*COMPUTER ENGINEERING DEPARTMENT*

# BLM413E PARALLEL PROGRAMMING

## LECTURE 1: INTRODUCTION TO PARALLEL PROGRAMMING CONCEPTS

*ASST. PROF. DR. SÜHA TUNA*

# WHAT IS PARALLELISM?

- Paralellism refers to the simultaneous occurence of events on a computer.
- An event typically means one of the following:
  - An arithmetical operation
  - A logical operation
  - Accessing memory
  - Performing input or output (I/O)

# TYPES OF PARALLELISM

- Parallelism can be examined at several levels.
  - Job level: several independent jobs simultaneously run on the same computer system.
  - Program level: several tasks are performed simultaneously to solve a single common problem.
  - Instruction level: the processing of an instruction, such as adding two numbers, can be divided into sub-instructions. If several similar instructions are to be performed their sub instructions may be overlapped using a technique called pipelining.
  - Bit level: when the bits in a word are handled one after the other this is called a bit-serial operation. If the bits are acted on in parallel the operation is bit-parallel.

- **In this course we deal with mostly with parallelism at the program and instruction level.**
- ***Concurrent processing* is the same as parallel processing.**

# AS A RESULT

- Traditional computers (serial computers) are not powerful enough to solve most of the real life problems. Hence, we do need to use parallel processing and/or distributed processing to solve evergrowing or real life problems.

# NEED FOR MORE COMPUTING POWER

- In today science, one trillion operations each second is actually somewhat modest.
- Suppose we wish to predict the weather over the United states and the Canada for the next two days. And also suppose that we want to model the atmosphere from sea level to an altitude of 20 kilometers. We need to make a prediction of the weather at each hour for the next two days.
- Suppose we use a cubical grid, with each cube measuring 0.1 kilometer on each side. Since the area of the United States and Canada is about 20 million square km,

2.0 x 107 km2 x 20 km x 103 cubes per km3 = 4 x 1011 grid points.

• If it takes 100 calculations to determine the weather at a typical point, then in order to predict the weather one hour from now, we'll need to make about 4 x 1013 calculations. Since we want to predict the weather at each hour for 48 hours, we'll need to make a total of about

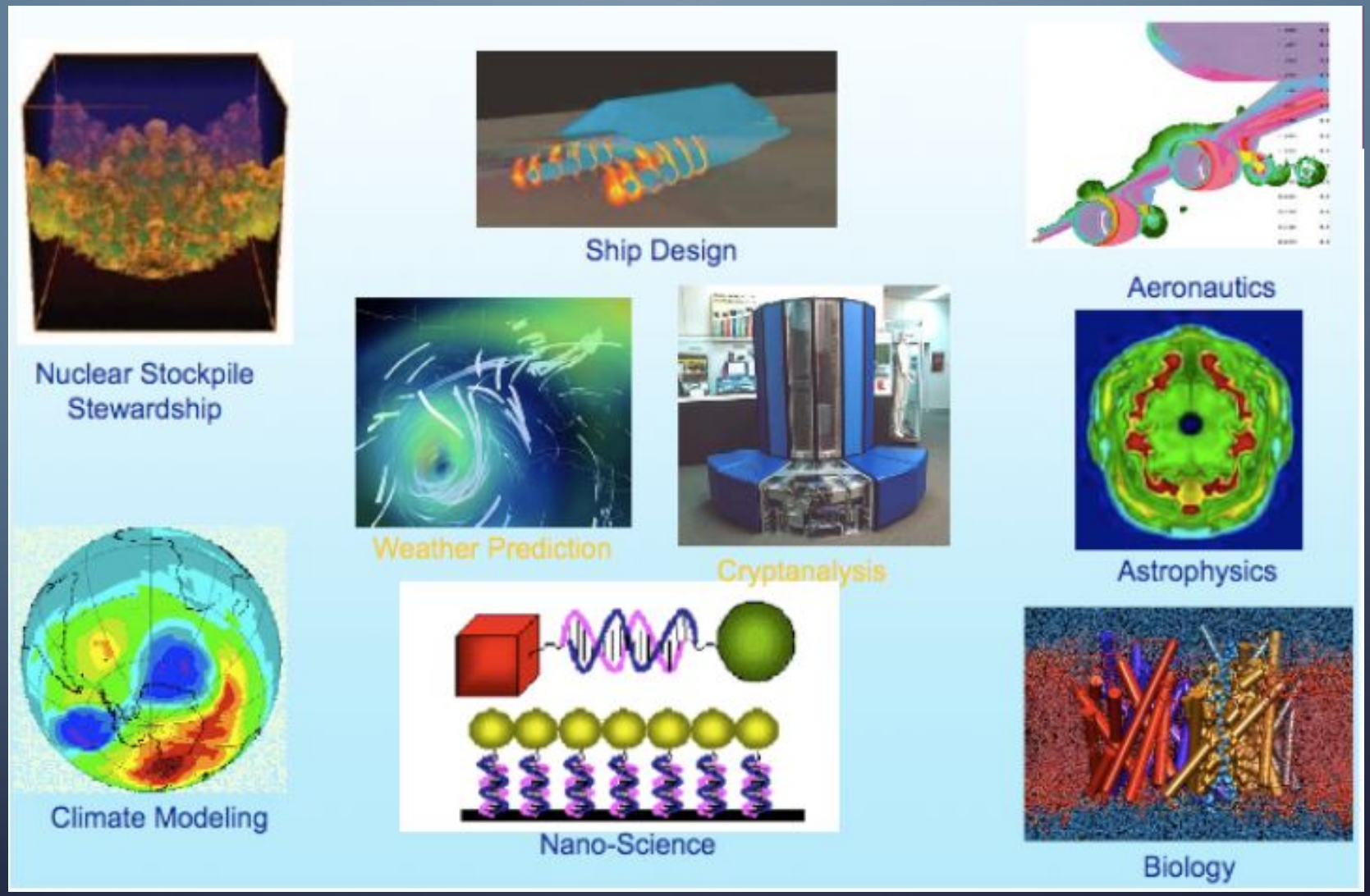
4 x 1013 calculations x 48 hours ~ 2 x 1015 calculations.

• If our computer can execute one billion (109) calculations per second, it will take about

2 x 1015 calculations/109 calculations per second = 2 x 106 seconds ~23 days

- In other words, the calculation is hopeless if we can only carry out one billion operations per second. If, on the other hand, we can carry out one trillion (1012) calculations per second, it will take us about half an hour to carry out the calculations.
- If you replace it with the entire world, it'll take 13 hours (first 12 hours are useless).

# APPLICATIONS OF HIGH-END COMPUTING



# UNITS OF MEASURE IN HPC

- High Performance Computing (HPC) units are:
  - Flop: floating point operation
  - Flops/s: floating point operations per second
  - Bytes: size of data
- Typical sizes are millions, billions, trillions, …

