Problem Solving with Computer and Algorithm Development

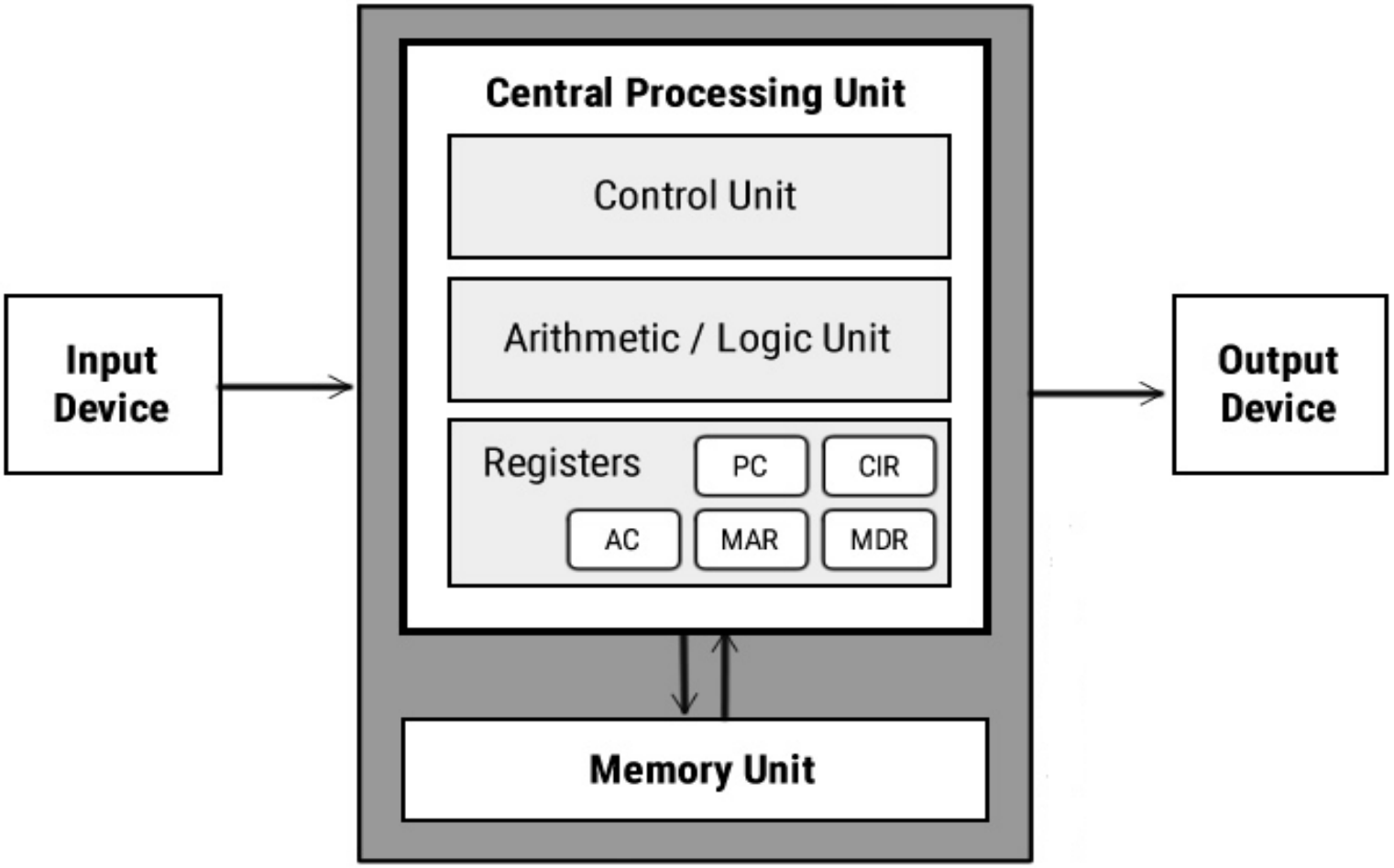
Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

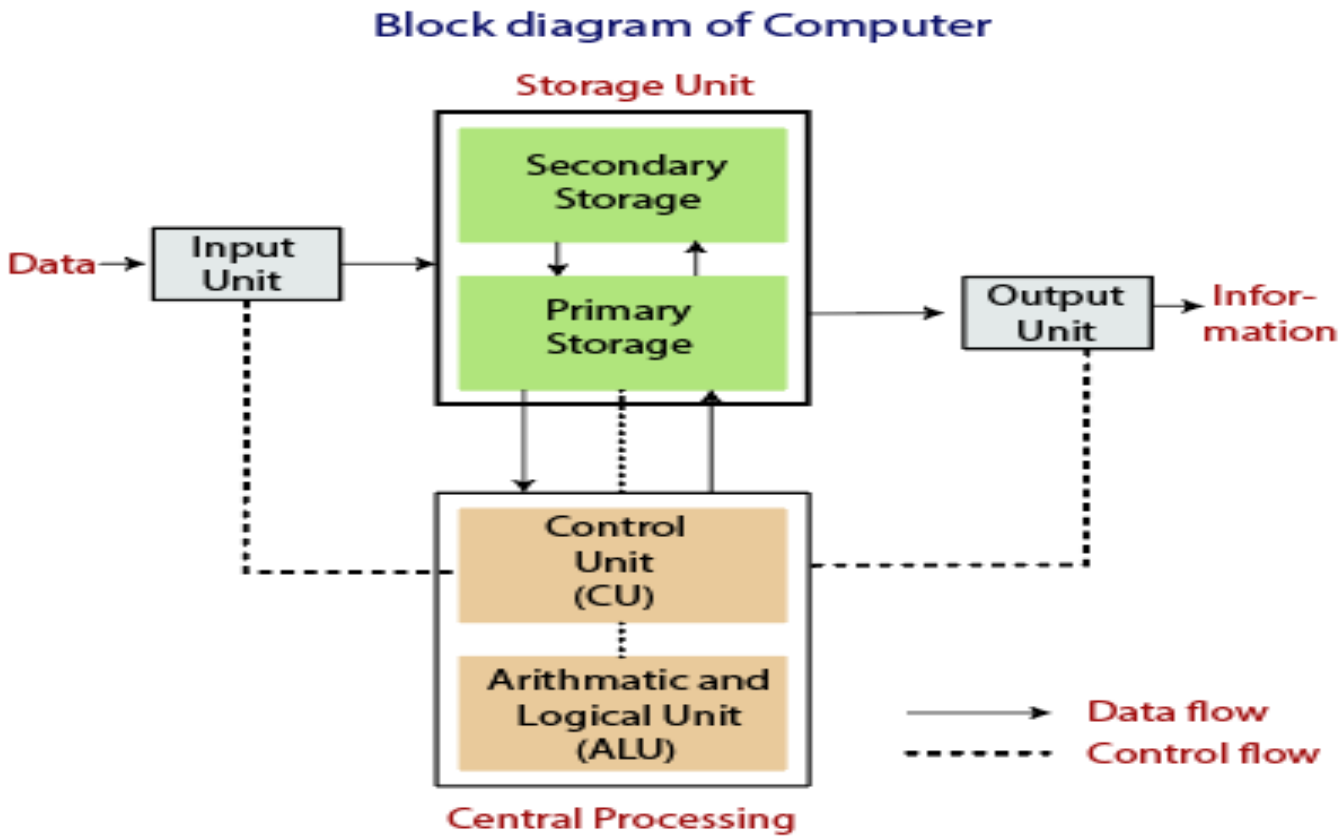
Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Von Neumann Architecture

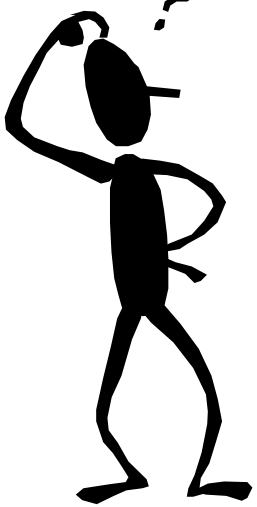


Von Neumann Architecture

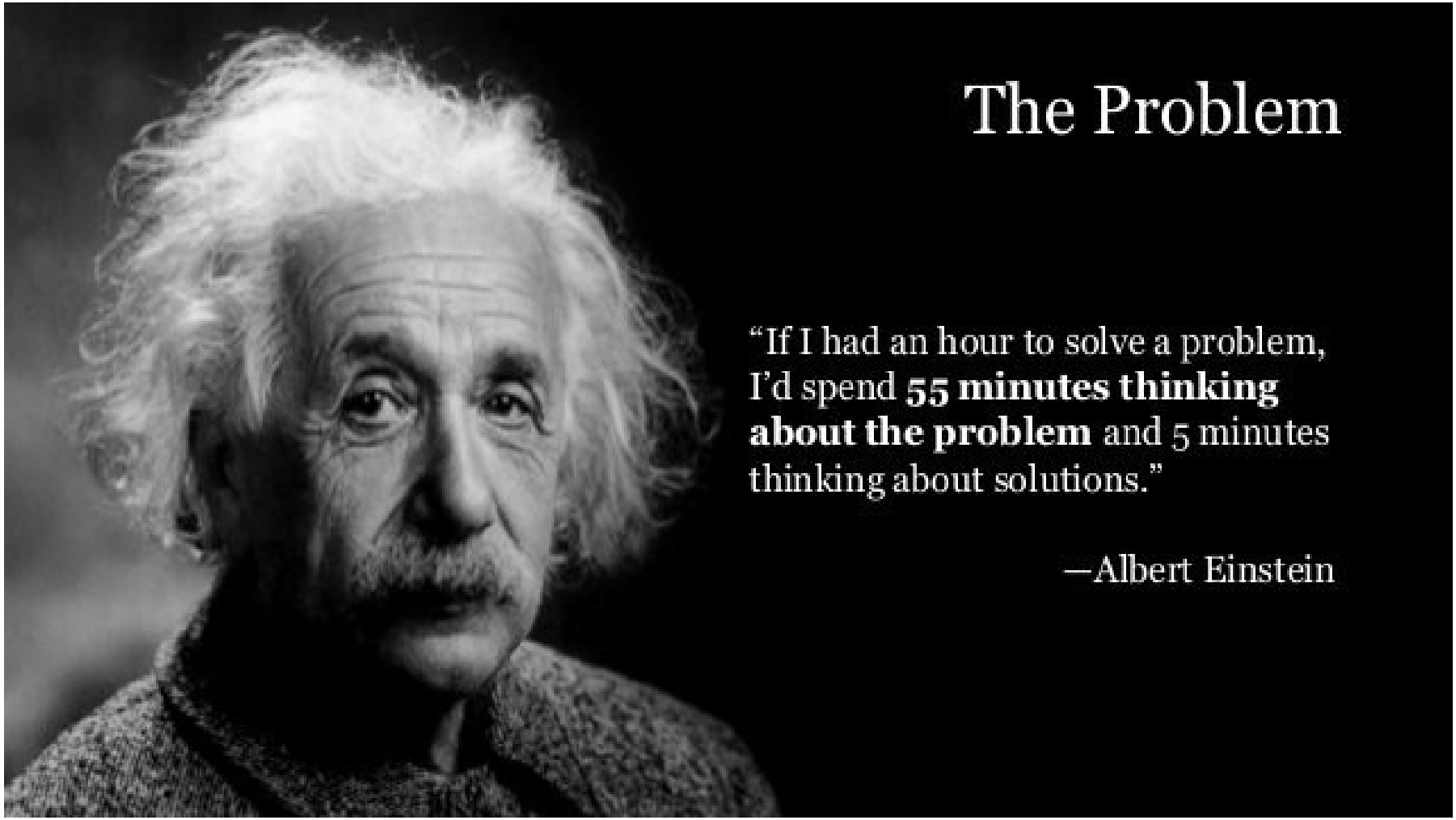
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory.



