INTRODUCTION TO PARALLEL COMPUTING

Parallel Computers and Parallel Computing

- What is a parallel computer?
  - A collection of processing elements that communicate and cooperate to solve problems.
  - What is a processing element?
    - A basic unit that can execute a computation task
    - Depending on the context, it can be a whole computer, a CPU, a CPU core, a GPU core, or a functional unit.
  - Three key elements in a parallel computer:
    1. More than one processing element – this allows multiple computations to be performed in parallel!
    2. The processing elements are connected, and can communicate with one another when needed
    3. The processing elements cooperate to solve problems.

Most Contemporary Computing Devices are Parallel Computers

- Mobile devices, IoT devices, many have multi-core CPUs
  - IPhone 13, A15 – 6 CPU cores, 16 Neural Engine cores, 4 GPU cores
- Desktop or laptop, multi-core CPU
- A high-end gaming computer (CPU + GPU)

Uniprocessor systems

- Multi-core server
- Linprog cluster
- FSU Research Computing Center (RCC)
- Cloud Computing platforms (Amazon AWS, Google cloud).
- Frontier supercomputer at Oak Ridge National Laboratory (No. 1 in June 2024 , 1714.81 Peta flops per second peak performance)

Multi-processor systems

- **In today's world, almost any computing device is a parallel computing device!**

Question

- Can the programs that you wrote take advantage of the parallel computing capability of modern computing devices?

- Parallel computing typically means utilizing many computing resources in a device or a system (which may be a cluster of devices) at the same time to solve ONE problem.
  - Solving the problem of the same size faster (e.g High Frequency Trading)
  - Solving the problem of bigger sizes in the same time (e.g. Large Language Models like ChatGPT)

Example: Calculating PI

$$\mathrm{PI} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \frac{4.0}{1+(1+\frac{i-0.5}{n} * \frac{i-0.5}{n})}$$

```
h, sum = 1.0 / 1000000000, 0.0
for in in range(1000000000):
    x = h * (i-0.5)
    sum + = 4.0 / (1.0 + x * x)
```

Solve problem faster with 2 CPUs:

CPU 0

```
h, sum = 1.0 / 1000000000, 0.0
for in in range(500000000):
    x = h * (i-0.5)
    sum + = 4.0 / (1.0 + x * x)
```

CPU 1

```
h, sum = 1.0 / 1000000000, 0.0
for in in range(500000000, 1000000000):
    x = h * (i-0.5)
    sum + = 4.0 / (1.0 + x * x)
```

Solve a larger problem with 2 CPUs:

```
h, sum = 1.0 / 2000000000, 0.0
for in in range(1000000000):
    x = h * (i-0.5)
    sum + = 4.0 / (1.0 + x * x)
```

```
h, sum = 1.0 / 2000000000, 0.0
for in in range(1000000000, 2000000000):
    x = h * (i-0.5)
    sum + = 4.0 / (1.0 + x * x)
```

Parallel Computers and Parallel Computing

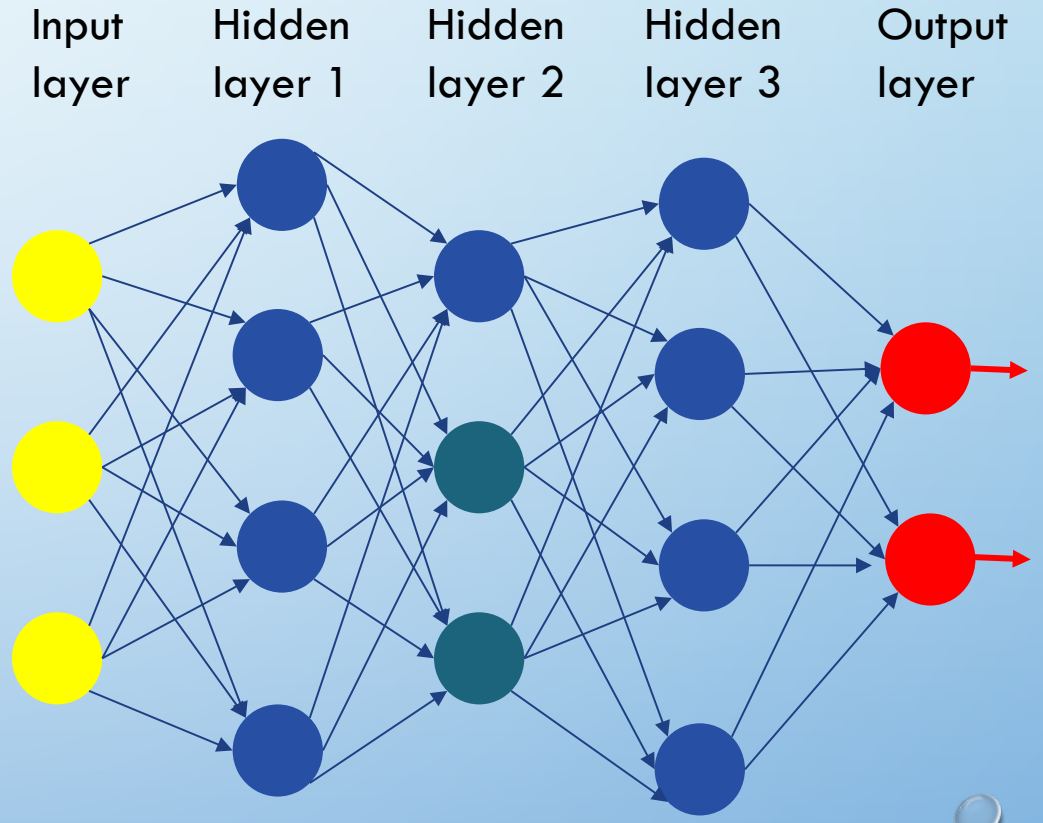
- Parallel computer
  - A collection of processing elements that communicate and cooperate to solve problems.
- Parallel computing
  - Parallel computing is *a type of computation where many calculations or processes are performed simultaneously*.
  - Parallel computing is performed on a parallel computer where multiple processing elements are utilized simultaneously.
- Parallel programs
  - Parallel programs are programs that can simultaneously utilize multiple processing elements, usually having multiple threads of execution.
  - Sequential programs: programs where instructions are executed in sequence.