  Mega Mflop/s = 106 flop/sec --> Mbyte = 220 = 1048576 ~ 106 bytes

  Giga Gflop/s = 109 flop/sec --> Gbyte = 230 ~ 109 bytes

  Tera Tflop/s = 1012 flop/sec --> Tbyte = 240 ~ 1012 bytes

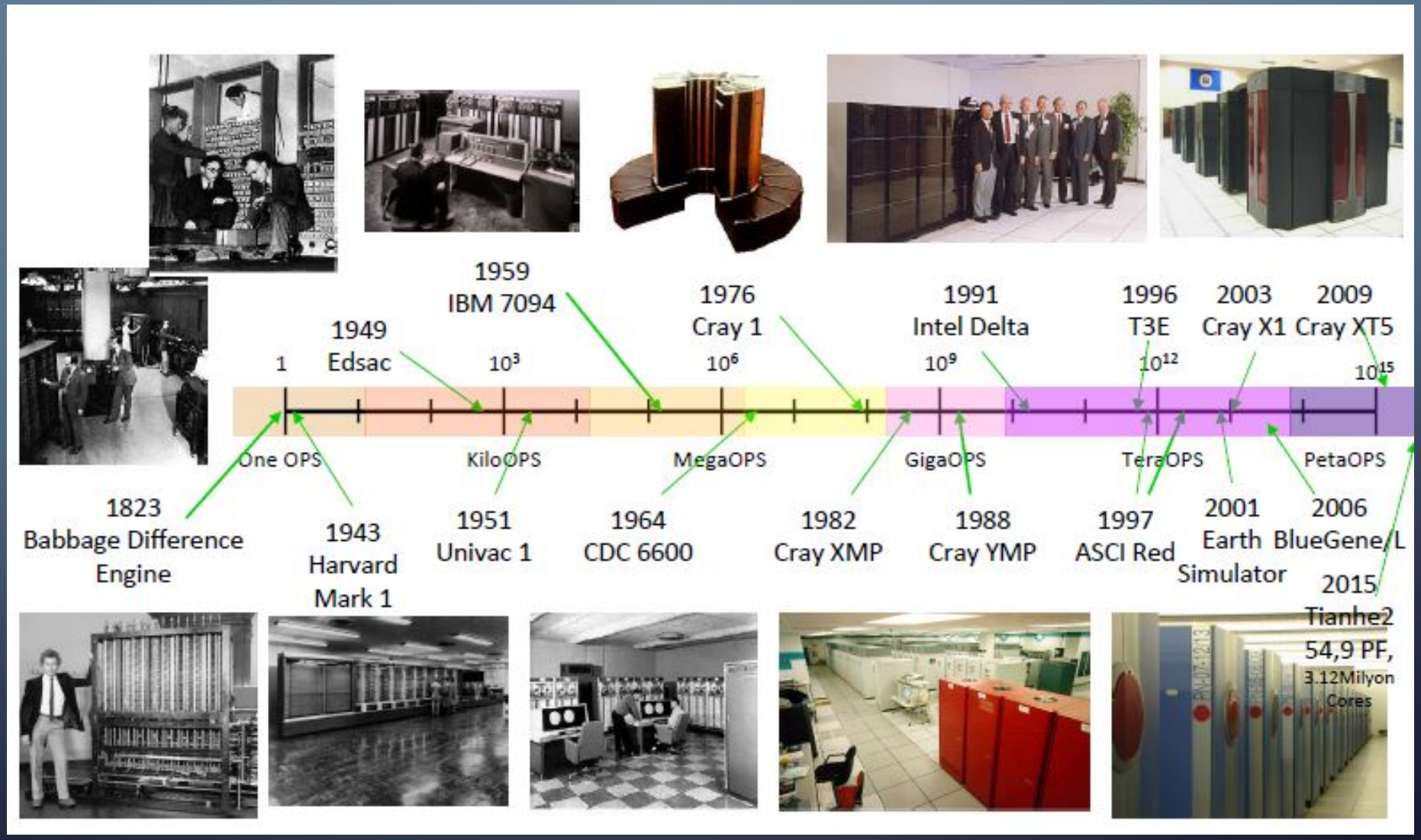
  Peta Pflop/s = 1015 flop/sec --> Pbyte = 250 ~ 1015 bytes

  Exa Eflop/s = 1018 flop/sec --> Ebyte = 260 ~ 1018 bytes

  Zetta Zflop/s = 1021 flop/sec --> Zbyte = 270 ~ 1021 bytes

  Yotta Yflop/s = 1024 flop/sec --> Ybyte = 280 ~ 1024 bytes

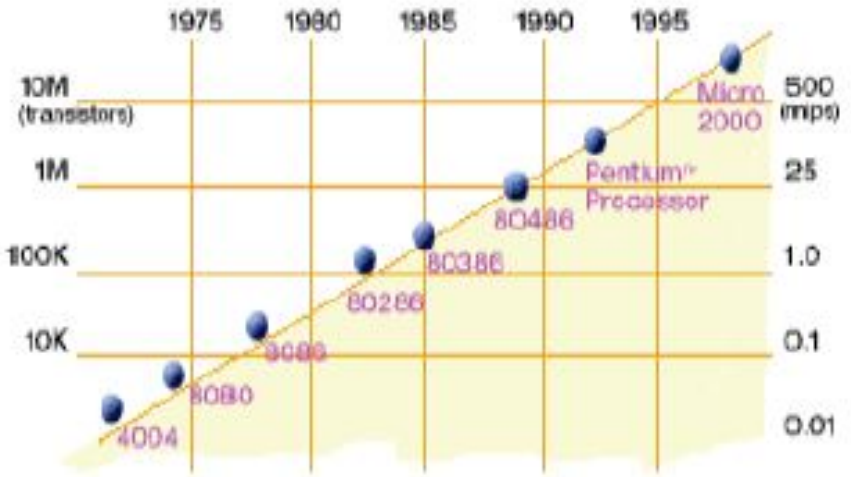
# EVOLUTION OF HPC



# WHY DO WE MAKE PARALLEL COMPUTING

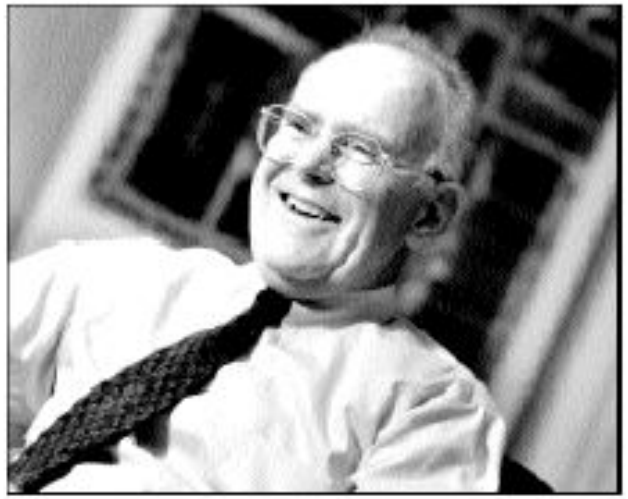
- Time: Reduce the turnaround time of applications
- Performance: Parallel computing is the only way to extend performance toward the TFLOP realm
- Cost/Performance: Traditional vector computers become too expensive as one pushes the performance barrier
- Memory: Applications often require memory that goes beyond that addressable by a single processor
- Whole classes of important algorithms are ideal for parallel execution. Most algorithms can benefit from parallel processing such as Laplace equation, Monte Carlo, FFT (signal processing), image processing
- Life itself is a set of concurrent processes
  - Scientists use modeling so why not model systems in a way closer to nature

# TECHNOLOGY TRENDS: MICROPROCESSOR CAPACITY




2X transistors/Chip Every 1.5 years
Called "Moore's Law"