Problem ?



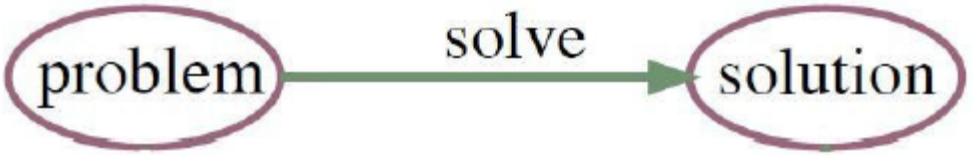
If we are solving some problem, we are usually looking for some solution, which will be the best among others.


The Problem
"If I had an hour to solve a problem, I'd spend **55 minutes thinking about the problem** and 5 minutes thinking about solutions."
—Albert Einstein

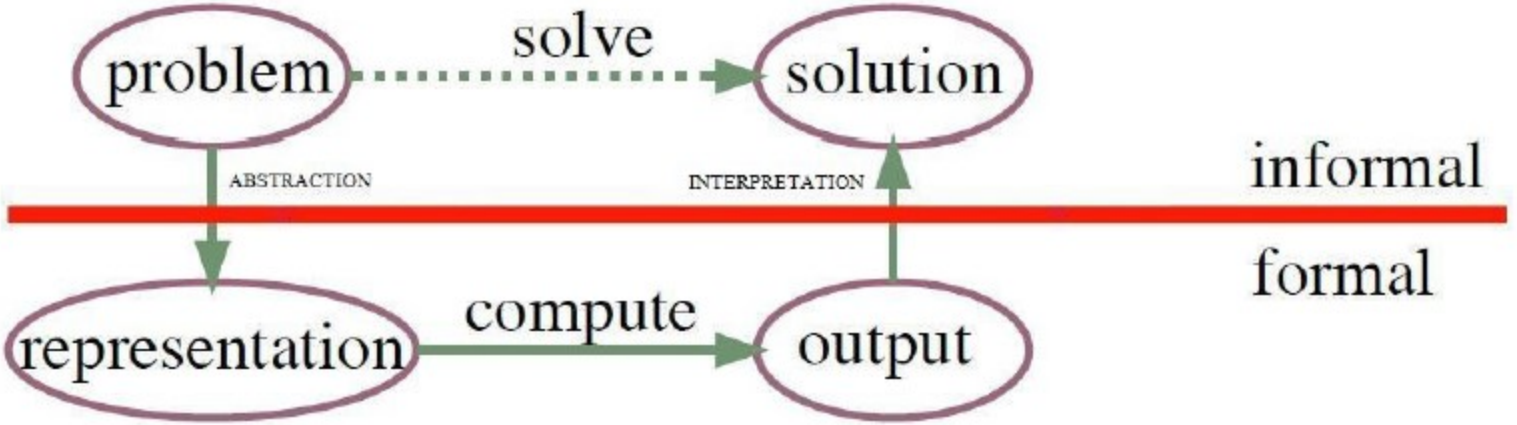
What is Problem solving ?

We have a problem and want to find a solution !

Ideally

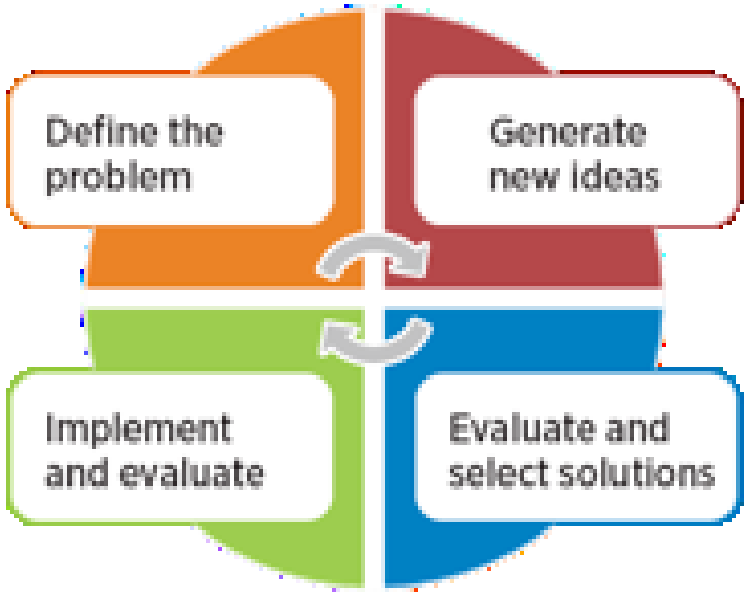


In Practice



What is problem solving?

- Problem solving is the act of **defining a problem**; determining the cause of the problem; identifying, prioritizing, and selecting alternatives for a solution; and implementing a solution.



The problem-solving process

Step	Characteristics
1. Define the problem	•Differentiate fact from opinion •Specify underlying causes •Consult each faction involved for information •State the problem specifically •Identify what standard or expectation is violated •Determine in which process the problem lies •Avoid trying to solve the problem without data
2. Generate alternative solutions	•Postpone evaluating alternatives initially •Include all involved individuals in the generating of alternatives •Specify alternatives consistent with organizational goals •Specify short- and long-term alternatives •Brainstorm on others' ideas •Seek alternatives that may solve the problem
3. Evaluate and select an alternative	•Evaluate alternatives relative to a target standard •Evaluate all alternatives without bias •Evaluate alternatives relative to established goals •Evaluate both proven and possible outcomes •State the selected alternative explicitly
4. Implement and follow up on the solution	•Plan and implement a pilot test of the chosen alternative •Gather feedback from all affected parties •Seek acceptance or consensus by all those affected •Establish ongoing measures and monitoring •Evaluate long-term results based on final solution

5-steps to Problem Solving

1. **Define the problem.**

- In understanding and communicating the problem effectively, we have to be clear about what the issue is. Remember to focus on the behaviour instead of attacking the person.

2. **Gather information.**

- What were the circumstances? What are the non-verbal messages being sent?
- Who does it affect? What behaviour is typical for the child's age? Are your expectations reasonable?

3. **Generate possible solutions.**

- Work together to brainstorm on all possible solutions. Do not judge whether the ideas are good or bad at this point.

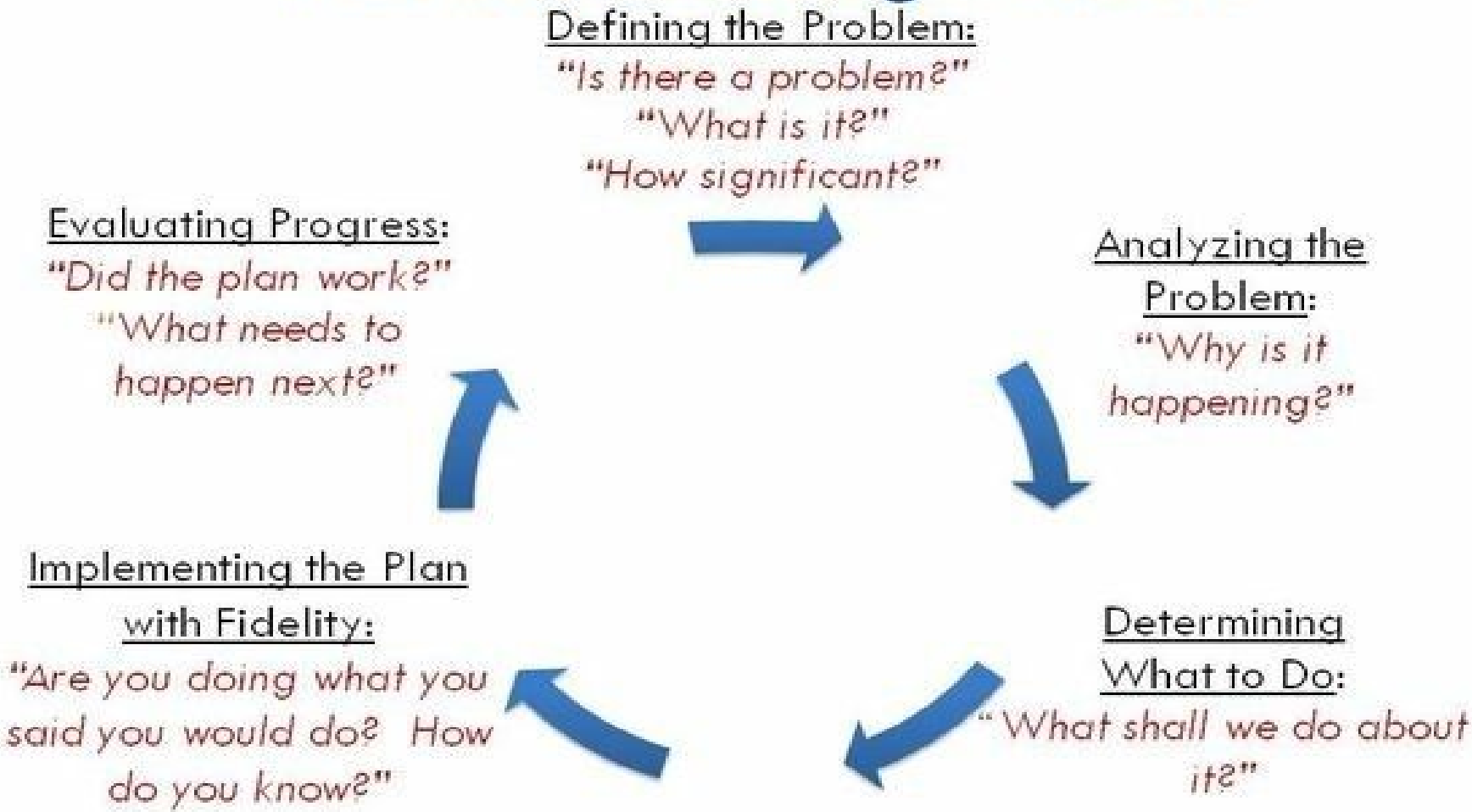
4. **Evaluate ideas and then choose one.**

- Decide which options you like and which you don't like. After weighing the pros and cons of each, choose an option that you both feel comfortable with. Once a solution has been found to meet everyone's needs while keeping everyone's self-respect and self-esteem intact, then make a plan to follow through and do it. Put down a time-frame for action.

5. **Evaluate.**

- Did it work? If yes, great! Consider how your solution may be applicable to other different problems. Ask how the problem can be prevented from happening again.
- What if it didn't work? Go back to step one or try out the other possible solutions that you and your child made in step 3.

