Costa Rica HPC School 2022

# Distributed Memory Systems

HPC SCHOOL

COSTA RICA

Gabriel P. Silva

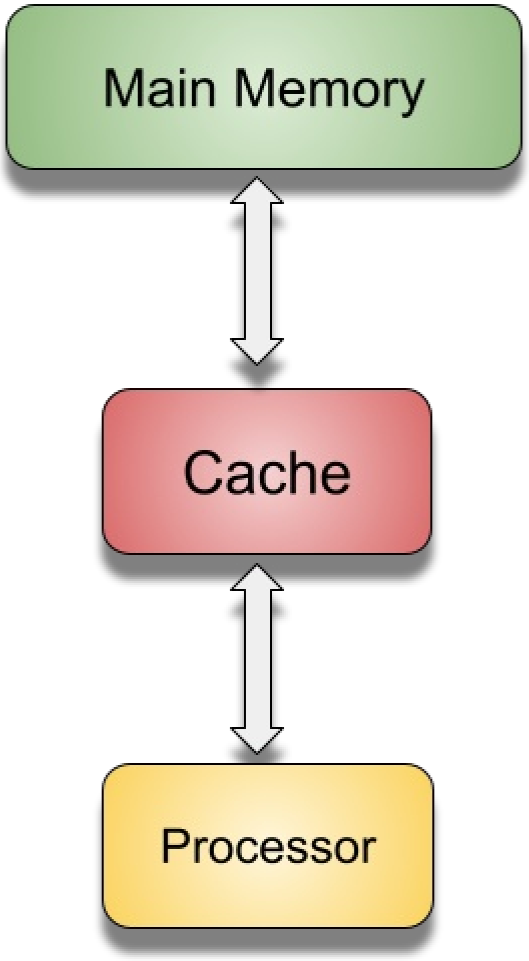
## Parallel Architectures

# Paralellism

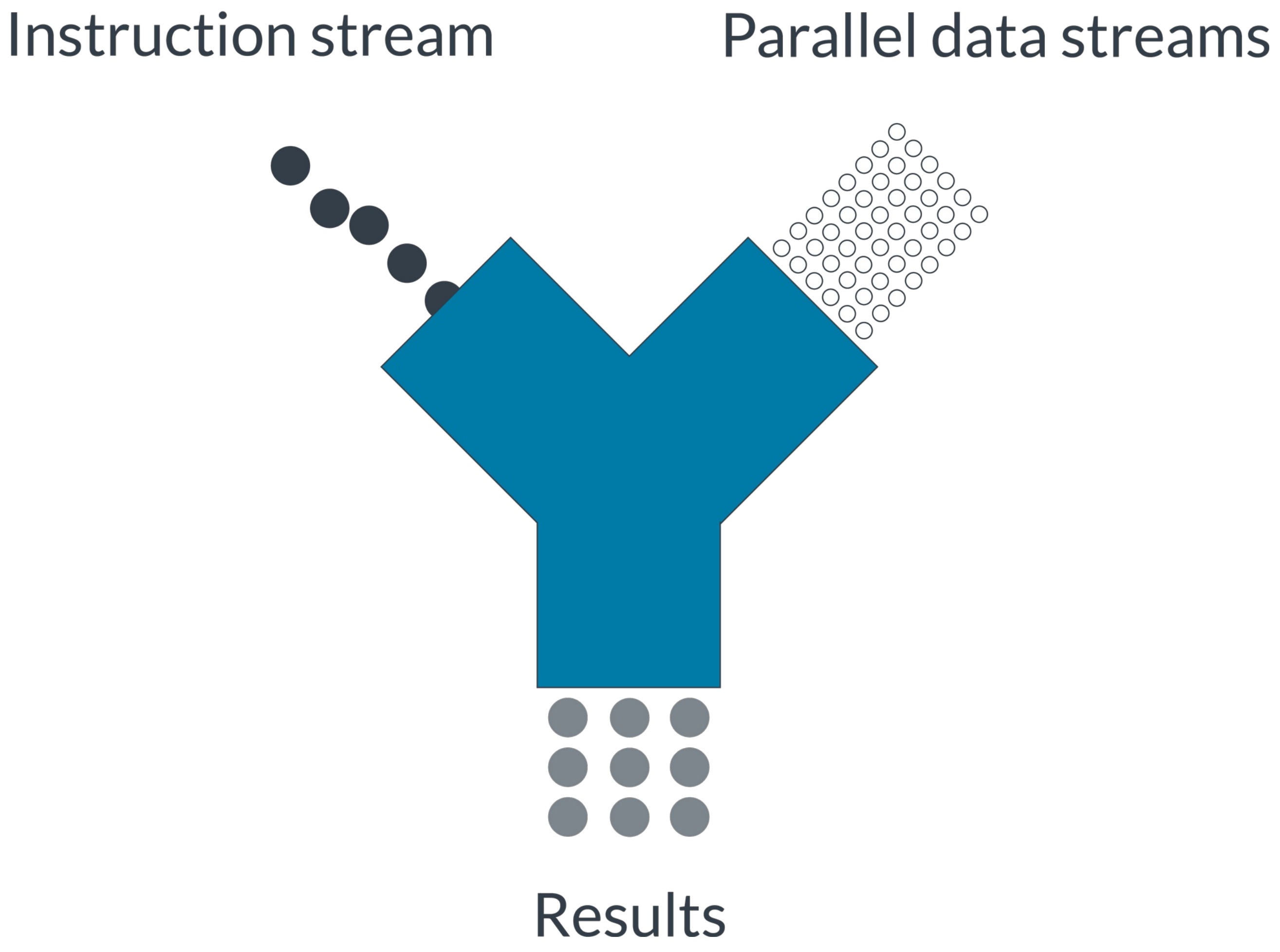
- Flynn Taxonomy (1966):
  - Single Instruction Stream (SI)
  - Multiple Instruction Streams(MI)
  - Single Data Stream (SD)
  - Multiple Data Streams (MD)

- Computer Architecture Categories
  - SISD (Conventional Processors)
  - SIMD (Vector Processors)
  - MIMD (Multiprocessors)

## SISD Architecture



# SIMD Architecture

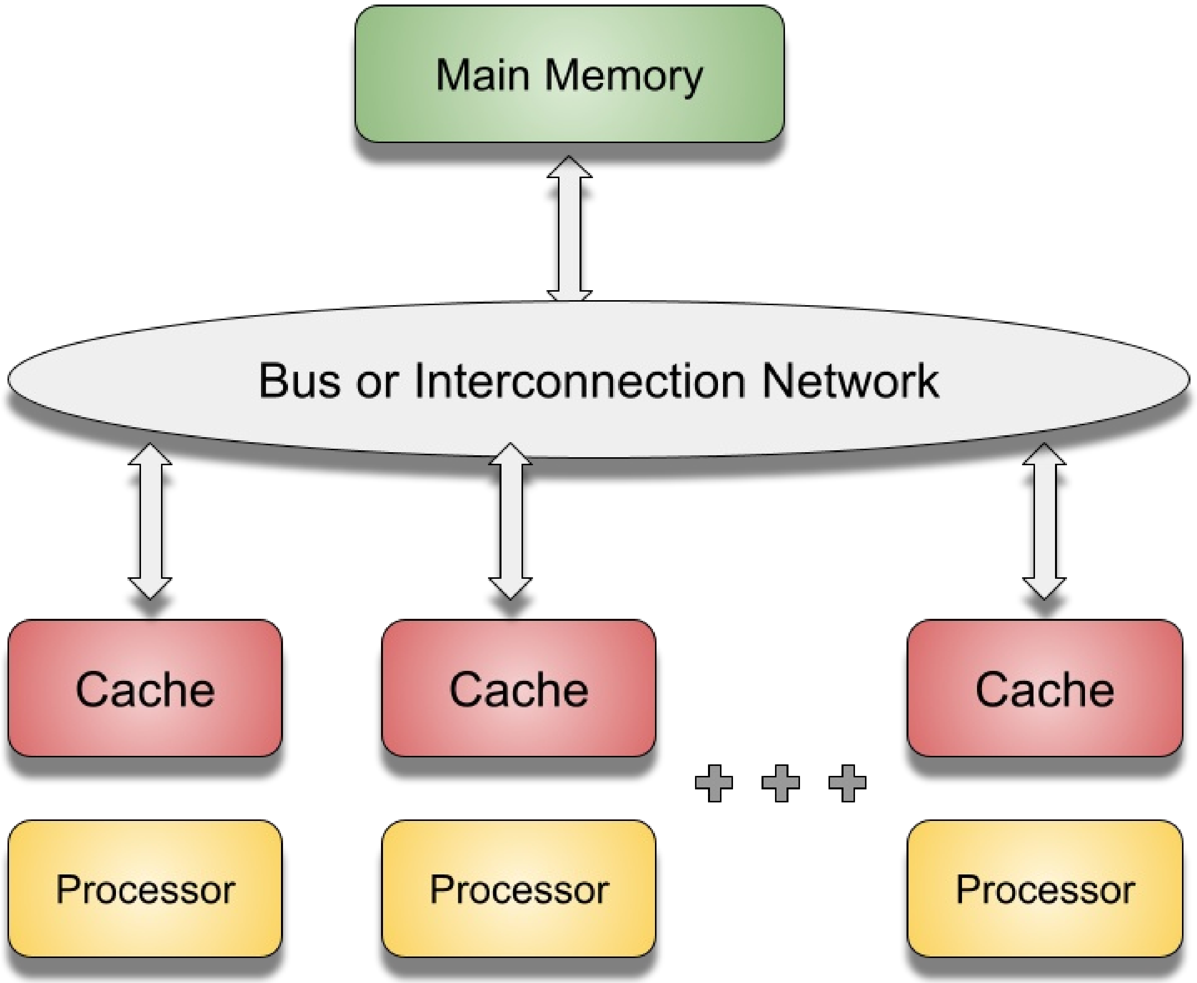


https://www.seekpng.com/ipng/u2q8u2a9a9u2q8a9_overview-simd-cpu-architecture/

## MIMD Architecture

- MIMD architectures are divided into two main categories:
  - Distributed Memory MIMD Architectures
    - Each processor can only access its local memory.
  - Shared Memory MIMD Architectures
    - Any processor can access the entire system memory space, whether local or not.