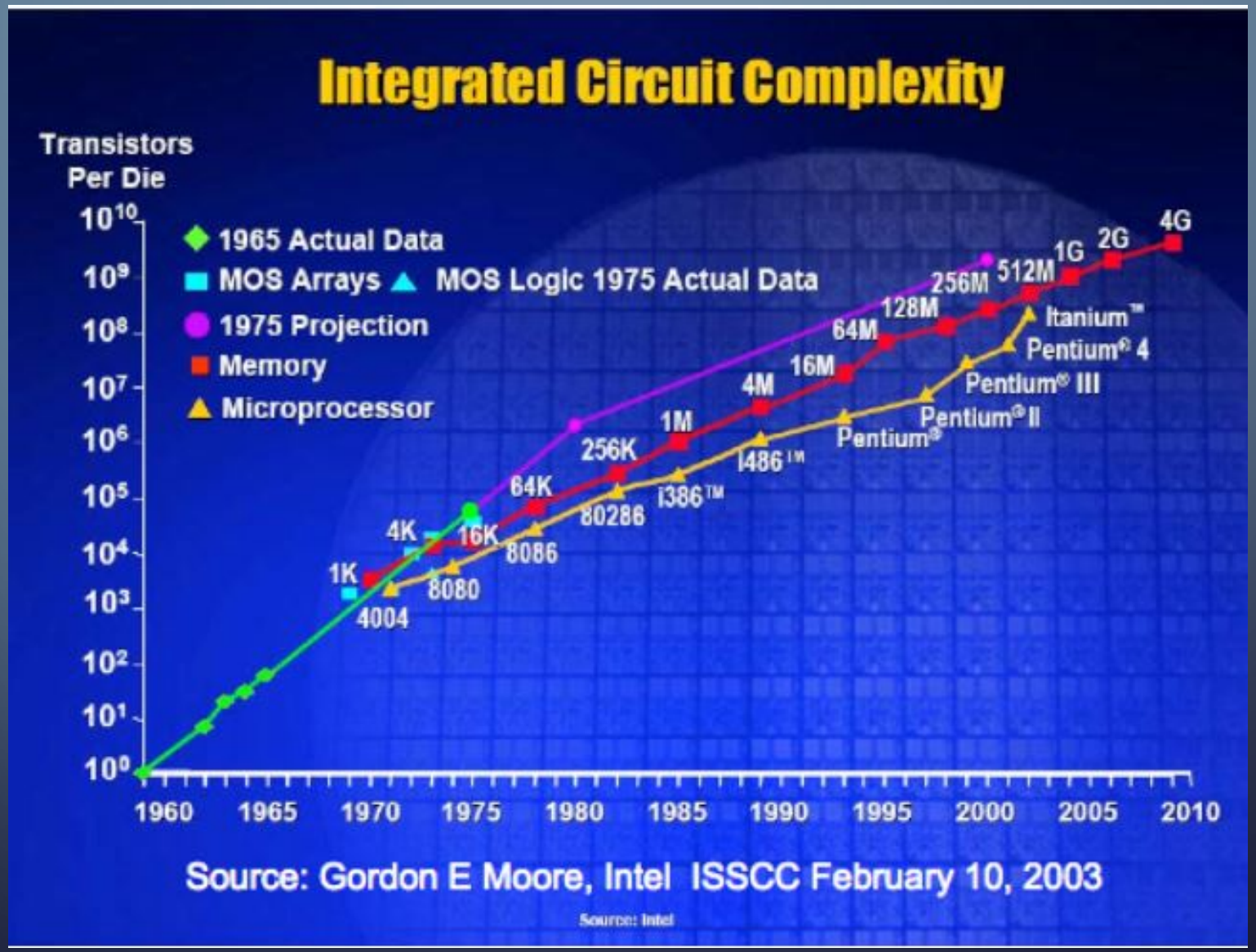
Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

# INTEGRATED CIRCUIT COMPLEXITY

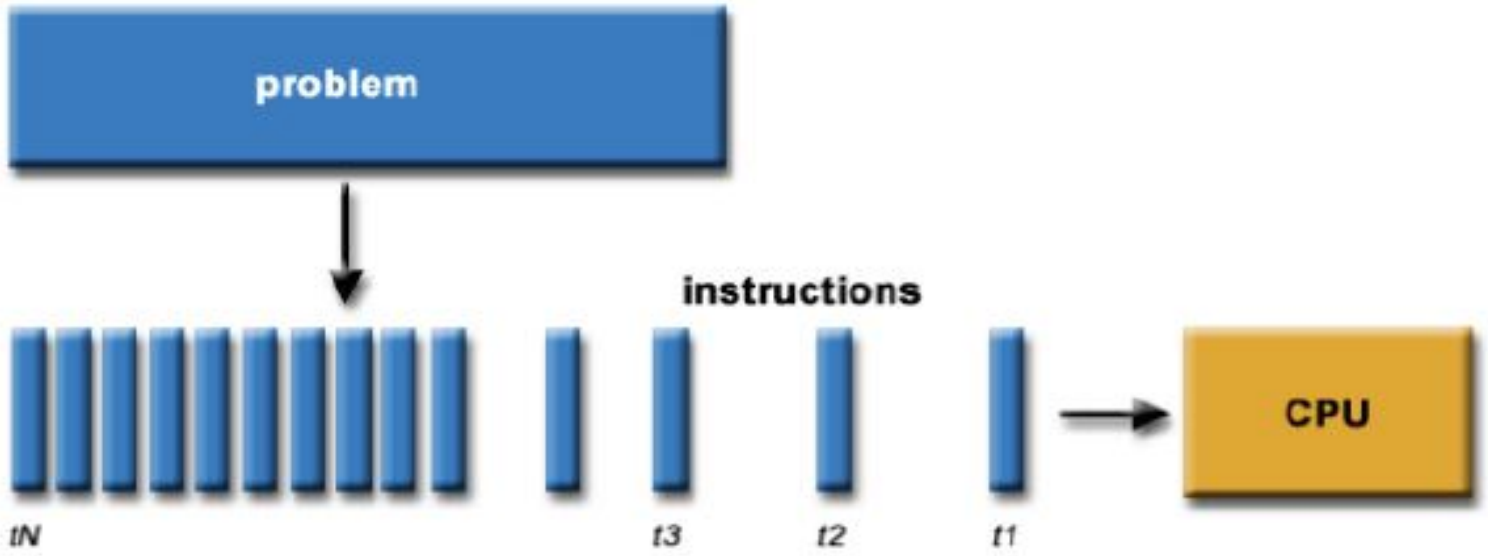


# VON NEUMANN ARCHITECTURE

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. Named after the Hungarian mathematician John von Neumann.
- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.
- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
  - A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then sequentially performs them.

# WHAT IS SERIAL COMPUTING

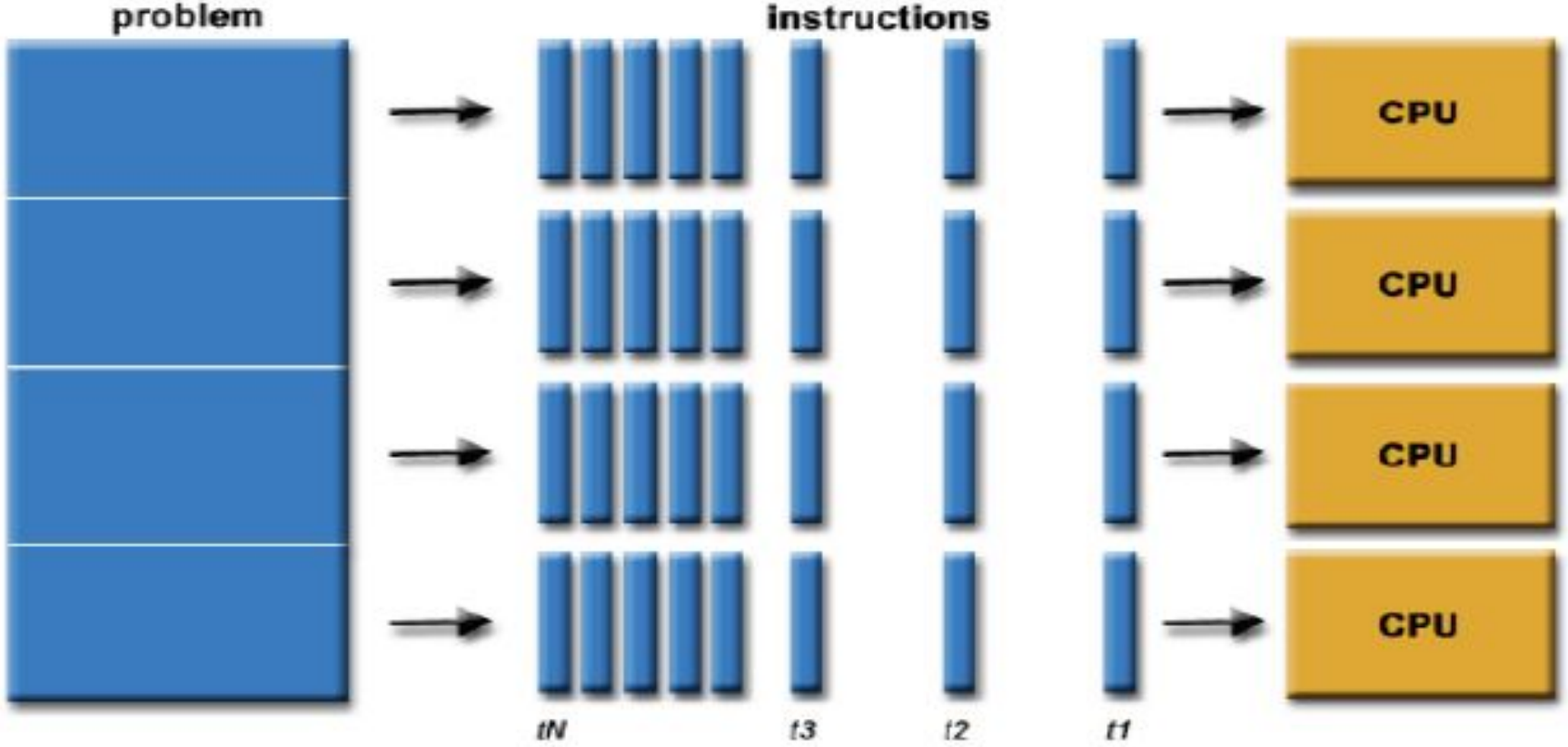
- Traditionally, software has been written for serial computation:
  - o To be run on a single computer having a single Central Processing Unit (CPU);
  - o A problem is broken into a discrete series of instructions.
  - o Instructions are executed one after another.
  - o Only one instruction may execute at any moment in time.



