INTRODUCTION TO PARALLEL PROGRAMMING

Caspar van Leeuwen
Carlos Teijeiro Barjas

SURF SARA

Goals

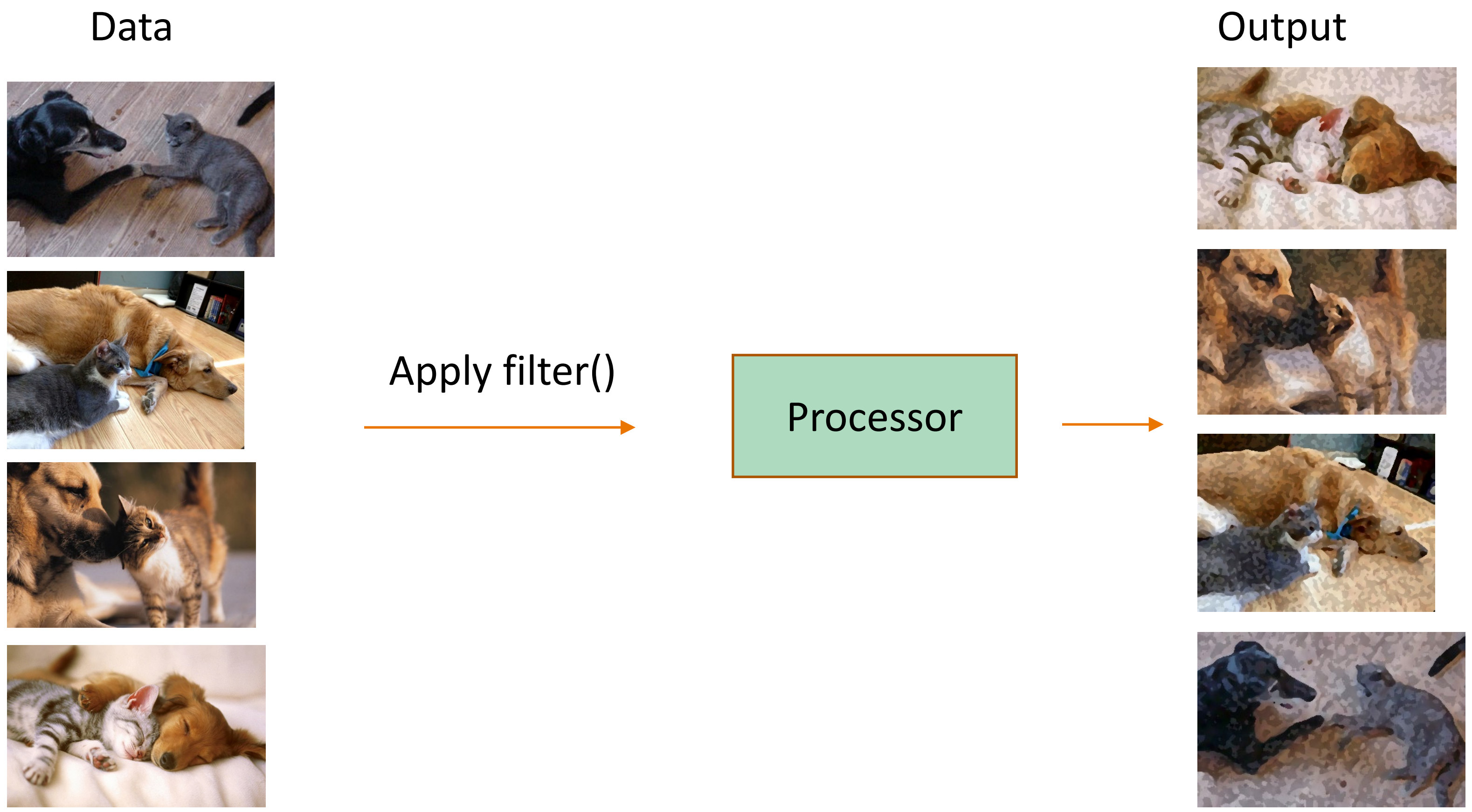
- The goal is to understand:
  - Merits and limits of parallel computing
  - Parallel programming models (task / data parallelism)
  - Differences between shared and distributed memory systems

Content

- What is parallel computing?
- Parallel programming models
- Different types of systems

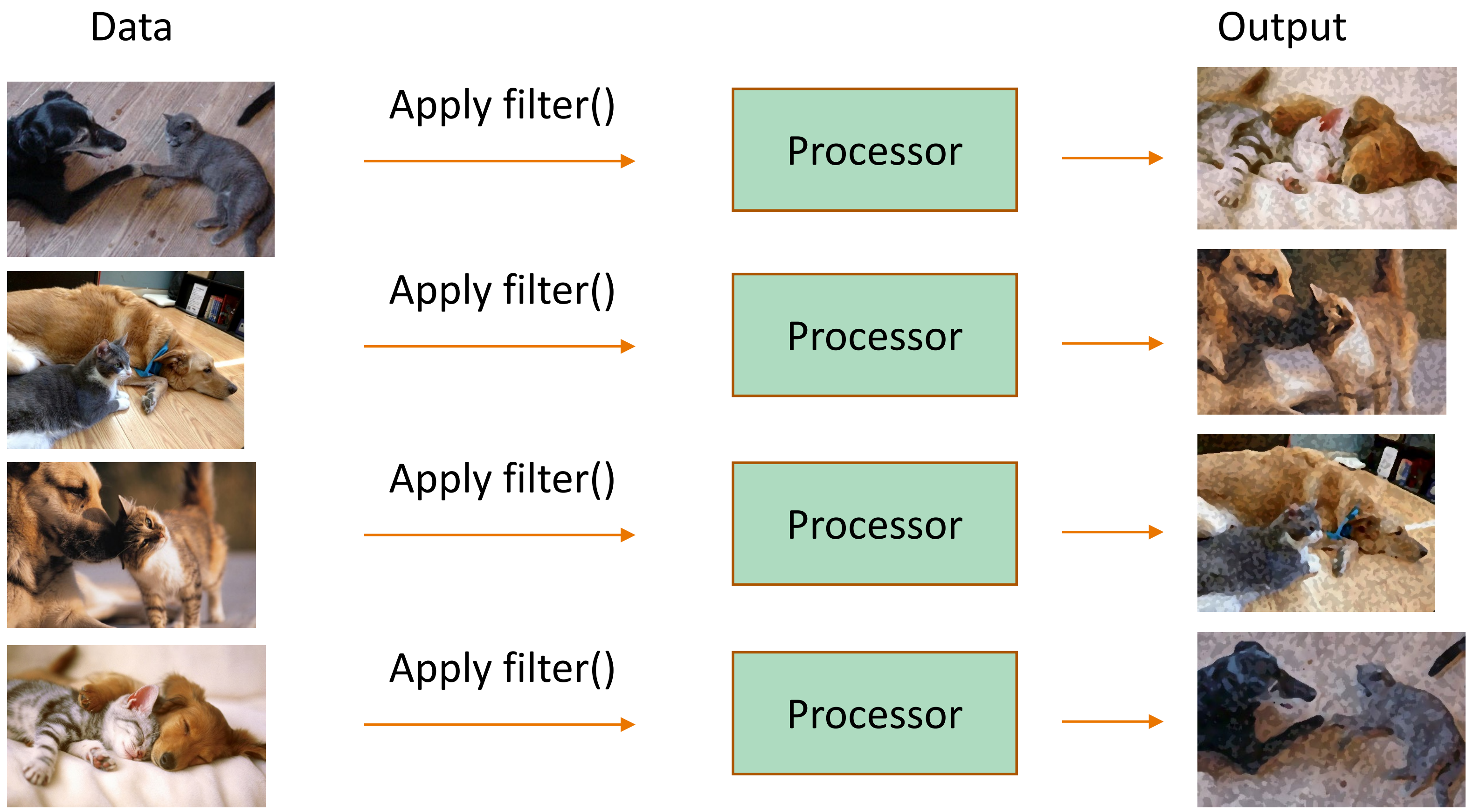
What is parallel computing?

- Parallel computing
  - Multiple processors or computers working on a single computational problem



What is parallel computing?

- Parallel computing
  - Multiple processors or computers working on a single computational problem