Why parallel computing (more) ?

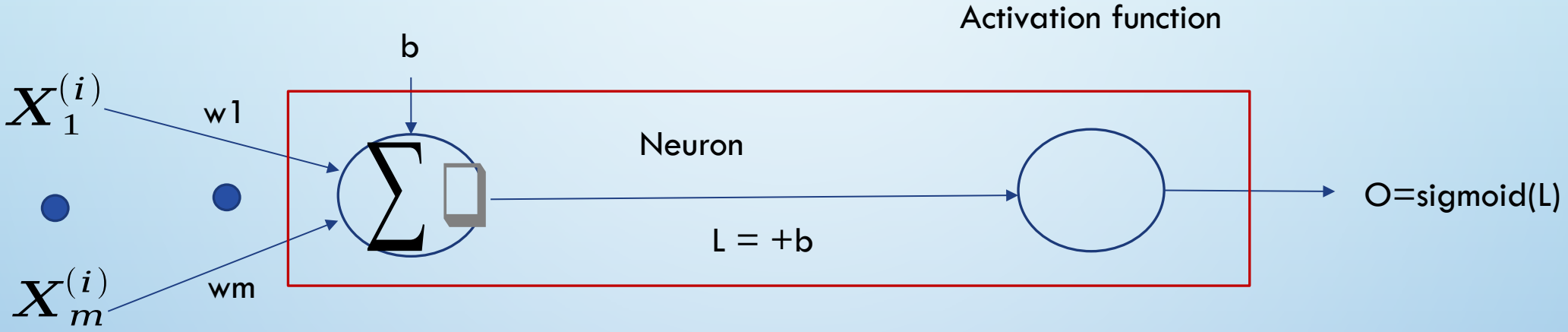
- Some large-scale applications can use any amount of computing power.
  - Scientific computing applications
    - ❖ Weather simulation. More computing power means more finer granularity and prediction longer into the future.
    - ❖ Training of large machine learning models in many domains, such as ChatGPT.
- In small scale, we would like our programs to run faster as the technology advances – conventional sequential programs are not going to do that.
  - CPU frequence has stopped at around 4GHz for more than 15 years, and single threaded programs are not running much faster during this period of time.
  - What we see during the period is that the number of Cores in CPUs steadily increases.

LLMs: today's driven force for Parallel Computing

- A parallel computing example: ChatGPT
  - Under the hood of a large language model is a large deep neural network, which consists of layers of artificial neurons and connections between them.
  - Each connection is associated with a weight, each neuron is also associated with a bias.
  - Training of a neural network is to get to the right weights and biases such that the error across the training data is minimized.

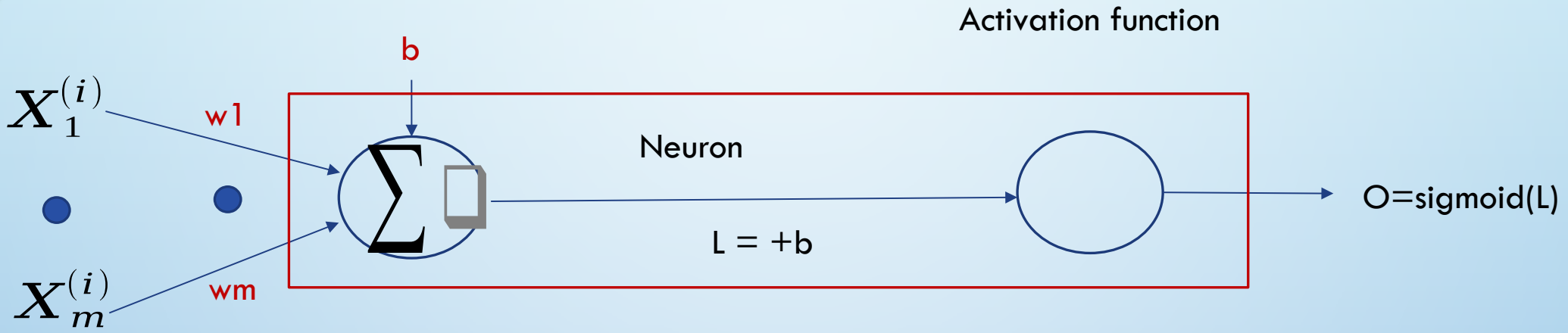


A single neuron



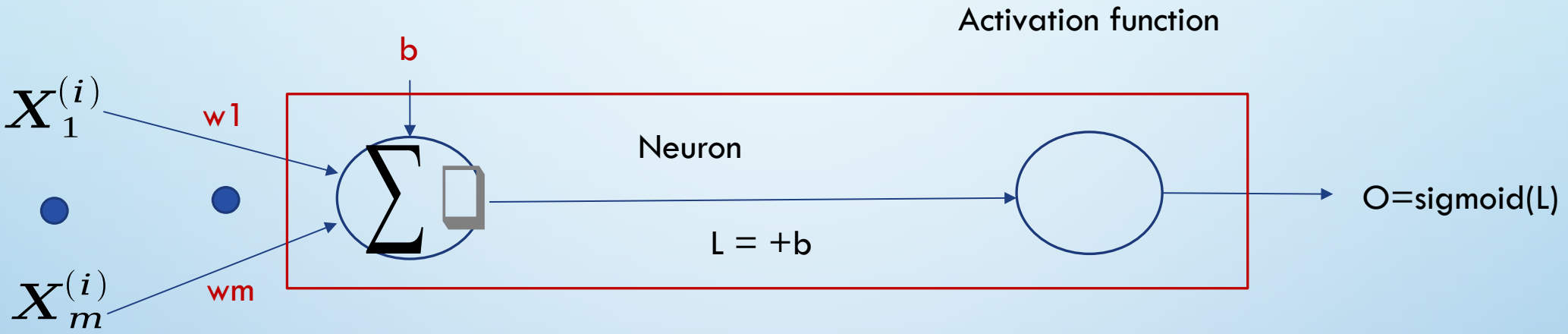
- An artificial neuron has two components: (1) weighted sum and (2) a nonlinear activation function.
  - Many activation functions, a commonly used one is sigmoid:

Training of a single neuron



- The training problem:
  - Given a set of training data [(,), (,), …, (,),], find the value for w1, w2, …, wm and b that minimize the overall errors.

