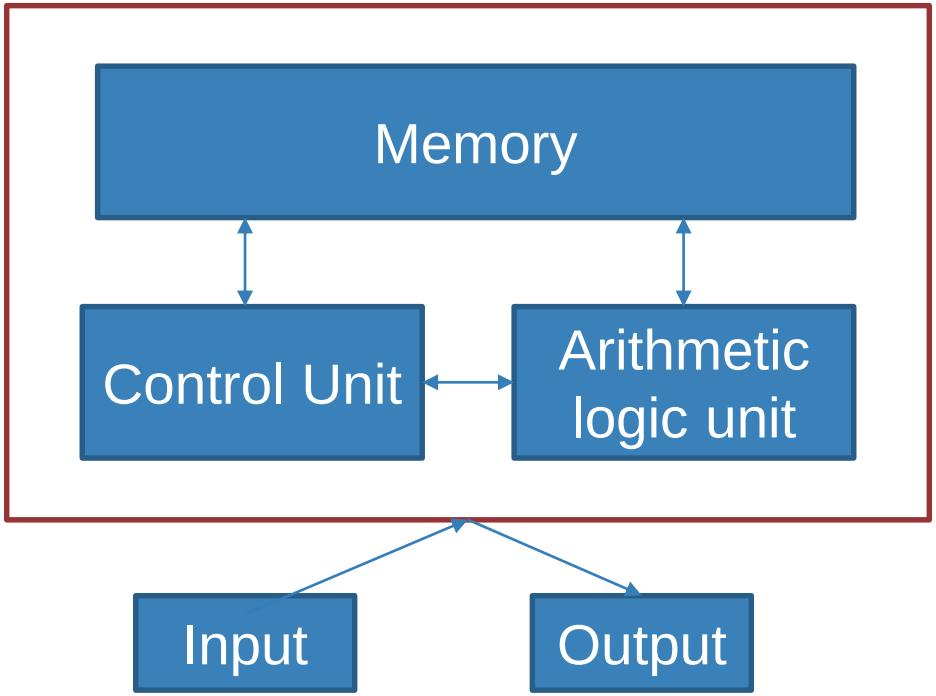
OpenMP - Modern hardware

Matic Brank

Lecad laboratory

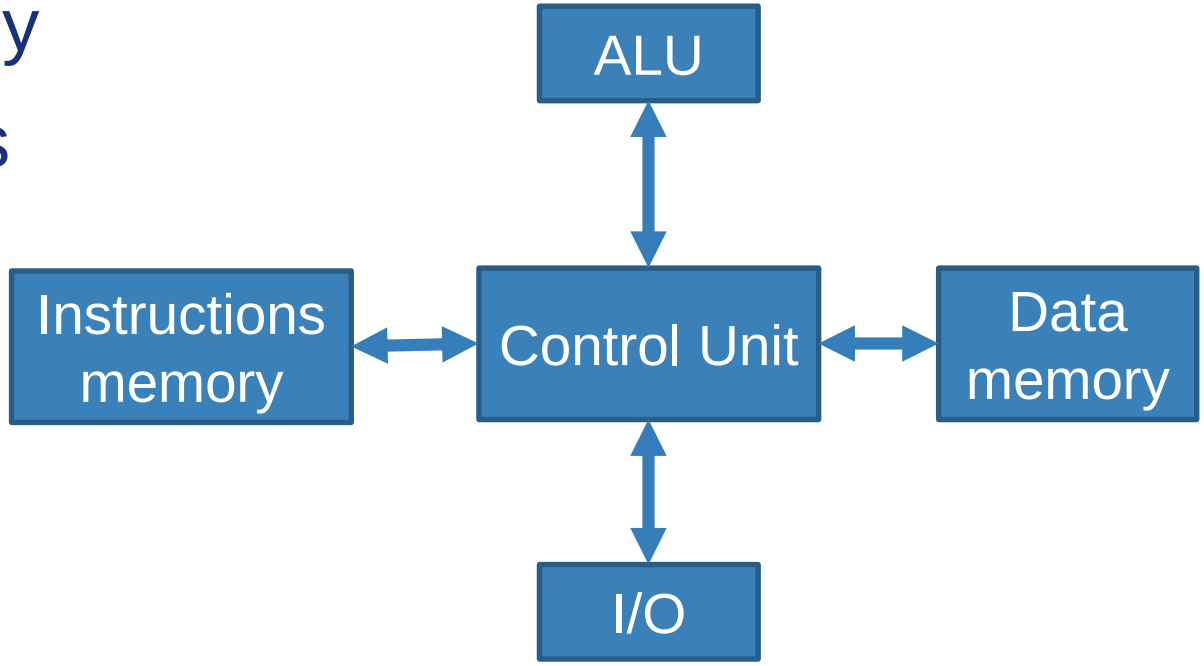
Von Neumann Architecture

- Hungarian-American mathematician John von Neumann proposed the design in 1945
  - Part of the proposal of computer EDVAC, which followed ENIAC
- Four main components:
  - Memory
  - Control Unit
  - Arithmetic Logic Unit
  - Input/Output



