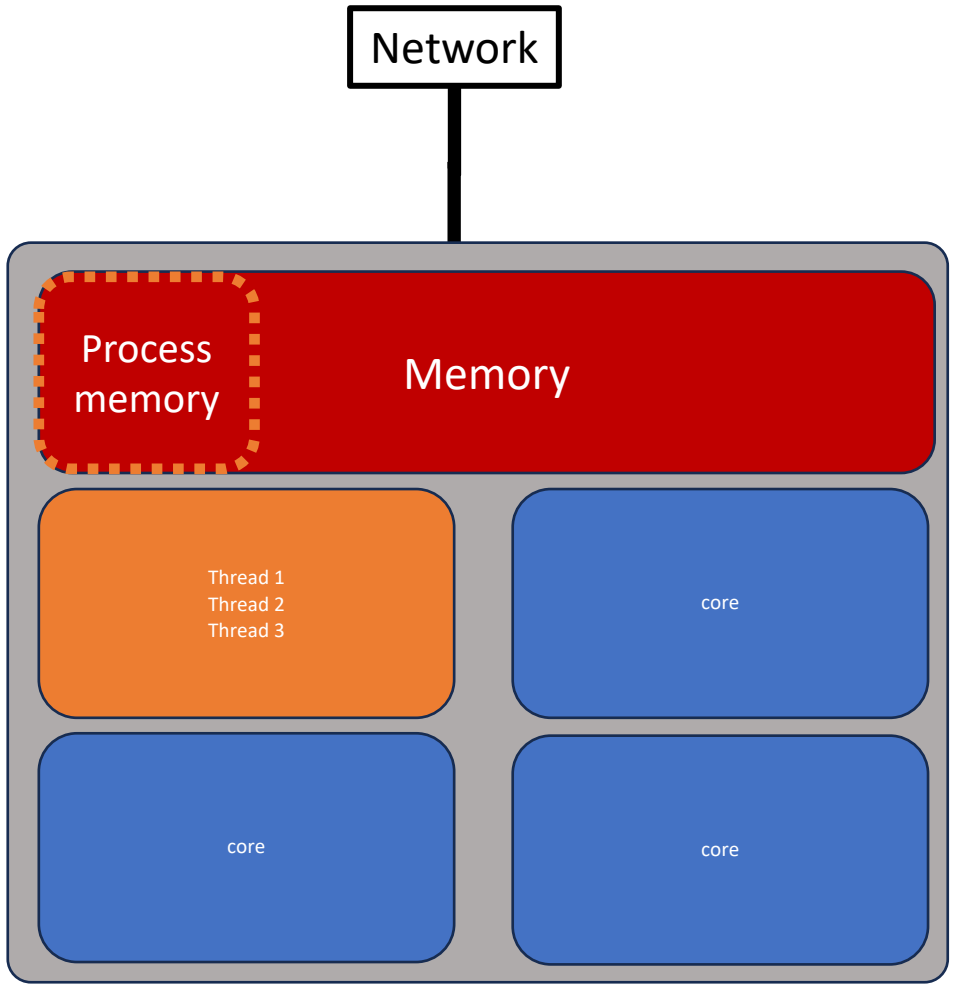
Parallel computing in Python in context of our cluster