Training of a single neuron



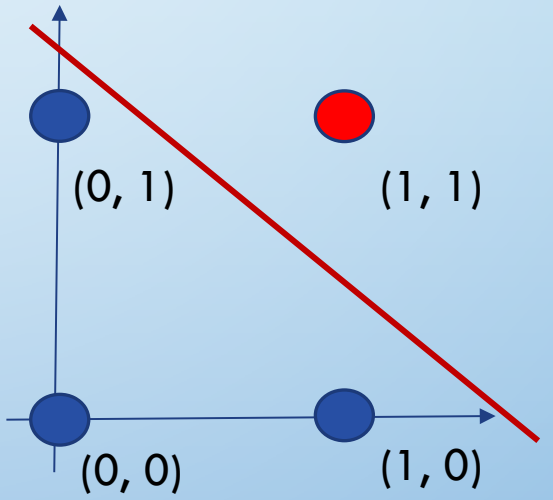
- The training algorithm (assume error)
  - Initialize the parameters w1, …, wm and b with some random number (or all 0's)
  - For each training data (,):
    - Perform forward propagation: compute L, O, and E
    - Perform backward propagation of errors: compute the gradient of the loss in O, which is ; the gradient of loss in L, which is , gradients for w1 …, wm and b.
    - Update the parameters wi = wi – learning_rate * gradients for_wi

Training for the logic AND with a single neuron

- In general, one neuron can be trained to realize a linear function.
- Logic AND function is a linear function:

0	0	0
0	1	0
1	0	0
1	1	1

Logic AND () operation



Training for the logic AND with a single neuron

- Consider training data input (=0,), output Y=0.
- Forward propagation:
  - L = +
  - O = Sigmoid(L) = sigmoid (0) = = 0.5
  -

Training for the logic AND with a single neuron

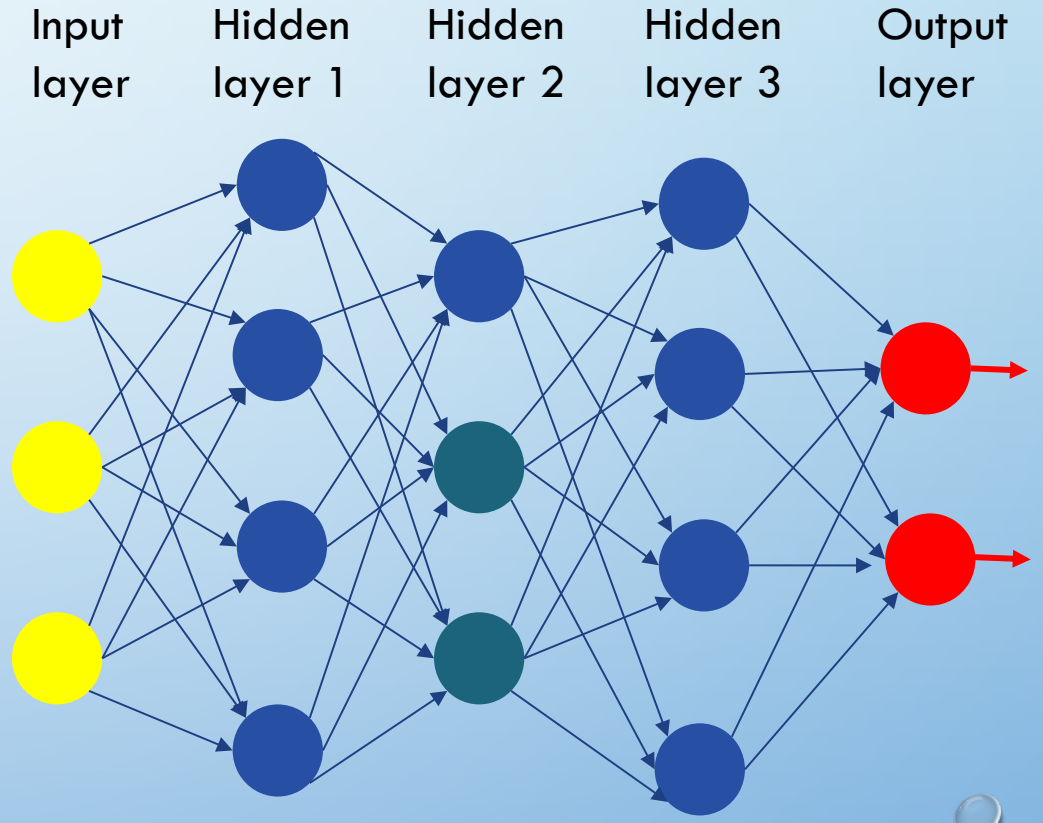
- Consider training data input (=0,), output Y=0.
- Backword propagation of gradients:
  - Gradient for O: =
  - Gradient for L: = = 0.5 * = 0.5 * sigmoid(L) (1-sigmoid(L)) = 0.5 * 0.5 (1-0.5) = 0.125,
  - Gradient for w2: = 0.125 * = 0.125
  - Gradient for w1=0, and Gradient for b=0.125

Training for the logic AND with a single neuron (See lect5/neuron_and1.py)