# Shared Memory MIMD Architecture



## Shared Memory MIMD Architecture

- Pros:
  - Uniprocessor programming techniques are easily adapted to multiprocessor environments since there is no need for data or code partitioning.
  - Also, data don´t need to be transferred or moved when two or more processors communicate.
  - As a result, communication between processes or threads is quite effective.

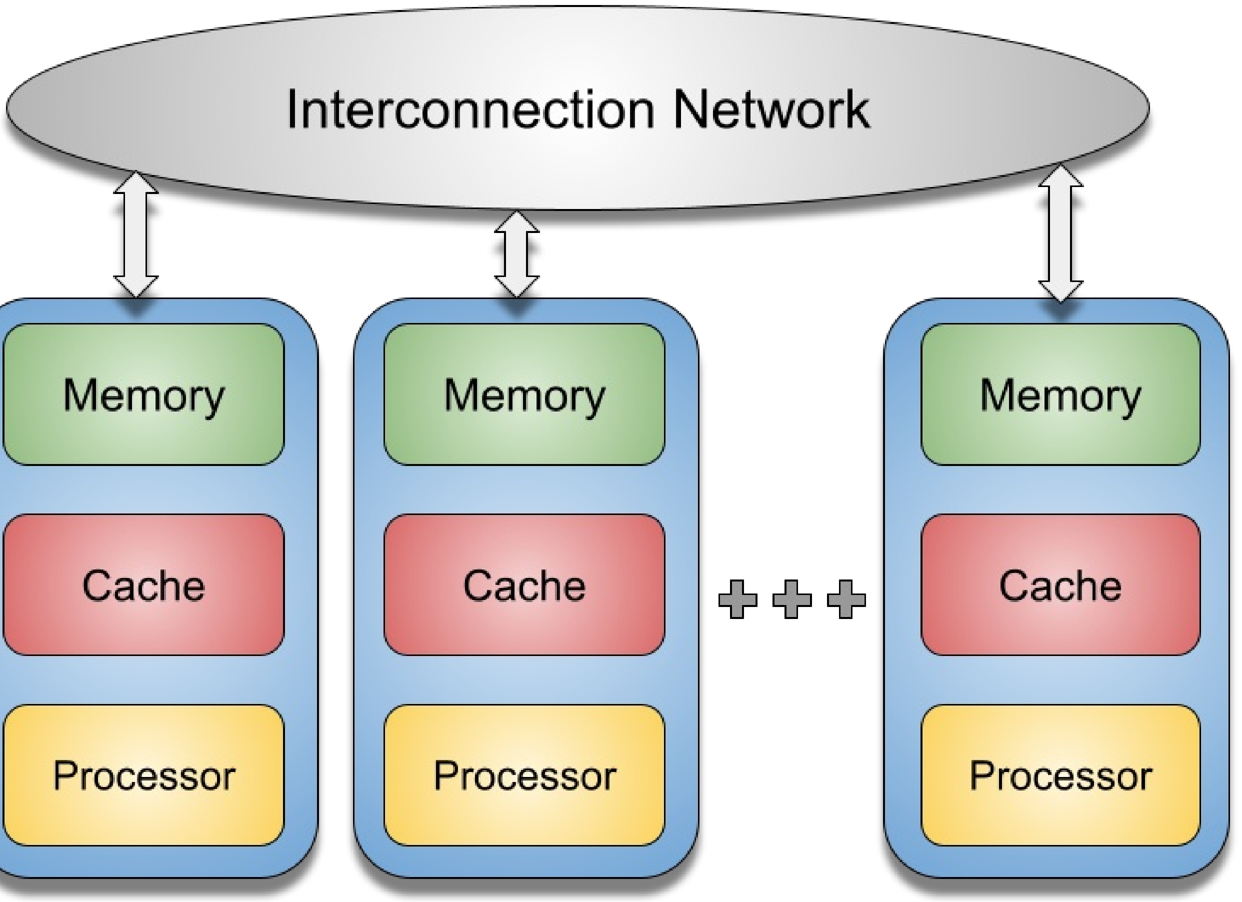
## Shared Memory MIMD Architecture

- Cons:
  - Special synchronization primitives are required when accessing shared memory regions to ensure correct computation results.
  - There is a lack of scalability due to memory contention issues. After a certain number of processors, adding more processors does not increase performance.

## Programming Shared Memory

- All processes/threads share a common address space.
- Communication takes place through variables in shared memory.
- Use of threads and/or processes for task mapping.
- Threads or processes are created dynamically or statically.
- Synchronization:
  - Mutual Exclusion
  - Barrier
- Languages: OpenMP and Pthreads

# Distributed Memory MIMD Architecture



## Distributed Memory MIMD Architecture

- Pros:
  - It is highly scalable and allows for building massively parallel processors.
  - Communication between processors takes place through message passing.
  - Message passing solves both communication and synchronization problems.

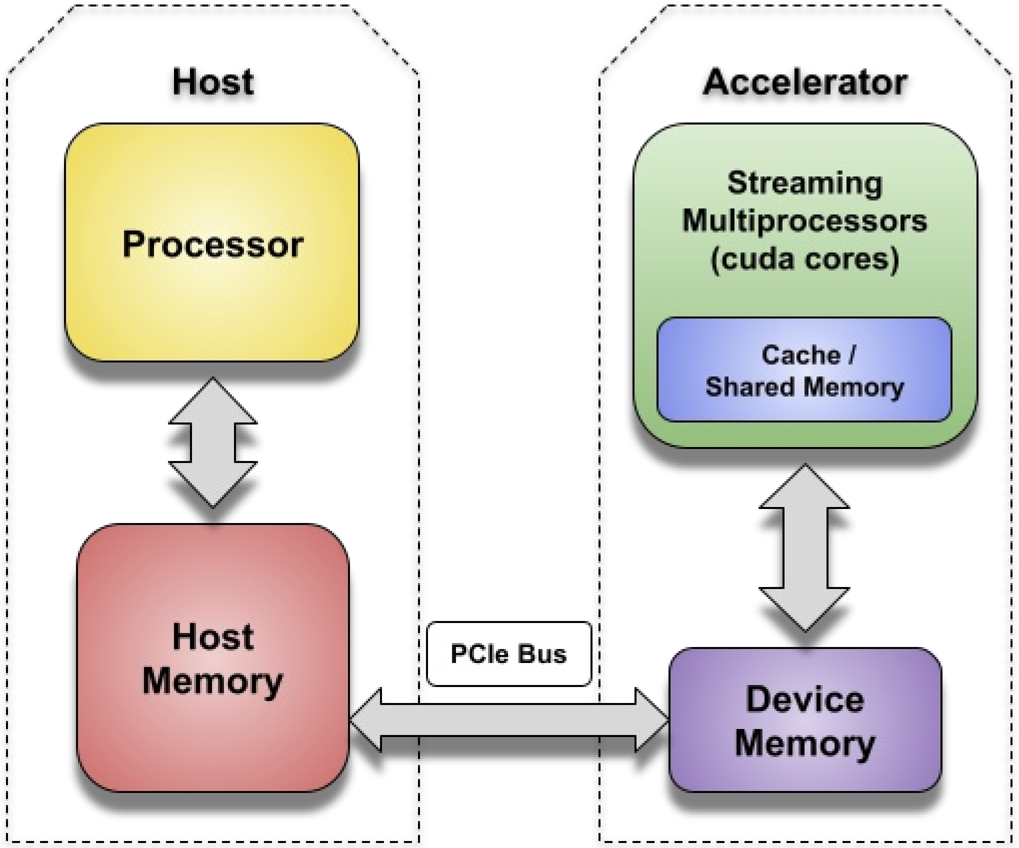
## Distributed Memory MIMD Architecture

- Cons:
  - An effective workload balancing between processors is required, either automatically or manually.
  - It is necessary to avoid deadlock situations both at the application and operating system levels.
  - It is a less natural programming model.

# Programming Distributed Memory

- Communication is made through explicit message exchanging.
- Tasks are mapped into processes, each with its private memory.
- Processes can be created statically or dynamically.
- Synchronization is done implicitly by message exchanging or collective synchronization operations (barriers).
- The processes/threads have no access to shared memory for synchronization and the communication is done by message passing.
- Languages: PVM and MPI.