# WHAT IS PARALLEL COMPUTING

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.

- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs



# PRINCIPALS OF PARALLEL COMPUTING

- Parallelism and Amdahl's Law
- Finding and exploiting granularity
- Preserving data locality
- Load balancing
- Coordination and synchronization
- Performance modeling

- *All of these things makes parallel programming even harder than sequential programming.*

# AUTOMATIC PARALLELISM IN MODERN MACHINES

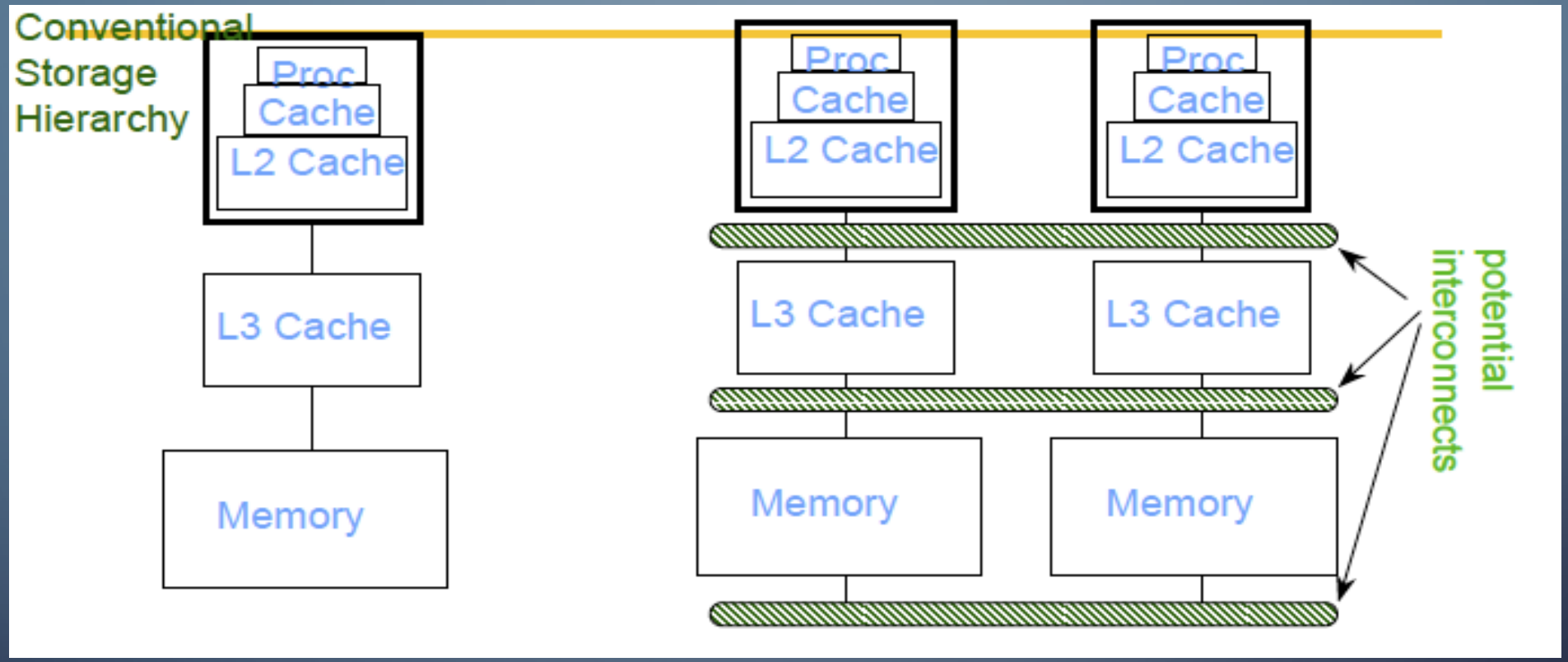
- Bit level parallelism
  - within floating point operations, etc.
- Instruction level parallelism (ILP)
  - multiple instructions execute per clock cycle
- Memory system parallelism
  - overlap of memory operations with computation
- OS parallelism
  - multiple jobs run in parallel on commodity SMPs

*Limits to all of these -- for very high performance, need user to identify, schedule and coordinate parallel tasks*

# OVERHEAD OF PARALELLISM

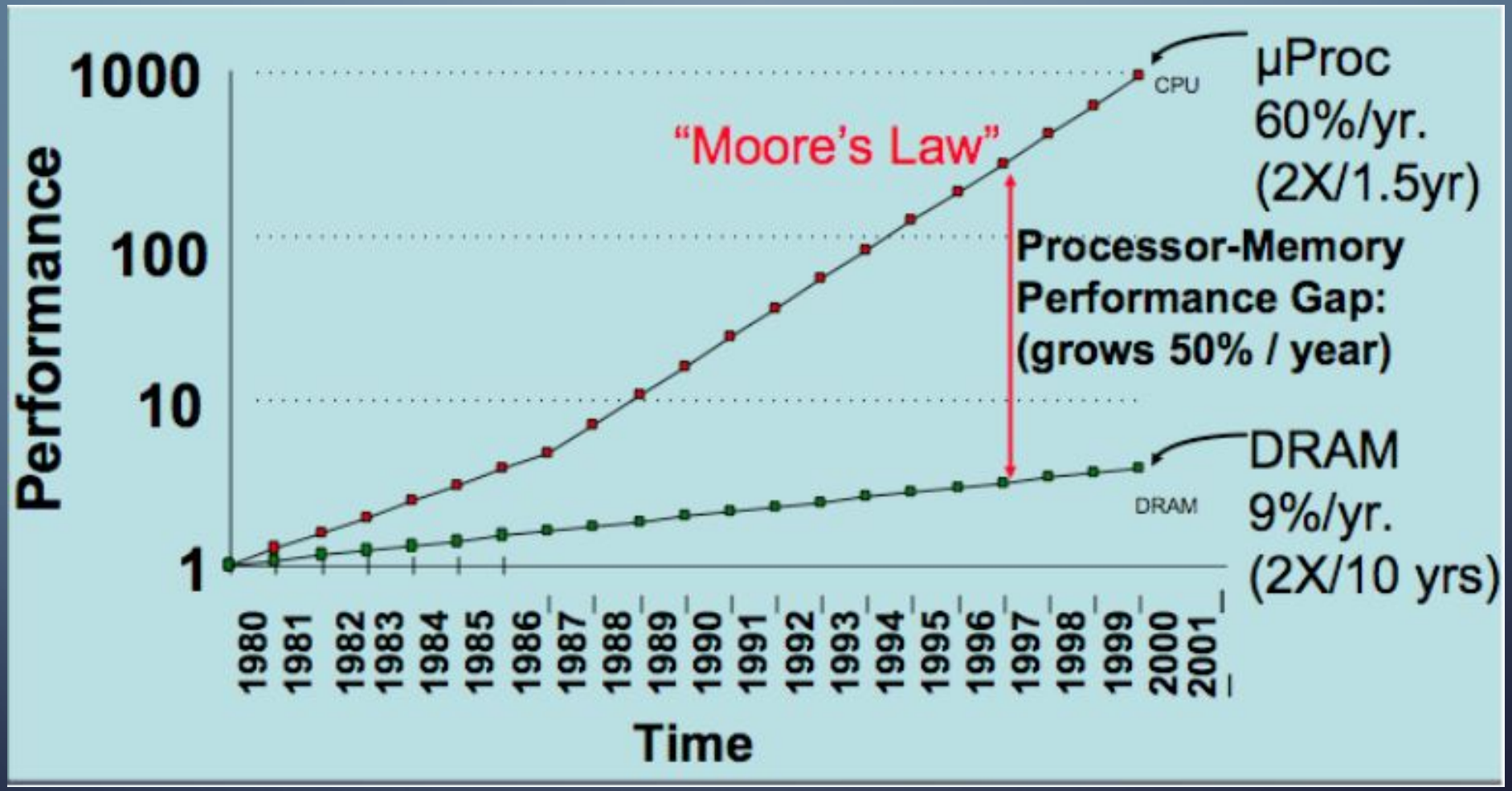
- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work