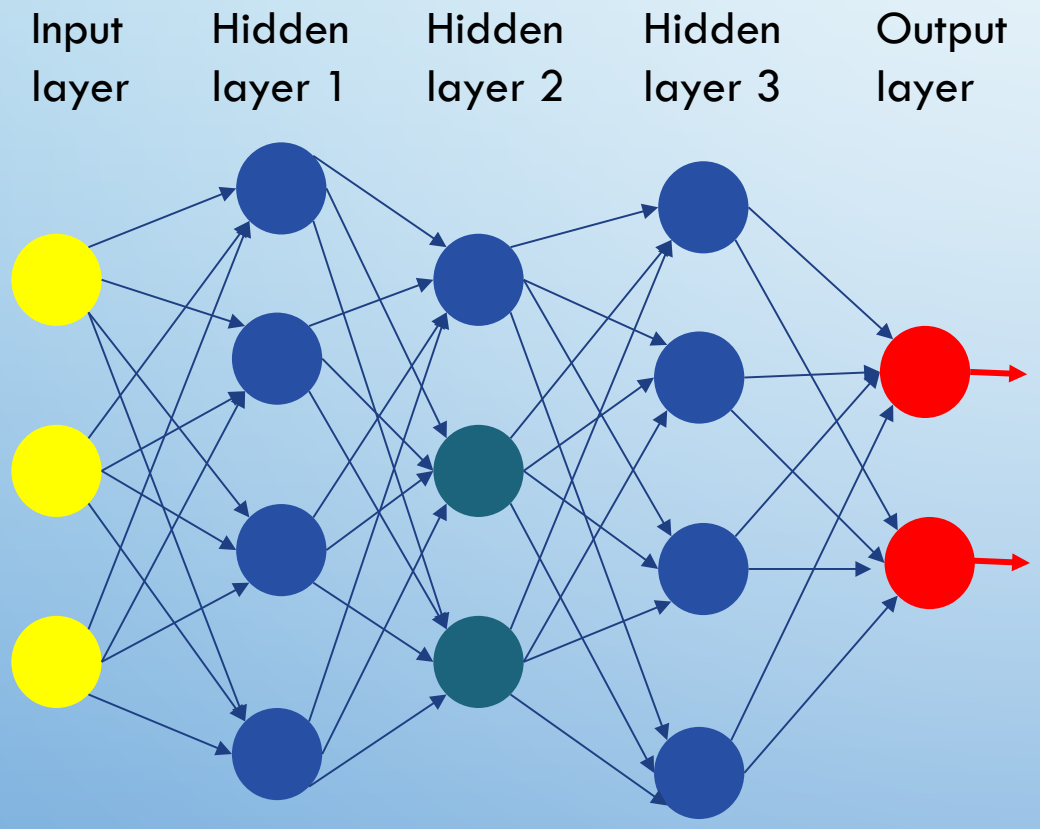
- Consider training data input (=0,), output Y=0.
- Update parameters (learning rate = 0.1)
  - w1 = w1 – learning_rate * gradient_for_w1 = 0 – 0.1 * 0 = 0
  - w2 = w2 – learning_rate * gradient_for_w2 = 0 – 0.1 * 0.125 = -0.0125
  - b = b – learning_rate * gradient_for_b = 0 – 0.1 * 0.125 = -0.0125

Training the deep neural network has the same process as training one neuron

- Initialize parameters (weights and biases) with some random numbers (or all 0's)
- For each training data (,):
  - Perform forward propagation to compute neural network output and error for the input
  - Perform backward propagation of gradients: compute the gradient for each parameter.
  - Update the parameters wi = wi –learning_rate * gradient_for_wi
- Time complexity for one training data input: ?



Training ChatGPT



- ChatGPT 4 has 1 trillion parameters.
- The size of the training data is also huge, in trillions.
- Total computations?
- Time it takes to train ChatGPT with parallel computing:
  - 5-6 months with 10,000 V100 GPU.

Why parallel computing?

- Bigger: Solving larger problems in the same amount of time.
- Faster: Solving the same sized problem in a shorter time.

- We would not have had ChatGPT without parallel computing!
  - Major world powers are racing to build larger and larger supercomputers.

Classifying parallel computing systems: Flynn's Taxonomy

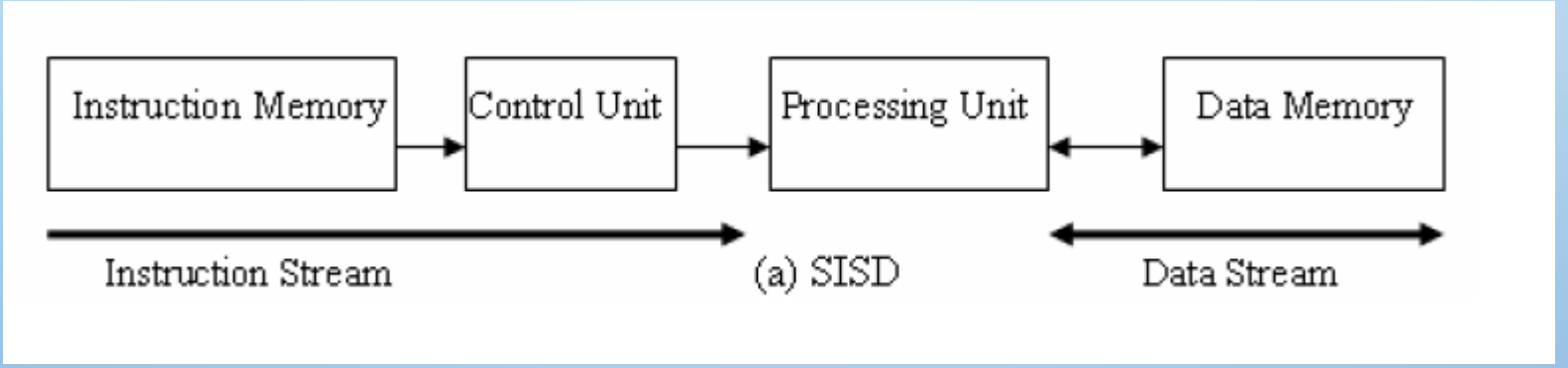
- Computing is basically executing instructions that operate on data.
- Flynn's taxonomy classifies the system based on the parallelism in instruction stream and parallelism in data stream.
  - single instruction stream or multiple instruction streams.
  - single data stream or multiple data streams.