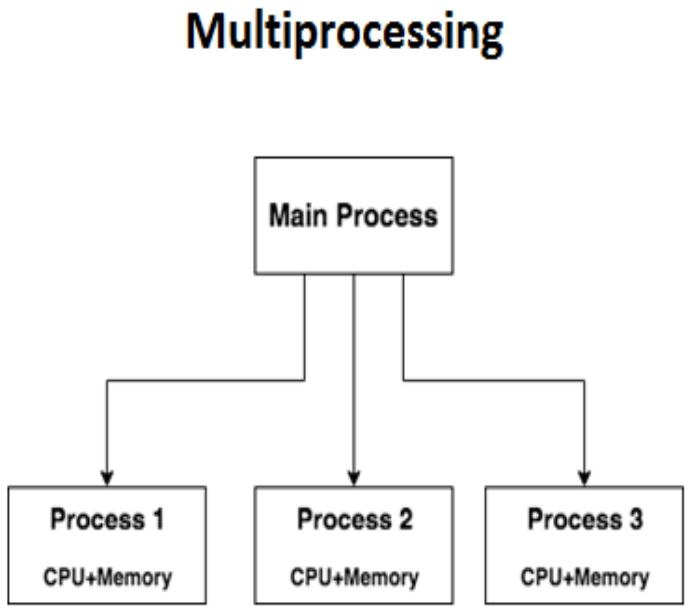


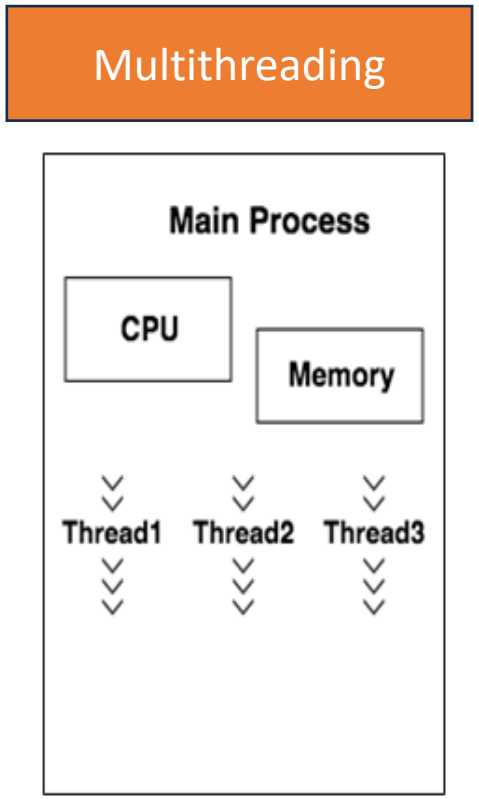




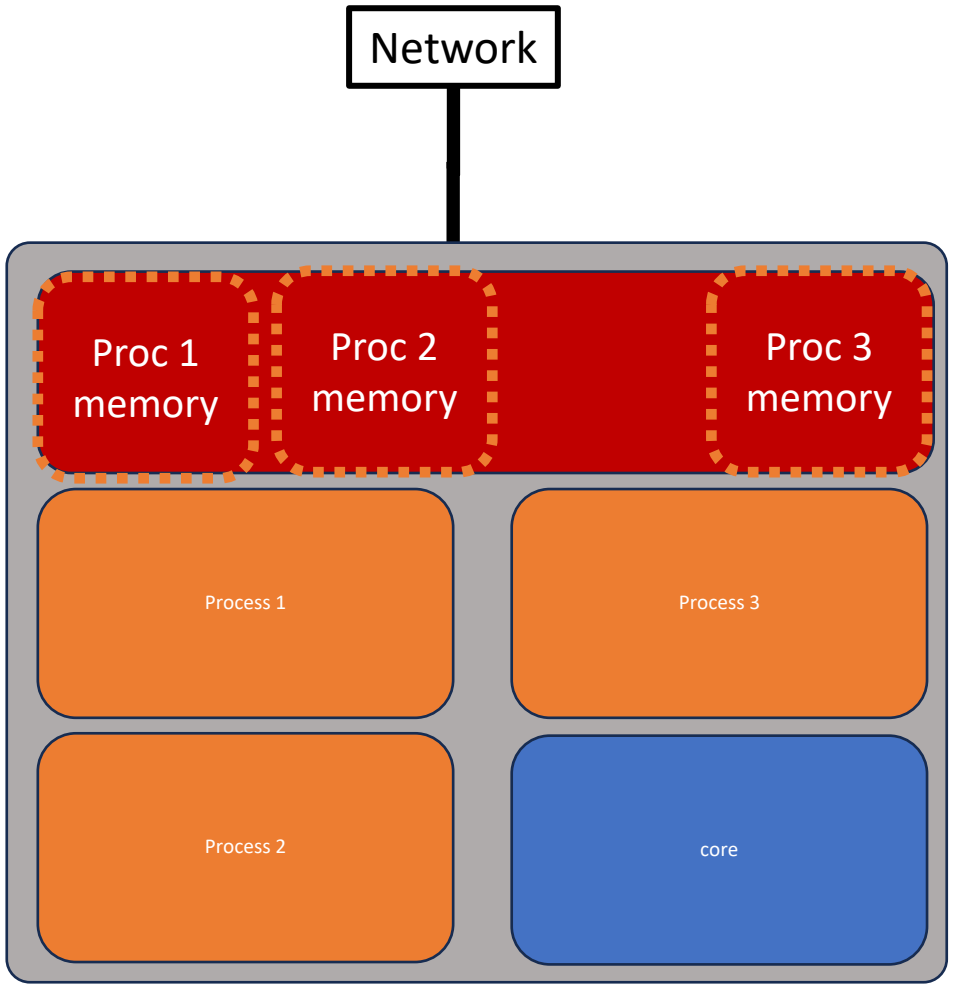
Parallel computing in Python in context of our cluster