# LOCALITY AND PARALLELISM




- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast on average
- Parallel processors, collectively, have large, fast memories
  - the slow accesses to "remote" data we call "communication"
- Algorithm should do most work on local data

# LATENCY: PROCESSOR – DRAM GAP



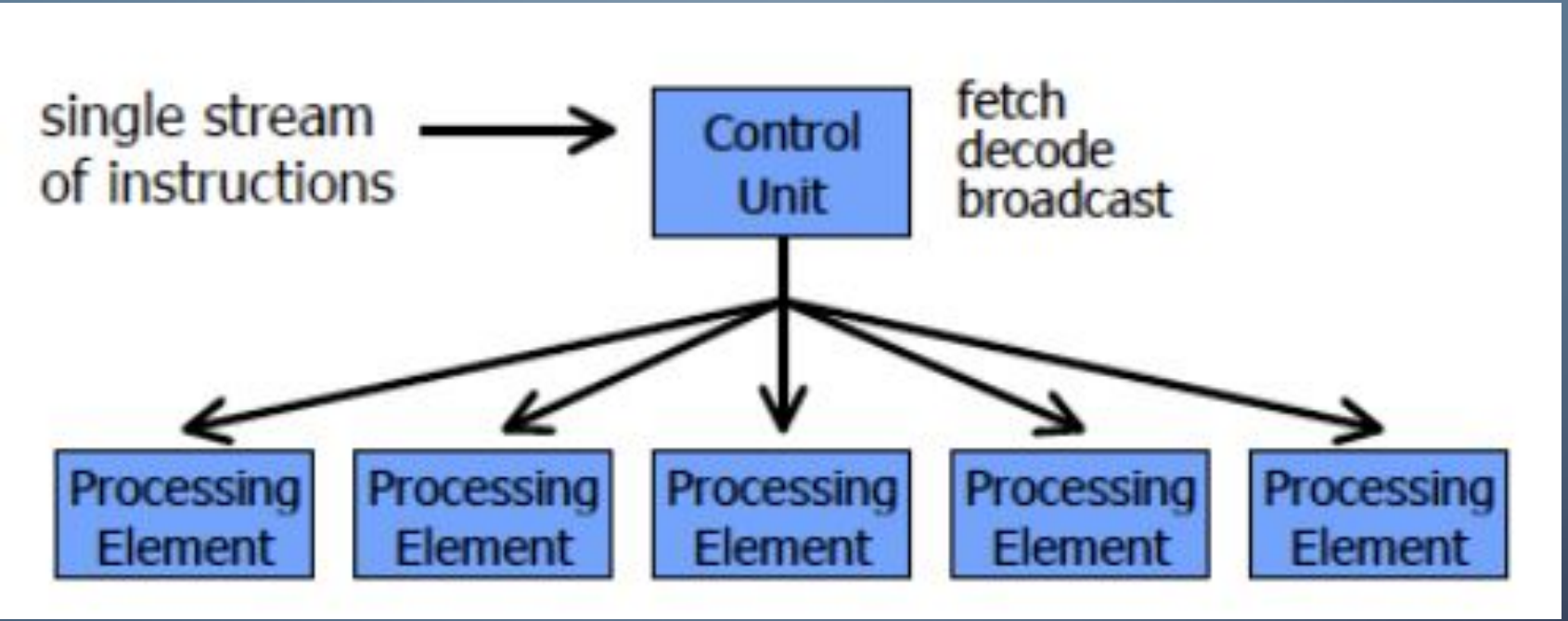
# LOAD IMBALANCE

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - adapting to “interesting parts of a domain”
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm needs to balance load
  - but techniques the balance load often reduce locality

# THE FLYNN TAXONOMY

- Proposed in 1966!!!
- Functional taxonomy based on the notion of streams of information: data and instructions
- Platforms are classified according to whether they have a single (S) or multiple (M) stream of each of the above
- Four possibilities:
  - SISD (sequential machine)
  - SIMD
  - MIMD
  - MISD (rare, no commercial system, systolic arrays)

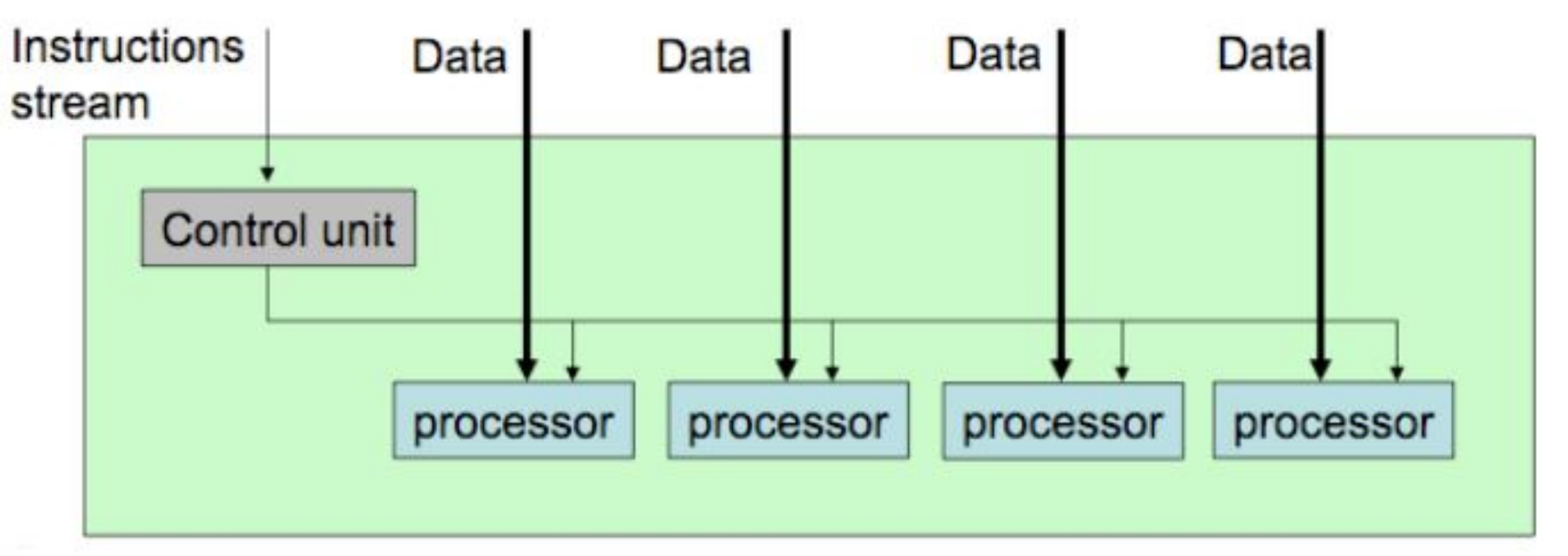
# SINGLE INSTRUCTION MULTIPLE DATA




- PEs can be deactivated and activated on-the-fly
- Vector processing (e.g., vector add) is easy to implement on SIMD