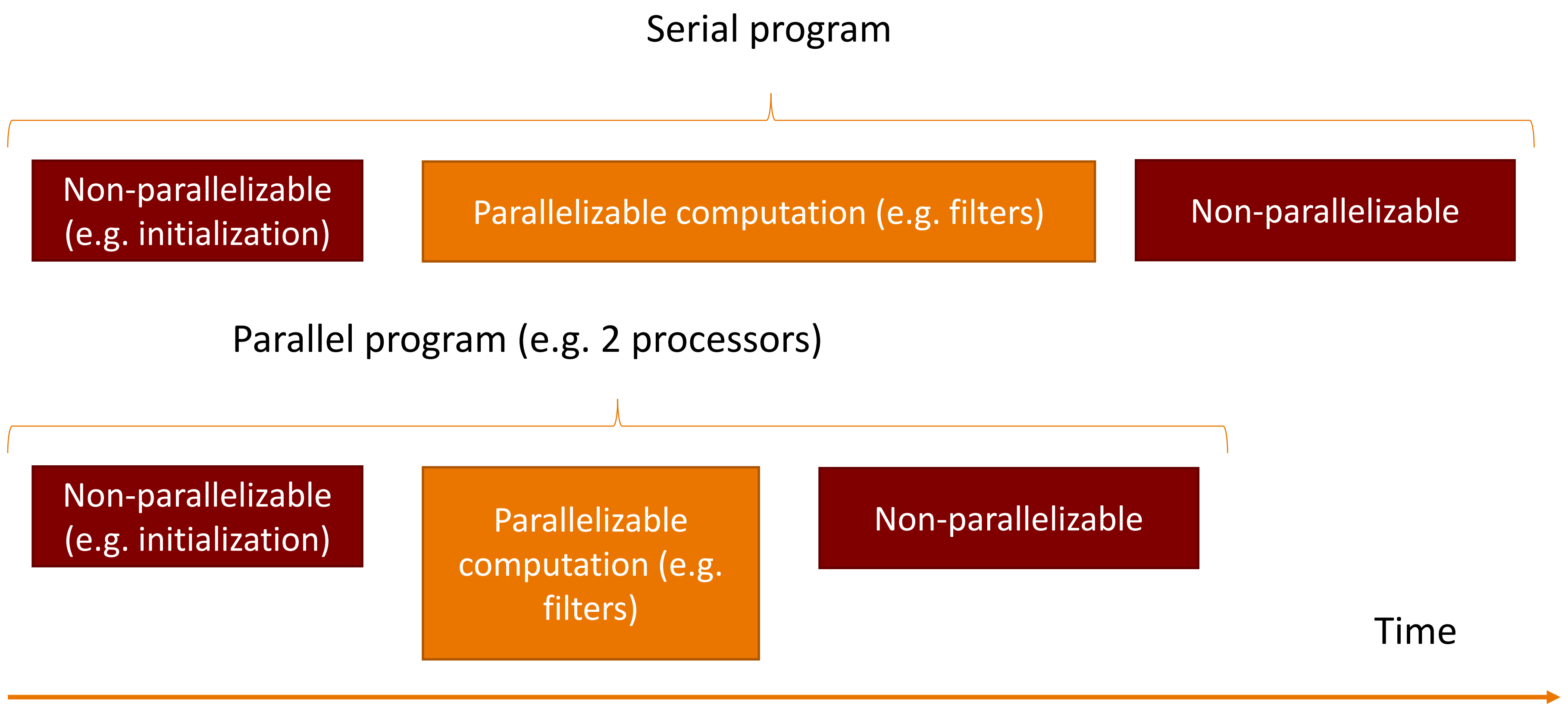


Parallel computing: benefits

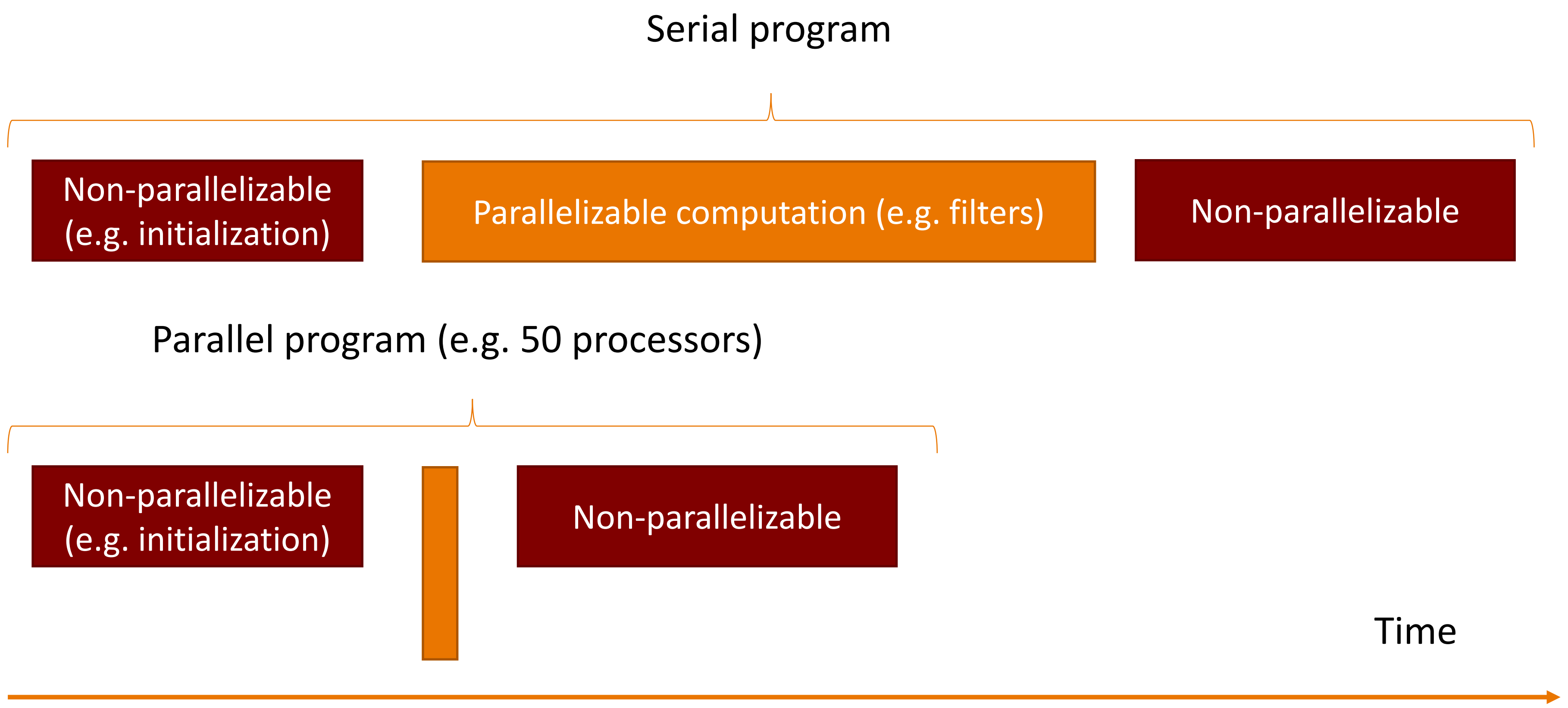
- Benefits
  - Solve computationally intensive problems (speedup)
  - Solve problems that don’t fit a single memory (multiple computers)

- Requirements:
  - Problem should be divisible in smaller tasks

Real problem: often partly parallelizable


Serial program
Non-parallelizable (e.g. initialization)
Parallelizable computation (e.g. filters)
Non-parallelizable
Parallel program (e.g. 2 processors)
Non-parallelizable (e.g. initialization)
Parallelizable computation (e.g. filters)
Non-parallelizable
Time

Real problem: often partly parallelizable

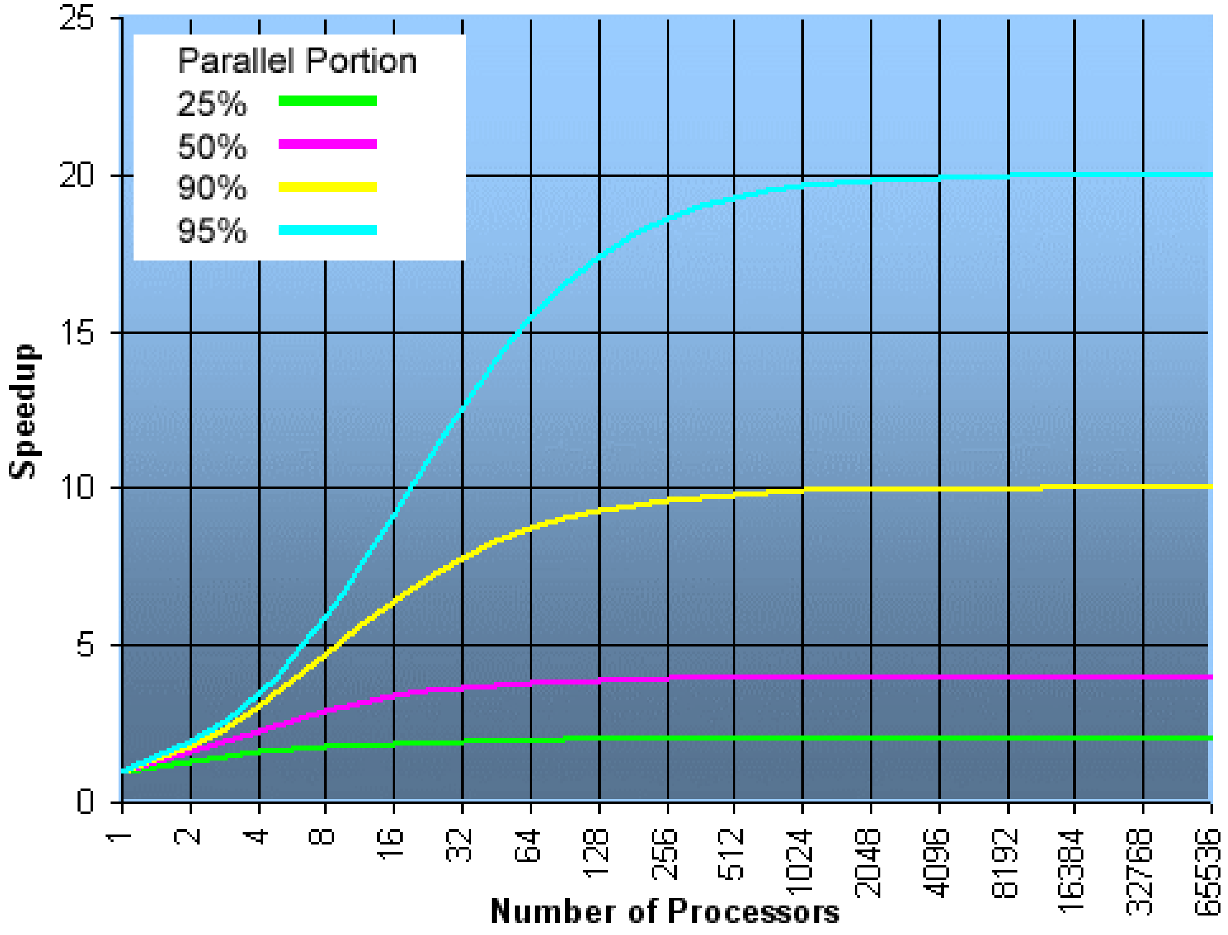

Serial program
Non-parallelizable (e.g. initialization)
Parallelizable computation (e.g. filters)
Non-parallelizable
Parallel program (e.g. 50 processors)
Non-parallelizable (e.g. initialization)
Non-parallelizable
Time

Amdahl's Law

$$Speedup = \frac{1}{(1-p)+\frac{p}{N}}$$

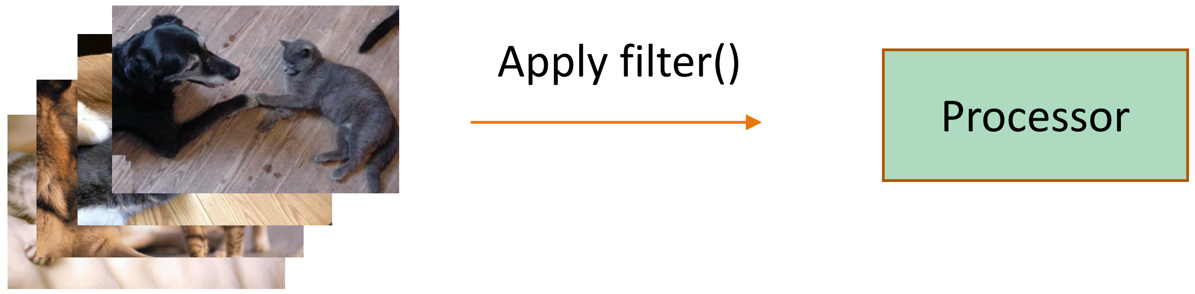
N = number of processors

p = parallel fraction



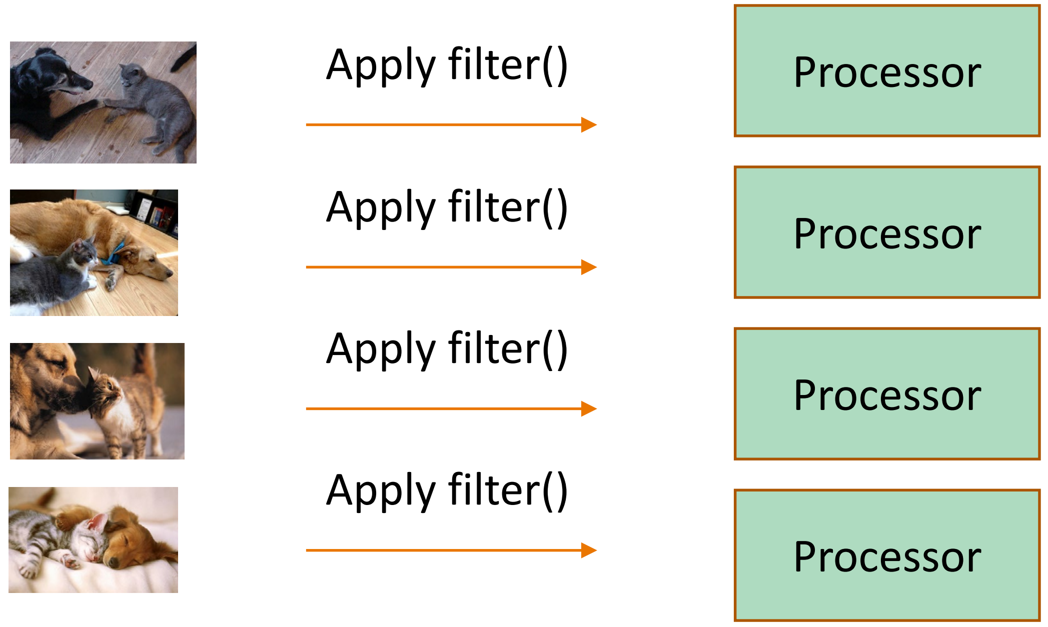
Amdahl’s law describes *strong scaling*

- Strong scaling
  - Variation of solution time with #processors for fixed *total* problem size
  - Possibilities to run the same problem in shorter time



Amdahl’s law describes *strong scaling*

- Strong scaling
  - Variation of solution time with #processors for fixed *total* problem size
  - Possibility to run the same problem in shorter time



Amdahl’s law is theoretical…

- … and reality is often worse!