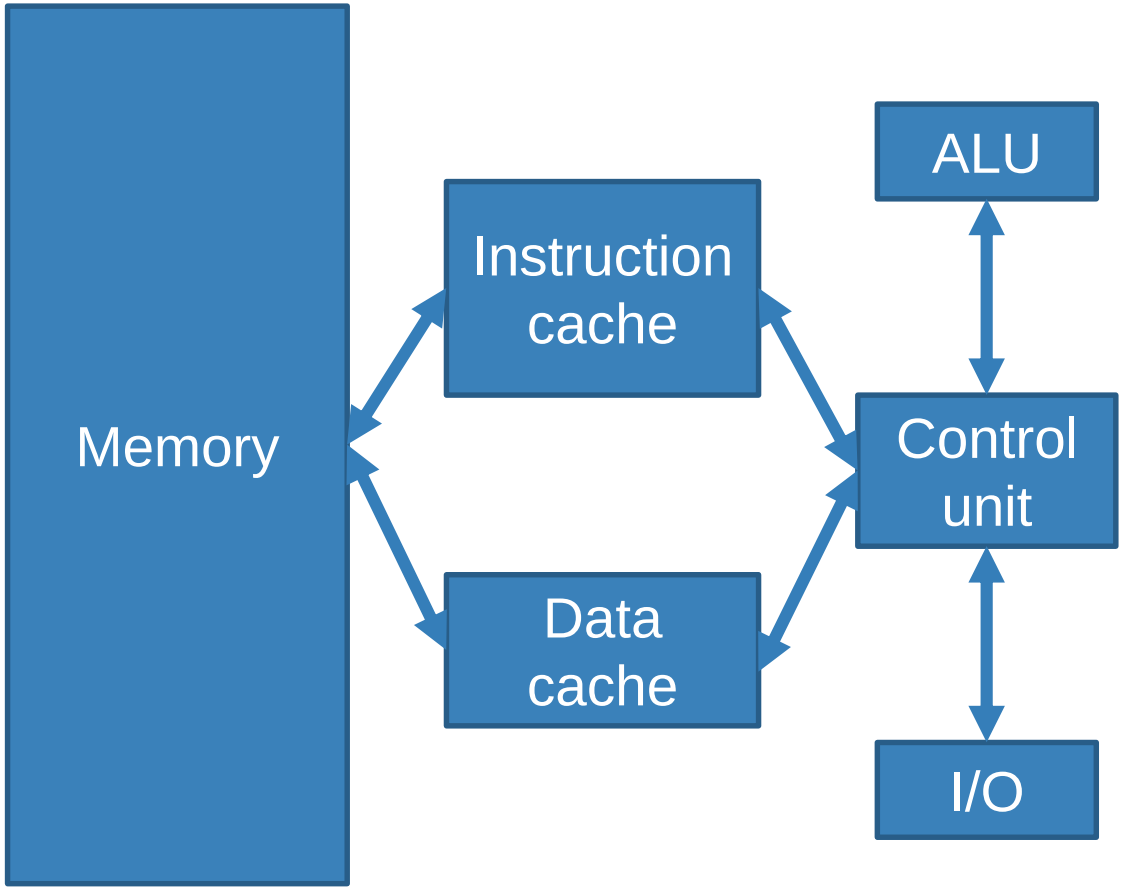
Harvard architecture

- Instruction read and data memory access can be performed simultaneously
- Different address spaces for program and data memories

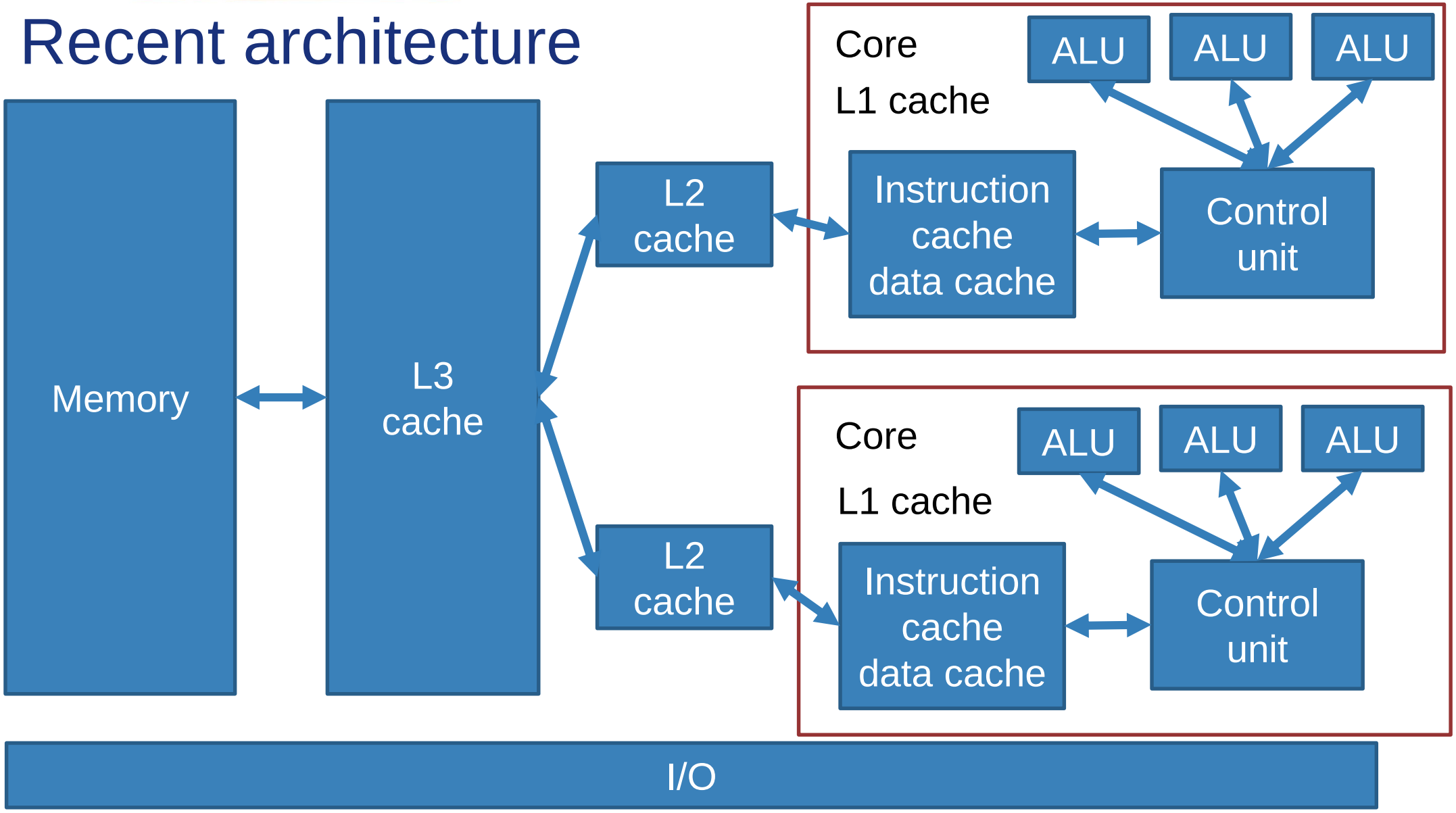


Modified Harvard architecture

- Introduction of a caching system: smaller but much faster memory closer to the CPU (short access latency)
- Mix of both previous architectures: Harvard to access caches; Von Neumann to access memory