# SINGLE INSTRUCTION MULTIPLE DATA (CONT.)

- Also known as Array-processors
- A single instruction stream is broadcasted to multiple processors, each having its own data stream
  - Still used in graphics cards today



# MULTIPLE INSTRUCTION MULTIPLE DATA

- Most general category
- Pretty much everything in existence today is a MIMD machine at some level
- This limits the usefulness of the taxonomy
- But you had to have heard of it at least once because people keep referring to it, somehow...
- Other taxonomies have been proposed, none very satisfying
- Shared- vs. Distributed- memory is a common distinction among machines, but these days many are hybrid anyway

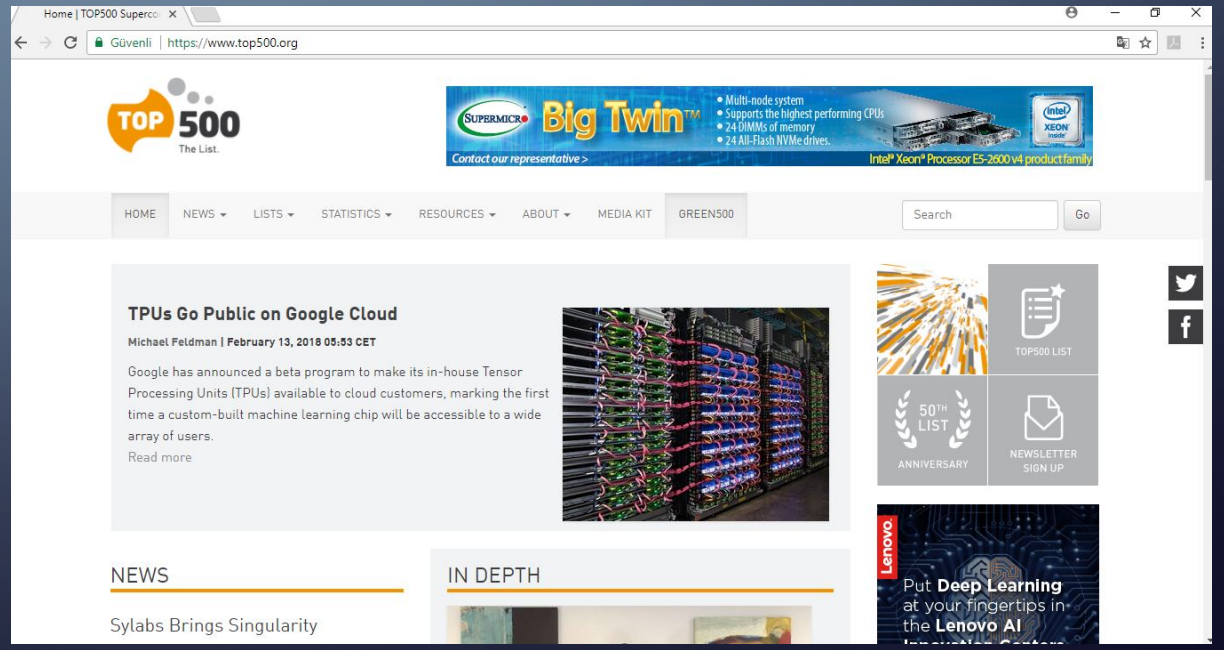
# MULTIPLE INSTRUCTION MULTIPLE DATA (CONT.)

- Each processing elements is capable of executing a different program independent of the other processors
- Most multiprocessor can be classified in this category)
- SPMD: Single program multiple data
- Typically more expensive hardware than SIMD but more general purpose

# A HOST OF PARALLEL MACHINES

- There are (have been) many kinds of parallel machines
- For the last 15 years their performance has been measured and recorded with the LINPACK benchmark, as part of Top500
- It is a good source of information about what machines are (were) and how they have evolved

http://www.top500.org



# PERFORMANCE METRİCS FOR PARALLEL SYSTEMS: SPEEDUP

- *Speedup (S)* is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with *p* identical processing elements.

# SPEEDUP FACTOR

$$S(p) = \frac{\text{Execution time using one processor (best sequential algorithm)}}{\text{Execution time using a multiprocessor with } p \text{ processors}} \quad \frac{t_s}{t_p}$$

where $t_s$ is execution time on a single processor and $t_p$ is execution time on a multiprocessor.

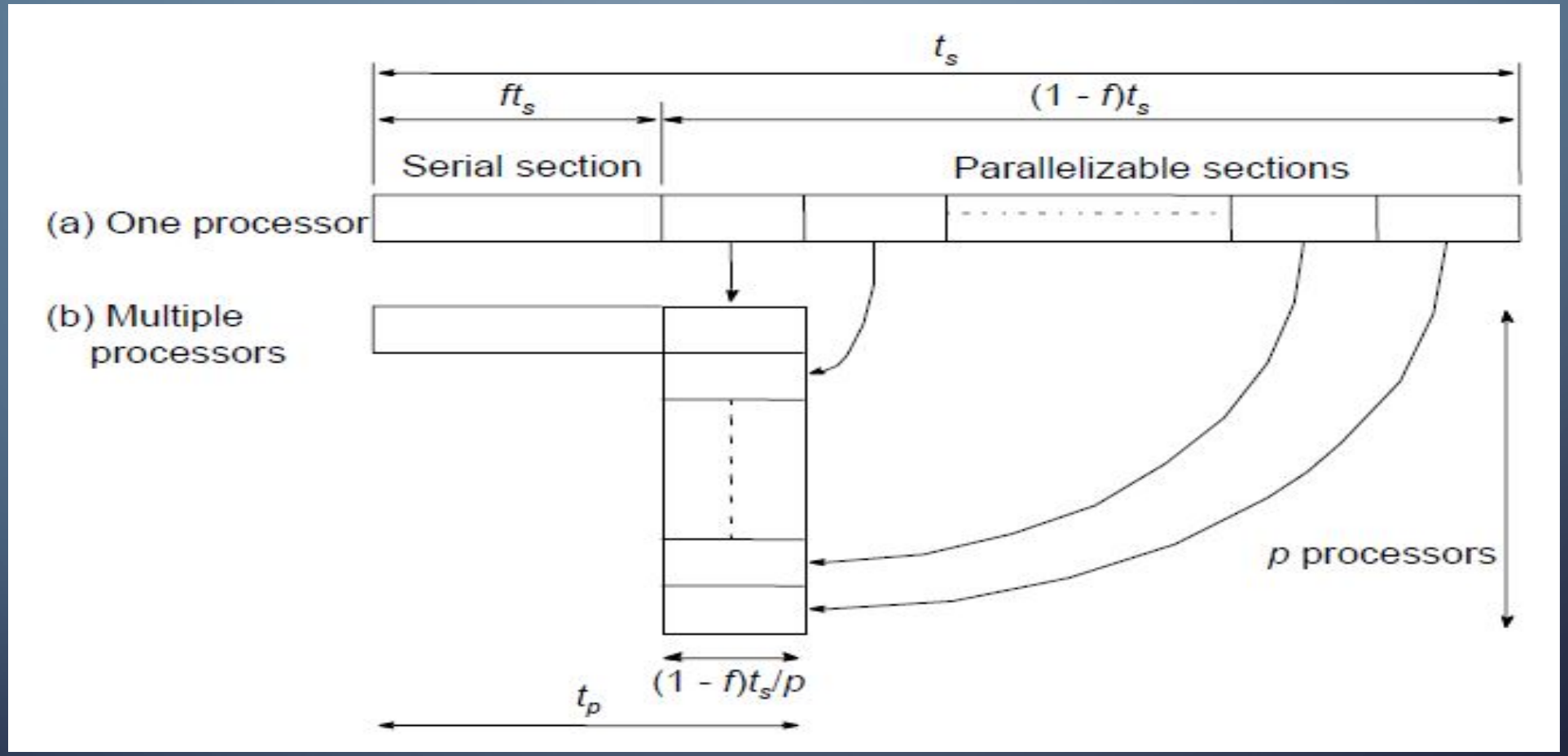
$S(p)$ gives increase in speed by using multiprocessor.