- Further issues to tackle
  - Load balancing
  - Process scheduling
  - Communication overhead
  - File Input/Output
  - …

Do we have hope…???

Gustafson's Law

$$Speedup = 1 + p \cdot (N - 1)$$

N = number of processors

p = parallel fraction

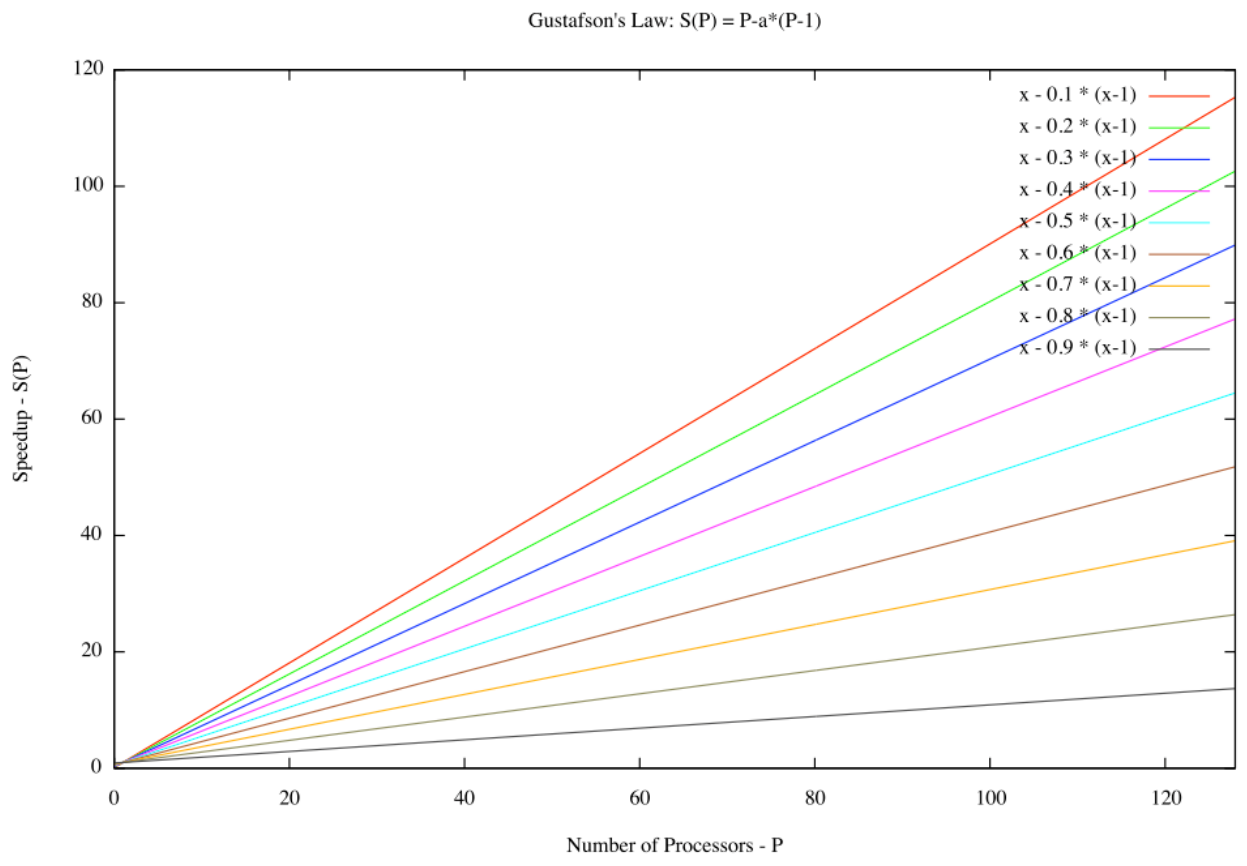
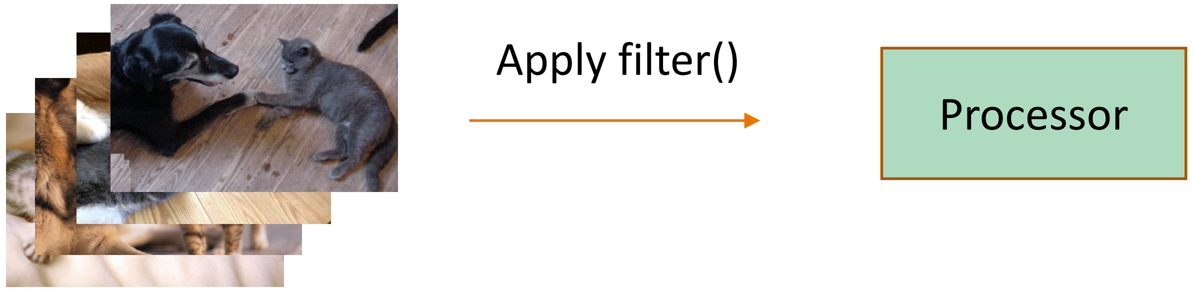


Image source: https://computing.llnl.gov/tutorials/parallel_comp

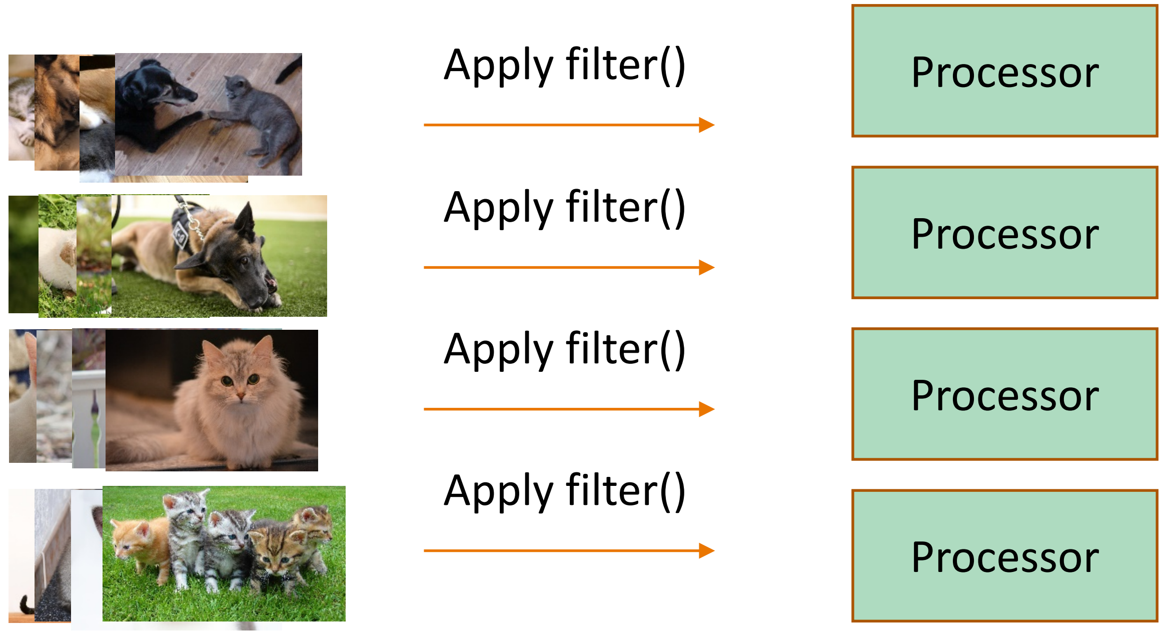
Gustafson's law describes *weak scaling*

- Weak scaling
  - Variation of solution time with #processors for fixed problem size*per processor*
  - Possibilities to run a bigger problem in the same time



Gustafson's law describes *weak scaling*

- Weak scaling
  - Variation of solution time with #processors for fixed problem size*per processor*
  - Possibilities to run a bigger problem in the same time



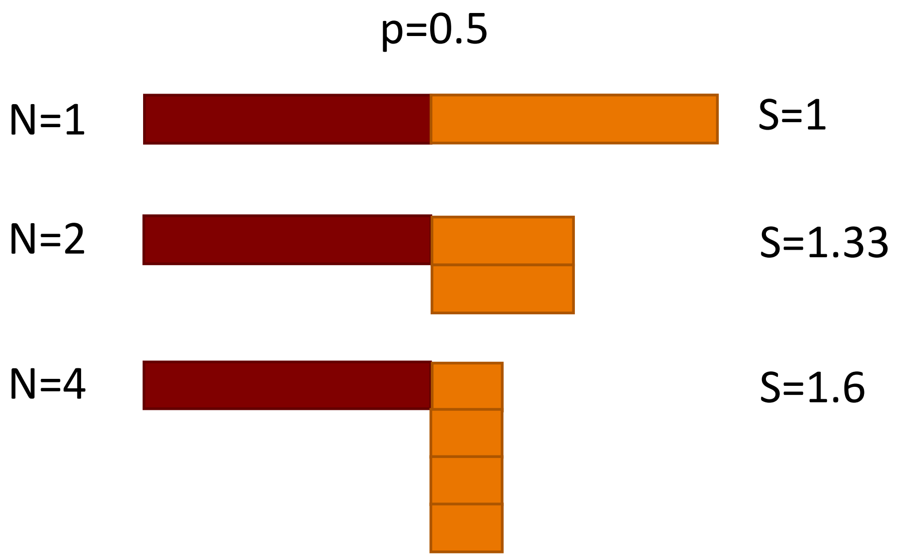
Strong scaling vs. weak scaling

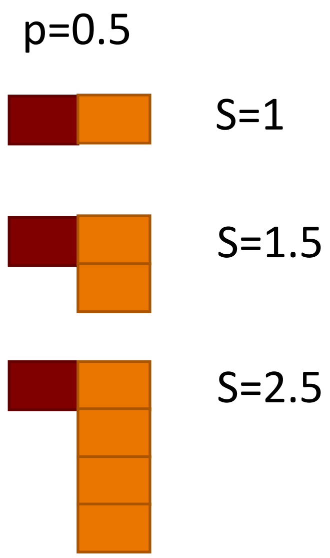
- Strong scaling: fixed work

$$Speedup = \frac{1}{(1-p)+\frac{p}{N}}$$

- Weak scaling: fixed work per processor

$$Speedup = 1 + p \cdot (N-1)$$





Weak scaling is often the most relevant to HPC computations

- Some examples
  - Physics: ”I can only run my fluid simulation a small domain / low resolution on my local PC”
  - Chemistry: “I can simulate a small molecule on my PC, but I want to simulate a big one”
- Common background
  - A big system / molecule increases the *total* work
  - Distributing larger work over multiple processors keeps the work *per processor* constant
  - … this is weak scaling!