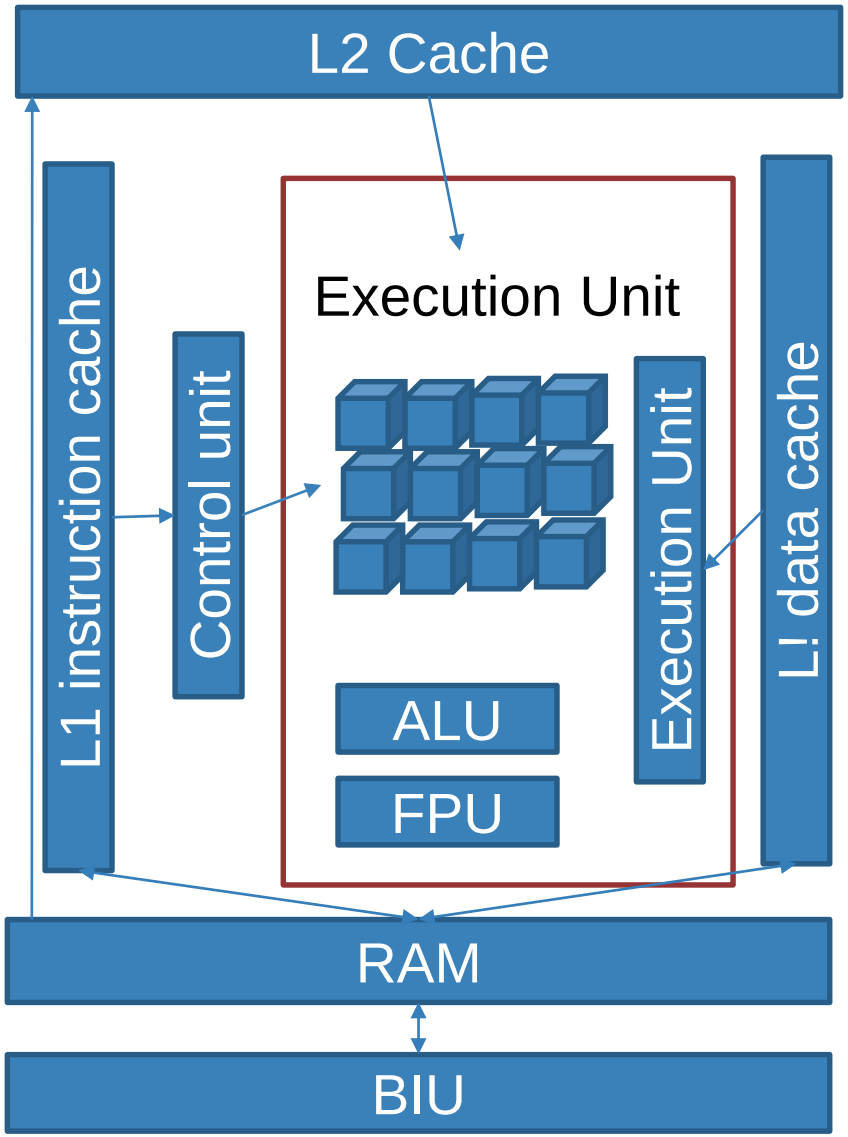


Recent architecture



Modified Harvard architecture

- PROCESS: is a task being run by a computer, often simultaneously with many other tasks each taking turns on a single central processing unit;
- CPU (Central Processing Unit): single- or multi-core chips that can handle multiple processes;
  - CU: Schedules the order of instruction execution
  - ALU: Performs arithmetic and logical calculations
  - FPU: Performs operations on floating point numbers; available in many modern processors



Basic Concepts - Memory

- **REGISTER**: small amount of storage where data resides that is being used right now;
- **CACHE**: small area of fast memory where data resides when it is about to be used and/or has been used recently;
- **MAIN MEMORY** (also called RAM “Random Access Memory”): where data resides when it is being used by a program that is currently running. It may be volatile or not.

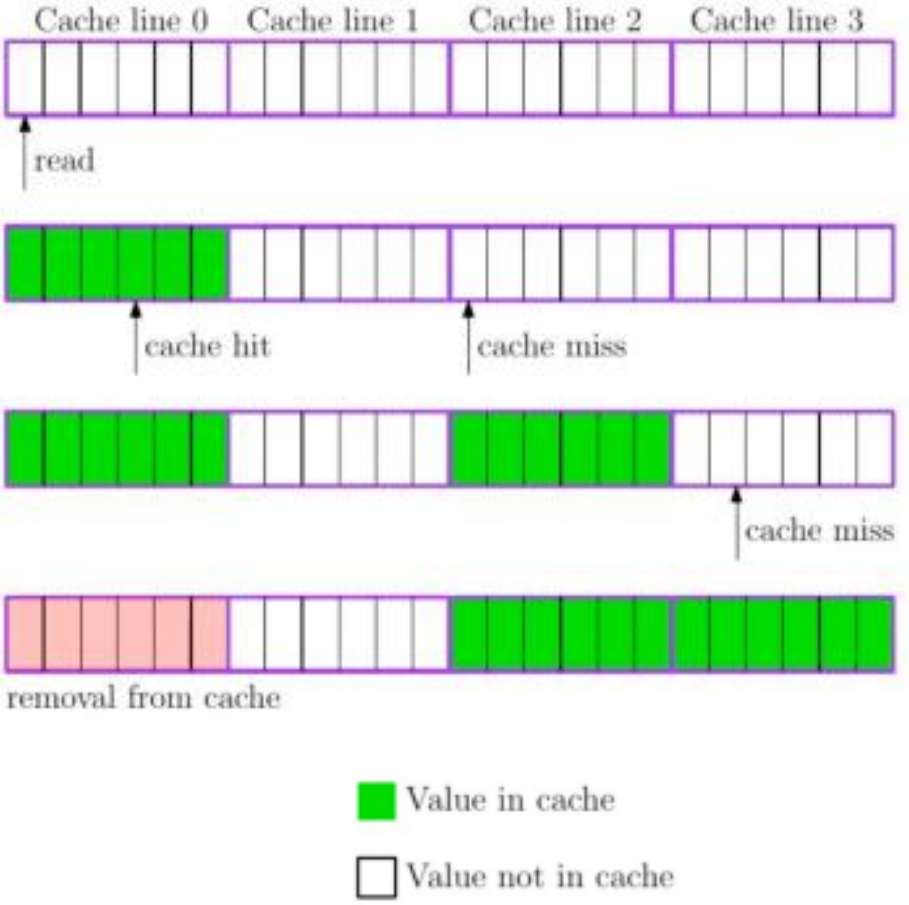
Compute bound vs memory bound

- A problem is compute bound if the performance is dictated by how many arithmetic operations the CPU can perform
  - Monte Carlo methods
- A problem is memory bound if the performance is dictated by the bandwidth of main memory
  - Stencil codes
  - Sparse linear systems
- How many instructions?

```
for(int i=0; i<n-1; i++)
   y[i] = x[i+1]-x[i]
```

Cache

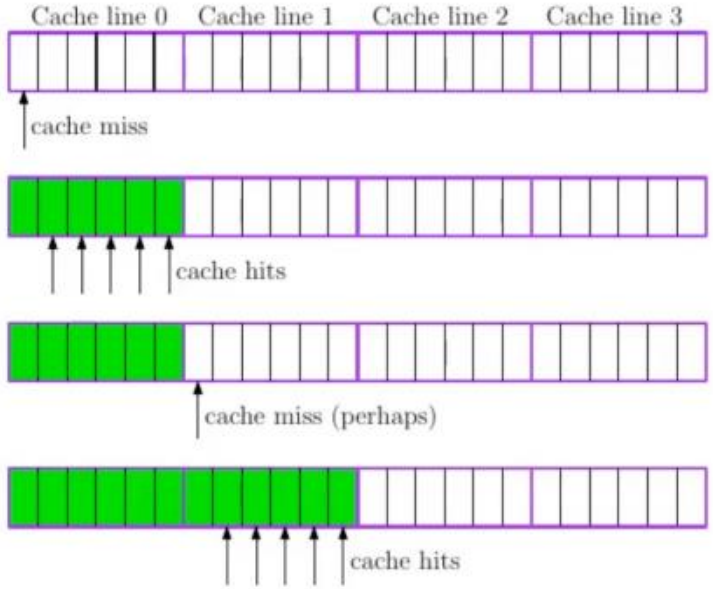
- Important for performance that you understand how caches work
- Transfers data in chunks of fixed size (cache lines)
  - 64-256 bytes (8-32 doubles)
- First read of any byte in a cache line transfers entire line