Flynn's taxonomy

- Single Instruction Single Data (SISD)
- Single Instruction Multiple Data (SIMD)
- Multiple Instructions Multiple Data (MIMD)
- Multiple Instructions Single Data (MISD)

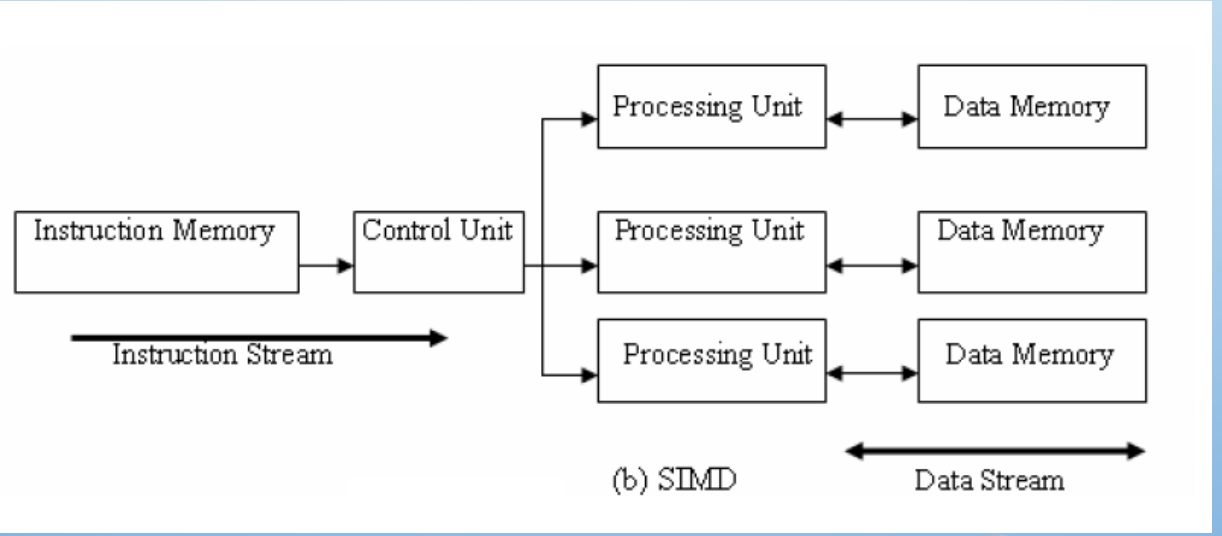
SISD

- At one time, one instruction operates on one data
  - Traditional sequential architecture, Von Neumann architecture.



SIMD

- Single control unit and multiple processing units. The control unit fetches an instruction and broadcast control to all processing units. The instruction operates on different data.
  - Can achieve massive processing power with minimum control logic
  - SIMD instructions allow for sequential reasoning.

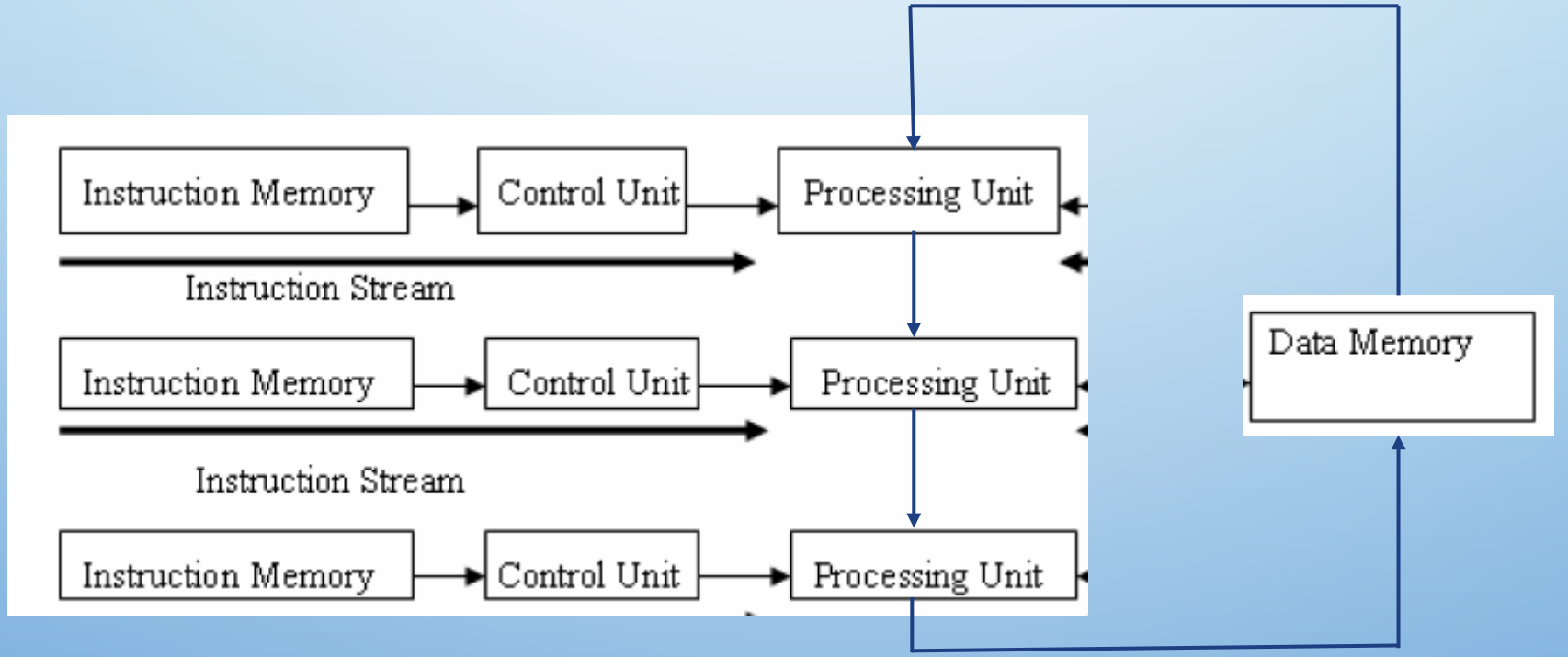


SIMD

- Exploit data-level parallelism
  - Matrix-oriented scientific computing and deep learning applications
  - Media (image and sound) processing
- Vector machines, MMX, SSE (Streaming SIMD Extensions), AVX (Advanced Vector eXtensions), GPU (extend SIMD or SIMT)