So now how can we program in parallel?

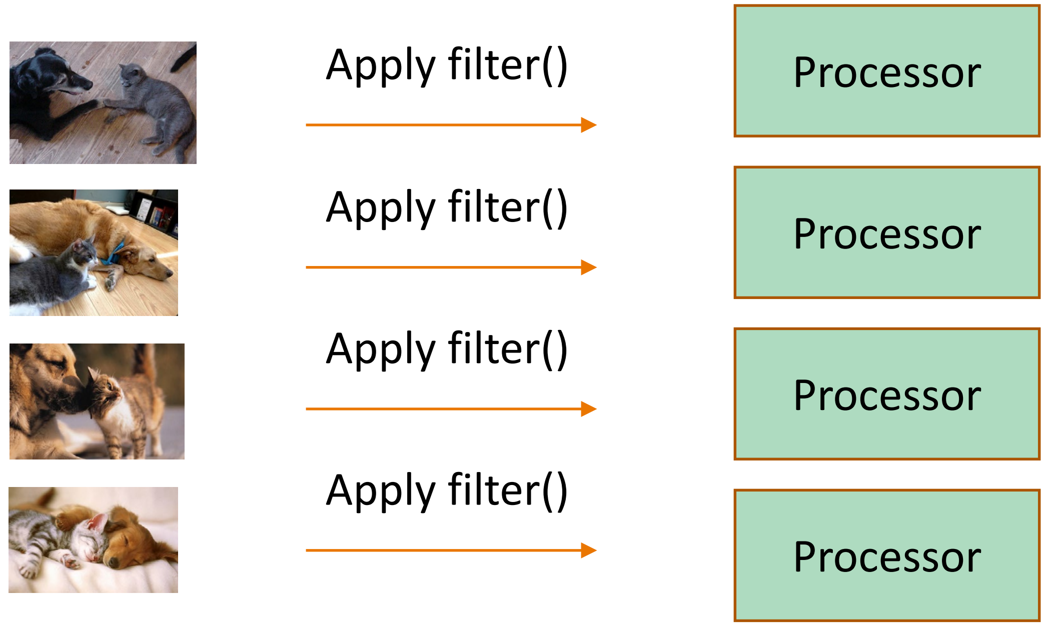
Parallel programming models

- “How do I parallelize my code?”

- Two well-known programming models
  - Data parallelism
  - Task parallelism

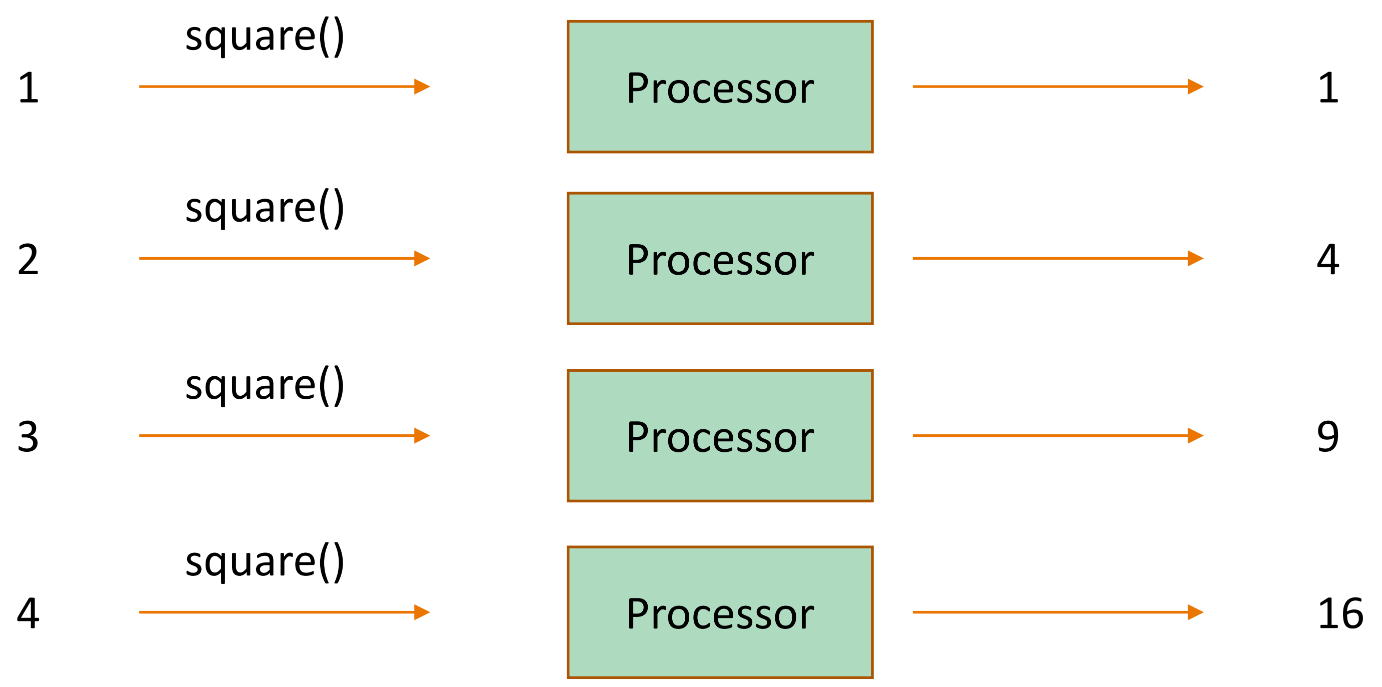
Parallel programming models: data parallelism

- Each processor performs the same task on different data



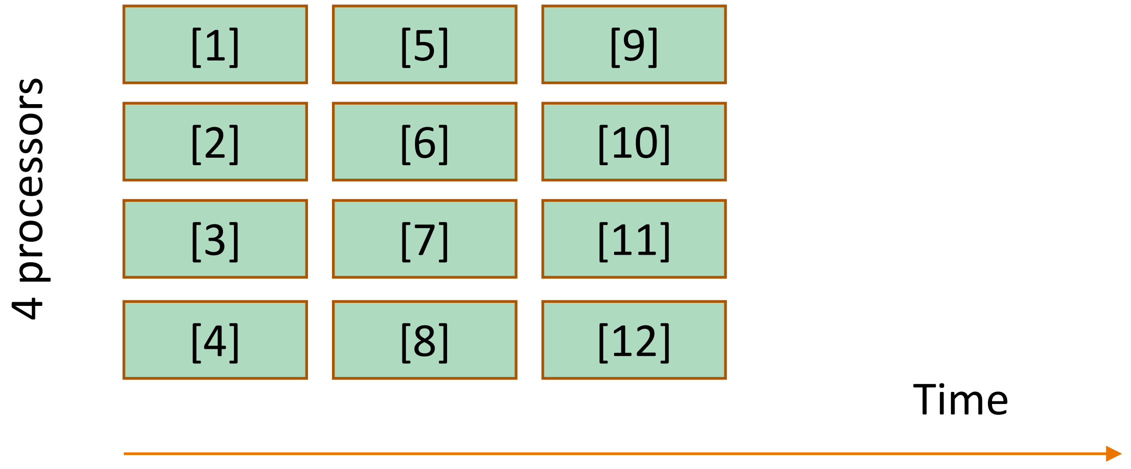
Parallel programming models: data parallelism

- Each processor performs the same task on different data



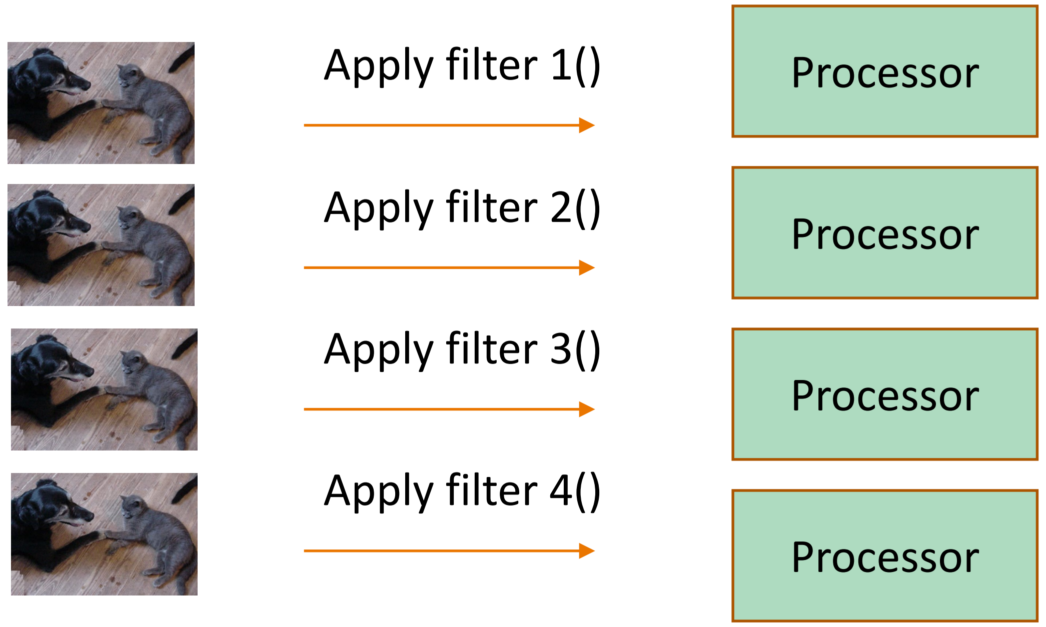
Parallel programming models: data parallelism

- Amount of parallelization depends on input data size
- Load balancing may be relatively easy
  - Same task on each data element
  - Approximately same time per element