Caches and memory access

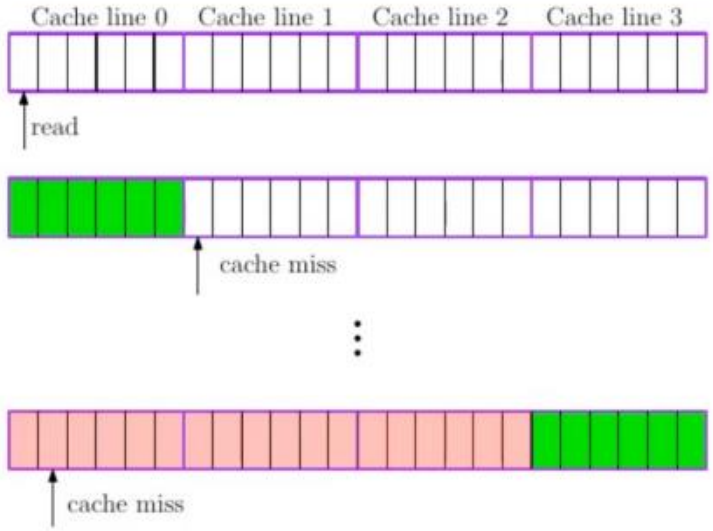
- // Access with stride 1

```
for (int i=0; i<n; i++)
    out[i] = in[i];
```