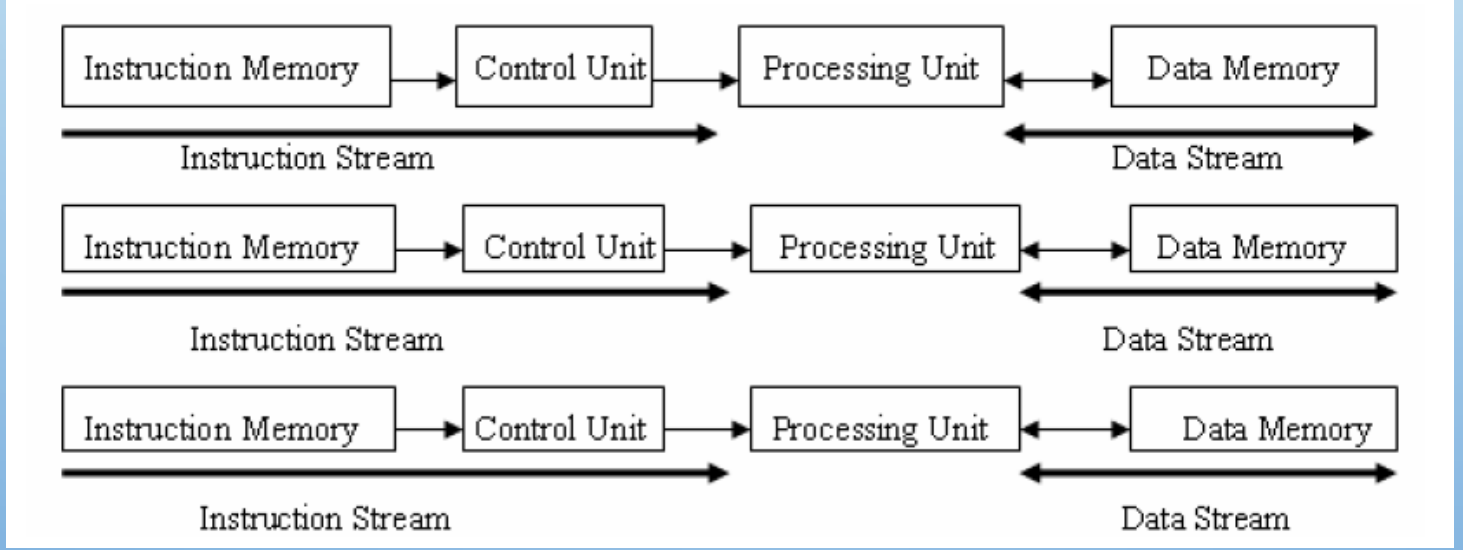
MISD

- Not commonly seen, no general purpose MISD computer has been built.
- Systolic array is one example of an MISD architecture.



MIMD

- Multiple instruction streams operating on multiple data streams
  - MIMD can be thought of as many copies of SISD machine.
  - Distributed memory multi-computers, shared memory multi-processors, multi-core computers.