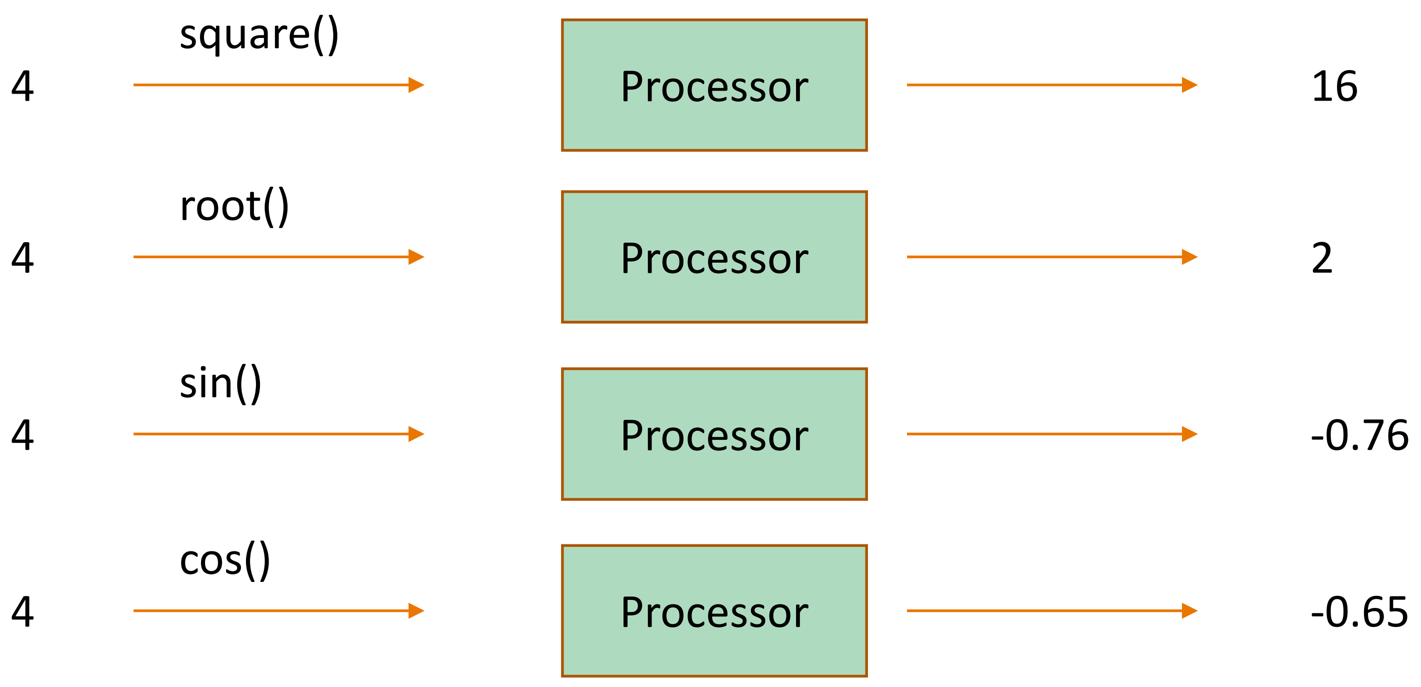
Parallel programming models: task parallelism

- Each processor performs a different task on the same data



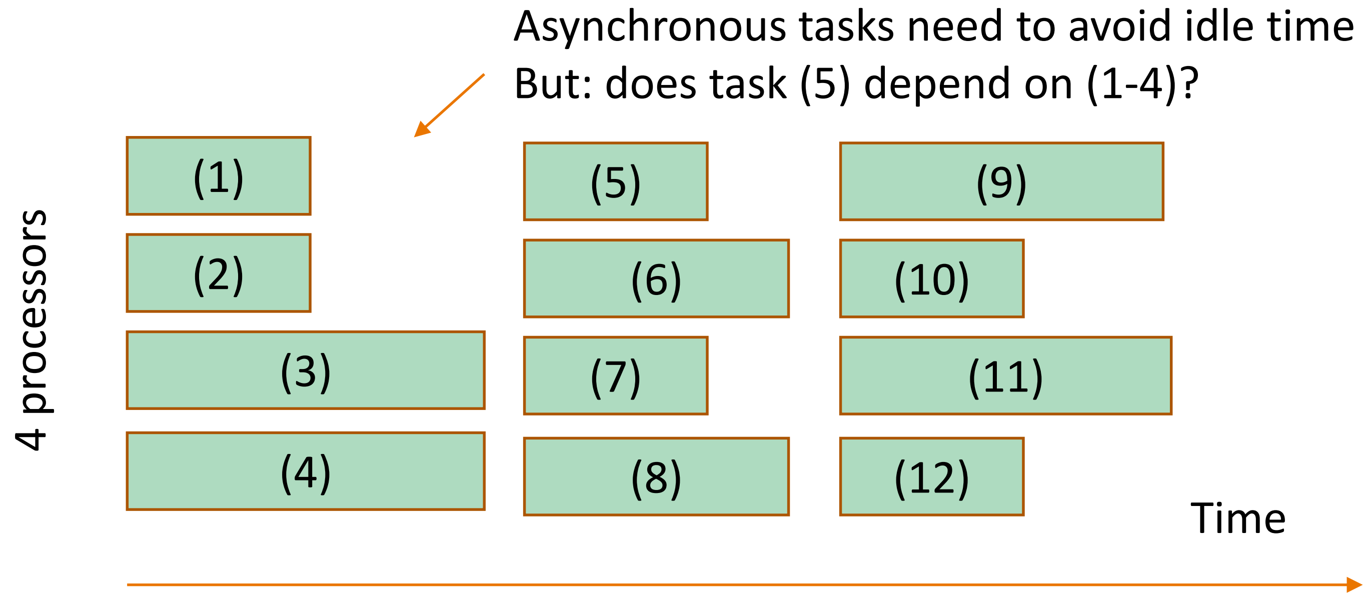
Parallel programming models: task parallelism

- Each processor performs a different task on the same data



Parallel programming models: task parallelism

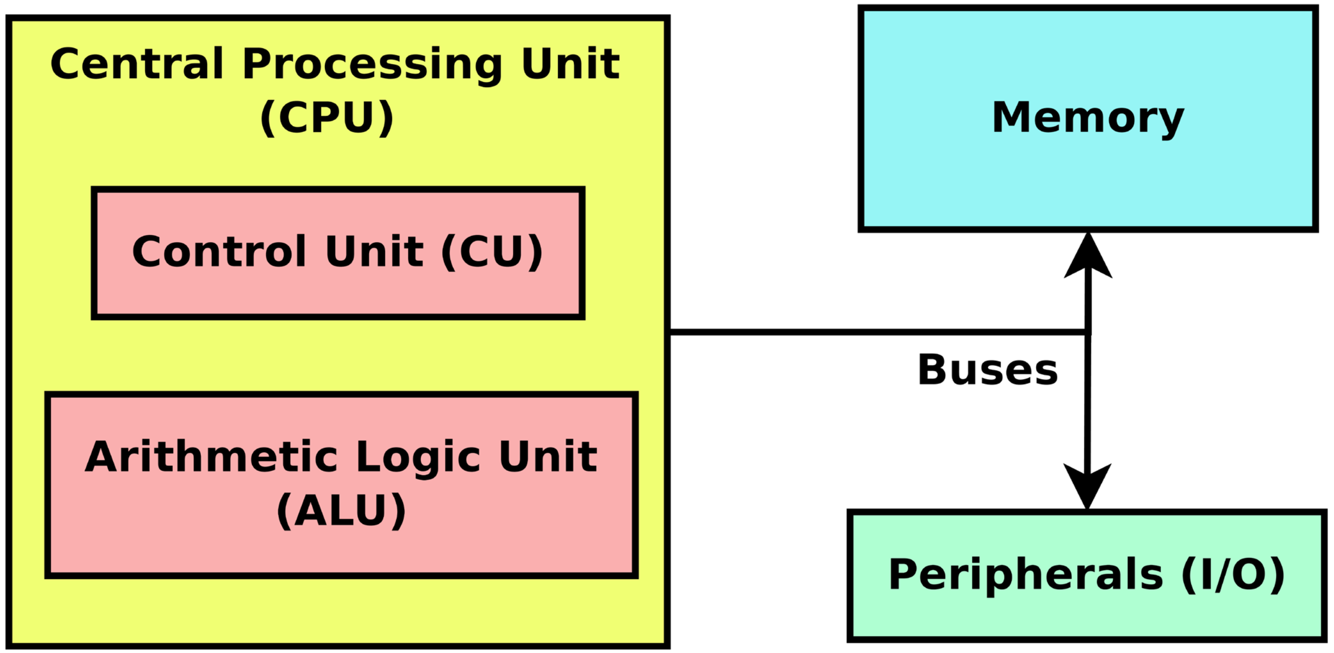
- Amount of parallelization depends on the number of tasks
- Load balancing can be very difficult
  - Heterogeneous tasks may be executed over the same data
  - Each task may take a very different amount of time


Asynchronous tasks need to avoid idle time
But: does task (5) depend on (1-4)?
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
4 processors
Time

Theory is essential to understand concepts…

but parallel programming also requires understanding of the systems!