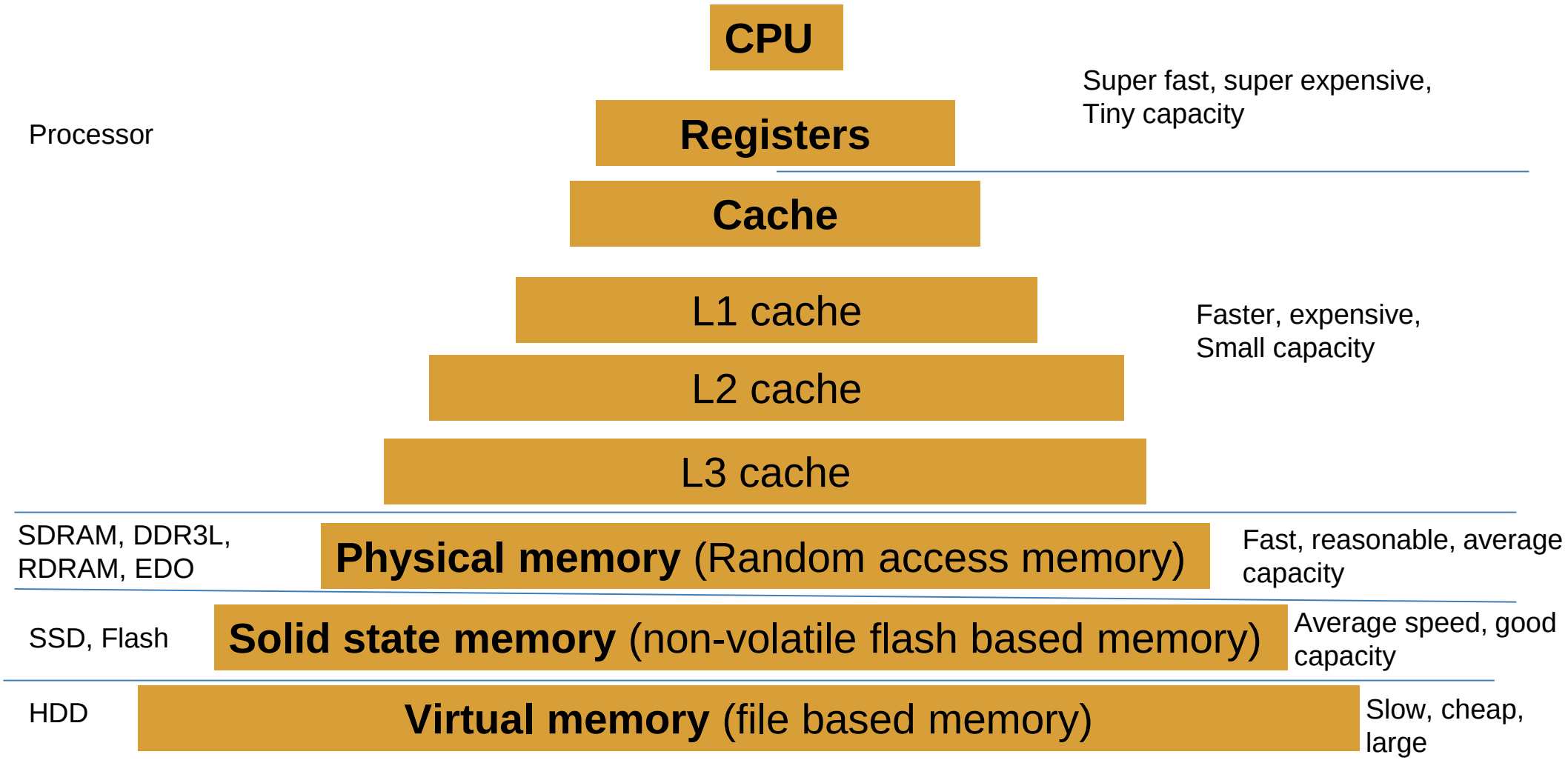


- //Access with stride 6

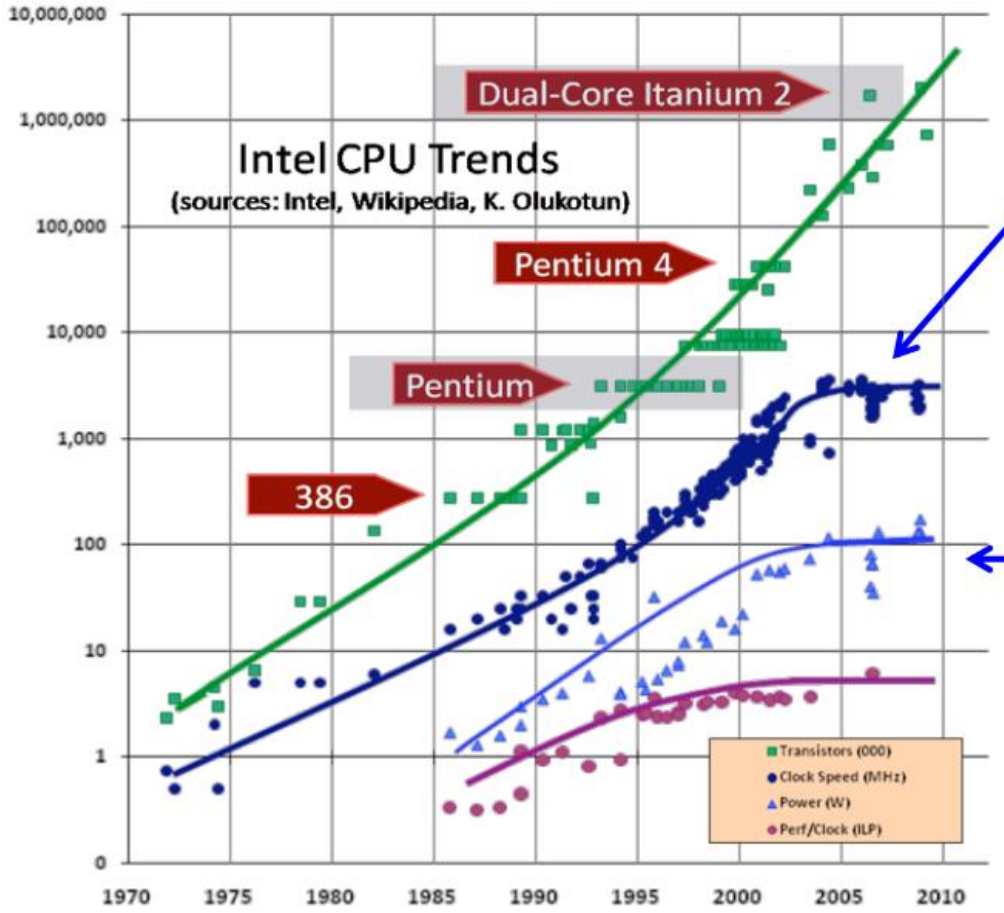
```
for(int j=0; j<6; j++)
  for (int i=0; i<n/6; i++)
    out[j+6*1] = in[j+6*i]
```



Memory hierarchy



Moore's law



"The Free Lunch is over":[1]

Processor clock-speeds are not getting any faster. Will remain ≤ 3 GHz Improved future performance must come from finding and exploiting **parallelism**.

Holding down power consumption is the new imperative

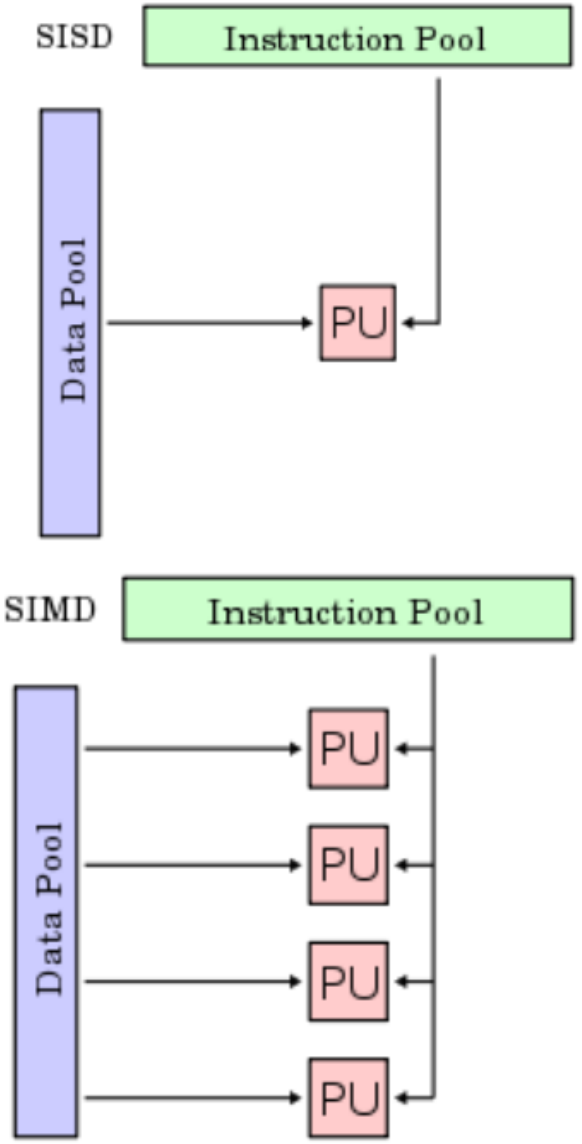
[1] article from Herb Sutter published in 2005. It stated that microprocessor serial-processing speed is reaching a physical limit, which leads to two main consequences:

- processor manufacturers will focus on products that better support multithreading (such as multi-core processors), and
- software developers will be forced to develop massively multithreaded programs as a way to better use such processors.

HPC Architectures

FLYNN'S TAXONOMY:

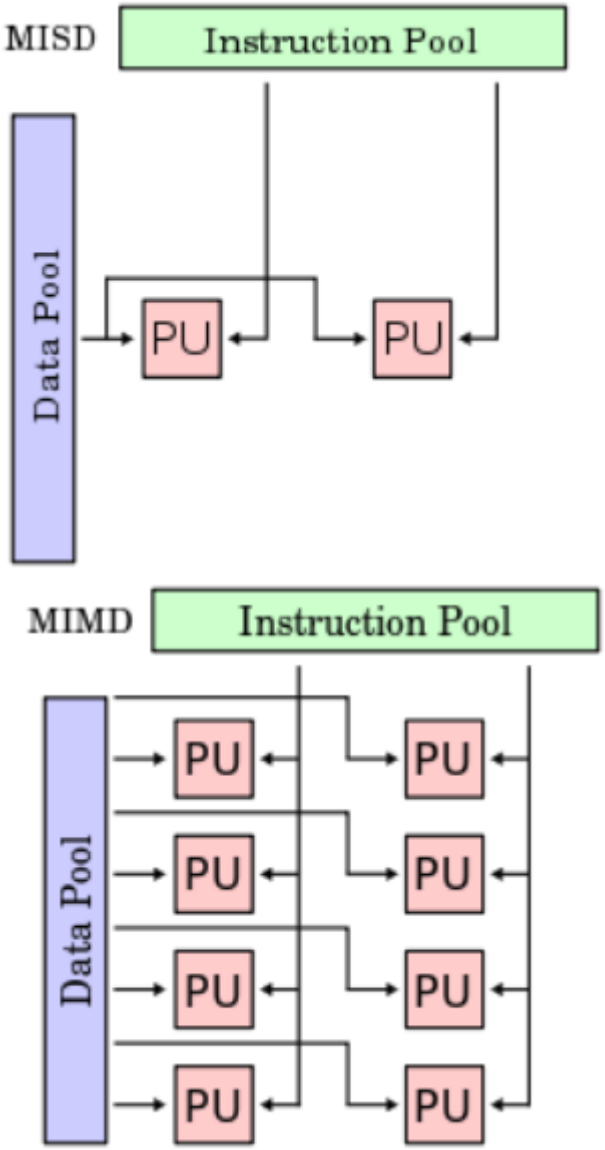
- Single Instruction Single Data (SISD): at any one time only a single instruction is executed, operating on a single data item.
- Single Instruction Multiple Data (SIMD): multiple processors, each operating on its own data item, but they are all executing the same instruction on that data item