# Accelerators



## Accelerators

- Pros:
  - GPUs Enable Exascale!
    - 7 of top 10 supercomputers
    - 9 of top 10 power Green500
  - GPUs enable ~3.5x more FLOPs/Watt efficiency
    - Lower Costs
    - Eases Access
  - GPUs are designed inherently parallel, when CPU cores designed for complex serial tasks.
  - GPUs can be used also for AI and Deep Learning.

## Accelerators

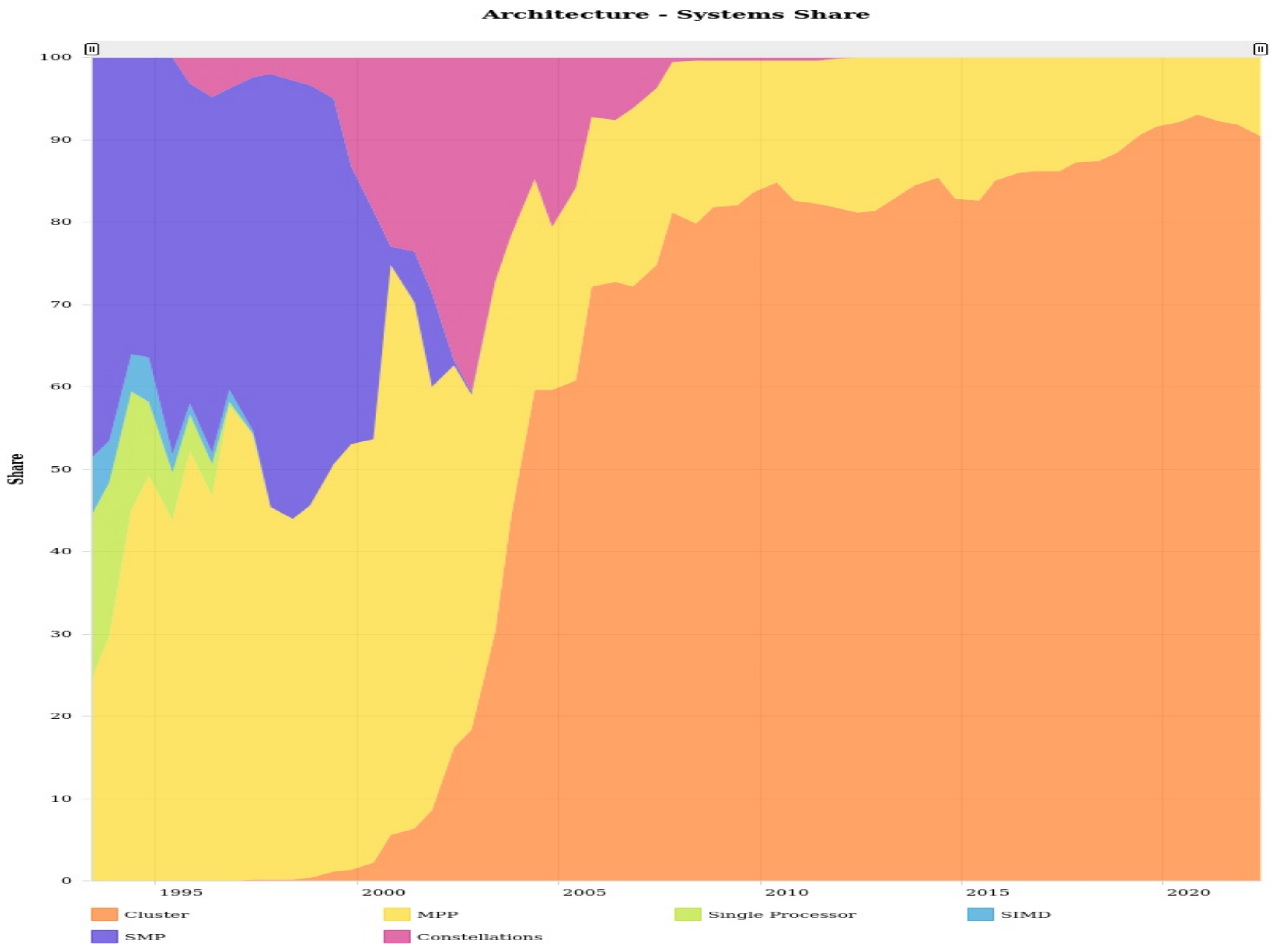
- Cons:
  - Requires data transfer between host memory and accelerator memory
  - It only can be used on very intensive computing kernels
  - Does not to have speed gains to many classes of scientific problems.
  - Can be very difficult to program if using CUDA

## Programmnig Accelerators

- The most computationally intensive loops (kernels) are transferred and processed in the accelerators.
- There are separate memories for the host and accelerator.
- Synchronization is done with special routines.
- It can be used both with distributed or shared memory computers.
- Languages: OpenACC, CUDA, OpenMP and OpenCL.

- https://www.amd.com/en/graphics/instinct-server-accelerators

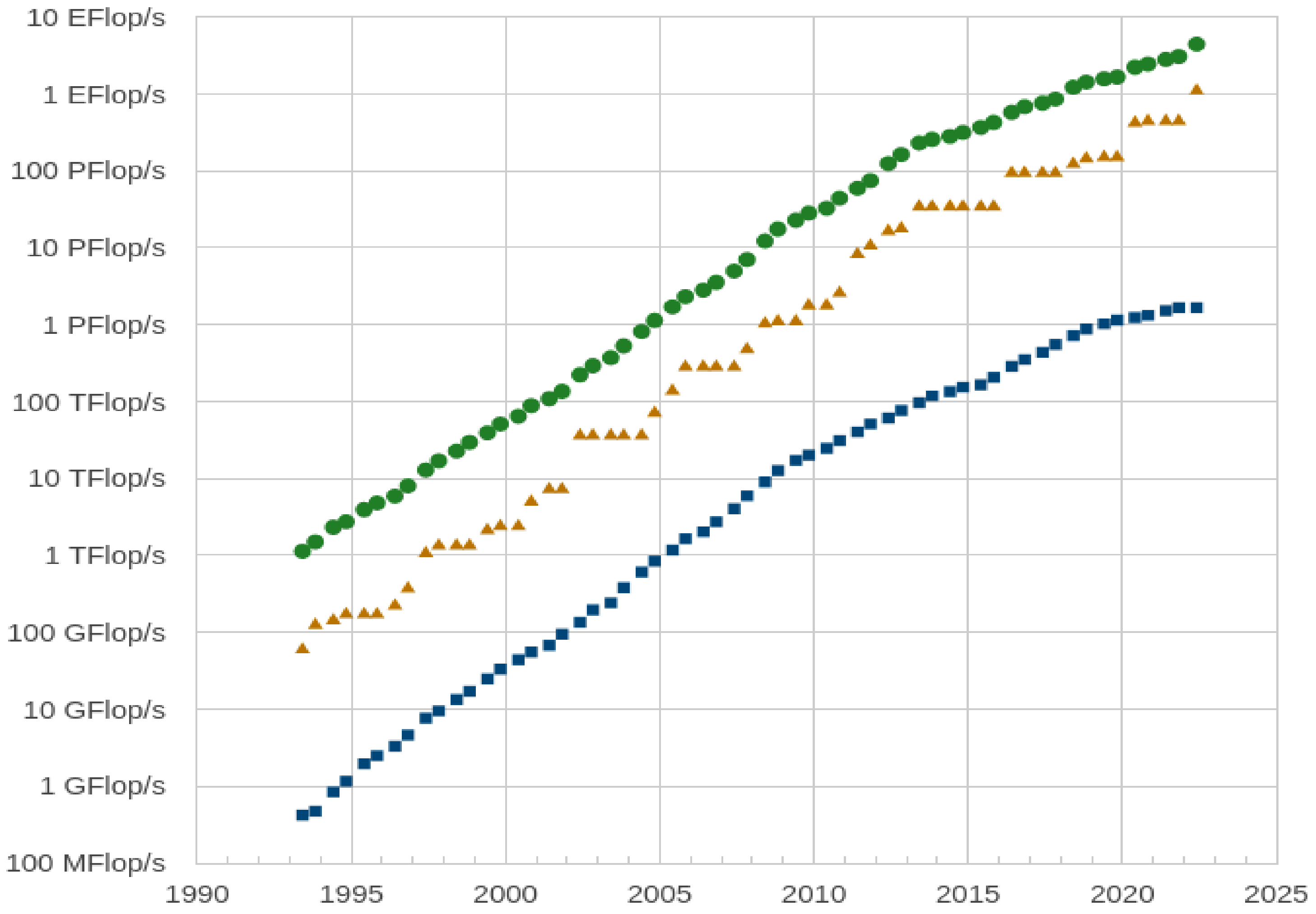
# Top500 Evolution



# Top500 Evolution

- 1 GFlop/s – 1988:
  - Cray Y-MP; 8 Processors
  - Finite element static analysis
- 1 TFlop/s – 1998:
  - Cray T3E; 1024 Processors
  - Metallic magnets atoms modeling
- 1 PFlop/s – 2008:
  - Cray XT5; $1.5\text{x}10^5$ Processors
  - Superconductors materials
- 1 Eflop/s – 2022(!):
  - Frontier; $8.7\text{x}10^6$ Processors
  - Energy, economic and national security