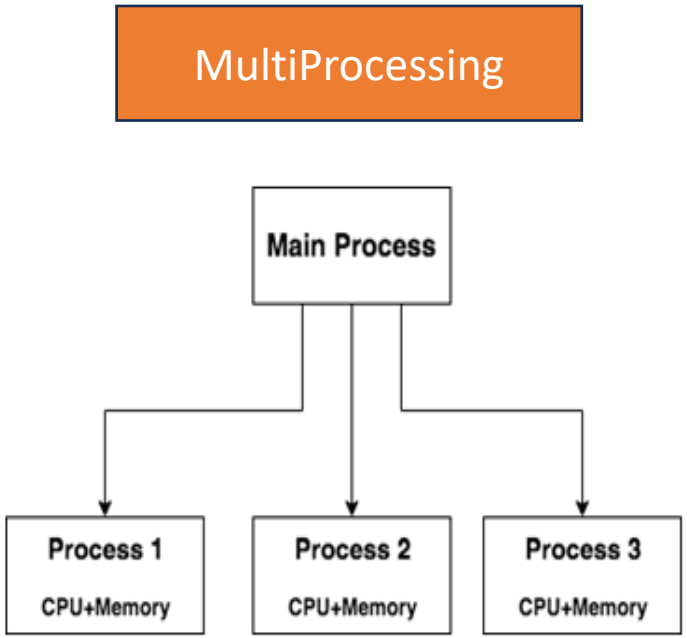


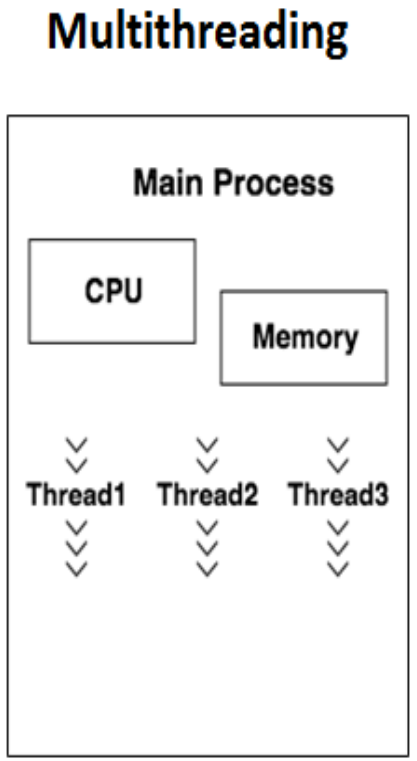




Parallel computing in Python in context of our cluster



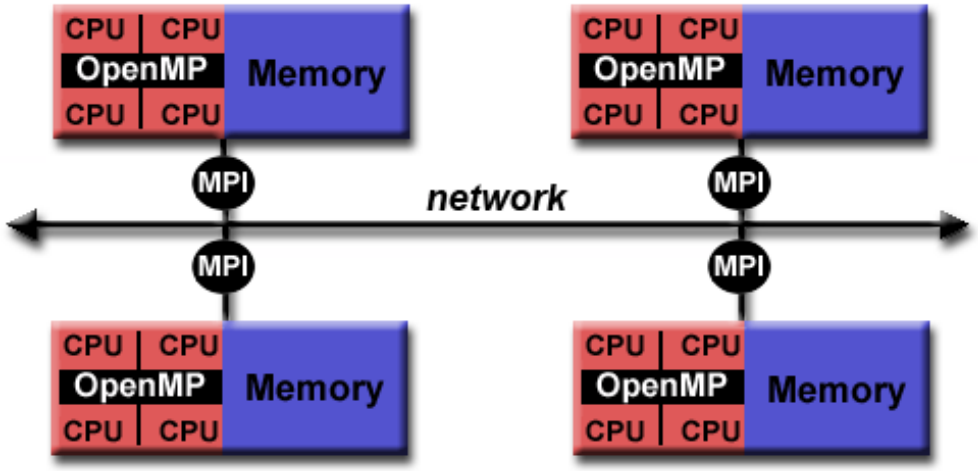




Introduction to parallel computing

Parallel computing implementations

- Hybrid model (e.g. Supercomputers)