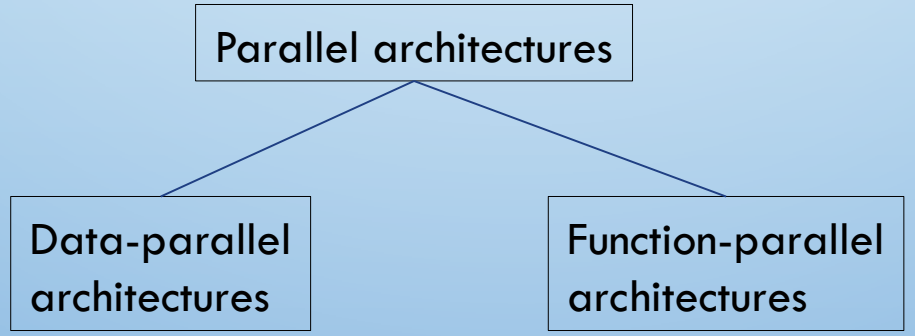


Flynn's Taxonomy

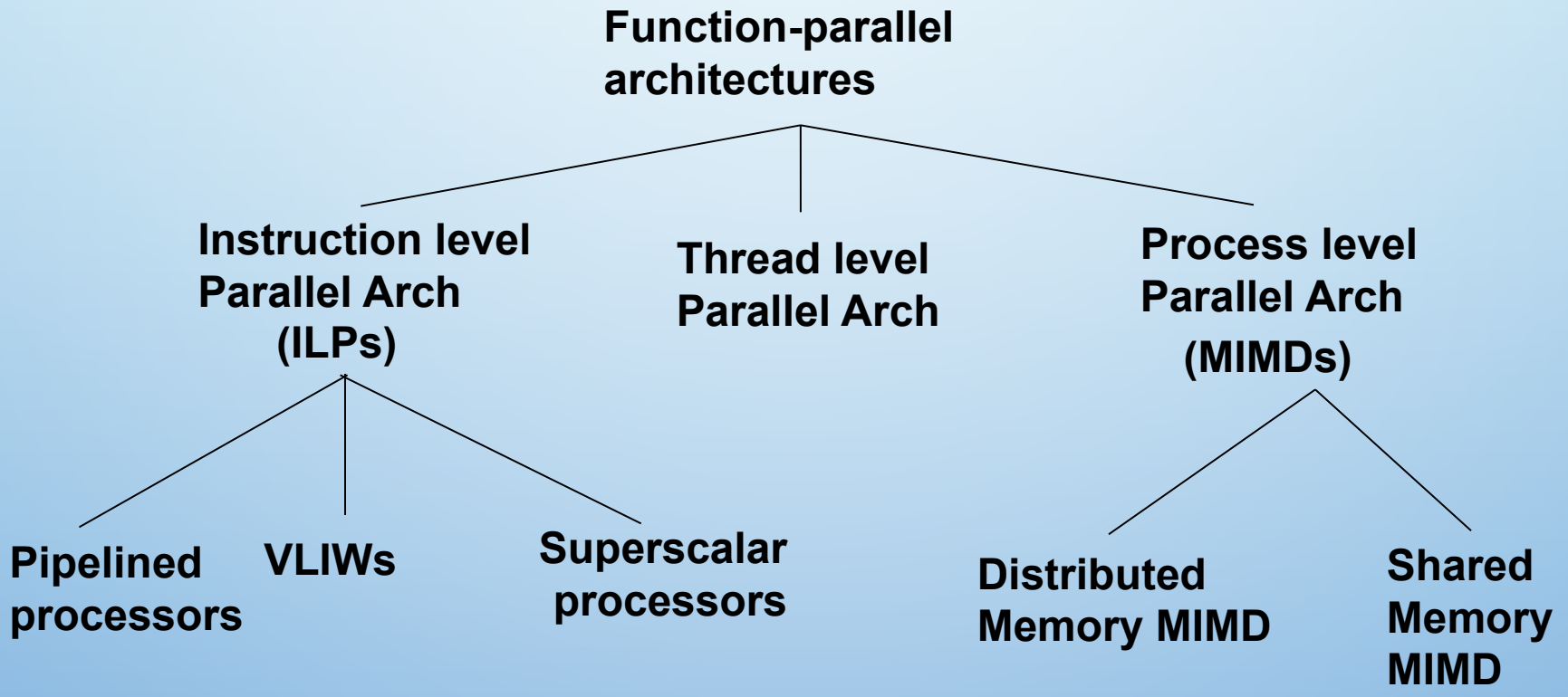
Type	Instruction Streams	Data Streams	Examples
SISD	1	1	Early computers, Von Neumann architecture, turing machine
SIMD	1	N	Vector architectures, MMX, SSE, AVX, GPU
MISD	N	1	No general purpose machine, systolic array
MIMD	N	N	Multi-core, multi-processor, multi-computer, cluster

Modern classification (Sima, Fountain, Kacsuk)

- Based on how parallelism is achieved
  - Data parallelism: same function operating on many data
  - Function parallelism: performing many functions in parallel
    - Control parallelism, task parallelism depending on the level of the functional parallelism.



Functional-parallel architectures



What types of (sequential) applications can be improved by parallel computing

- If we have an infinite number of processors, it is possible to make sequential program run much faster?

```
A = 2;
B = 4;
C = 6;
D = 10;
```

```
A = 2;
B = A + A;
C = B * A;
D = C+1;
```

```
For(i=0; i<100000; i+= 1)
    a[i] = b[i] + c[i];
```

Dependency and Parallel Execution

- When two instructions operate on the same data with at least one instruction write to the data, we say that there exists data dependency between the two instructions.
  - True dependency – read after write
  - Output dependency – write after write
  - Anti-dependency – write after read
- Parallel execution can change the timing to execute the instruction. With data dependence, the instructions have to be executed in exactly the same order to have the same semantic (we would say the instructions cannot be parallelized).

```
A = 1
B =  A * 2
```

```
A = 1
A = 2
```

```
B = A +1
A = 2
```

Dependency

- In addition to data dependency, a program may also have control dependency.

- Due to the dependencies, a typical program will have a portion that can be parallelized, and the other part that cannot be parallelized.

```
A = 2;
B = A + A;
C = b * A;
D = c+1;

For(i=0; i<100000; i=+)
    a[i] = 0;
```

Speedup and Scalability

- Let be the execution time for a computation to run on 1 processor and be the execution time for the computation (with the same input – same problem) to run on P processors.

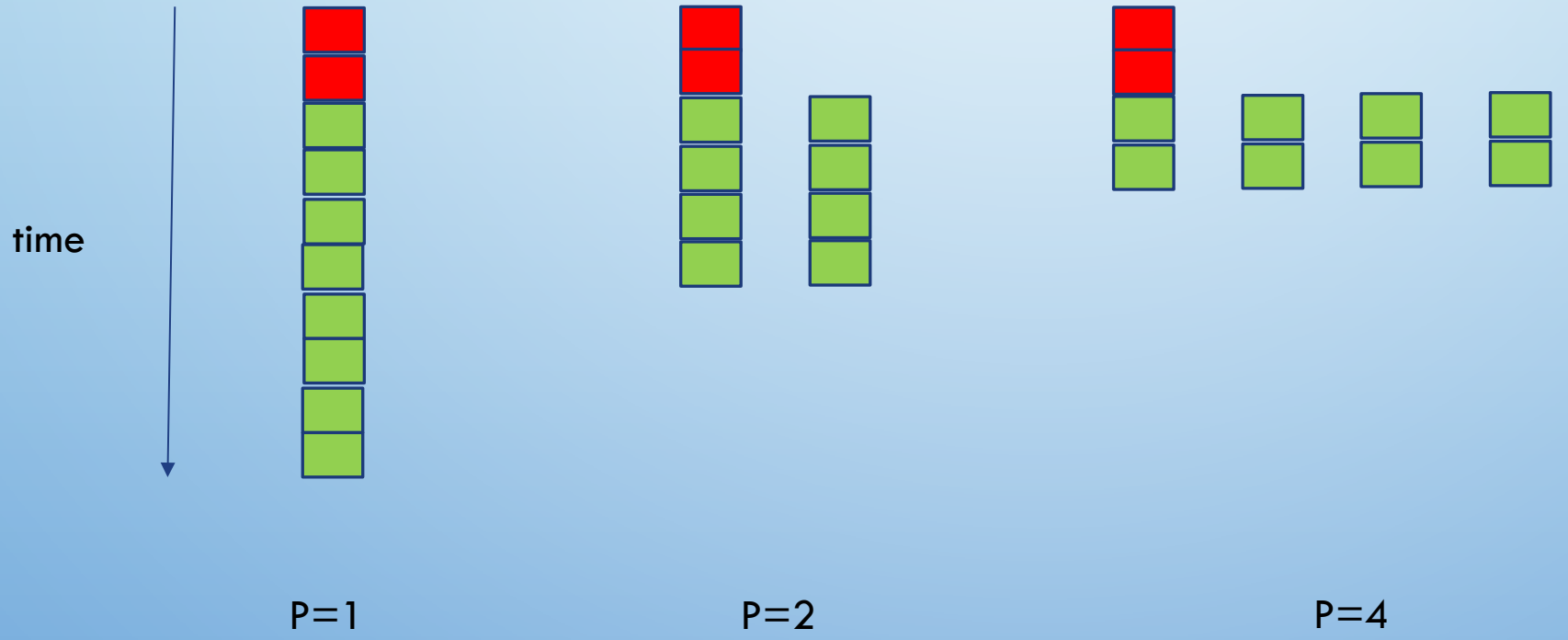
- Scalability of a program measures how many processors that the program can make effective use of.
  - When the problem size is fixed, this is referred to as “strong scaling”.
  - When the problem size can increase with the number of processors, it is referred to as “weak scaling”.