Problem Solving Method



Problem solving

Real-World Problem

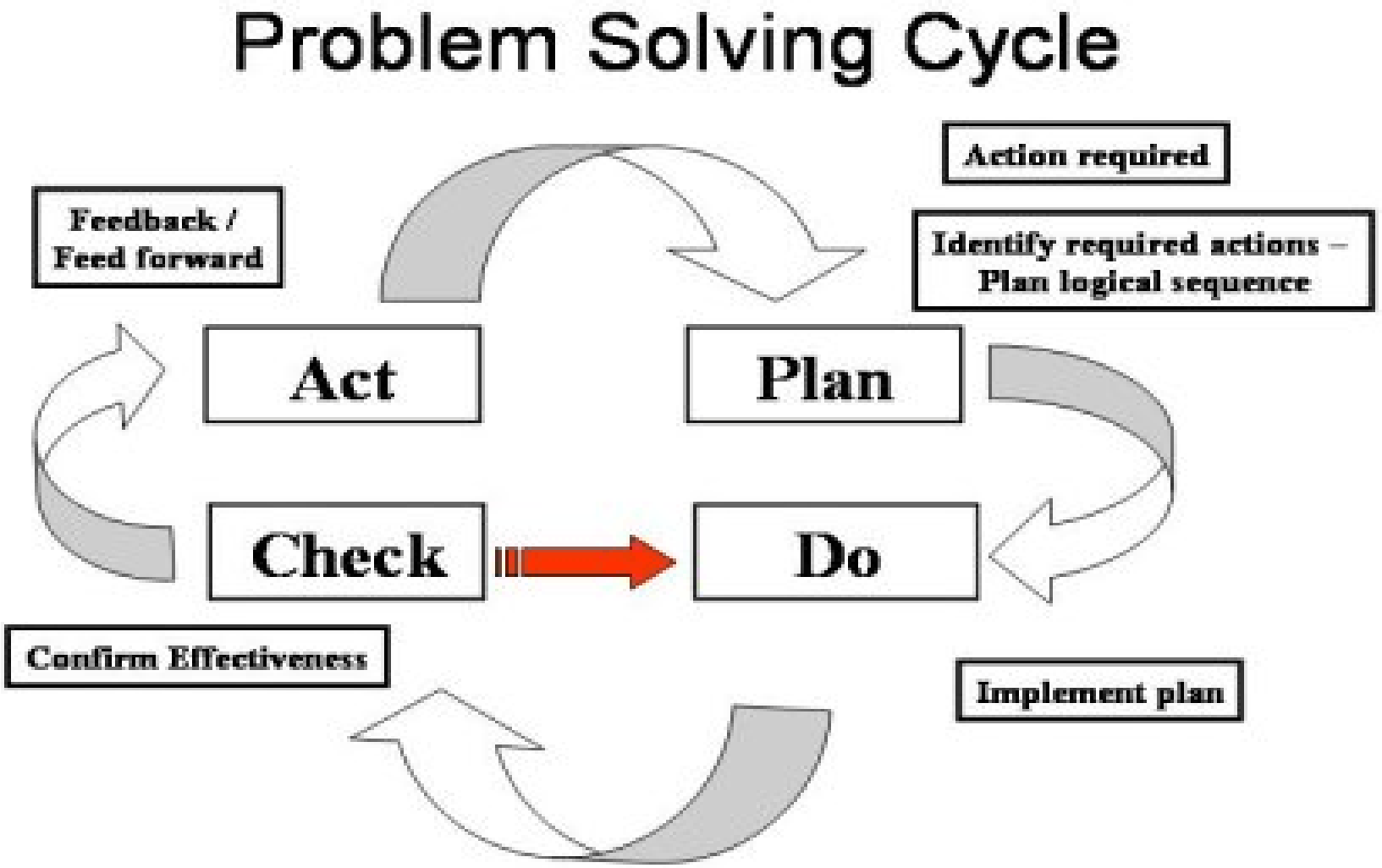
Formulation of Abstract Problem

Solve the Abstract Problem

Interpret the Solution

Implement the Solution

Problem Solving Cycle



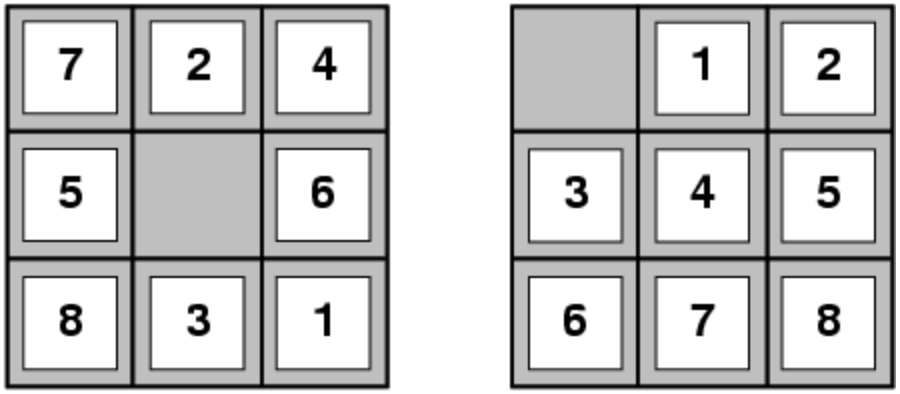
Two types of problem approaches

- **Toy Problem:** It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.

- **Real-world Problem:** It is real-world based problems which require solutions. Unlike a toy problem, it does not depend on descriptions, but we can have a general formulation of the problem.

Example: Toy Problem

- **8 Puzzle Problem:** Here, we have a 3×3 matrix with movable tiles numbered from 1 to 8 with a blank space. The tile adjacent to the blank space can slide into that space.
- In the figure, our task is to convert the current state into goal state by sliding digits into the blank space.



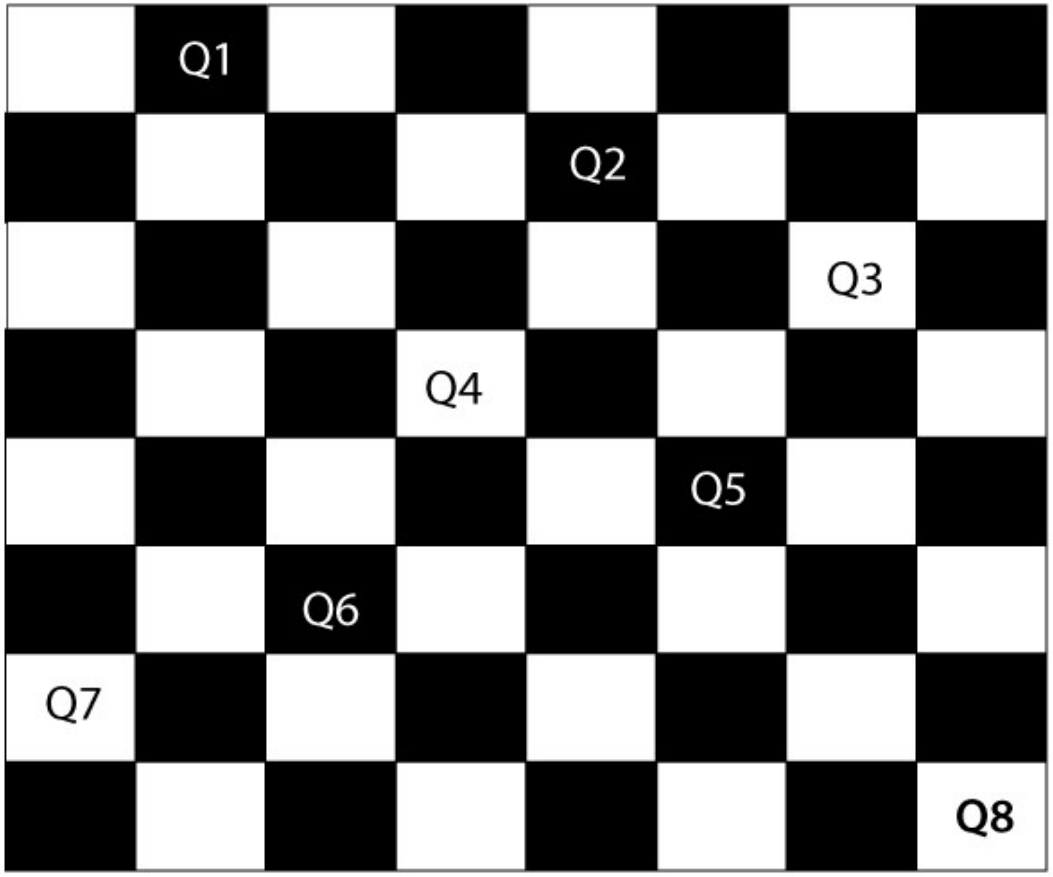
8 Puzzle Problem formulation

1. **States:** It describes the location of each numbered tiles and the blank tile.
2. **Initial State:** We can start from any state as the initial state.
3. **Actions:** Here, actions of the blank space is defined, i.e., either **left, right, up or down**
4. **Transition Model:** It returns the resulting state as per the given state and actions.
5. **Goal test:** It identifies whether we have reached the correct goal-state.
6. **Path cost:** The path cost is the number of steps in the path where the cost of each step is 1.

- **Note:** The 8-puzzle problem is a type of **sliding-block problem** which is used for testing **new search algorithms** in **artificial intelligence.**

8-queens problem

- **8-queens problem:** The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either **diagonally or in same row and column.**



Problem formulation : [two main kinds of formulation]

[1] Incremental formulation: It starts from an empty state where the operator augments a queen at each step.

Following steps are involved in this formulation:

1. **States:** Arrangement of any 0 to 8 queens on the chessboard.
2. **Initial State:** An empty chessboard
3. **Actions:** Add a queen to any empty box.
4. **Transition model:** Returns the chessboard with the queen added in a box.
5. **Goal test:** Checks whether 8-queens are placed on the chessboard without any attack.
6. **Path cost:** There is no need for path cost because only final states are counted.

- In this formulation, there is approximately **1.8×10^{14}** possible sequence to investigate.

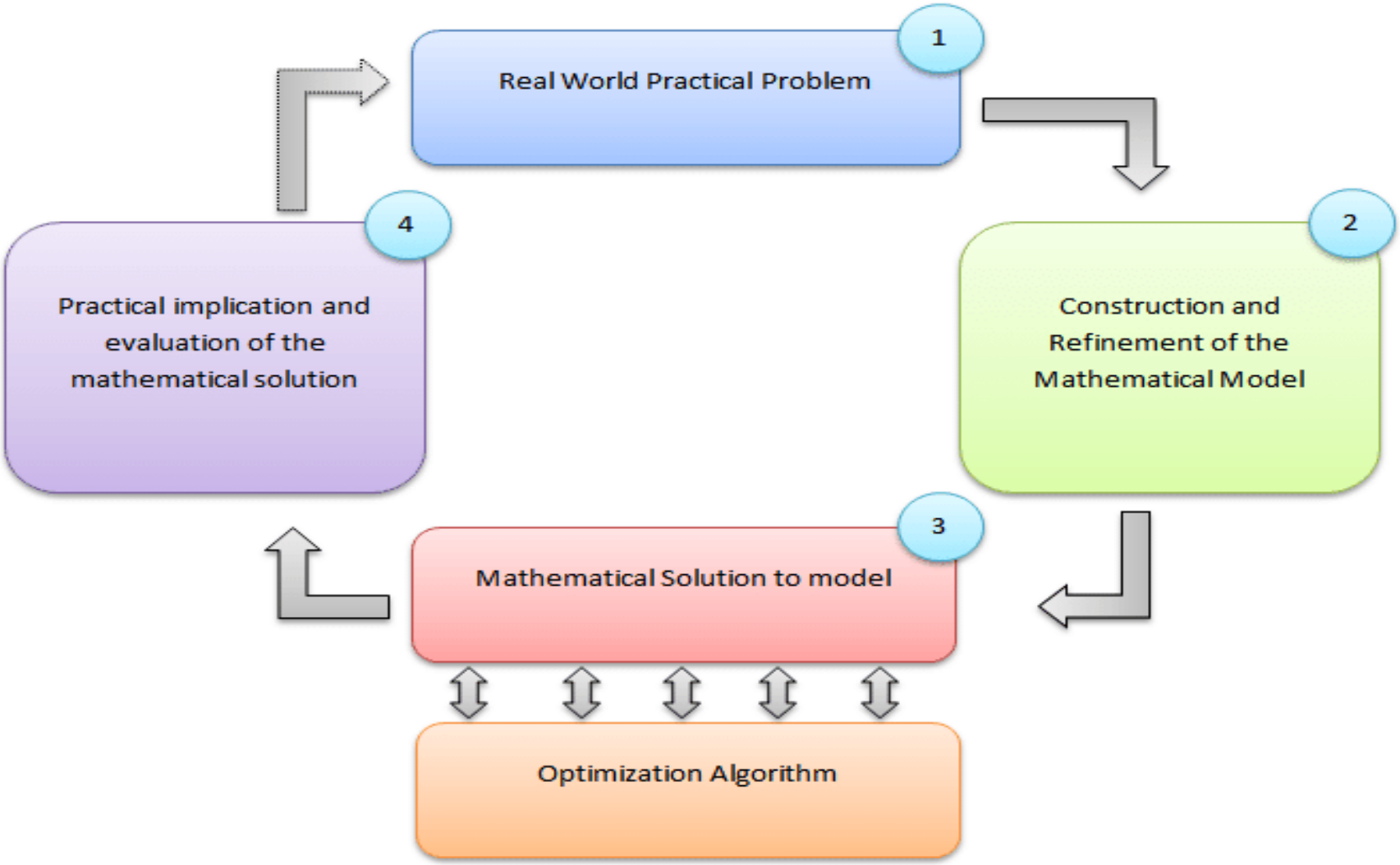
Problem formulation : [two main kinds of formulation]

[2] Complete-state formulation: It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.