Use best sequential algorithm with single processor system. Underlying algorithm for parallel implementation might be (and is usually) different.

# MAXIMUM SPEEDUP AND SUPERLINEARITY

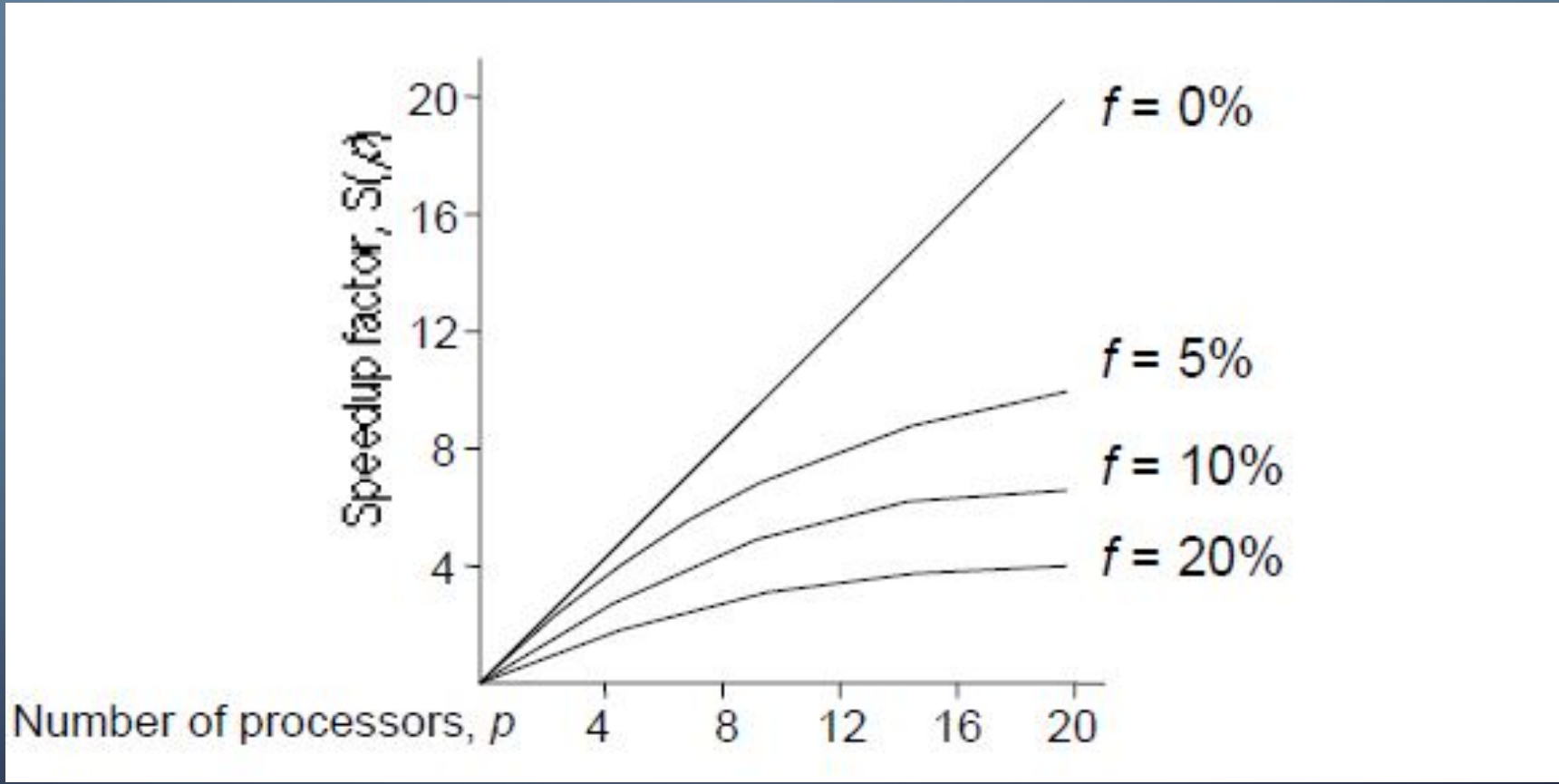
- Maximum speedup is usually *p* with *p* processors (linear speedup).
- Possible to get superlinear speedup (greater than *p*) but usually a specific reasons such as:
  - A parallel version does less work than the corresponding serial algorithm.
  - Resource-based superlinearity: The higher aggregate cache/ memory bandwidth can result in better cache-hit ratios, and therefore superlinearity.

# MAXIMUM SPEEDUP: AMDAHL'S LAW




$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1 + (p-1)f}$$

# SPEEDUP AGAINST NUMBER OF PROCESSORS



# EXAMPLE OF AMDAHL'S LAW

- Suppose that a calculation has a 4% serial portion, what is the limit of  speedup on 16 processors?
  - *16 / ( 1 + (16 – 1) * .04) = 10*
- What is the maximum speedup?
  - *1 / 0.04 = 25*

# EFFICIENCY

- The parallel efficiency of an application is defined as *Efficiency(p) = Speedup(p)/p*
  - Efficiency(p) <= 1
  - For perfect speedup *Efficiency(p) = 1*
  - For superlinear speedup *Efficiency(p)>1*
- We will rarely have perfect speedup.
  - Lack of perfect parallelism in the application or algorithm
  - Imperfect load balancing (some processors have more work)
  - Cost of communication
  - Cost of contention for resources, e.g., memory bus, I/O
  - Synchronization time
- Understanding why an application is not scaling linearly will help finding ways improving the applications performance on parallel computers.

# MORE ON AMDAHL'S LAW

- Amdahl's law works on a fixed problem size
  - This is reasonable if your only goal is to solve a problem faster.
  - What if you also want to solve a larger problem?
    - Gustafson's Law (Scaled Speedup)

# GUSTAFSON'S LAW

- Fix execution time on a single processor
  - s + p = serial part + parallel part = 1 (normalized serial time)
  - (s = same as f previously)
  - Assume problem fits in memory of serial computer
- Fixed-size speedup (Amdahl's Law)