HPC Architectures

FLYNN'S TAXONOMY:

- Multiple Instructions Single Data (MISD): All processors do different operations on the same data
- Multiple Instructions Multiple Data (MIMD): multiple CPUs operate on multiple data items, each executing independent instructions
  - Shared Memory
  - Distributed Memory



Process vs. thread

Process	Thread
Instance of a program running on a computer	A dispatchable unit of work within a process
Heavy weight operation	Light weight operation
Every process has its own memory space	Threads use the memory of the process they belong to
Inter process communication is slow as processes have different memory address	Inter thread communication as fast as threads of the same process share the same memory address of the process they belong to
Context switching between process is more expensive	Context switching between threads is less expensive
Processes don’t share the memory with other processes	Threads share the same memory with other threads of the same process

Parallel performance

Parallel Performance: Basic Concepts

FLOP: Floating Point Operation (addition, multiplication, square root, and so on)

- FLOP/sec, FLOP/s, FLOPS: a common performance metric
- FLOPS units:
  - megaFLOPS (MFLOPS) 106
  - gigaFLOPS (GFLOPS) 109 single CPU performance
  - teraFLOPS (TFLOPS) 1012 small-middle scaled supercomputers
  - petaFLOPS (PFLOPS) 1015 we are here right now
  - exaFLOPS (EFLOPS) 1018 the next milestone

The most powerful supercomputers nowadays operate in petaFLOPS range

The TOP500 table shows the 500 most powerful commercially available computer systems worldwide: https://www.top500.org/lists/top500/

Parallel Performance: Basic Concepts

- Peak Performance:
  - The theoretical maximum performance
  - Little indication on how the system will perform in practice, can never be reached.
  - *[# FLOPs/cycle#] × clock rate [Hz] × # cores × # sockets × # nodes*
- Terminology:
  - The cores perform FLOPs.
  - A core can do a certain number of FLOPs every time its internal clock ticks. These ticks are called cycles and measured in Hertz (Hz).
  - A processor contains one or more cores.
  - A socket holds one processor.
  - A node contains one or more sockets.
  - A "chassis" houses one or more nodes

Speedup

- The goal of speedup is to use P processors to solve a problem P times faster
- Relative Speedup:
  - compare the same (parallel) algorithm on one and multiprocessor system

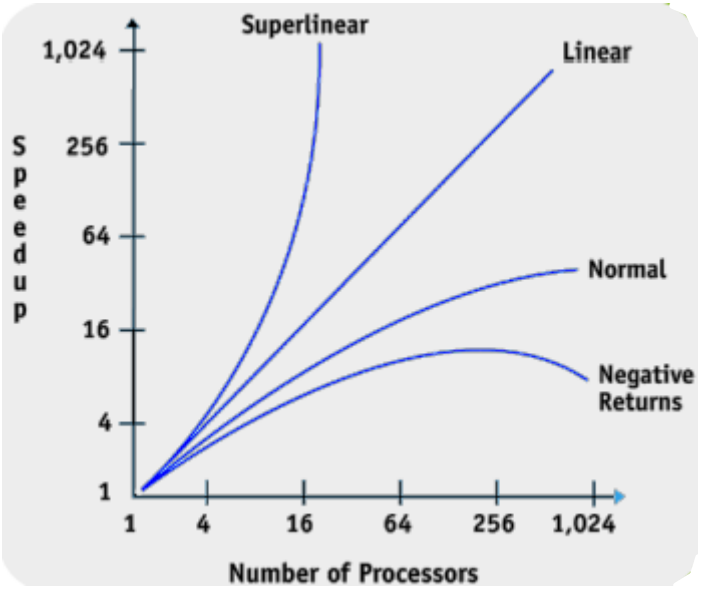
$$S_r(p) = \frac{T_{par}(1)}{T_{par}(p)}$$

- Absolute Speedup:
  - best sequential algorithm for the one processor system is compared to the best parallel algorithm for the multi-processor system

$$S_r(p) = \frac{T_{seq}}{T_{par}(p)}$$

Speedup

- The execution time depends on what the program does!
- A parallel program spends time in:
  - Work
  - Synchronization
  - Communication
  - Extra work (overheads)



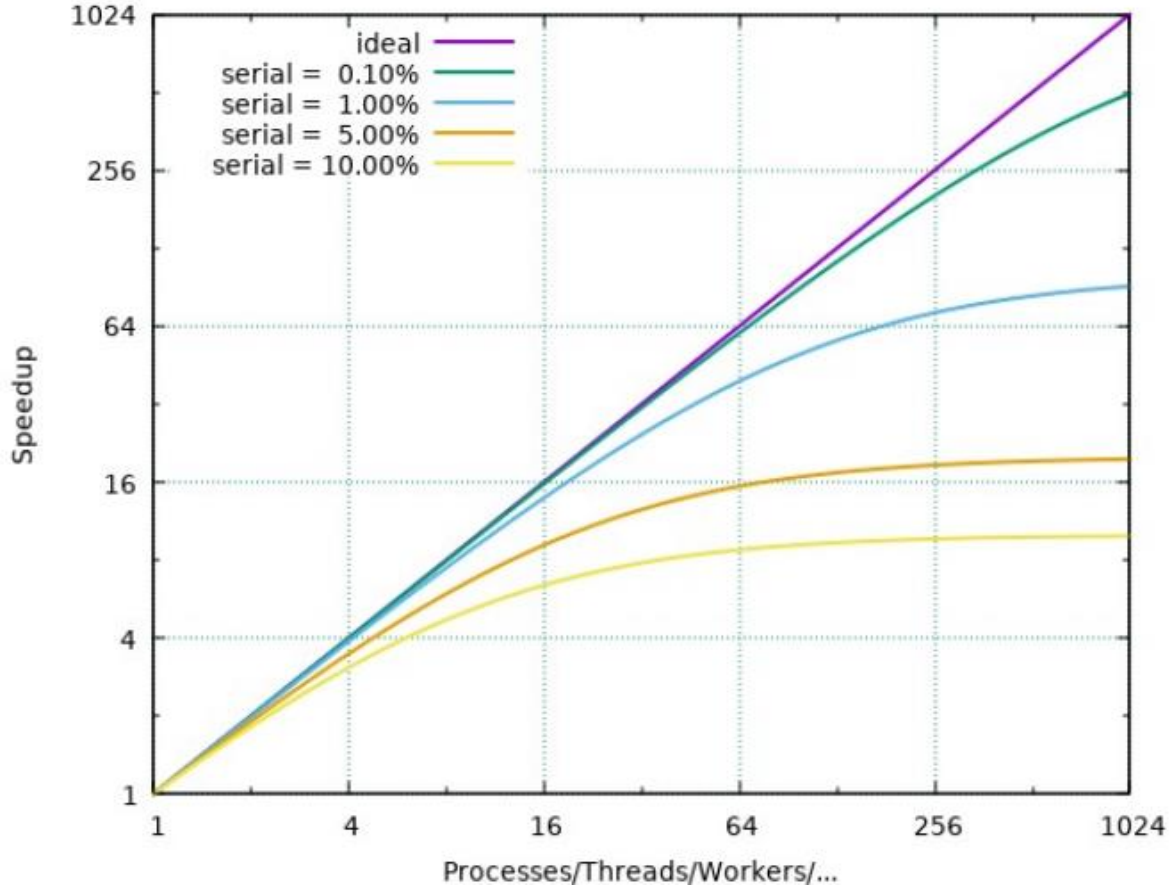
- A program implemented for a parallel machine is likely to do extra work (than a sequential program) even when running in a single processor machine