Following steps are involved in this formulation

- **States:** Arrangement of all the 8 queens one per column with no queen attacking the other queen.
- **Actions:** Move the queen at the location where it is safe from the attacks.
- This formulation is better than the incremental formulation as it reduces the state space from **1.8×10^{14}** to **2057**, and it is easy to find the solutions.

Solution to Real world problem



Some Real-world problems

1. **Traveling salesperson problem(TSP):** It is a **touring problem** where the salesman can visit each city only once. The objective is to find the shortest tour and sell-out the stuff in each city.
2. **VLSI Layout problem:** In this problem, millions of components and connections are positioned on a chip in order to minimize the area, circuit-delays, stray-capacitances, and maximizing the manufacturing yield. The layout problem is split into two parts:
   - **Cell layout:** Here, the primitive components of the circuit are grouped into cells, each performing its specific function. Each cell has a fixed shape and size. The task is to place the cells on the chip without overlapping each other.
   - **Channel routing:** It finds a specific route for each wire through the gaps between the cells.

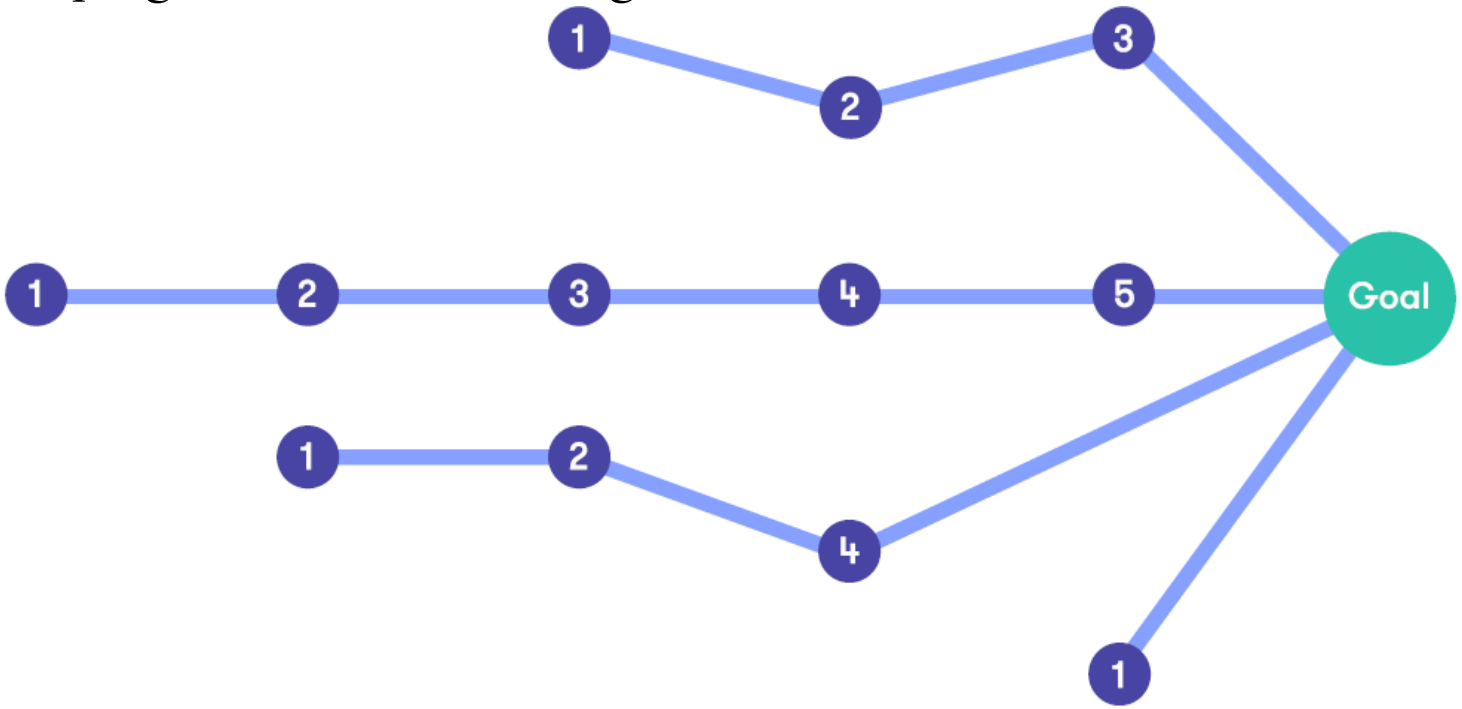
Search and problem solving

- Many problems can be phrased as search problems. This requires that we start by formulating the alternative choices and their consequences.
- **Search in practice: getting from A to B ?**

- This question belongs to the class of search and planning problems. Similar problems need to be solved by self-driving cars, and (perhaps less obviously) AI for playing games.

Many different ways to solve the problem

- Often there are many different ways to solve the problem, some of which may be more preferable in terms of time, effort, cost or other criteria.
- Different **search techniques** may lead to different solutions, and developing advanced search algorithms is an established research area.



Searching for solutions

- There may be many solutions to a particular problem. If you can think of the task you want your agent to perform in these terms, then you will need to write a problem solving **agent** which uses search.
- In Artificial Intelligence, Search techniques are universal problem-solving methods. Rational agents or Problem-solving agents in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result.
- Problem-solving agents are the goal-based agents and use atomic representation.

Problem Taxonomy

Problem Taxonomy

- **Un-decidable Problem:** Do not have algorithms of any known complexity (polynomial or super-polynomial)

- **Intractable problem:** Have algorithms with super polynomial time complexity.

- **Tractable problem:** Have good algorithms with polynomial time complexity (irrespective of the degree)

Different type of decidable Problem

- **Decision Problems:** The class of problems, the output is either yes or no.
- **Example** : Whether a given number is prime?

- **Counting Problem:** The class of problem, the output is a natural number
- **Example** :How many distinct factor are there for a given number

- **Optimization Problem:** The class of problem with some objective function based on the problem instance
- **Example** :**Finding a minimal spanning tree** for a weighted graph

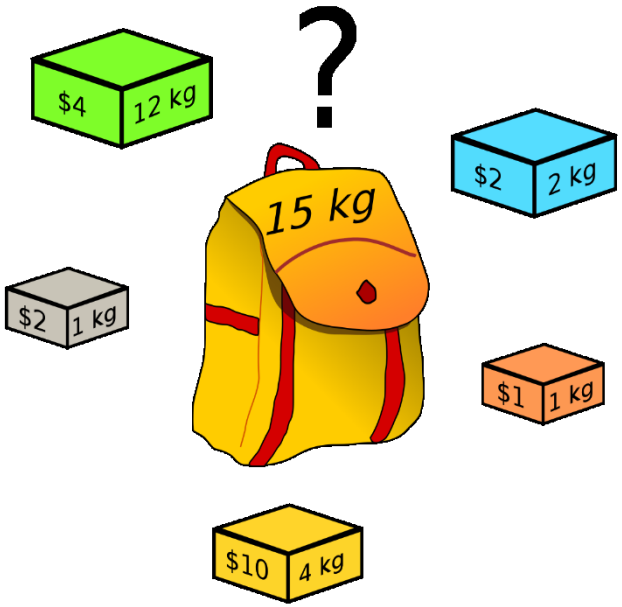
Three general categories of intractable problems

1. Problems for which polynomial-time algorithms have been found .
2. Problems that have been proven to be intractable.
3. Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found .

- It is a surprising phenomenon that most problems in computer sciences seem to fall into either the first or third category.

0-1 Knapsack Problem

0-1 Knapsack Problem
Given a set of items, each with a weight and benefit, determine the items to include in a collection so that the total weight is less than or equal to a given weight limit and the total benefit is maximized



Knapsack problem; introduction

- The classical knapsack problem assumes that each item must be put entirely in the knapsack or not included at all. It is this 0/1 property that makes the knapsack problem difficult.
- In the **0/1 Knapsack problem**, you are given a bag which can hold W kg. There is an array of items each with a different weight and price value assigned to them.
- The objective of the 0/1 Knapsack is to **maximize the value** you put into the bag. You are allowed to only take all or nothing of a given item.

Knapsack problem

- *Knapsack* problem : Suppose we have n integers $a_1, a_2, \ldots, a_n$ and a constant W. We want to find a subset of integers so that their sum is less than or equal to but is as close to W as possible. There are 2^n subsets. ($n = 100$, $2^n = 1.26765 \times 10^{30}$, it takes 4.01969×10^{12} years if our computer can examine 10^{10} subsets per second.)

- A problem is considered *tractable* (computationally easy, $O(n^k)$) if it can be solved by an efficient algorithm and *intractable* (computationally difficult, the lower bound grows fast than n^k) if there is no efficient algorithm for solving it.

- The class of *NP-complete* problems: There is a class of problems, including TSP and Knapsack, for which no efficient algorithm is currently known.

Knapsack problem

There are two types of knapsack problem.

1. **0-1 knapsack problem**: In 0-1 knapsack problem each item either be taken or left behind.

2. **Fractional knapsack problem**: In fractional knapsack problem fractions of items are allowed to choose.

Knapsack problem; introduction

- The classical knapsack problem assumes that each item must be put entirely in the knapsack or not included at all. It is this 0/1 property that makes the knapsack problem difficult.
- In the **0/1 Knapsack problem**, you are given a bag which can hold W kg. There is an array of items each with a different weight and price value assigned to them.
- The objective of the 0/1 Knapsack is to **maximize the value** you put into the bag. You are allowed to only take all or nothing of a given item.

Knapsack problem

There are two types of knapsack problem.

1. **0-1 knapsack problem**: In 0-1 knapsack problem each item either be taken or left behind.

2. **Fractional knapsack problem**: In fractional knapsack problem fractions of items are allowed to choose.

0-1 Knapsack Problem

Given a knapsack of capacity C and n objects of volumes $\{w_1, w_2 \ldots w_n\}$ and profits $\{p_1, p_2 \ldots p_n\}$, the objective is to choose a subset of n objects that fits into the knapsack and that maximizes the total profit.