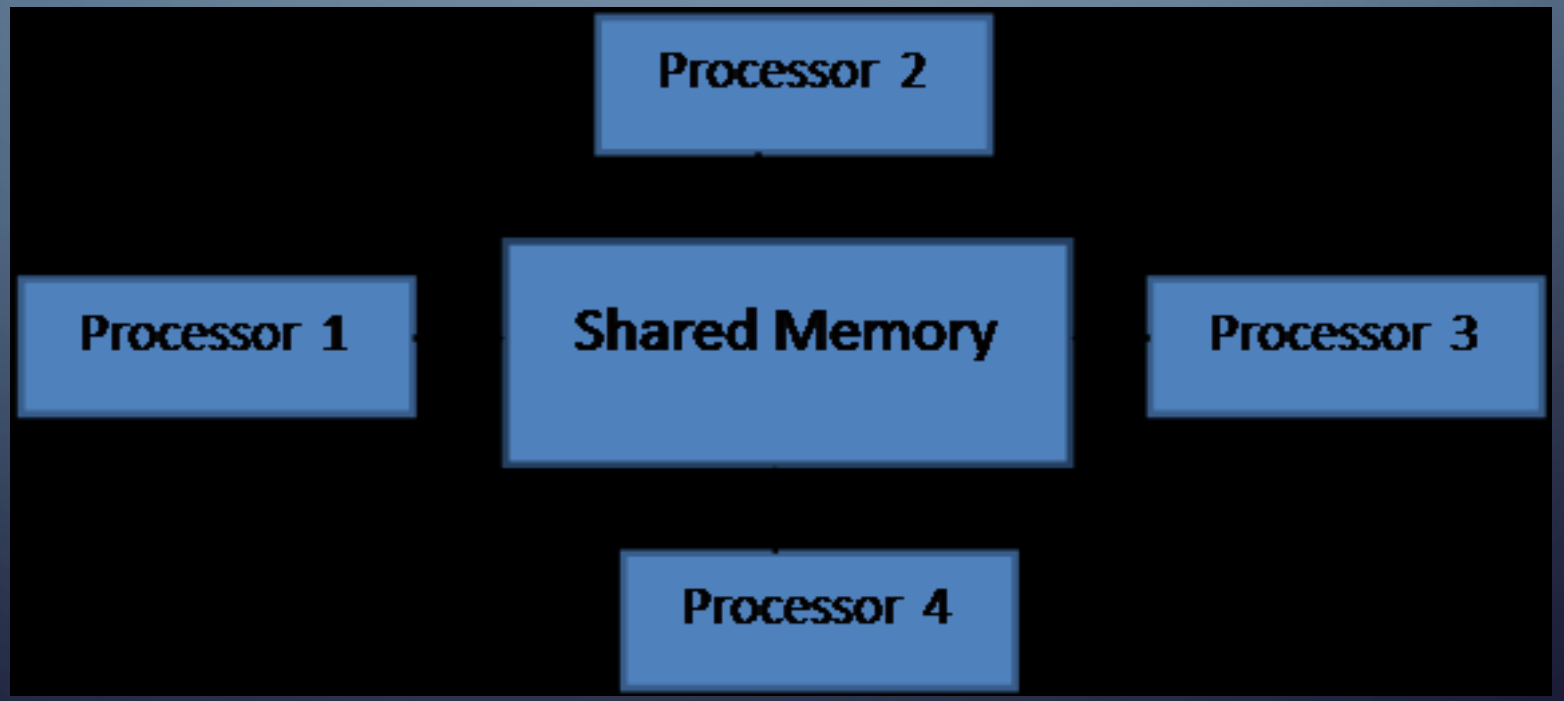
$$S_{fixed\_size} = \frac{s+p}{s+\frac{p}{n}} = \frac{1}{s+\frac{1-s}{n}}$$

- Fix execution time on a parallel computer
  - s + p = serial part + parallel part = 1 (normalized parallel time)
  - s + np = serial time on a single processor
  - Assume problem fits in memory of parallel computer
- Scaled Speedup (Gustafson's Law)

$$S_{scaled} = \frac{s+np}{s+p} = n+(1-n)s$$

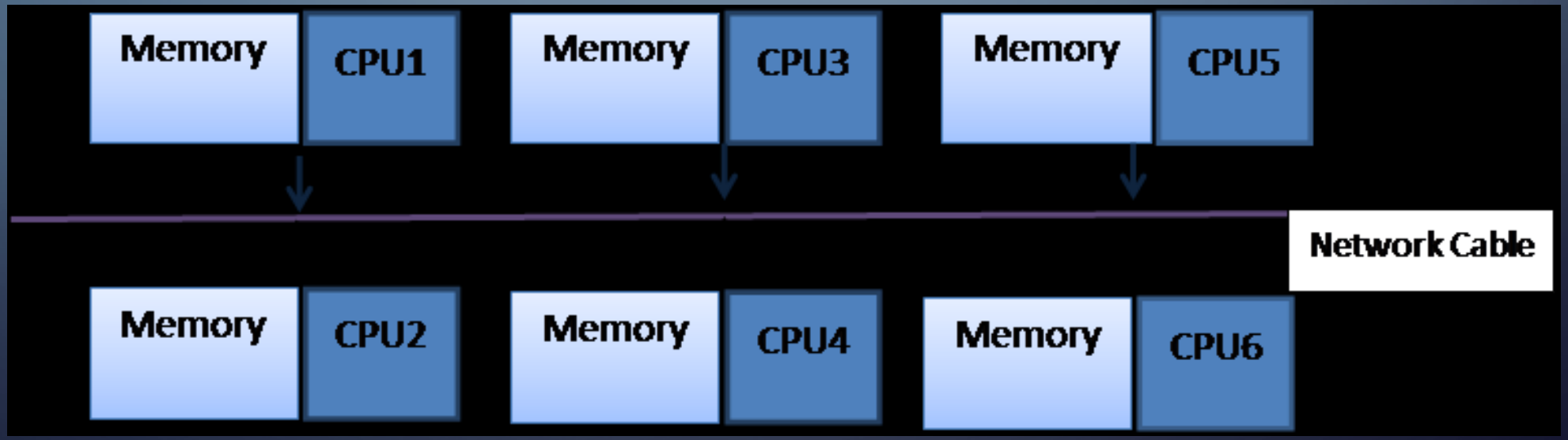
# SHARED MEMORY

- Shared memory is one where all processors can see whole of the memory that is available. Simple example is your desktop computer or laptop, where all processing units can see all the memory of system



# DISTRIBUTED MEMORY

- Distributed memory system is one where processor can see limited memory i.e two desktop computer connected in network . They can only see memory available to them only not of other



# WHAT IS PROCESS AND THREAD

- A Process is an executing instance of program. It has distinct address space. It different from other executing instance of program in way that it has separate resources .

- Thread is subset of a process. A process can have *n* number of threads as desired. Every thread of process share its all resources i.e. data as well address space of process that created it. Thread has to be part of some process. It can not be independent .

# MPI (MESSAGE PASSING INTERFACE)

- MPI stands for Message Passing Interface . These are available as API(Application programming interface) or in library form for C,C++ and FORTRAN.
- Different MPI's API are available in market i.e OpenMPI, MPICH, HP-MPI, Intel MPI, etc. Whereas many are freely available like OpenMPI, MPICH etc. , other like Intel MPI comes with license i.e. you need to pay for it .
- One can use any one of above to parallelize programs . MPI standards maintain that all of these APIs provided by different vendors or groups follow similar standards, so all functions or subroutines in all different MPI API follow similar functionality as well arguments.
- The difference lies in implementation that can make some MPIs API to be more efficient than other. Many commercial CFD-Packages gives user option to select between different MPI API. However HP-MPI as well Intel MPIs are considered to be more efficient in performance.
- When MPI was developed, it was aimed at distributed memory system but now focus is both on distributed as well shared memory system. However it does not mean that with MPI , one cannot run program on shared memory system, it just that earlier, we could not take advantage of shared memory but now we can with latest MPI 3

# OPENMP (OPEN MULTIPROCESSING)

- OpenMP stand for Open Multiprocessing . OpenMP is basically an add on in compiler. It is available in gcc (gnu compiler) , Intel compiler and with other compilers.
- OpenMP target shared memory systems i.e. where processor shared the main memory.
- OpenMP is based on thread approach . It launches a single process which in turn can create n number of thread as desired.  It is based on what is called "fork and join method" i.e. depending on particular task it can launch desired number of thread as directed by user.
- Programming in OpenMP is relatively easy and involve adding *pragma* directive . User need to tell number of thread it need to use.  (Note that launching more thread than number of processing unit available can actually slow down the whole program )

# MPI VS OPENMP