A computer is…



A memory is…

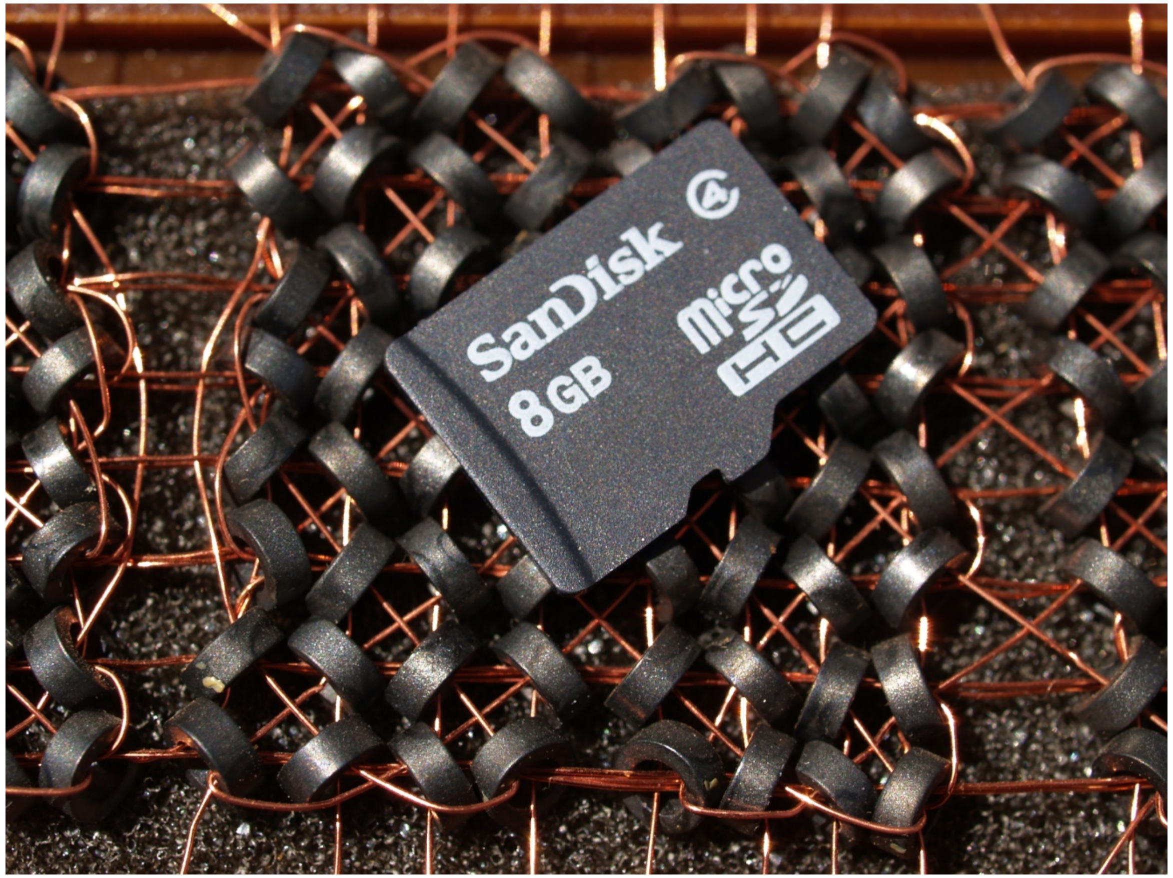
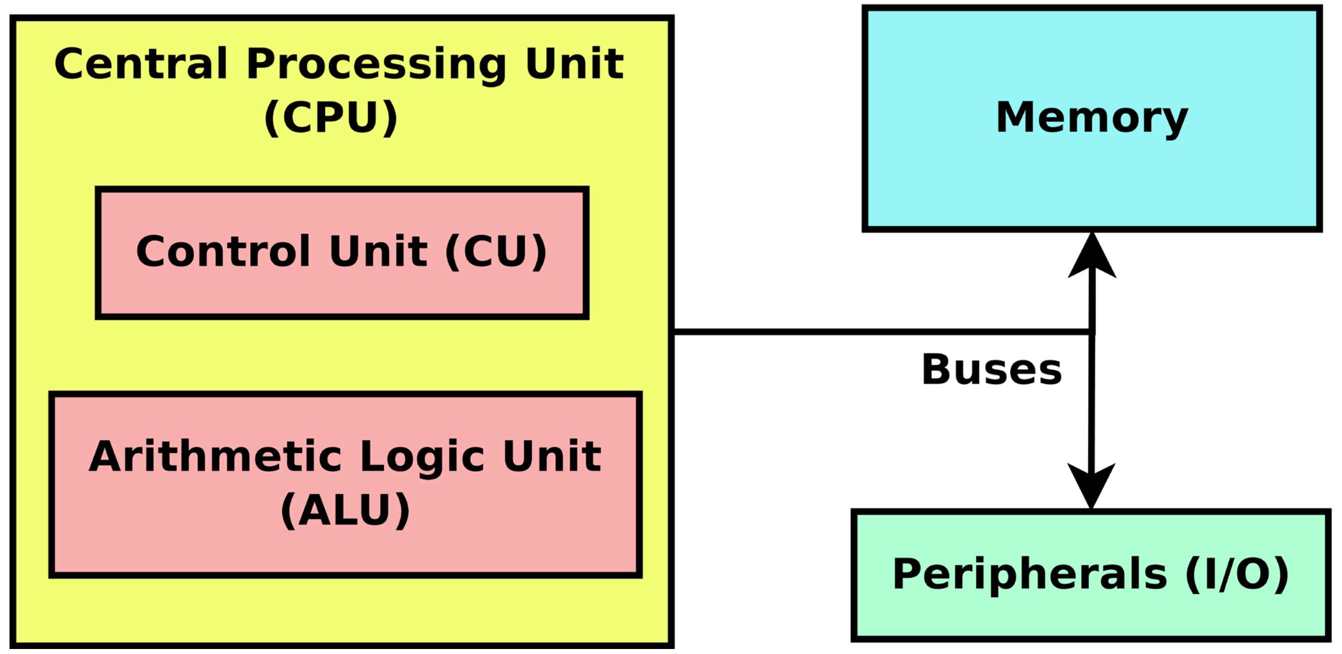


Image source: https://upload.wikimedia.org/wikipedia/commons/c/c0/8 bytes vs. 8Gbytes.jpg

A computer is…



A CPU is…

Example of the MIPS architecture: control unit lines in orange, data lines in black

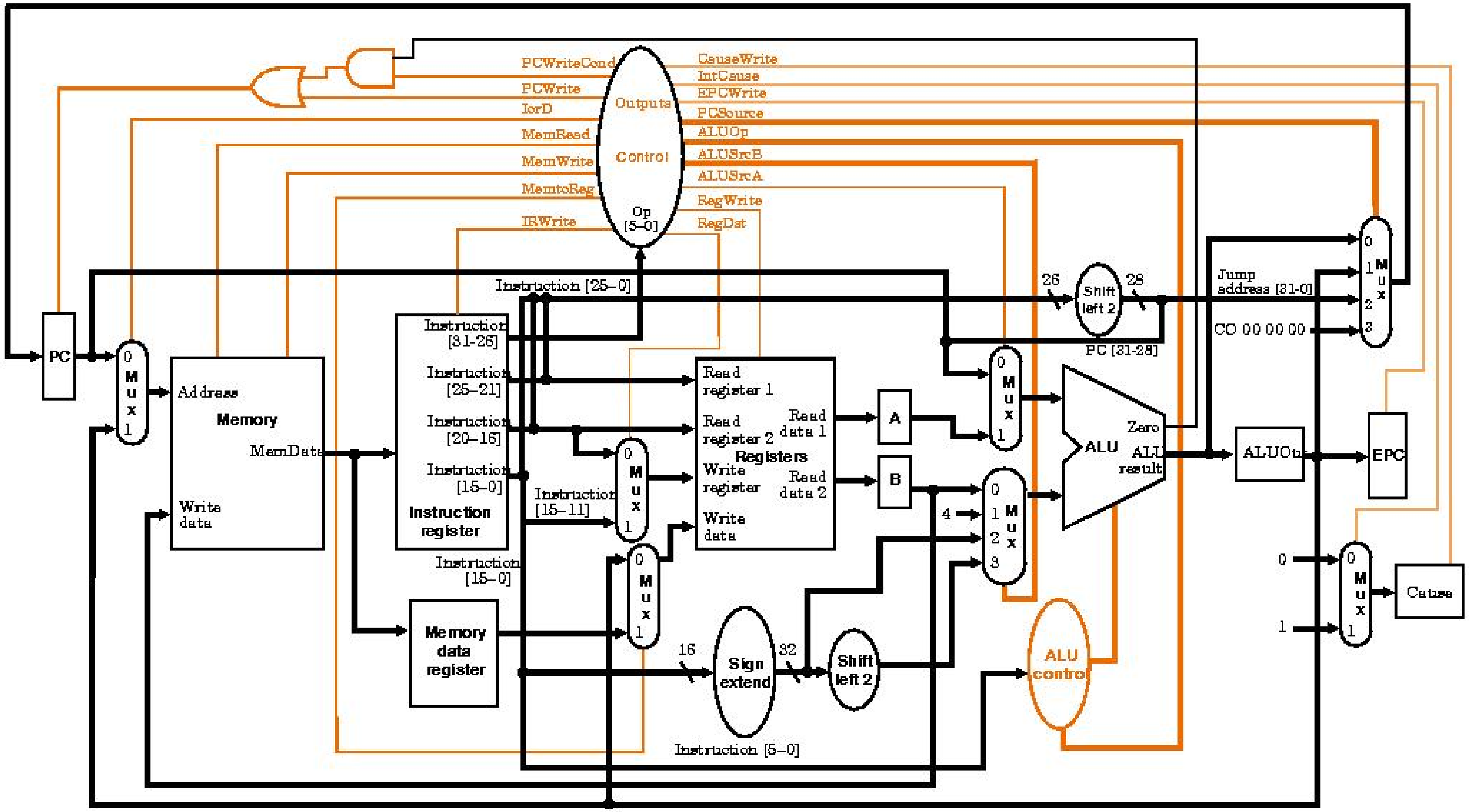
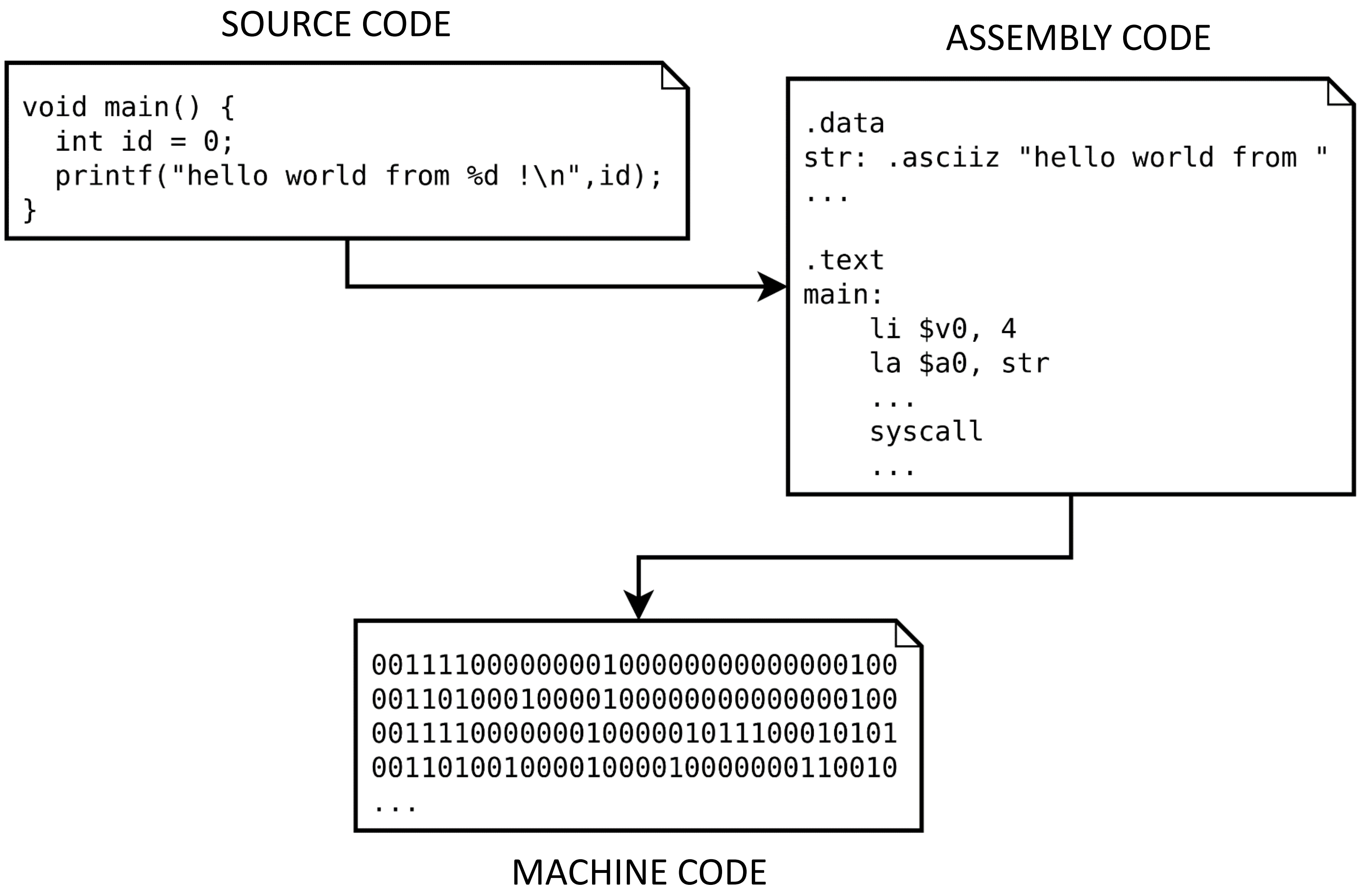