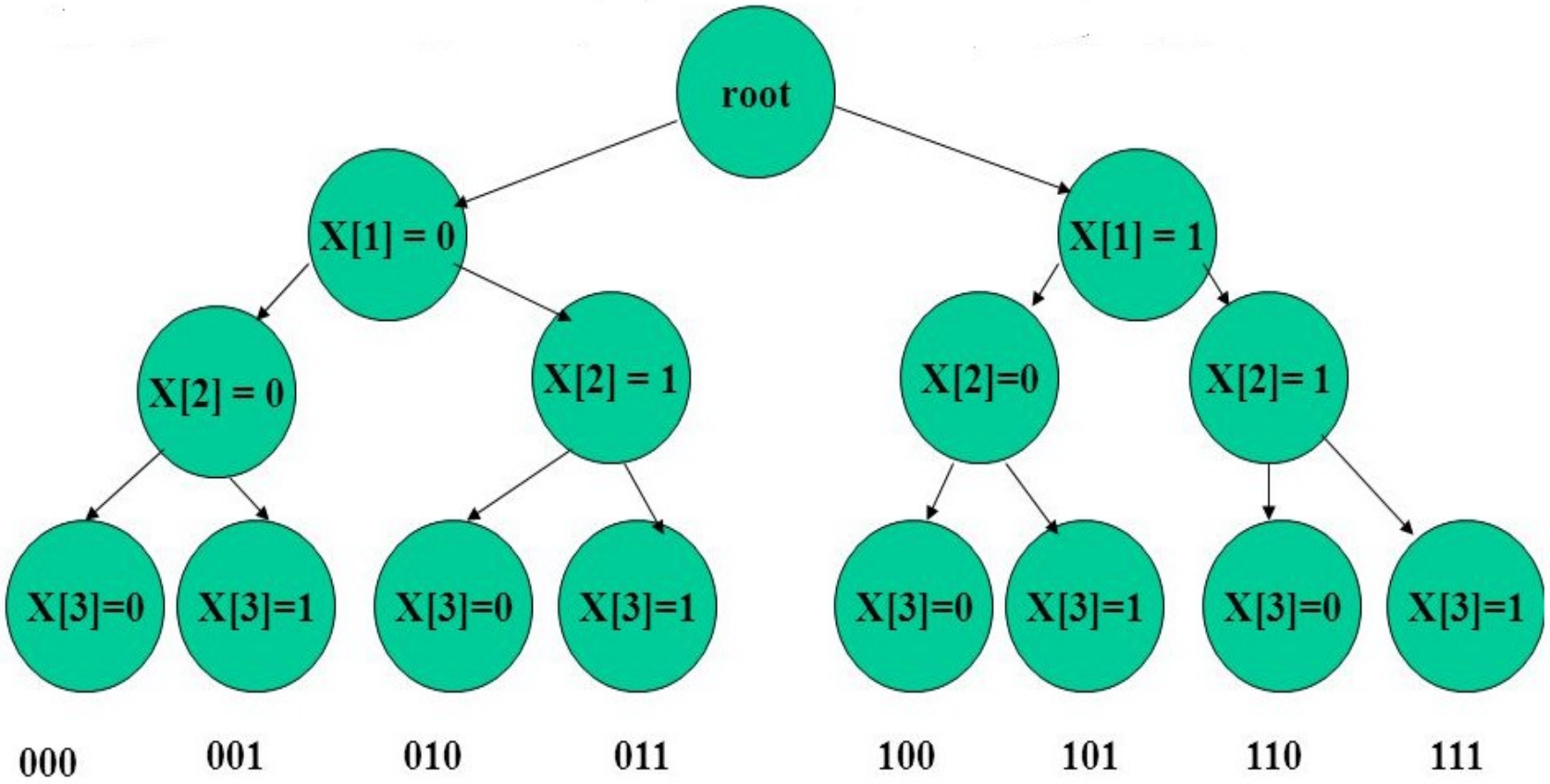
In more formal terms, let x_i be 1 if object i is present in the subset and 0 otherwise.

The knapsack problem can be stated as

$$\text{Maximize} \sum_{i=0}^{n} x_i \cdot p_i \quad \text{subject to} \quad \sum_{i=0}^{n} x_i \cdot w_i \leq C$$

Note that the constraint $x_i \in \{0, 1\}$ is not linear. A simplistic approach will be to enumerate all subsets and select the one that satisfies the constraints and maximizes the profits. Any solution that satisfies the capacity constraint is called a *feasible* solution. The obvious problem with this strategy is the running time which is at least 2^n corresponding to the power-set of n objects.

Solution tree for 3 item 0/1 knapsack problem



Knapsack problem

- *Knapsack* problem : Suppose we have n integers $a_1, a_2, \ldots, a_n$ and a constant W. We want to find a subset of integers so that their sum is less than or equal to but is as close to W as possible. There are 2^n subsets. ($n = 100$, $2^n = 1.26765 \times 10^{30}$, it takes 4.01969×10^{12} years if our computer can examine 10^{10} subsets per second.)

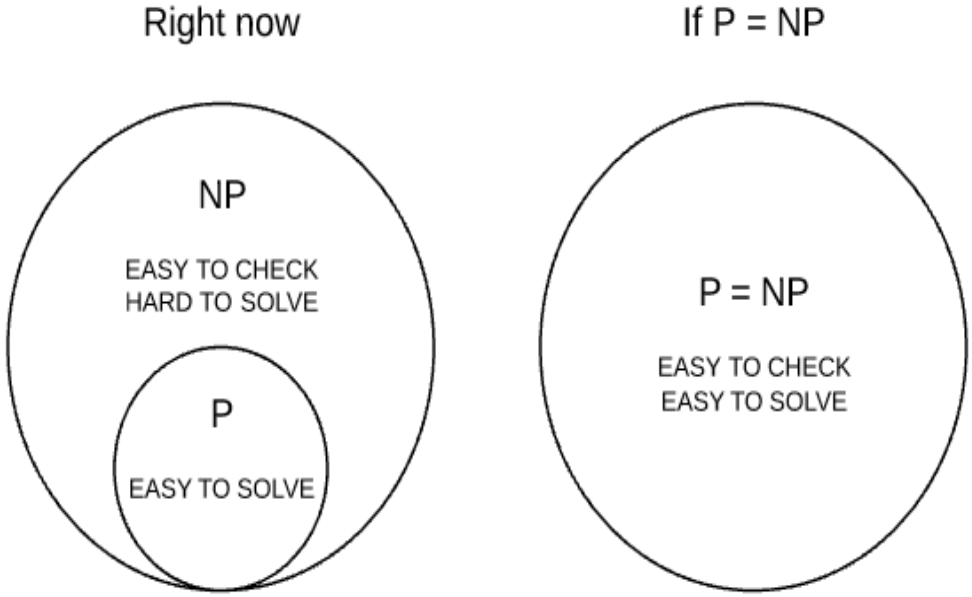
- A problem is considered *tractable* (computationally easy, $O(n^k)$) if it can be solved by an efficient algorithm and *intractable* (computationally difficult, the lower bound grows fast than n^k) if there is no efficient algorithm for solving it.

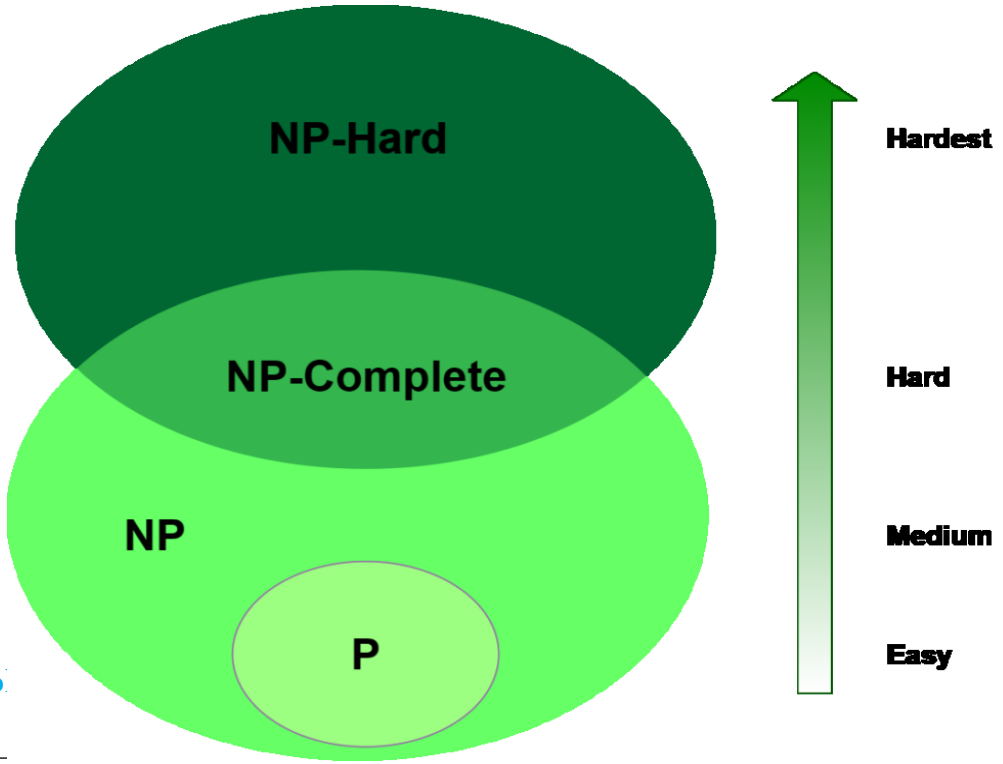
- The class of *NP-complete* problems: There is a class of problems, including TSP and Knapsack, for which no efficient algorithm is currently known.

knapsack problem is NP-complete

- The knapsack problem is NP-complete because the known NP-complete problem **subset-sum** is polynomially reducible to the **knapsack problem**, hence every problem in NP is reducible to the knapsack problem.
- The fact that we know an O(nW) algorithm for knapsack doesn't contradict the possibility that P≠NP because O(nW) is not polynomial in the length of the input to the algorithm, which is O(nlogW). If we ever did find an algorithm that ran in time polynomial in n and logW that would prove that P=NP.
- Algorithms that are polynomial in the magnitudes of coefficients in the input instead of their sizes are called *pseudo polynomial*. An NP-complete problem might or might not have a pseudo polynomial algorithm. Those that don't (assuming P≠NPP≠NP) are *strongly NP-complete*. The knapsack problem is not strongly NP-complete, but it is still NP-complete.