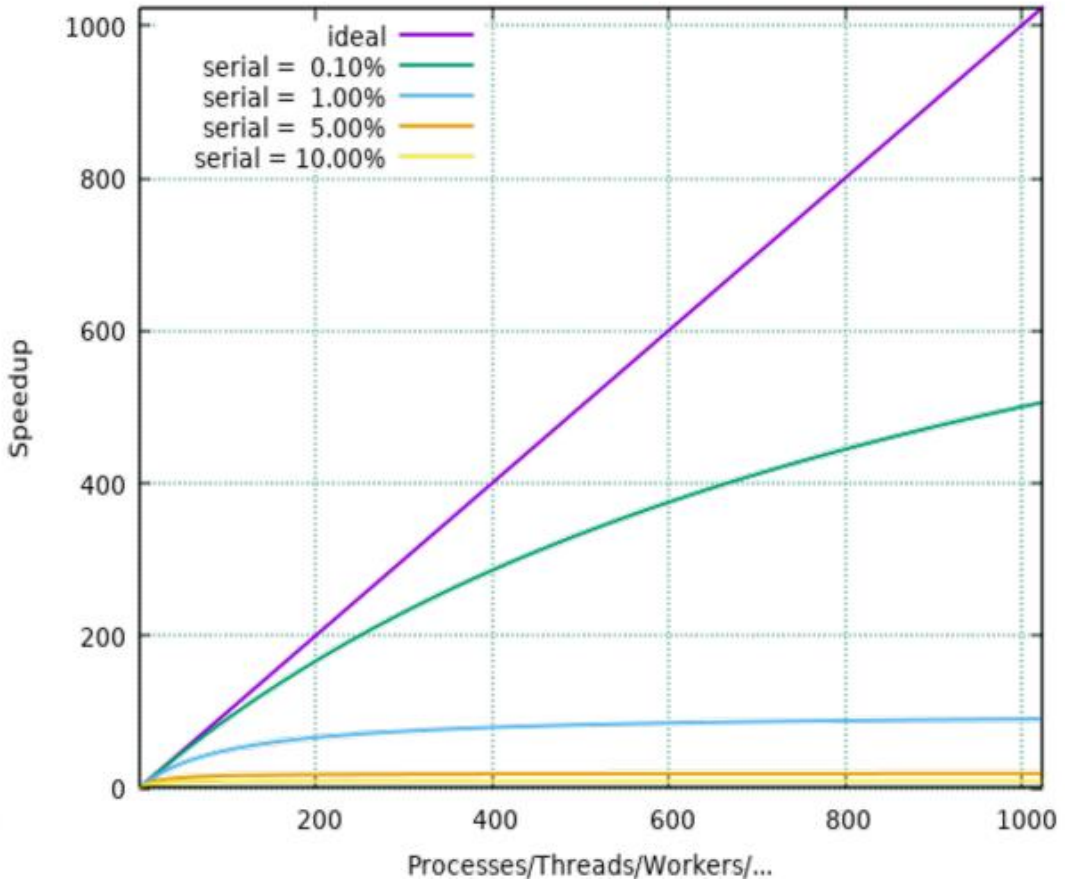
Limits to speedup and Amdahl's law

- If two processors are used, my program will run two times faster?
  - No, the speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program.
- Amdahl's law
  - Let f be the fraction of operations in a computation that must be performed sequentially, where 0 <= f <= 1. The maximum speedup achievable by a parallel computer with p processor:

$$S_{max}(f,p) = \frac{1}{f + \frac{1-f}{p}}$$

Amdahl's law


1024
256
64
16
4
1
Speedup
1
4
16
64
256
1024
Processes/Threads/Workers/...
ideal
serial = 0.10%
serial = 1.00%
serial = 5.00%
serial = 10.00%


1000
800
600
400
200
0
Speedup
200
400
600
800
1000
Processes/Threads/Workers/...
ideal
serial = 0.10%
serial = 1.00%
serial = 5.00%
serial = 10.00%

Limits to speedup

- Serial sections limit the parallel performance
- Parallel Overhead: the time required to coordinate parallel tasks and communicate information between processors
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - cost of extra (redundant) computation

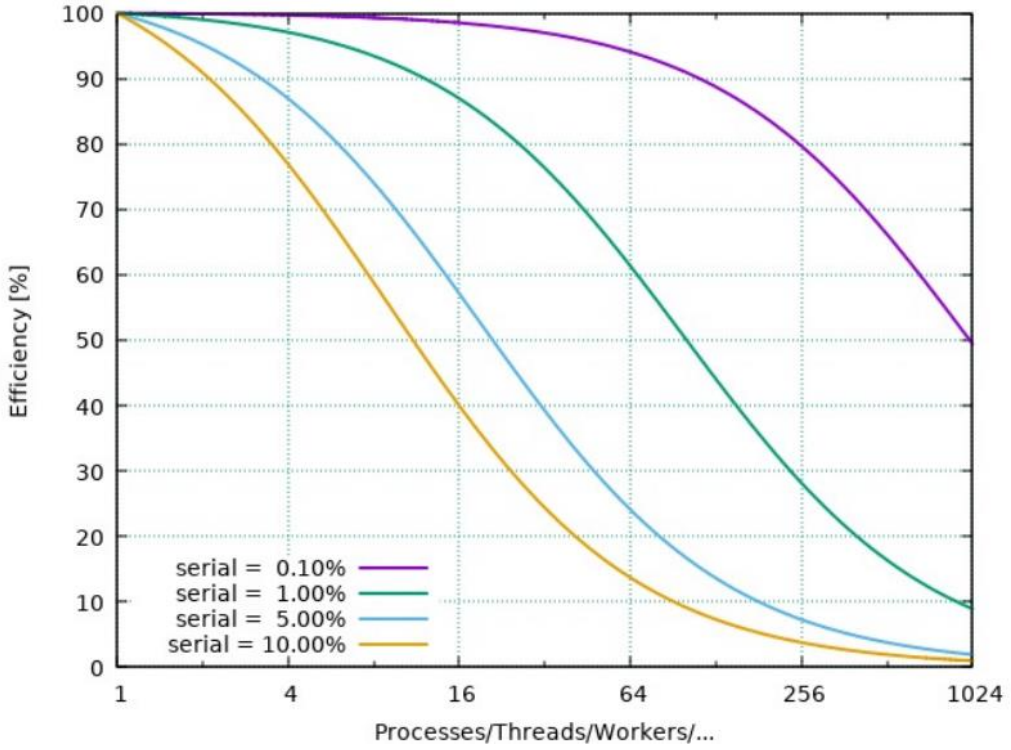
…the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude. — Gene Amdahl