# Top500 Performance Evolution

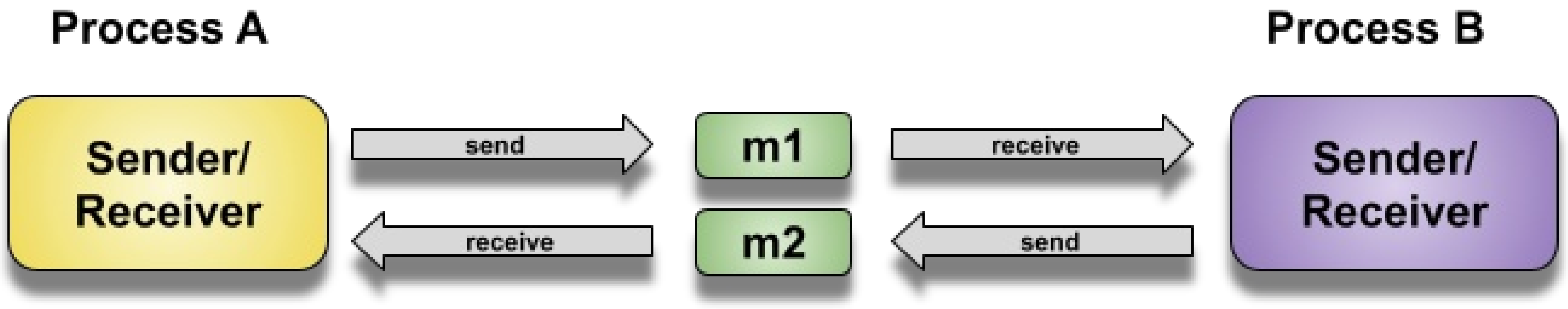


# Top500 (June/2022)

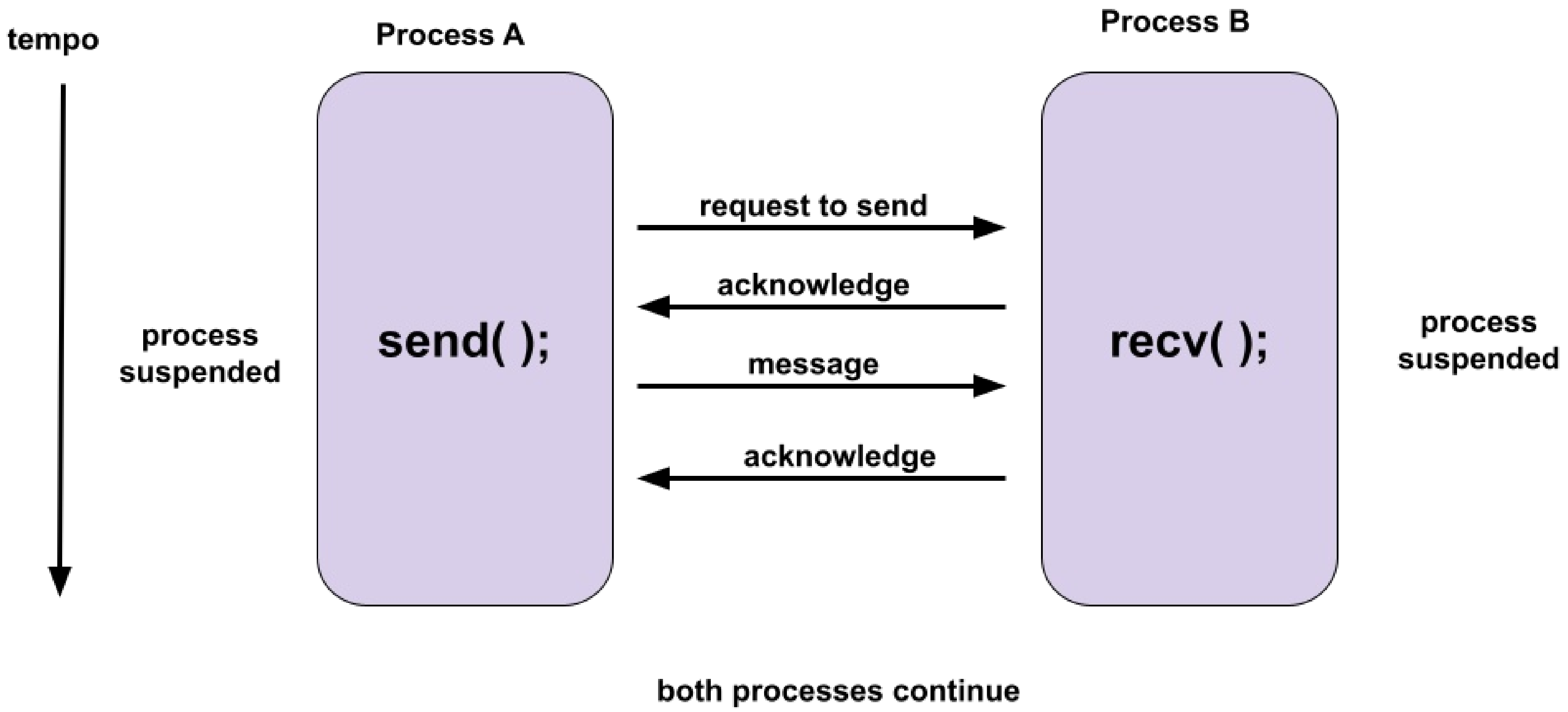
1) Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE – 1,102 PFlops (8,730,112 cores) – 21 MW – USA
2) Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu – 442 PFlops (7,630,848 cores) – 30 MW – Japan
3) LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE – 151 PFLops (1,110,144 cores) – 2.9 MW – Finland
4) Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM – 148 Pflops (2,414,592 cores) – 10 MW – USA
5) Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox- 94 PFlops (1,572,480 cores) – 7,5 MW – USA
6) Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC – 93 PFlops (10,649,600 cores) – 15 MW – China