P vs NP Problem





0-1 knapsack problem is pseudo-polynomial

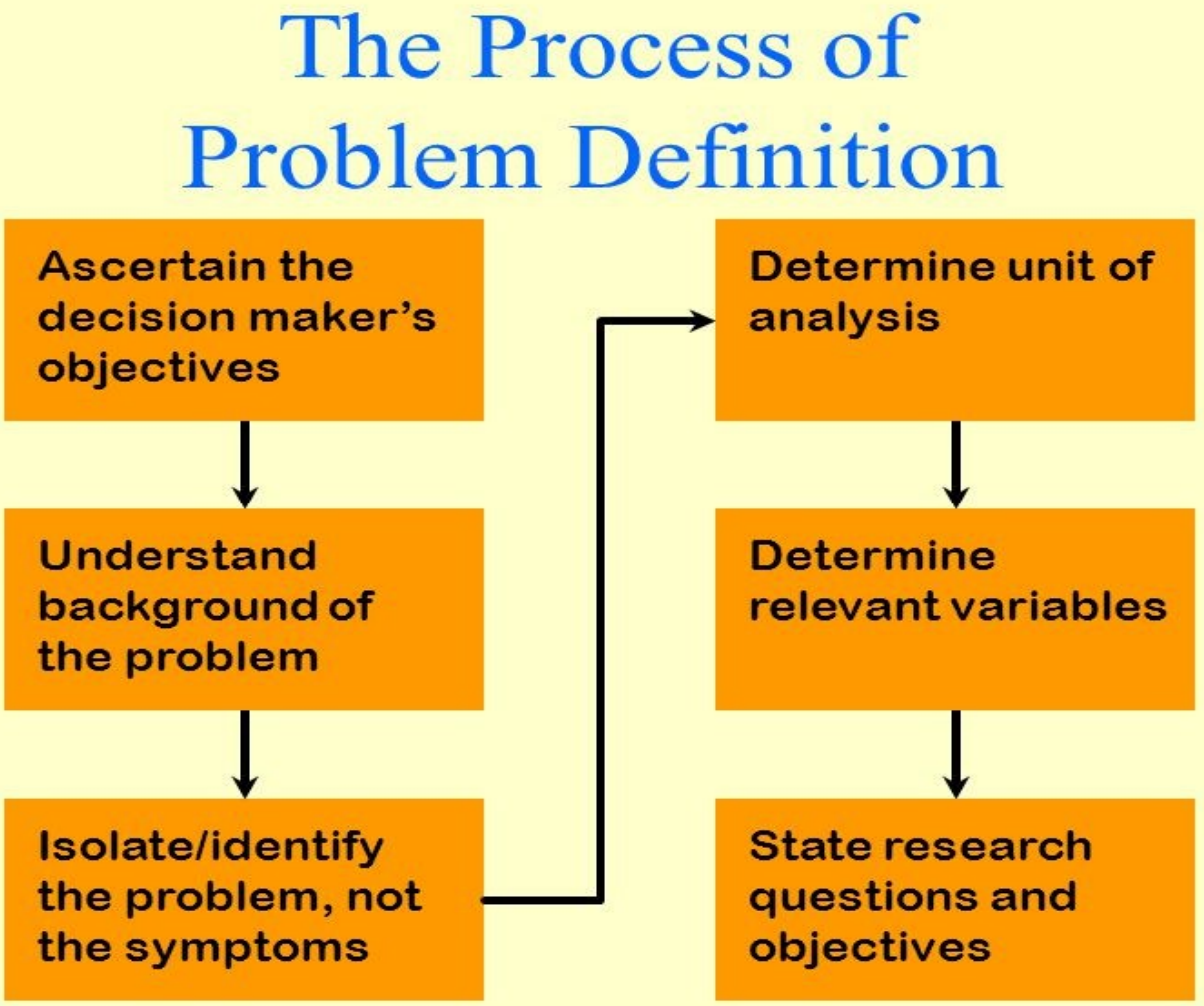
- knapsack problem is NP-complete while it can be solved by DP. They say that the DP solution is pseudo-polynomial, since it is exponential in the "length of input" (i.e. the numbers of bits required to encode the input).
- The running time is O(NW) for an unbounded knapsack problem with N items and knapsack of size W. W is not polynomial in the length of the input though, which is what makes it *pseudo*-polynomial.
- Consider W = 1,000,000,000,000. It only takes 40 bits to represent this number, so input size = 40, but the computational runtime uses the factor 1,000,000,000,000 which is $O(2^{40})$.
- So the runtime is more accurately said to be $O(N.2^{\text{bits in W}})$, which is **exponential**.

Running times for different sizes of input.

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3 nsec	0.01 μ	0.02 μ	0.06 μ	0.51 μ	0.26 μ
16	4 nsec	0.02 μ	0.06 μ	0.26 μ	4.10 μ	65.5 μ
32	5 nsec	0.03 μ	0.16 μ	1.02 μ	32.7 μ	4.29 sec
64	6 nsec	0.06 μ	0.38 μ	4.10 μ	262 μ	5.85 cent
128	0.01 μ	0.13 μ	0.90 μ	16.38 μ	0.01 sec	10^{20} cent
256	0.01 μ	0.26 μ	2.05 μ	65.54 μ	0.02 sec	10^{58} cent
512	0.01 μ	0.51 μ	4.61 μ	262.14 μ	0.13 sec	10^{135} cent
2048	0.01 μ	2.05 μ	22.53 μ	0.01 sec	1.07 sec	10^{598} cent
4096	0.01 μ	4.10 μ	49.15 μ	0.02 sec	8.40 sec	10^{1214} cent
8192	0.01 μ	8.19 μ	106.50 μ	0.07 sec	1.15 min	10^{2447} cent
16384	0.01 μ	16.38 μ	229.38 μ	0.27 sec	1.22 hrs	10^{4913} cent
32768	0.02 μ	32.77 μ	491.52 μ	1.07 sec	9.77 hrs	10^{9845} cent
65536	0.02 μ	65.54 μ	1048.6 μ	0.07 min	3.3 days	10^{19709} cent
131072	0.02 μ	131.07 μ	2228.2 μ	0.29 min	26 days	10^{39438} cent
262144	0.02 μ	262.14 μ	4718.6 μ	1.15 min	7 mnths	10^{78894} cent
524288	0.02 μ	524.29 μ	9961.5 μ	4.58 min	4.6 years	10^{157808} cent
1048576	0.02 μ	1048.60 μ	20972 μ	18.3 min	37 years	10^{315634} cent

Note: “nsec" stands for nanoseconds, “ μ”is one microsecond and “cent" stands for centuries. The explosive running time (measured in centuries) when it is of the order 2^n.

Problem definition



Need for Programming Language

- We have some problem.
- We have to solve the problem.
- We can solve the problem, but we don't want to repeatedly solve the similar problems.
- We want to use the computer to do this task for us whenever required.
- We have to tell the computer, "How to solve the problem."

Problem and Problem Solving

- **What is Problem?**

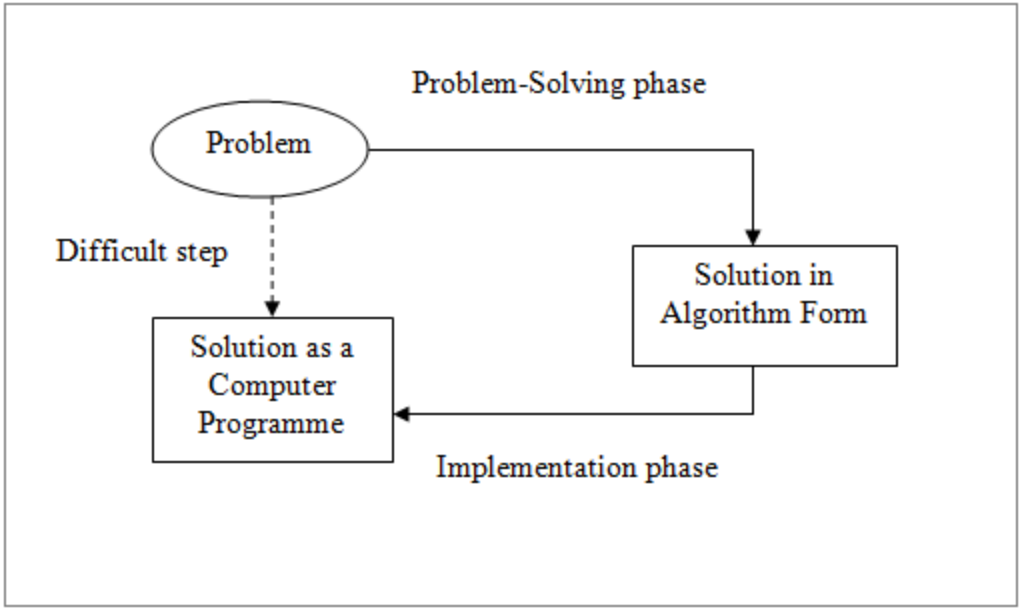
- A problem is an obstacle which makes it difficult to achieve a desired goal, objective or purpose.
- It refers to a situation, condition, or issue that is yet unresolved.
- In a broad sense, a problem exists when an individual becomes aware of a significant difference between what actually is and what is desired.

- **What is Problem Solving?**

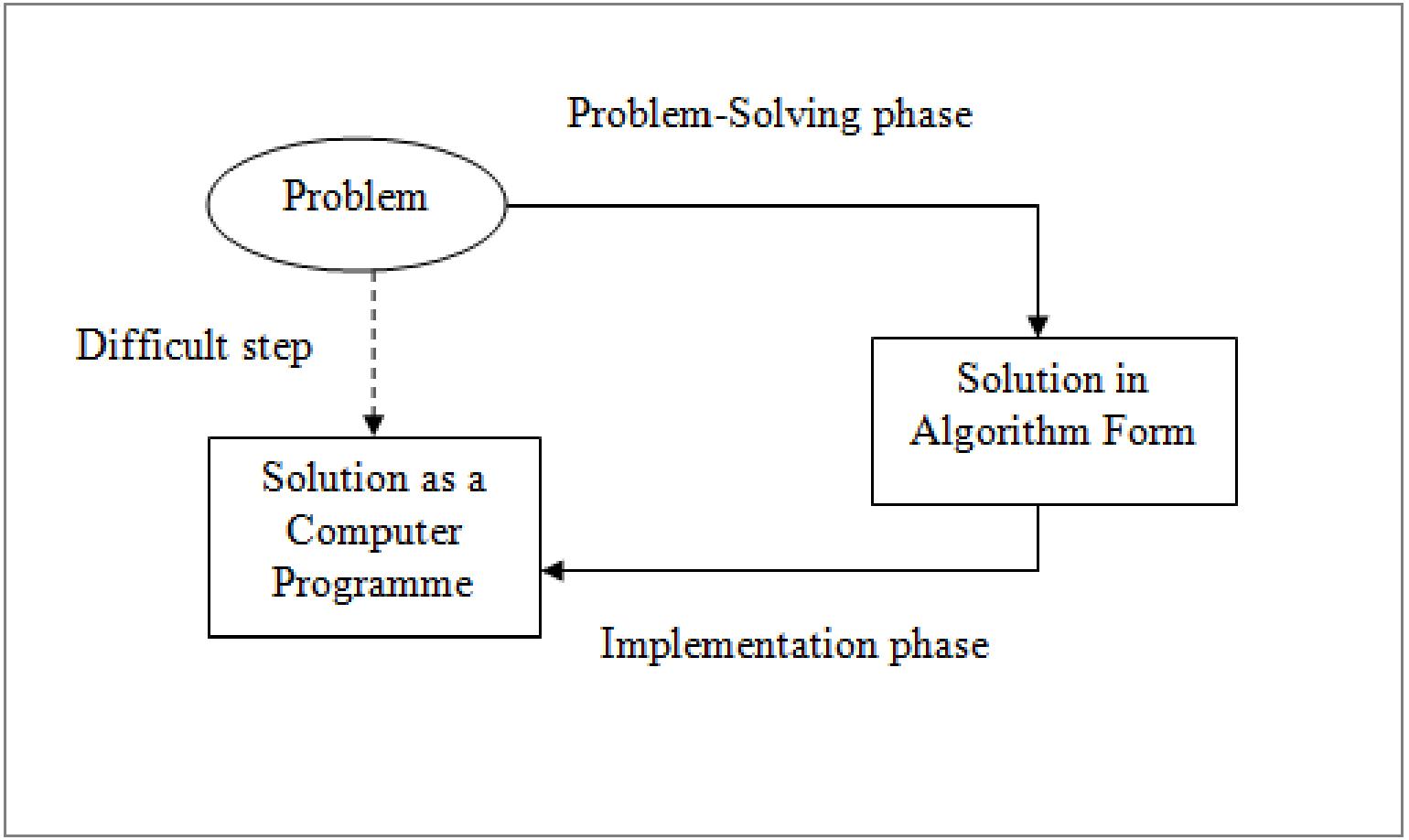
- Every problem in the real world needs to be identified and solved. Trying to find a solution to a problem is known as problem solving.
- Problem solving becomes a part of our day to day activity. Especially, when we use computers for performing our day to day activity.

How to solve a problem?

1. If you have a problem, either you can solve it manually or using computer.
2. If the problem is easy enough, solve it manually or else use computers.
3. Bigger problems can be sub-divided into smaller problems (sub-problems) and start solving them one by one.
4. After completing solving sub-problems, the entire big problem has been solved easily.