Efficicency

- A measure of how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization
- E(p) <= 1
- E(p) = 1 if there is linear speedup

$$E_r(p) = \frac{S_r(p)}{p}$$

$$E_a(p) = \frac{S_a(p)}{p}$$



Scalability

- The performance when the size of the problem or the number of processors changed.
- Assume a problem of size N using P processors takes time T ▫
  - Strong Scaling:
    - *As N remains fixed, kP processors solve the problem in T/k time.*
  - Weak Scaling:
    - *For a problem of size kN, kP processors solve the problem in time T.*

Communication

- For some problems, increasing the number of processors will:
  - Decrease execution time attributable to computation
  - May increase execution time attributable to communication
- Time required for communication is dependent upon system communication bandwidth parameters.
- The time (T) required to send W words between any two processors is given by

$$T = L + \frac{W}{B}$$

- Where
  - L = Latency: the fastest time in which a single byte can be sent
  - B = Bandwidth: how much data can be sent per unit time

Load imbalance

- Amdahl's and Gustafson's laws assume that all processors are equally busy
  - Not always true

Core	1	2	3	4	Total
# tasks	6	1	3	2	12

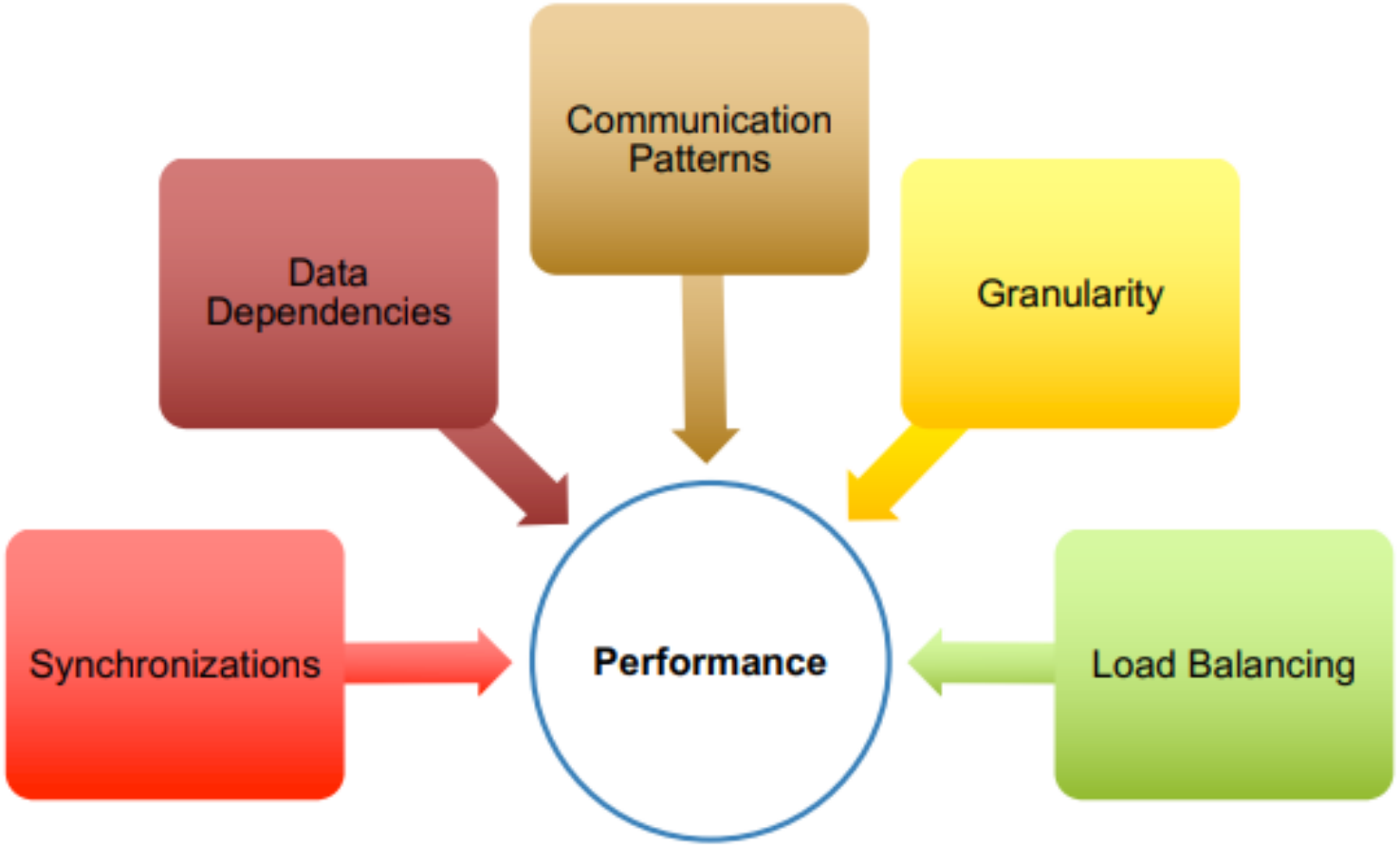
- Takes 6 minutes to complete as other cores waiting for core 1 to finish
- If work distributed evenly -> 3 minutes

- Make best use of processors at hand before increasing number of processors

Quantifying load imbalance

- Define Load Imbalance Factor
  - *LIF = maximum load/ average load*
  - For perfectly balanced problems LIF = 1.0, as expected
  - Usually; LIF > 1.0
  - LIF tell you how much faster your calculation could be with a balanced load
- Core tasks
  - LIF = 6/3 = 2
  - Initial time = 6 mins
  - Best time = LIF/2 = 3 mins

Parallel performance factors



THANK YOU FOR YOUR ATTENTION

www.prace-ri.eu