Image source: http://people.cs.pitt.edu/˜don/coe1502/current/Unit4a/Unit4a.html (last visited: 2018)

A program becomes a process…



```
void main() {
  int id = 0;
  printf("hello world from %d !\n",id);
}
```

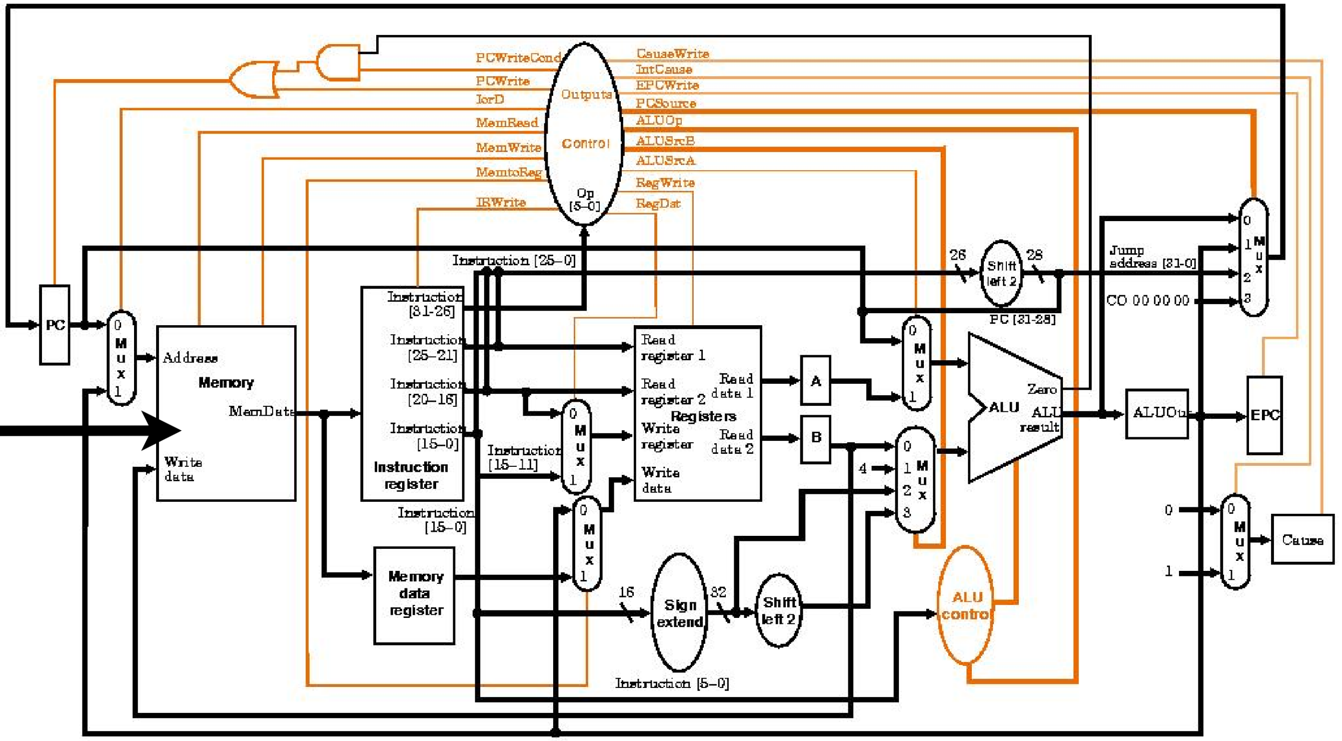
```
.data
str: .asciiz "hello world from "
...

.text
main:
    li $v0, 4
    la $a0, str
    ...
    syscall
    ...
```

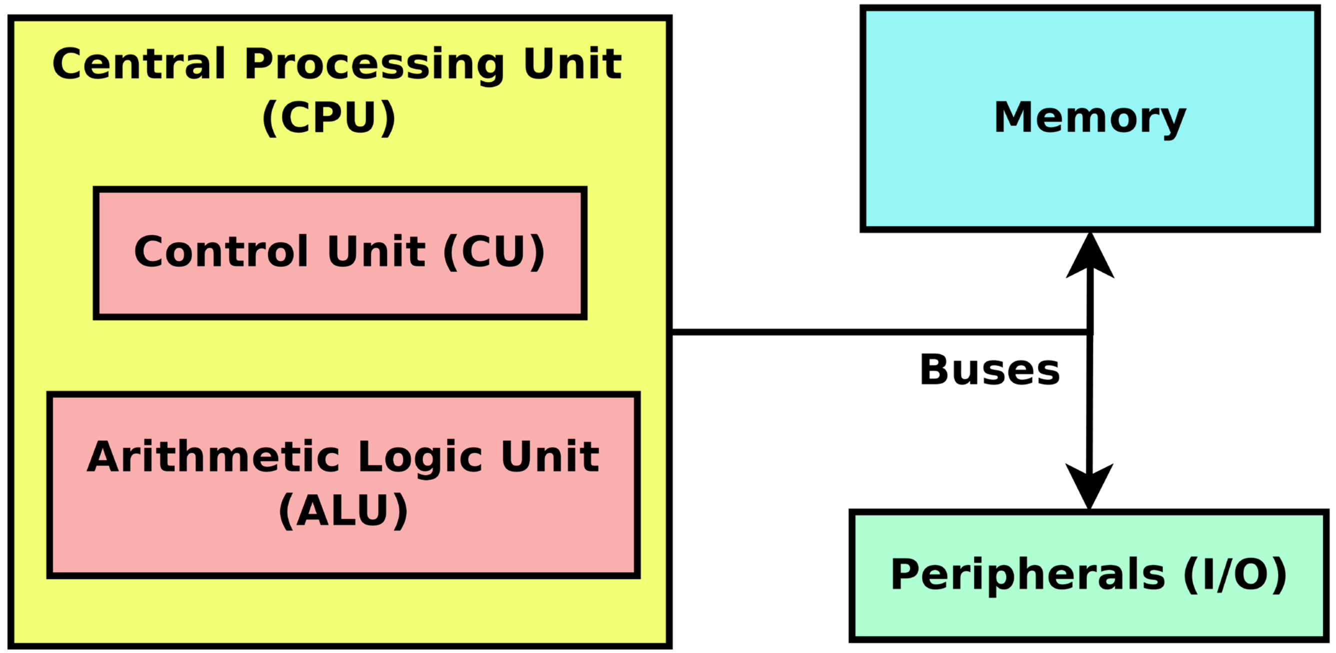
```
00111100000000010000000000000100
00110100010000100000000000000100
00111100000001000001011100010101
00110100100001000010000000110010
...
```

A program becomes a process…

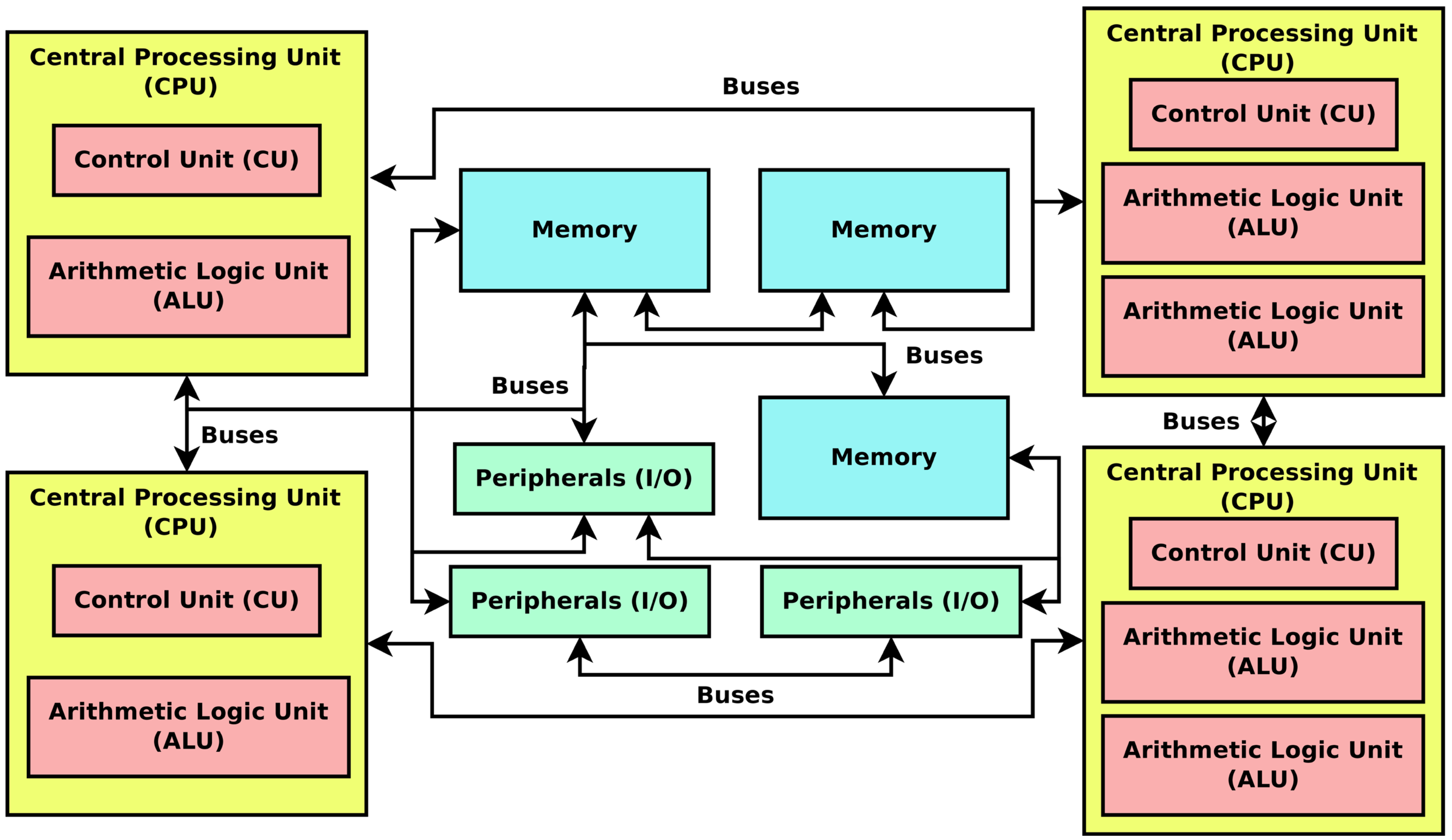
```
00111100000000010000000000000100
00110100001000010000000000000100
00111100000000100000101110001010 1
00110100010000100001000000110010
...
```



Different types of computer architectures



Different types of computers architectures (!!1!!!!!11!!eleven!!)