Amdahl's Law (fixed size speedup, strong scaling)

- Given a program, let *f* be the fraction that must be sequential and *1-f* be the fraction that can be parallelized
-
-
- When =
- Original paper: *Amdahl, Gene M. (1967).* *"Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities"* *. AFIPS Conference Proceedings (30): 483–485.*

Amdahl's law

Amdahl's law: As P increases, the percentage of work in the parallel region reduces, performance is more and more dominated by the sequential region.



Question

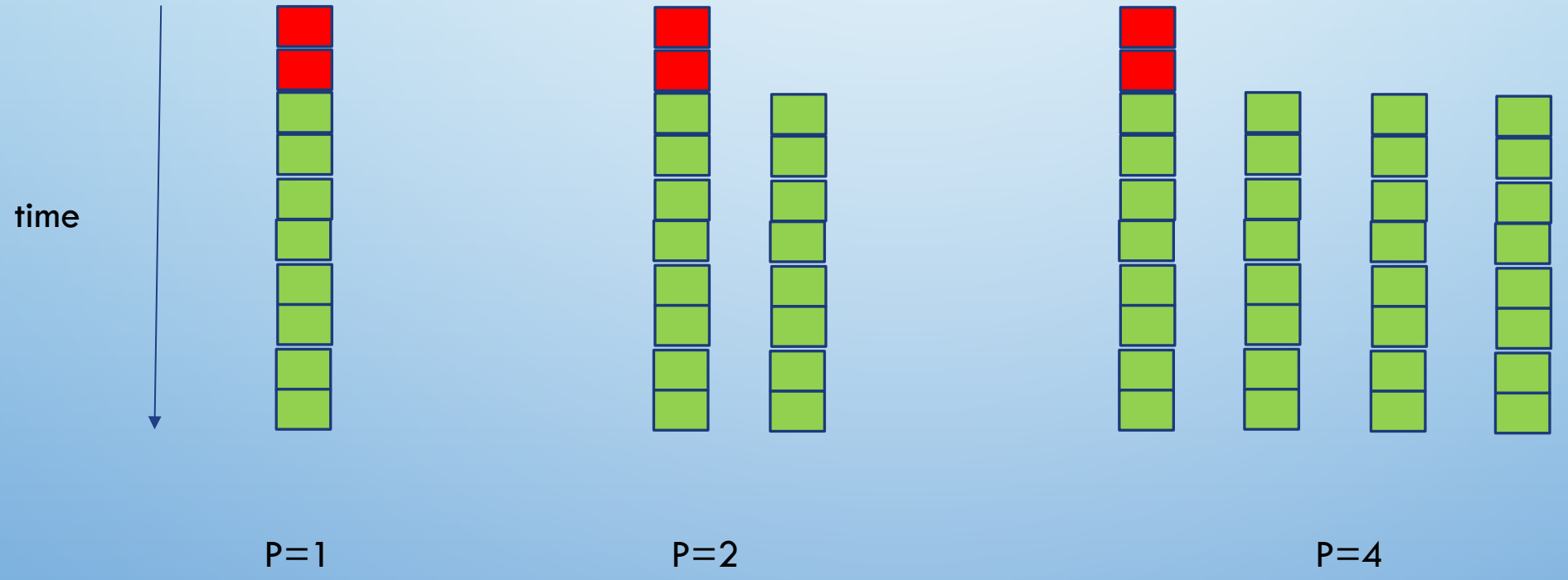
- Let a program has f=10% sequential component and 90% parallel component. What is the speedup one will get with 2 processors, 5 processors and 10 processors?
- If a person tell you that he is able to speed up the program by 15 times, what would be your responds?

Gustafson's Law (scaled speedup, weak scaling)

- Large scale parallel/distributed systems are expected to allow for solving problem faster or larger problems.
  - Amdahl's Law indicates that there is a limit on how faster it can go.
  - How about bigger problems? This is what Gustafson's Law sheds lights on!
- In Amdahl's law, as the number of processors increases, the amount of work in each node decreases (more processors sharing the parallel part).
- In Gustafson's law, as the number of processors increases, the amount of work in each node remains the same (doing more work collectively).

Gustafson's law

Gustafson's law: As P increases, the total work on each process remains the same. So the total work increases with P.



Gustafson's Law (scaled speedup, weak scaling)

- The work on each processor is 1 (f is the fraction for sequential program, (1-f) is the fraction for parallel program.
- With P processor (with the same), the total amount of useful work is . Thus,
- Thus, speedup(P) = .

No of PEs	Strong scaling speedup (Amdalh's law, f = 10%)	Weak scaling speedup (Gustafson's law, f = 10%)
2	1.82	1.9
4	3.07	3.7
8	4.71	7.3
16	6.40	14.5
100	9.90	90.1

Implication of Gustafson's law

- For weak scaling, speedup(P) =
  - Speedup is now proportional to P.
- Scalability is much better when the problem size can increase.
  - Many application can use more computing power to solve larger problems
    - Weather prediction, large deep learning models.
- *Gustafson, John L. (May 1988). "Reevaluating Amdahl's Law". Communications of the ACM. **31** (5): 532–3.*