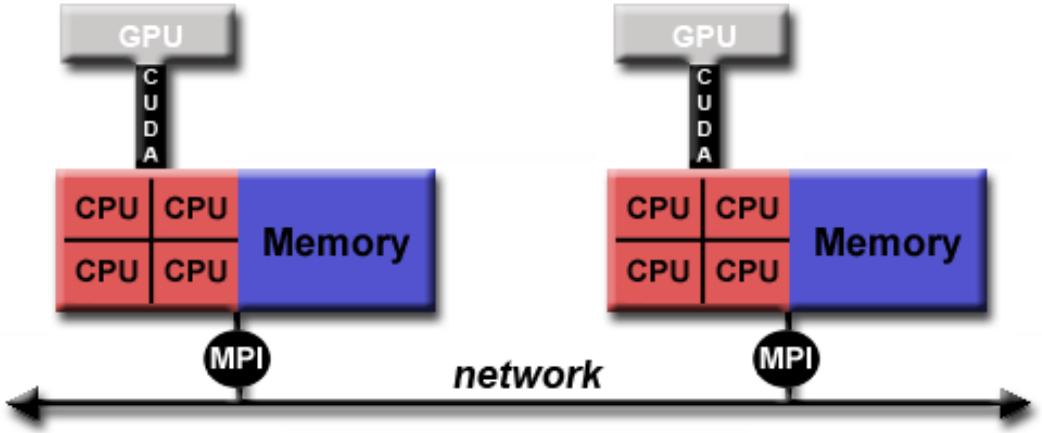
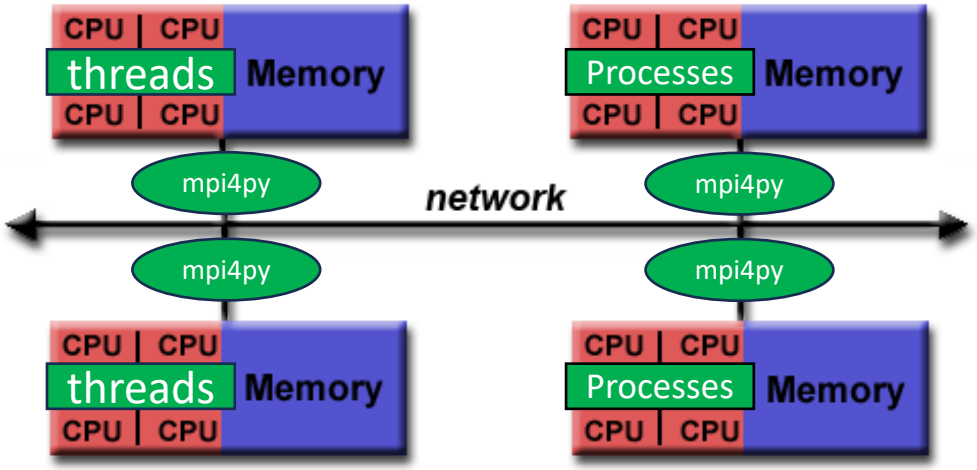


Image source: computing.llnl.gov

Introduction to parallel computing

Python parallel computing

- Hybrid model (e.g. Supercomputers)



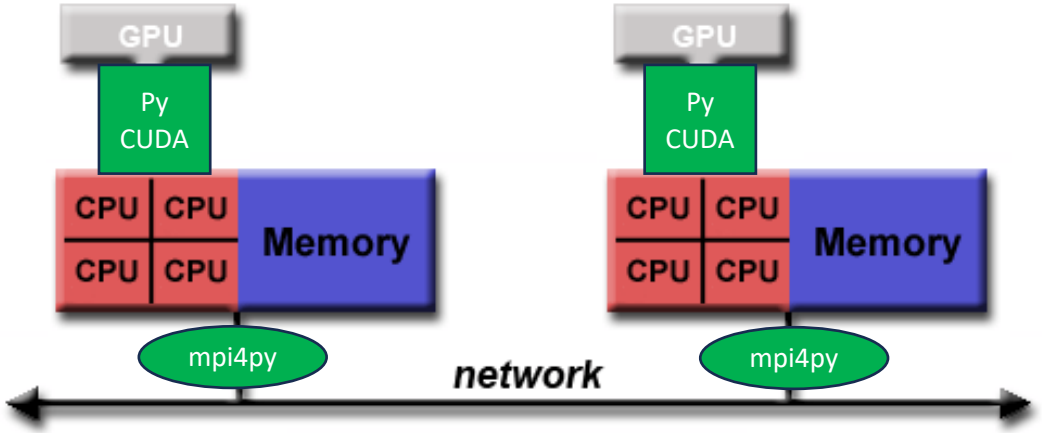


Image source: computing.llnl.gov