## Message Passing

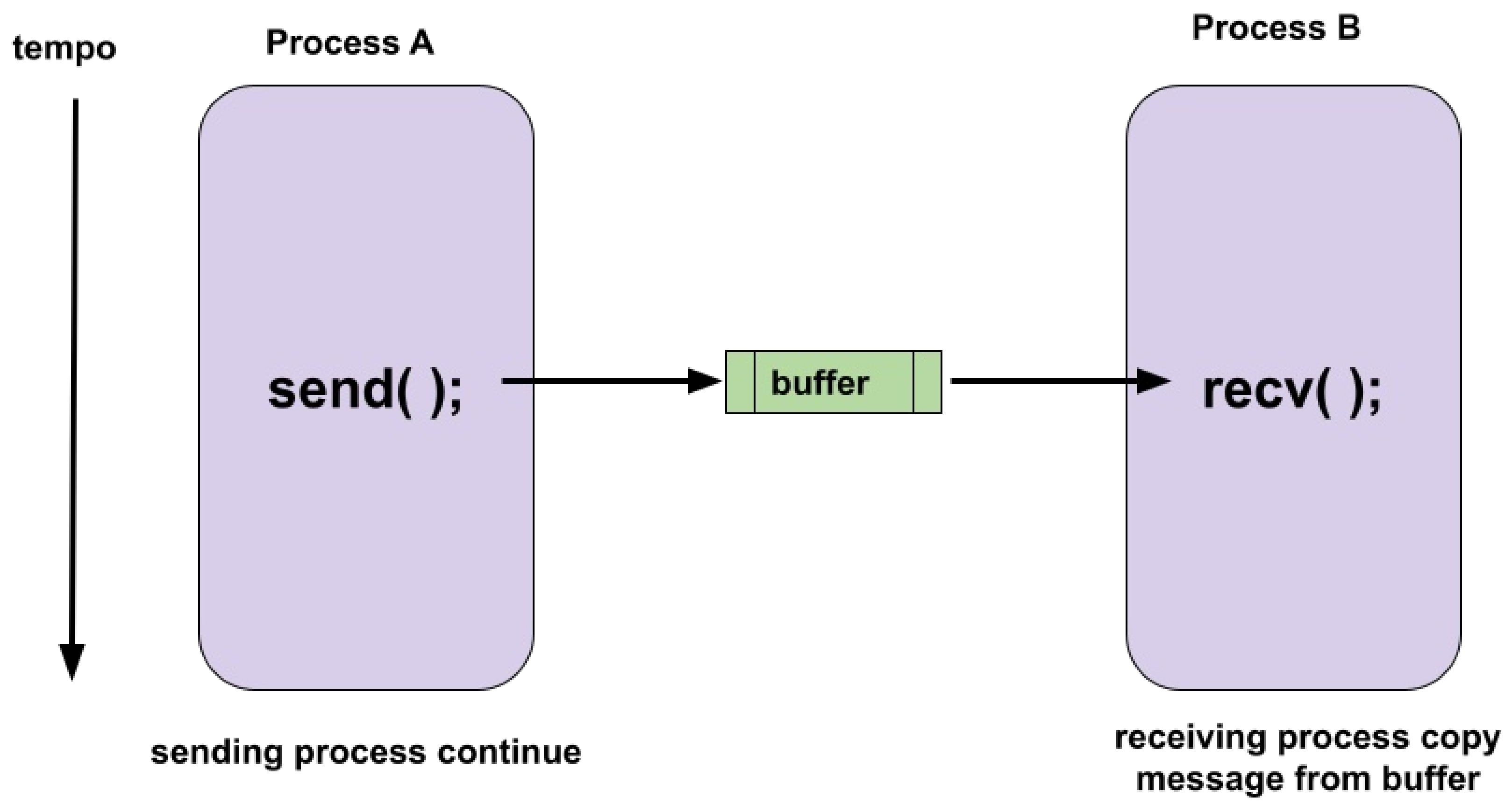
## Message Passing



## Synchronous Communication



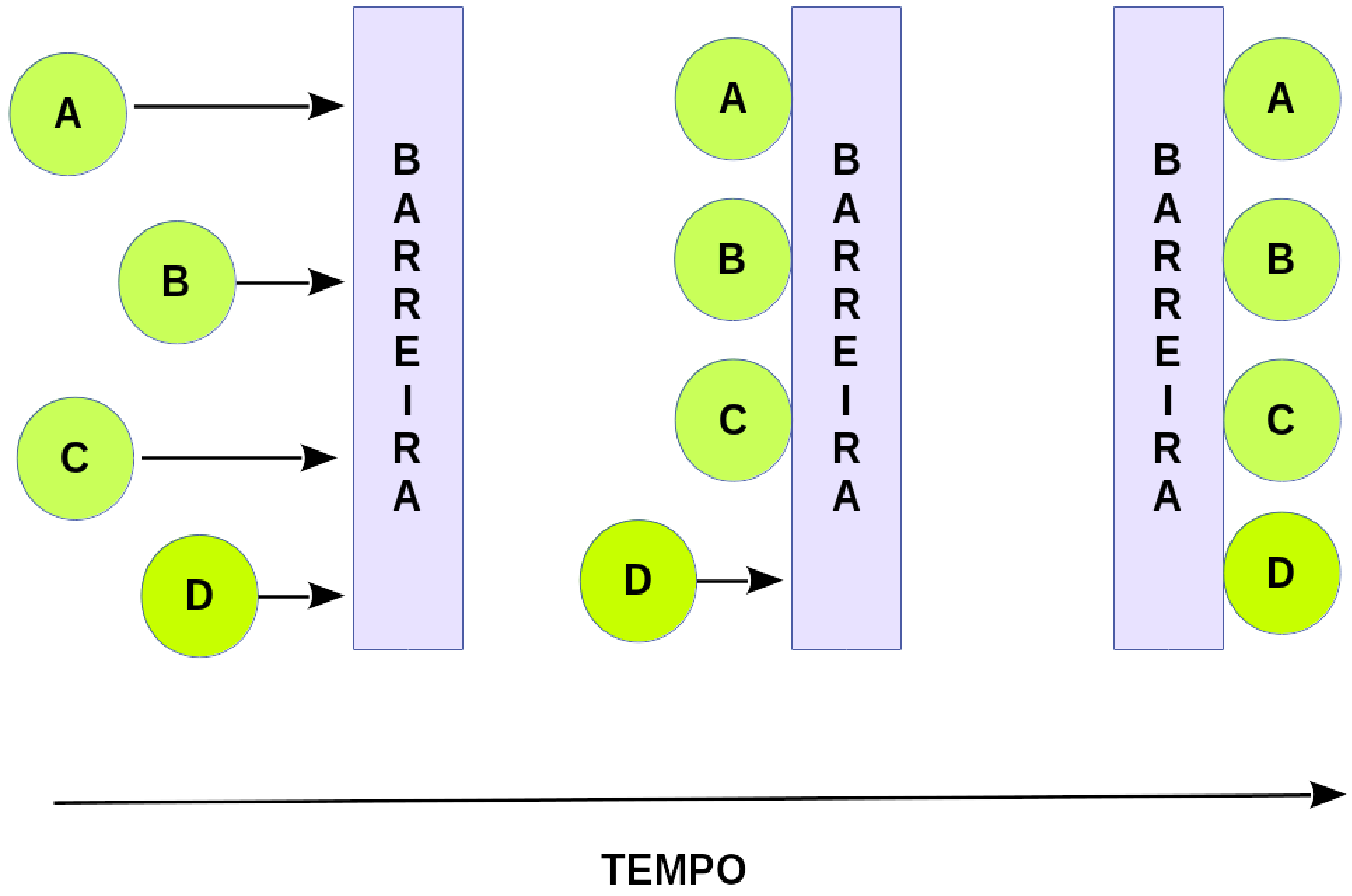
# Asynchronous Communication



## Synchronous vs Assynchronous

- Synchronous communication mode is simple and secure. However, it eliminates the possibility of overlapping application processing and message transmission, reducing overall parallelism.
- The asynchronous communication mode allows greater overlap between application processing and message transmission, increasing the possibilities of exploiting parallelism.

# Barrier



## Performance Evaluation

## Performance Metrics

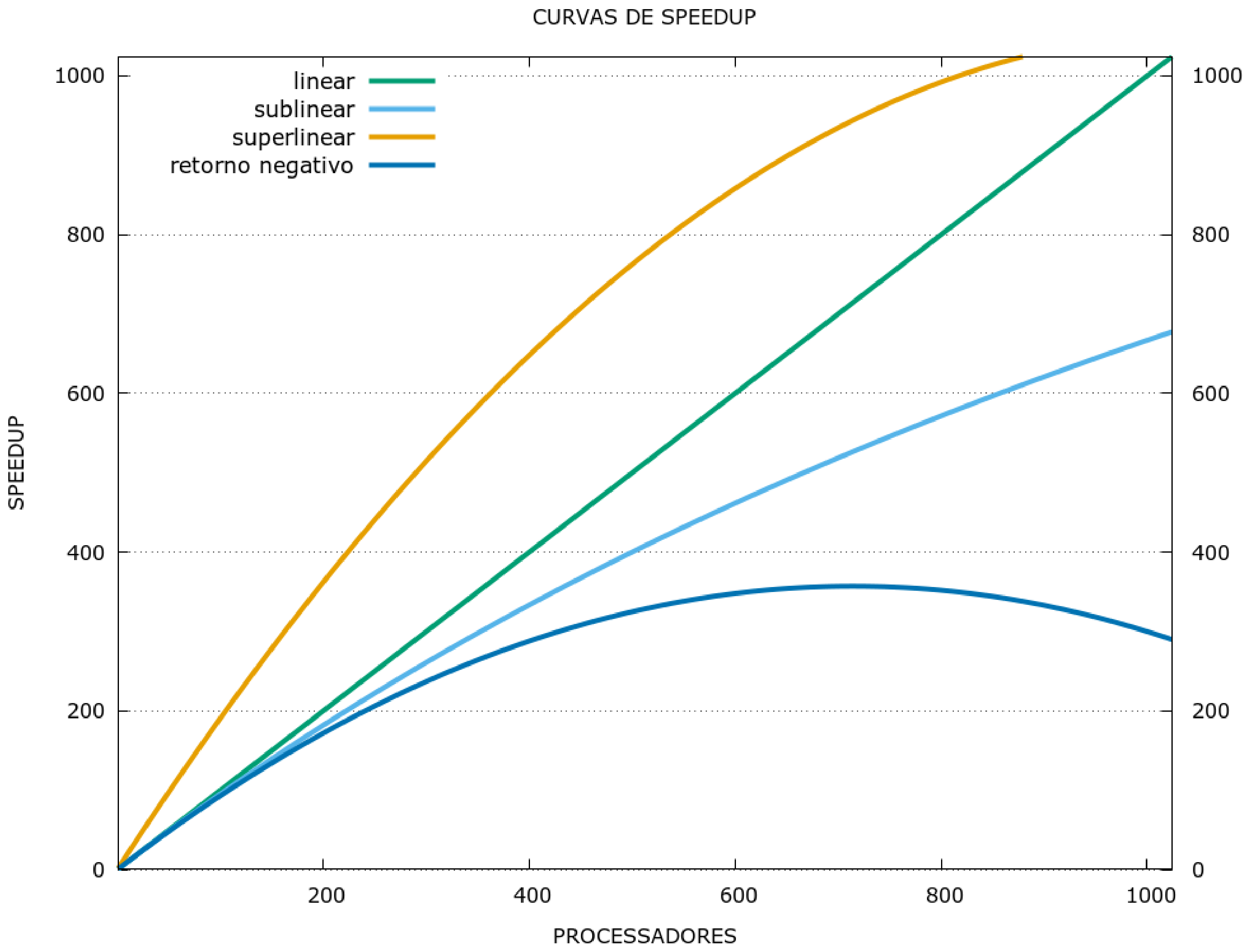
- Speedup:
  - It is the ratio between the time spent executing an algorithm or application on a single processor and the time spent executing it with n processors:

$$S(n) = T(1)/T(n)$$

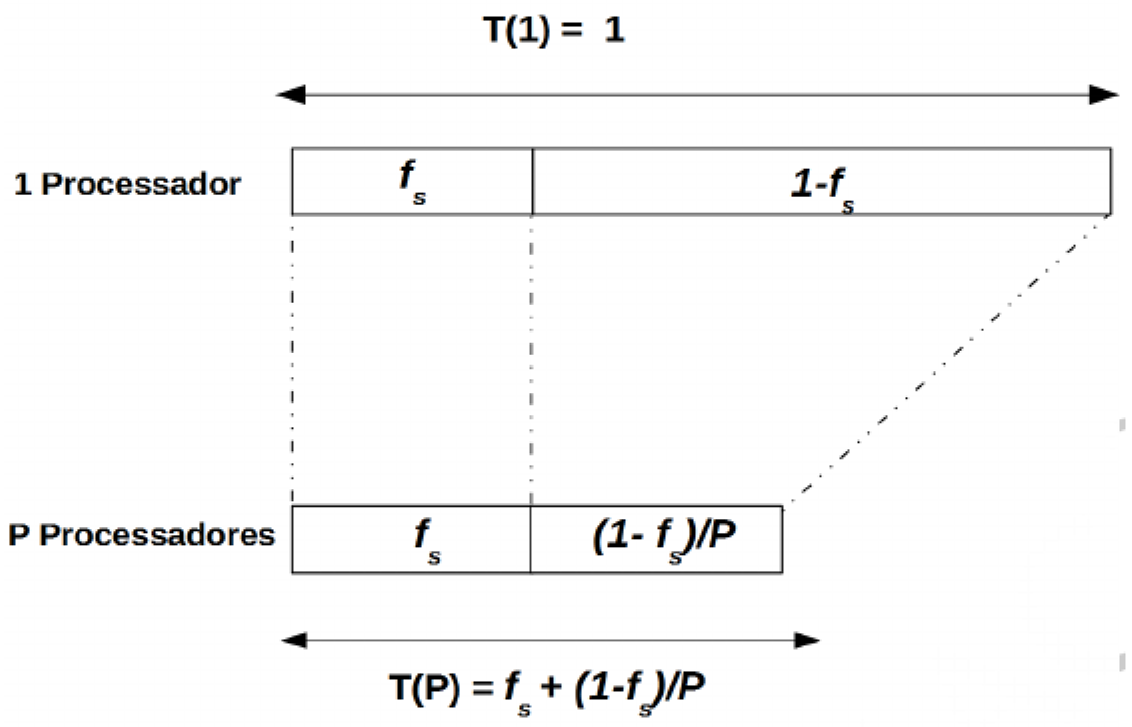
- Efficiency:

$$E(n) = S(n)/n$$

# Speed-up


CURVAS DE SPEEDUP
linear
sublinear
superlinear
retorno negativo
SPEEDUP
PROCESSADORES
0
200
400
600
800
1000

# Amdahl’s Law



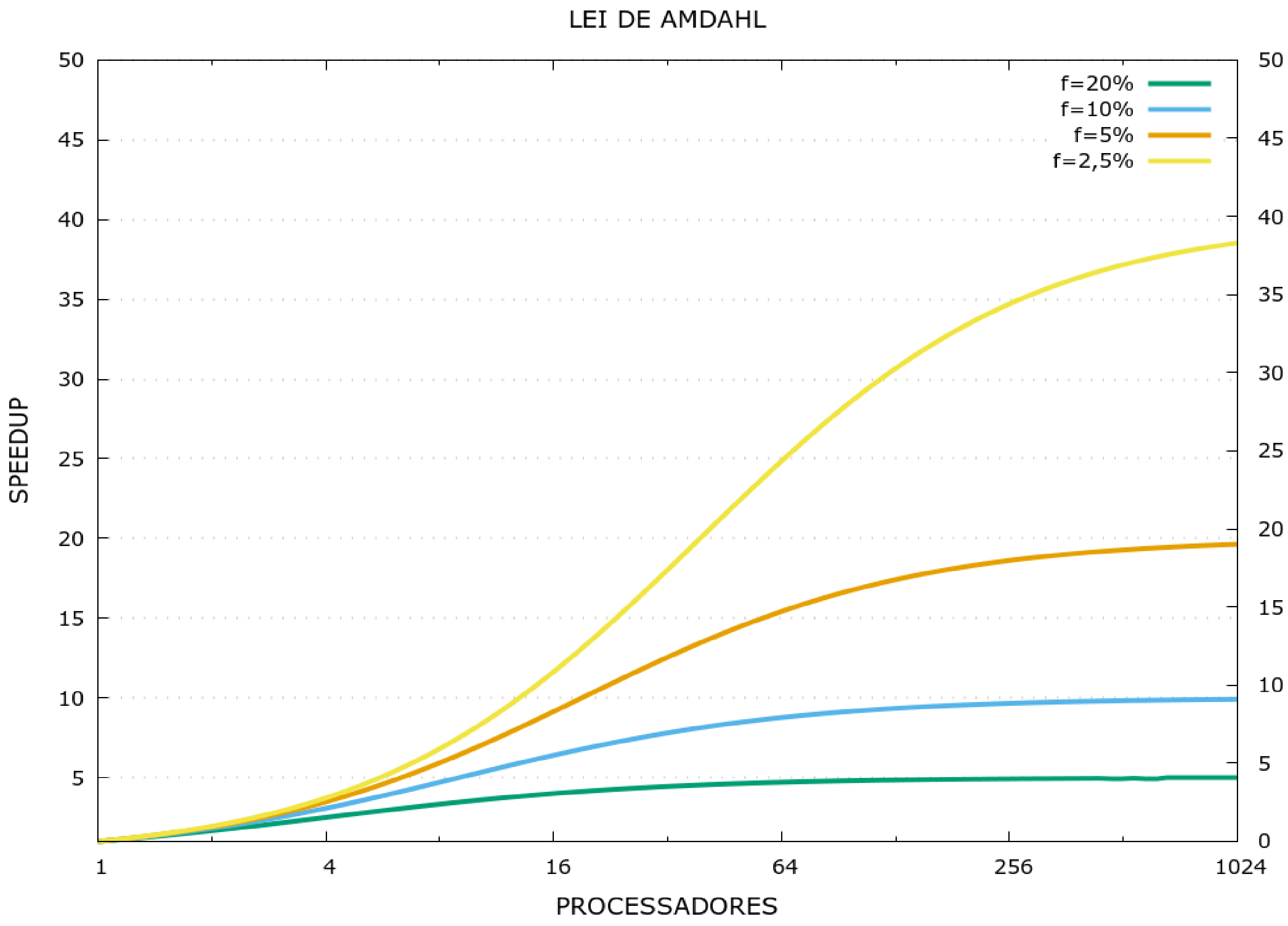
# Amdahl’s Law

$$S(P) = \frac{T(1)}{T(P)} \tag{2.4}$$

$$S(P) = \frac{f_s + (1 - f_s)}{f_s + \frac{(1 - f_s)}{P}} \tag{2.5}$$

$$S(P) = \frac{1}{f_s + \frac{(1 - f_s)}{P}} = \frac{P}{1 + (P - 1) f_s} \tag{2.6}$$

# Amdahl’s Law



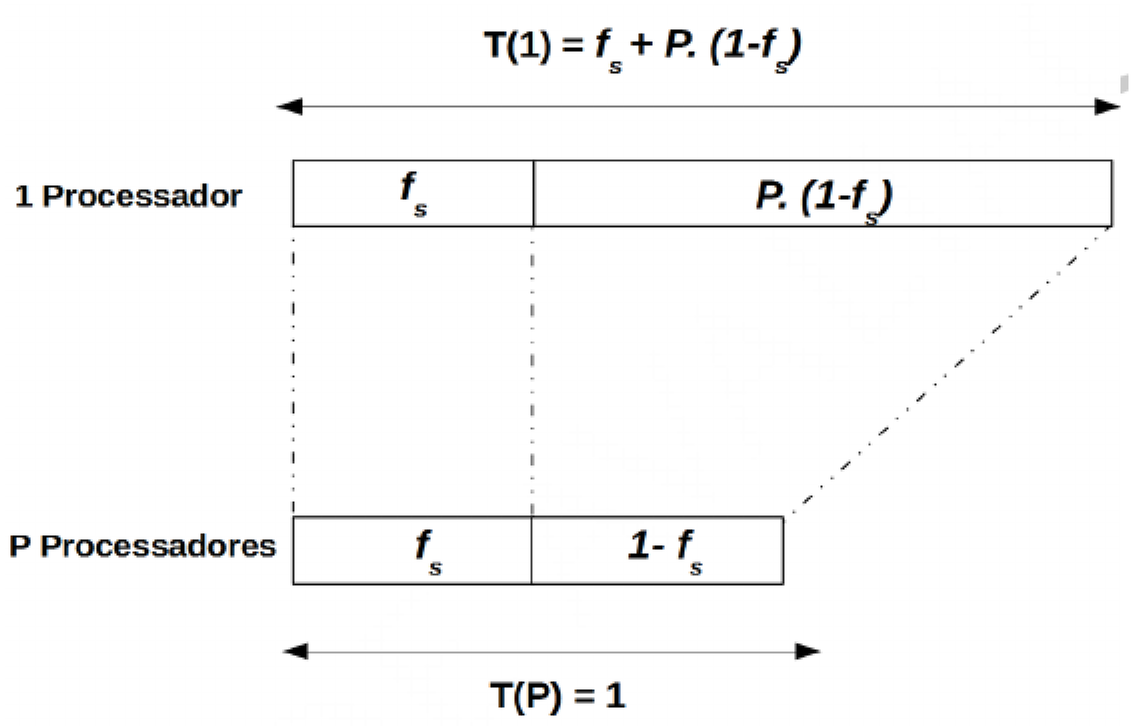
LEI DE AMDAHL
SPEEDUP
PROCESSADORES
f=20%
f=10%
f=5%
f=2,5%
0
5
10
15
20
25
30
35
40
45
50
1
4
16
64
256
1024