Central Processing Unit (CPU)
Control Unit (CU)
Arithmetic Logic Unit (ALU)
Buses
Memory
Memory
Central Processing Unit (CPU)
Control Unit (CU)
Arithmetic Logic Unit (ALU)
Arithmetic Logic Unit (ALU)
Buses
Buses
Buses
Memory
Buses
Peripherals (I/O)
Central Processing Unit (CPU)
Control Unit (CU)
Arithmetic Logic Unit (ALU)
Peripherals (I/O)
Peripherals (I/O)
Central Processing Unit (CPU)
Control Unit (CU)
Arithmetic Logic Unit (ALU)
Arithmetic Logic Unit (ALU)
Buses

Different types of computer architectures

- Shared memory: all processors access the same memory (e.g. typical laptop)
  - Early model: uniform memory access for every processor (UMA)

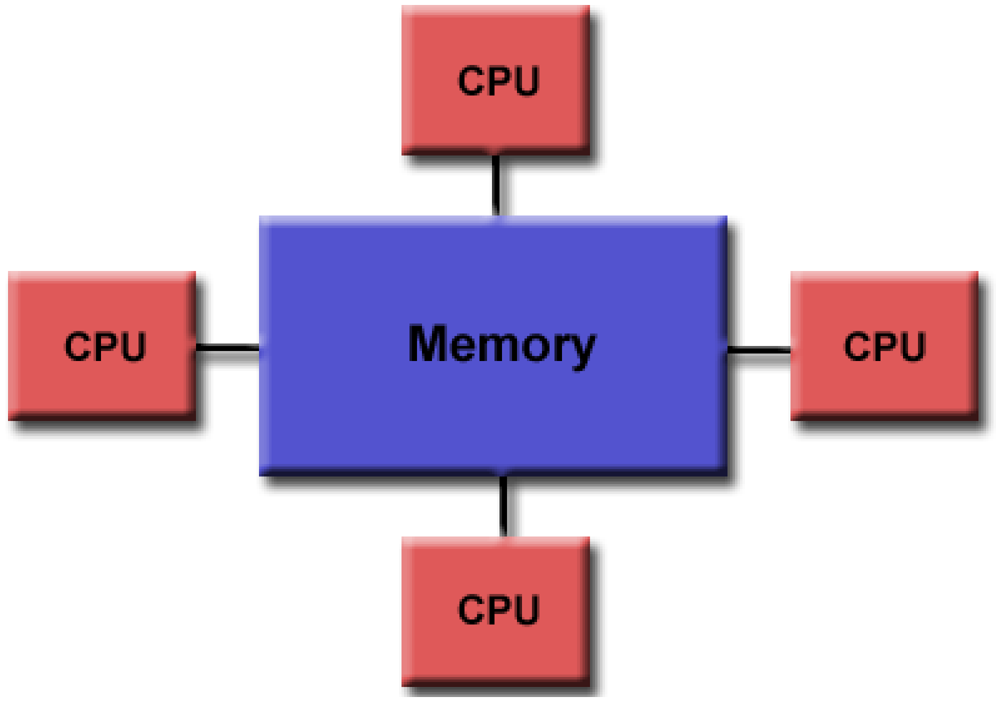


Image source: computing.llnl.gov

Different types of computer architectures

- Shared memory: all processors access the same memory (e.g. typical server)
  - General model: non-uniform memory access for every processor with protocol for cache coherency (ccNUMA)

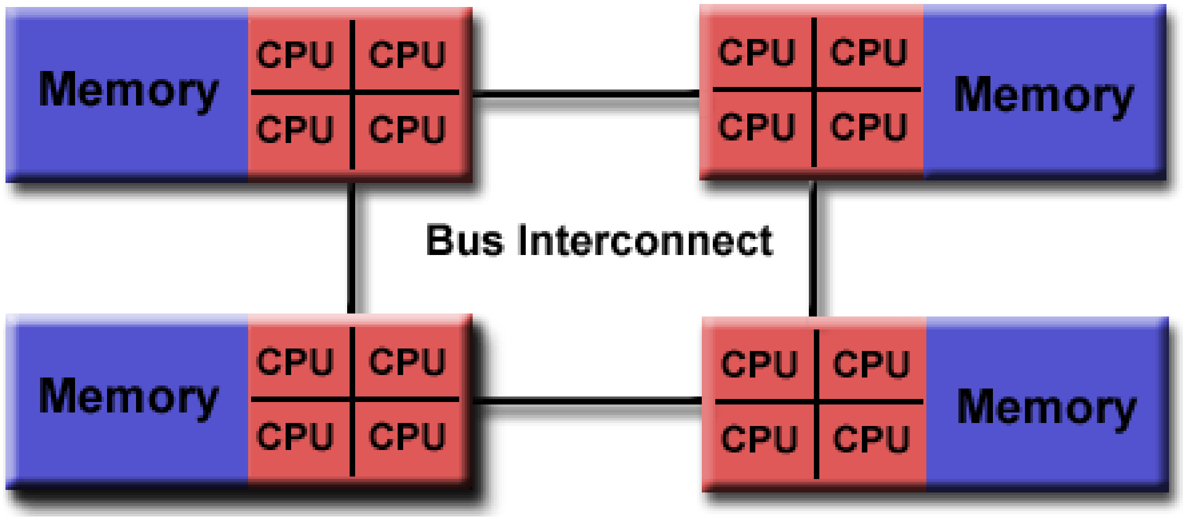
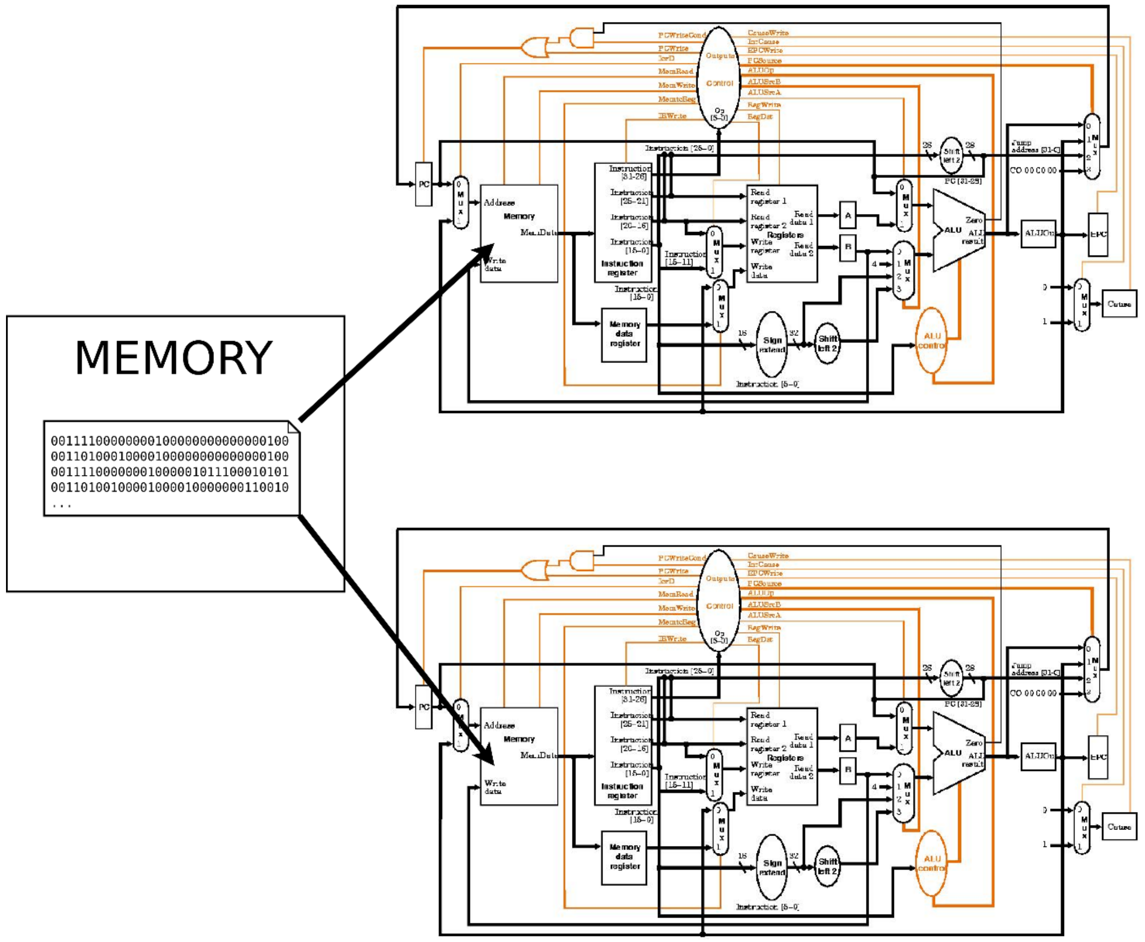


Image source: computing.llnl.gov

Different types of computer architectures

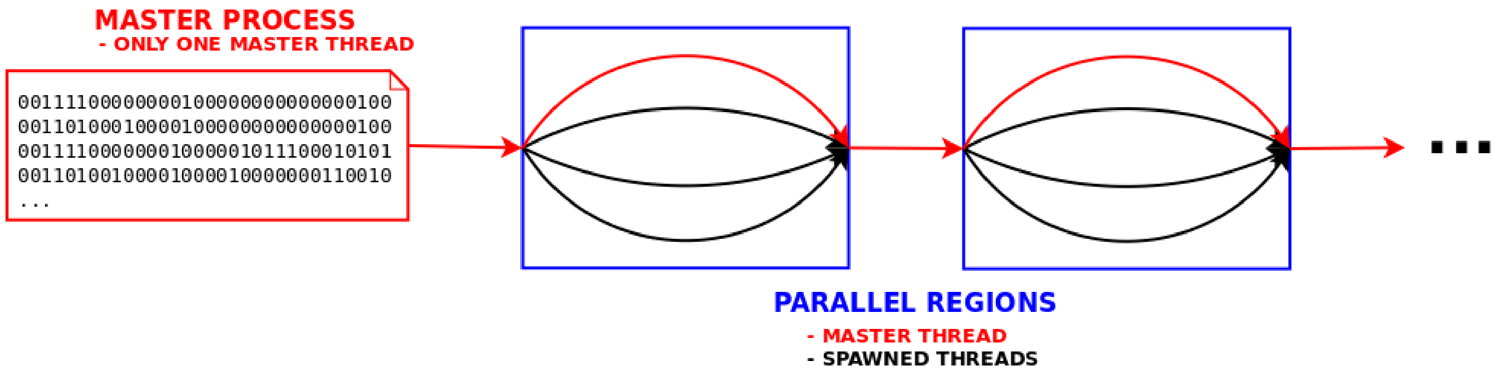
- Shared memory: all processors access the same memory
  - The programmer is seeing a single unified memory
  - Different sequences of execution (threads) run on the same process
  - Different memory modules may be used, but only one logical memory space is addressed
  - Communication between processors is done implicitly



```
00111100000000010000000000000000100
00110100010000100000000000000000100
00111100000000010000010111000010101
00110100100001000010000000110010
...
```

Many threads in a process: OpenMP

- A process creates lightweight instances of itself (threads) that are coordinated for simultaneous execution
  - A thread shares the program code and data section with all other threads inside the same process
  - The parallel computing approach is called fork-join
    - A sequential program begins with one process (that is, only one thread)
    - A parallel region is defined by creating (spawning) threads from the process and destroying the original thread remains



… and this is where we are going to start!

INTRODUCTION TO PARALLEL PROGRAMMING

Caspar van Leeuwen
Carlos Teijeiro Barjas

SURF SARA