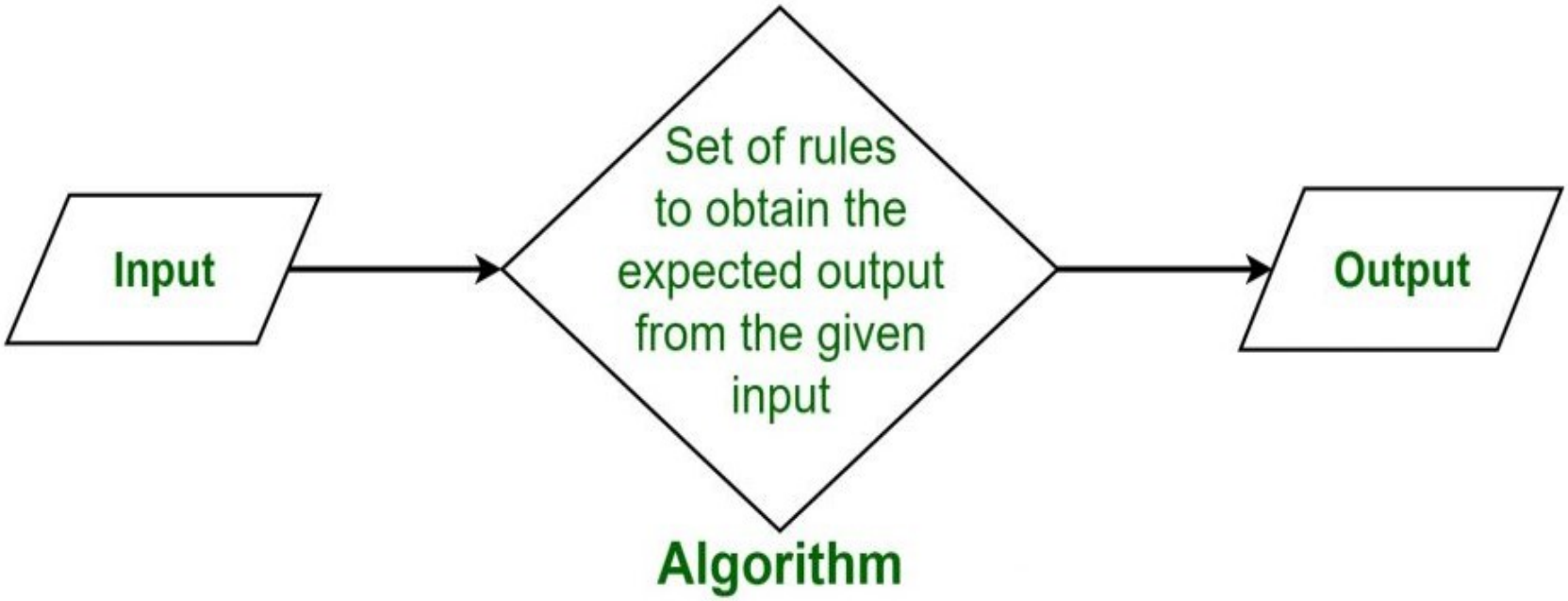
Problem solving with computer



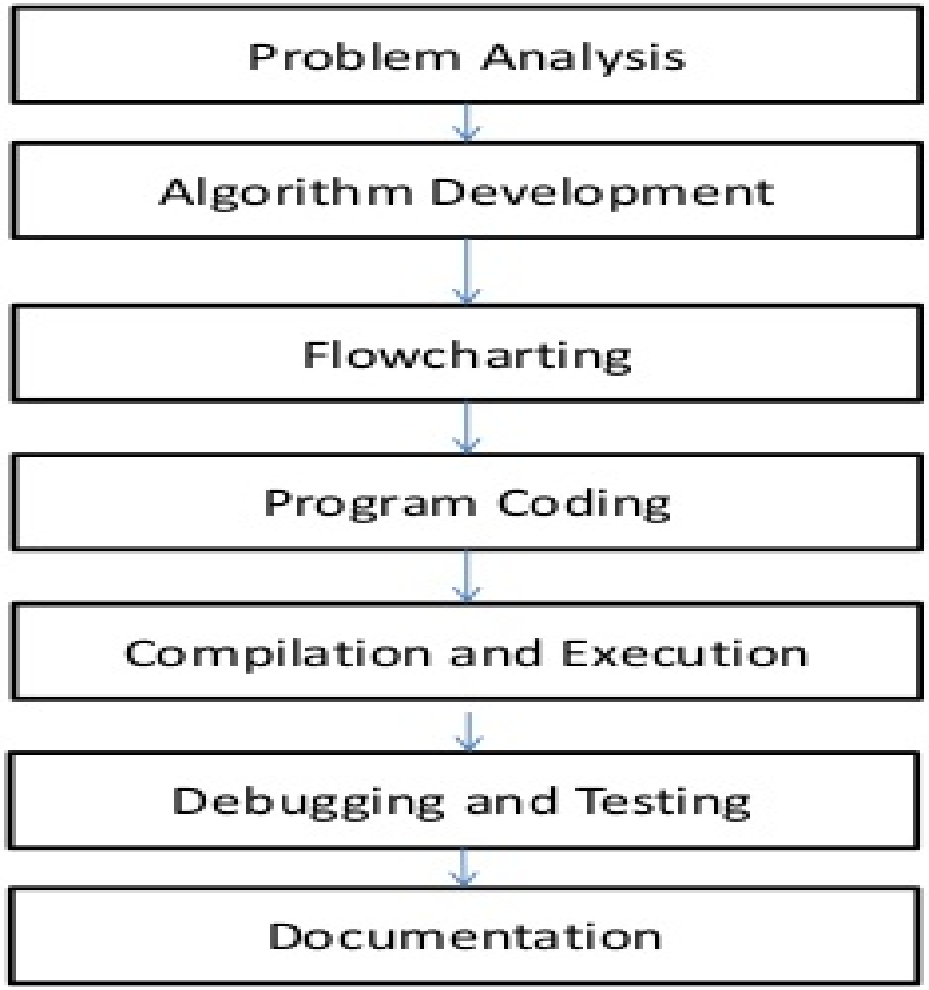
Problem Solving

- According to psychology, "*a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition.*"
- According to computer science, *a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.*

What is Algorithm?



Steps in Problem Solving

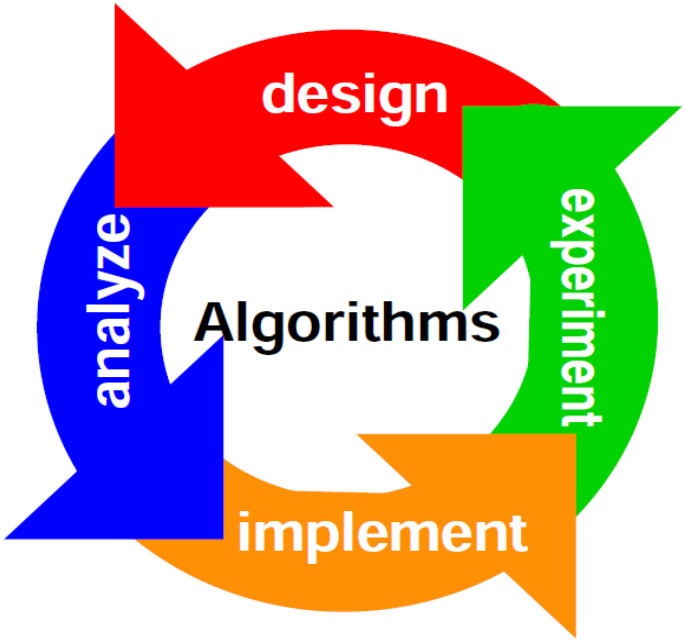


Algorithm development

- **Algorithm development** is the act of designing the steps that solve a particular problem for a computer or **any other device** to follow not excluding human being, but in this case computers only and computer like devices.

Steps in development of Algorithms

1. Problem definition
2. Development of a model
3. Specification of Algorithm
4. **Designing an Algorithm**
5. Checking the correctness of Algorithm
6. Analysis of Algorithm
7. Implementation of Algorithm
8. Program testing
9. Documentation Preparation