## Amdahl’s Law

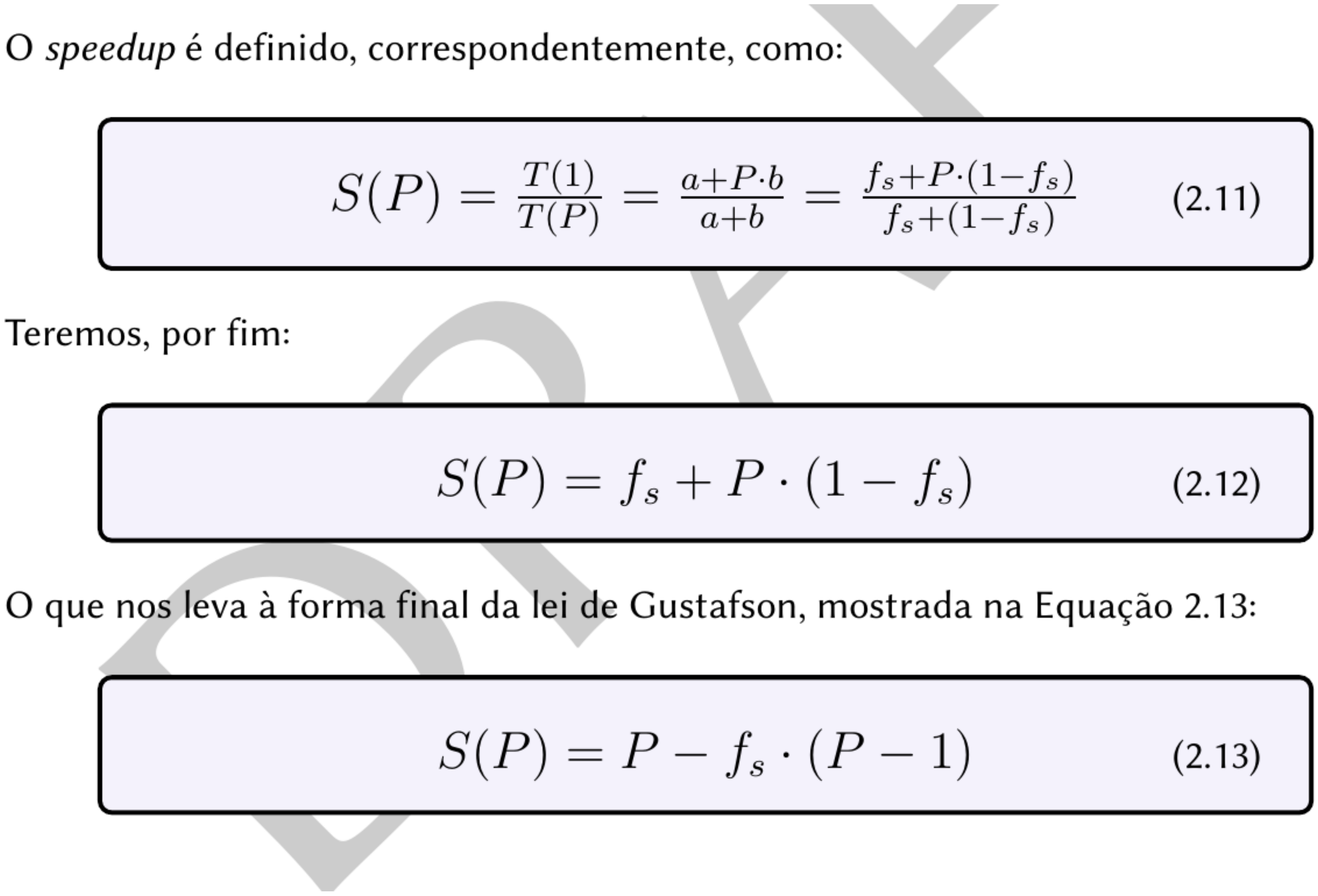
- Even with an infinite number of processors, the maximum speedup is limited to 1/f , where f is the serial fraction of the program.
- Example:
  - With only 5% of serial computation, the maximum speedup is 20, no matter how many processors are in use.

$$\boxed{S_t(P) = \frac{1}{\frac{1}{20}} = \frac{1}{0{,}05} = 20}$$

# Gustafson’s Law



# Gustafson’s Law



O *speedup* é definido, correspondentemente, como:

$$S(P) = \frac{T(1)}{T(P)} = \frac{a+P\cdot b}{a+b} = \frac{f_s+P\cdot(1-f_s)}{f_s+(1-f_s)} \tag{2.11}$$

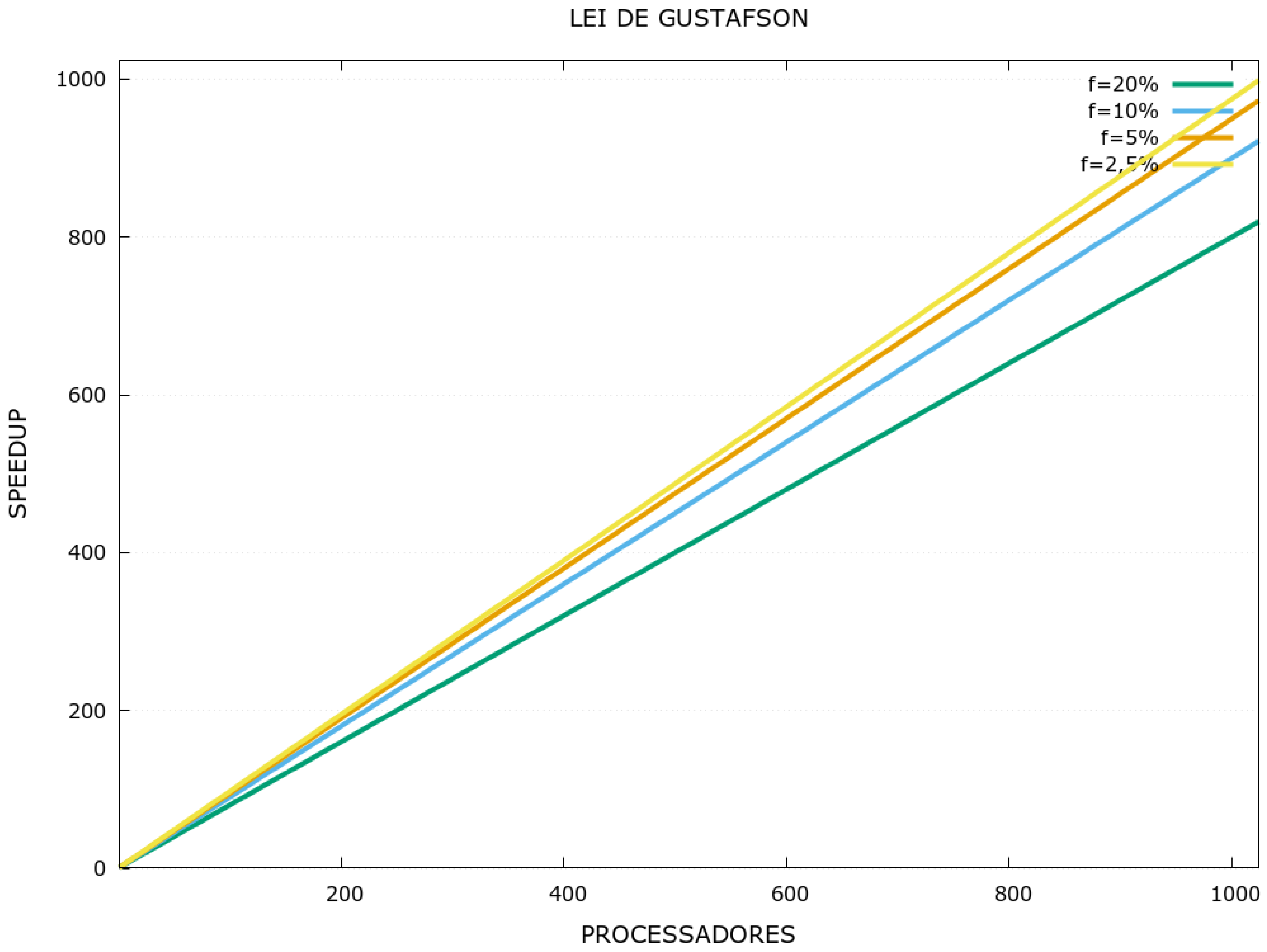
Teremos, por fim:

$$S(P) = f_s + P \cdot (1 - f_s) \tag{2.12}$$

O que nos leva à forma final da lei de Gustafson, mostrada na Equação 2.13:

$$S(P) = P - f_s \cdot (P - 1) \tag{2.13}$$

# Gustafson’s Law



## Scalability

- A system is said to be scalable when its efficiency remains constant as the number of processors (P) applied to the problem solution grows.
- But, if the problem size is kept constant as the number of processors increases, the communication overhead increases, and then efficiency decreases.
- A fair scalability analysis should increase the problem size to be solved proportionally as the number of processors grows. So it counterbalances the natural increase in the communication overhead when P grows.

## Scalability

- A problem of size S using P processors takes time T to execute.
- The system is said to be scalable if a problem of size 2S on 2P processors takes the same time T.
- Usually scalability is a more desirable property than speed-up.

## Final Remarks

# Some Considerations

- There are basically two approaches to create a parallel programs:
  - Data partitioning
  - Functional partitioning
- But, one should try to exploit data locality, reduce communication costs, parallelize communication and computation, reduce synchronization overhead, and provide good load balancing.
- If there are idle processors at any time, it is clear evidence of load unbalancing.
- Communication and synchronization costs can’t be higher than processing time. If so, you did something wrong.

## Performance Considerations

- The message-passing paradigm places no limits on how a parallel program is structured.
- However, some performance considerations you must take into account: the first is the granularity of the task, which typically refers to the ratio of the number of bytes received by a process to the number of floating point operations it performs.
- Increasing the granularity will decrease the amount of application communication, but it also will cause a reduction in available parallelism.

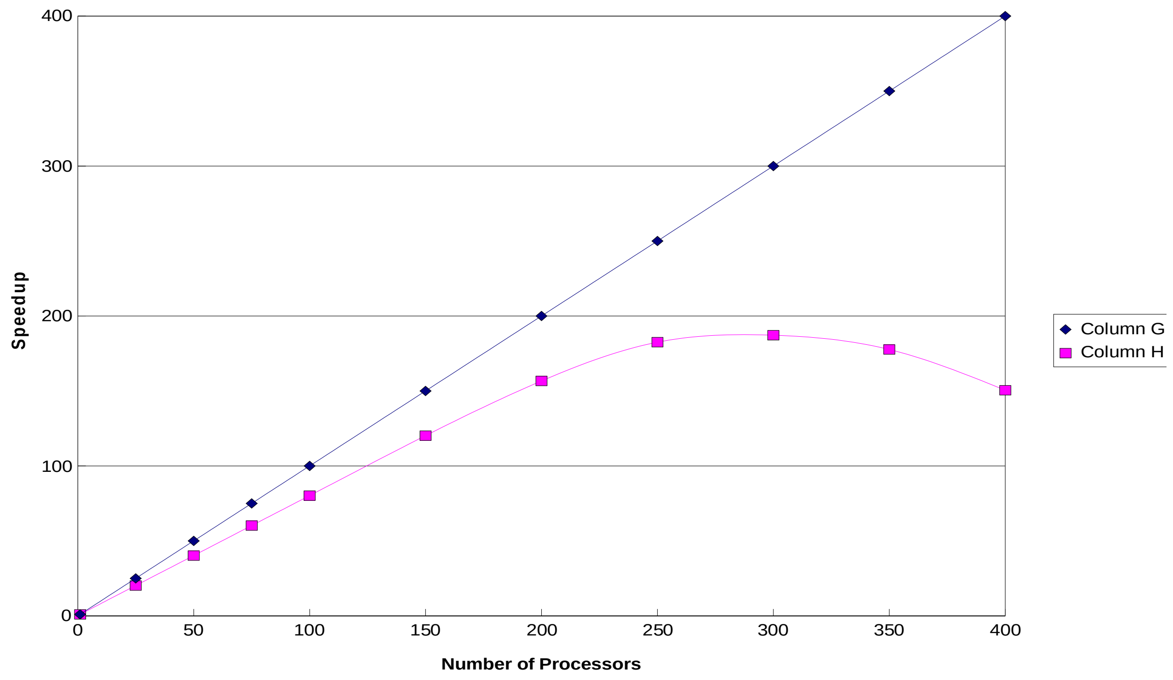
## Performance Considerations

- The total number of messages sent is also a factor to consider.
- As a general rule of thumb, sending a small number of large messages takes less time than sending a large number of small ones.
- It does not always apply, however. Some applications may overlap computation by sending small messages. The optimal number of messages is application specific and only should be known on a case-by-case basis, according to the experience of the programmer.

## Performance Considerations

- Even if all machines are of the same type and model, they may have some performance differences.
- Network considerations also matter if you are using a set of independent nodes.
- Network latencies can cause issues, and the available processing power can vary dynamically depending on the load on each machine.
- Some form of dynamic load balancing is required to avoid these problems.

# Load balancing

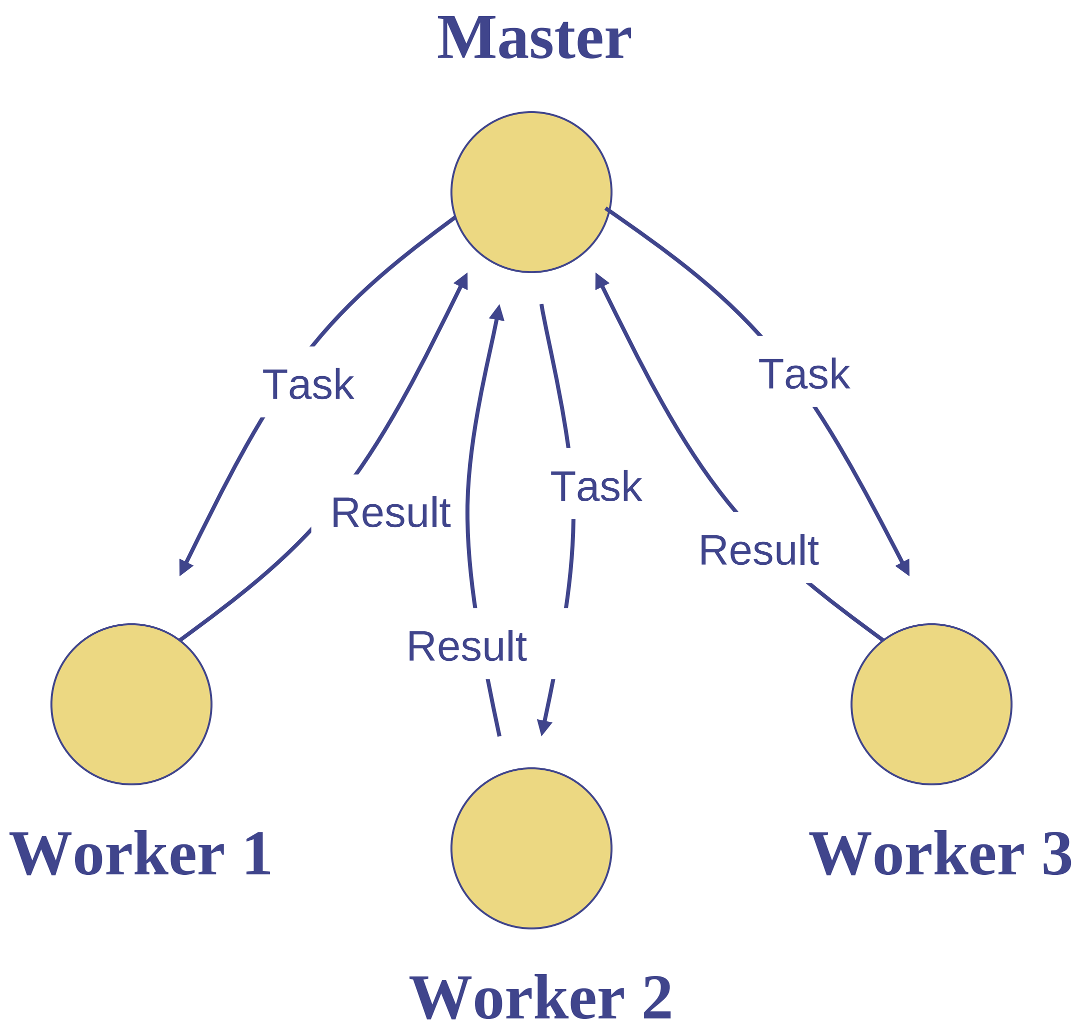


- Speed-up sublinear due to:
  - Sequential code fraction
  - Communication bottlenecks
  - Synchronization overhead
  - Bad tasks distribution

## Load Balancing

- The most simple load balancing scheme is the static one, but it can lead to further unbalancing when computation proceeds.
- So, some dynamic load balance should be applied. The simplest one is the bag of tasks.
- It is implemented typically as a master/worker program, where the master process manages a set of worker tasks.
- It sends jobs to workers as soon as they are idle.
- Various schemes and structures, such as queues and task vectors, can be used for this.

# Bag of Tasks


Master
Task
Result
Task
Result
Task
Result
Worker 1
Worker 2
Worker 3

## Dynamic Balancing

- The master-slave method is not suitable for applications that require a lot of task-to-task communication, as the tasks will start and end at arbitrary times.
- In this case, a third method should be used. At some predetermined time, all processes stop; workloads are re-examined and re-distributed if necessary.
- Variations of these methods are possible for specific applications.
- But if you apply more sophisticated balancing algorithms, you should be sure that the benefits overcome the additional overhead introduced by the algorithm.

## Bibliography

- Parallel Programming with MPI
  - Pacheco, P.S., Morgan Kaufmann Publishers, 1997.
- Using MPI-2: Advanced Features of the Message-Passing Interface
  - Gropp, W.; Lusk, E.; Thakur, R. The MIT Press, 1999.
- Parallel Computing: theory and pratice
  - Quinn, Michael J. – McGraw-Hill, 1994.
- Parallel Computer Architecture: A Hardware/Software Approach
  - David E. Culler and J. P. Singh – Morgana Kaufman 1999