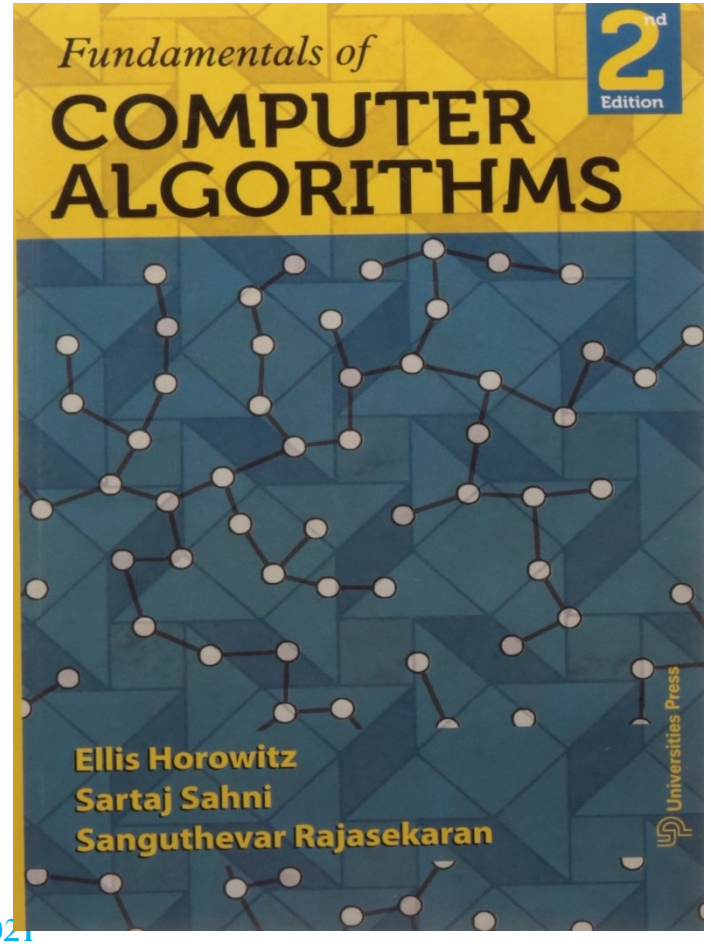


Algorithmic paradigm

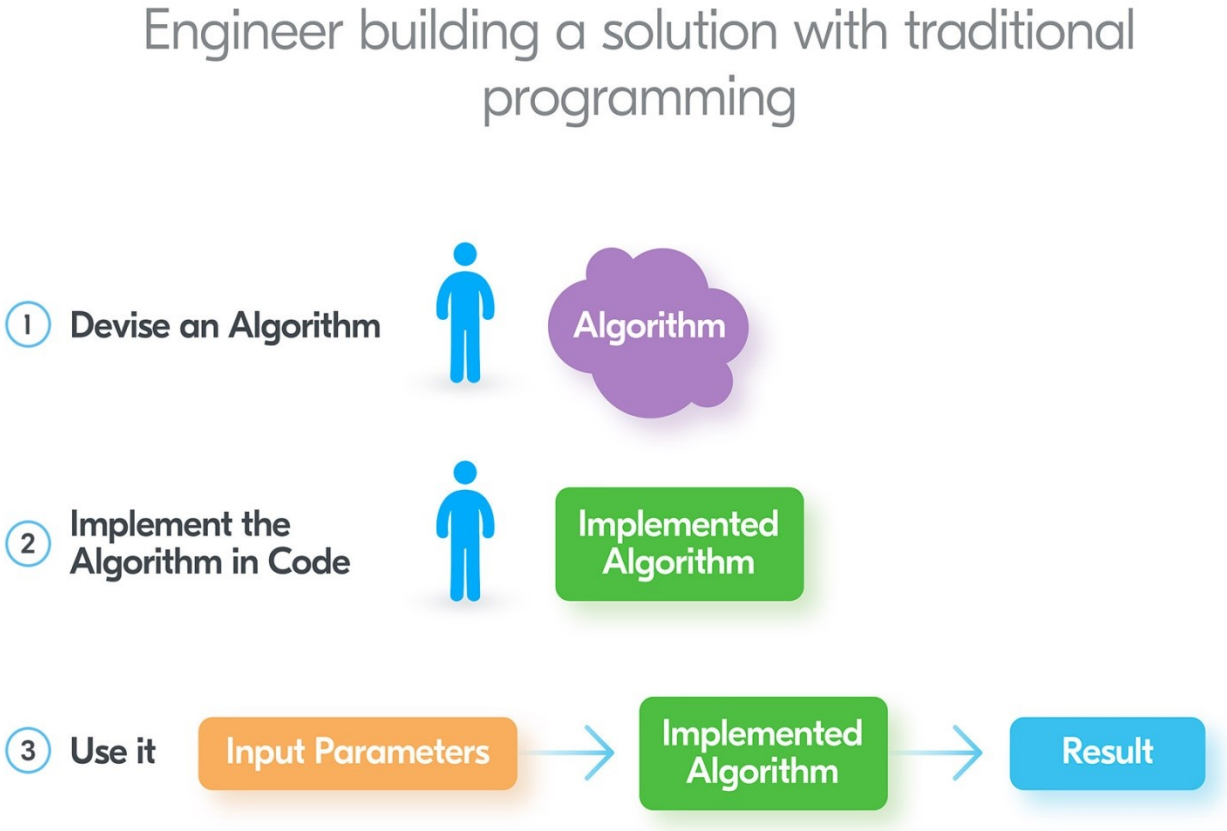
- An **algorithmic paradigm** or **algorithm design paradigm** is a generic model or framework which underlies the design of a **class of algorithms.**

Main algorithm design techniques:

- Brute-force or exhaustive search.
- Divide and Conquer.
- Greedy Algorithms.
- Dynamic Programming.
- Branch and Bound Algorithm.
- Backtracking.



Traditional Programming Approach



Problem Solving Techniques in Artificial Intelligence

- Problem-solving is commonly known as the method to reach the desired goal or finding a solution to a given situation.
- In computer science, problem-solving refers to artificial intelligence techniques, including various techniques such as forming efficient algorithms, heuristics, and performing root cause analysis to find desirable solutions.
- In **Artificial Intelligence**, the users can solve the problem by performing logical algorithms, utilizing polynomial and differential equations, and executing them using modeling paradigms.
- There can be various solutions to a single problem, which are achieved by different heuristic

Problem Solving in Artificial Intelligence

- The problem of AI is directly associated with the nature of humans and their activities. So we need a number of **finite st**eps to solve a problem which makes human easy works.
- **Goal Formulation:** This one is the first and simple step in problem-solving. It organizes finite steps to formulate a target/goals which require some action to achieve the goal. Today the formulation of the goal is based on AI agents.
- **Problem formulation:** It is one of the core steps of problem-solving which decides what action should be taken to achieve the formulated goal. In AI this core part is dependent upon software agent which consisted of the following components to formulate the associated problem.

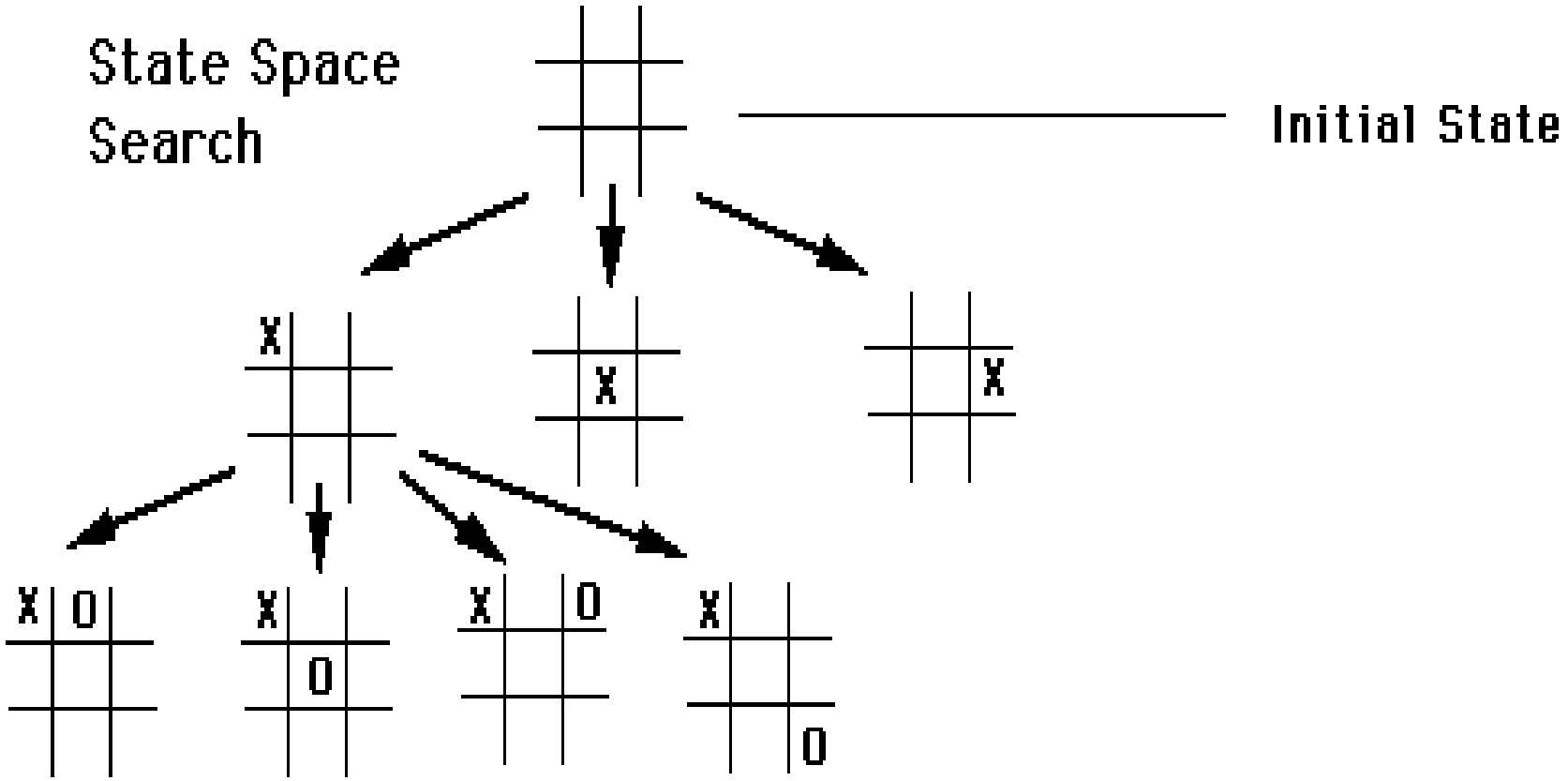
Steps performed by Problem-solving agent

- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
- **Initial State:** It is the starting state or initial step of the agent towards its goal.
- **Actions:** It is the description of the possible actions available to the agent.
- **Transition Model:** It describes what each action does.
- **Goal Test:** It determines if the given state is a goal state.
- **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure.

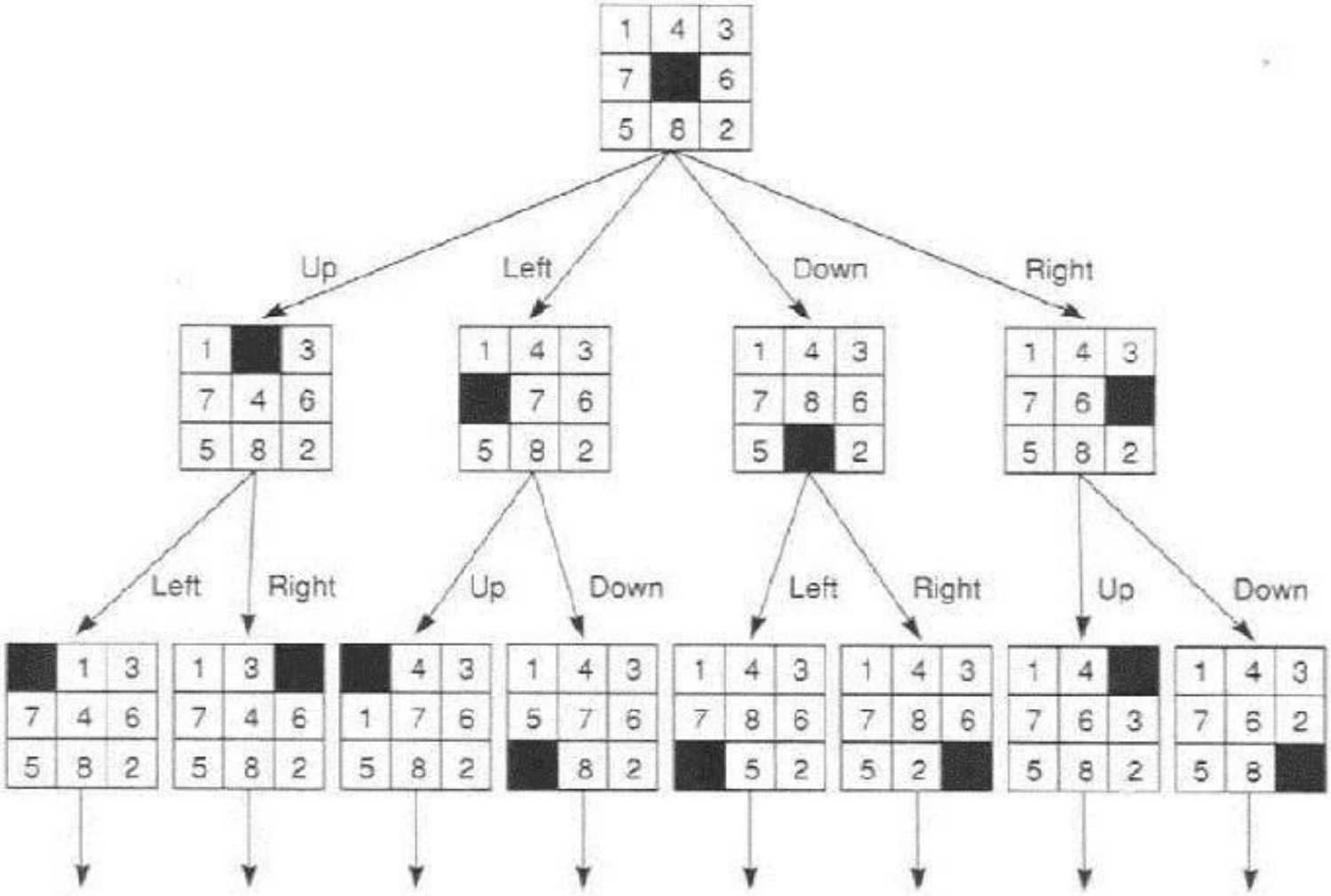
An optimal solution has the lowest path cost among all the solutions.

- **Note: Initial state, actions**, and **transition model** together define the **state-space** of the problem implicitly.
- State-space of a problem is a set of all states which can be reached from the **initial state** followed by any sequence of actions.
- The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.
- **Search:** It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

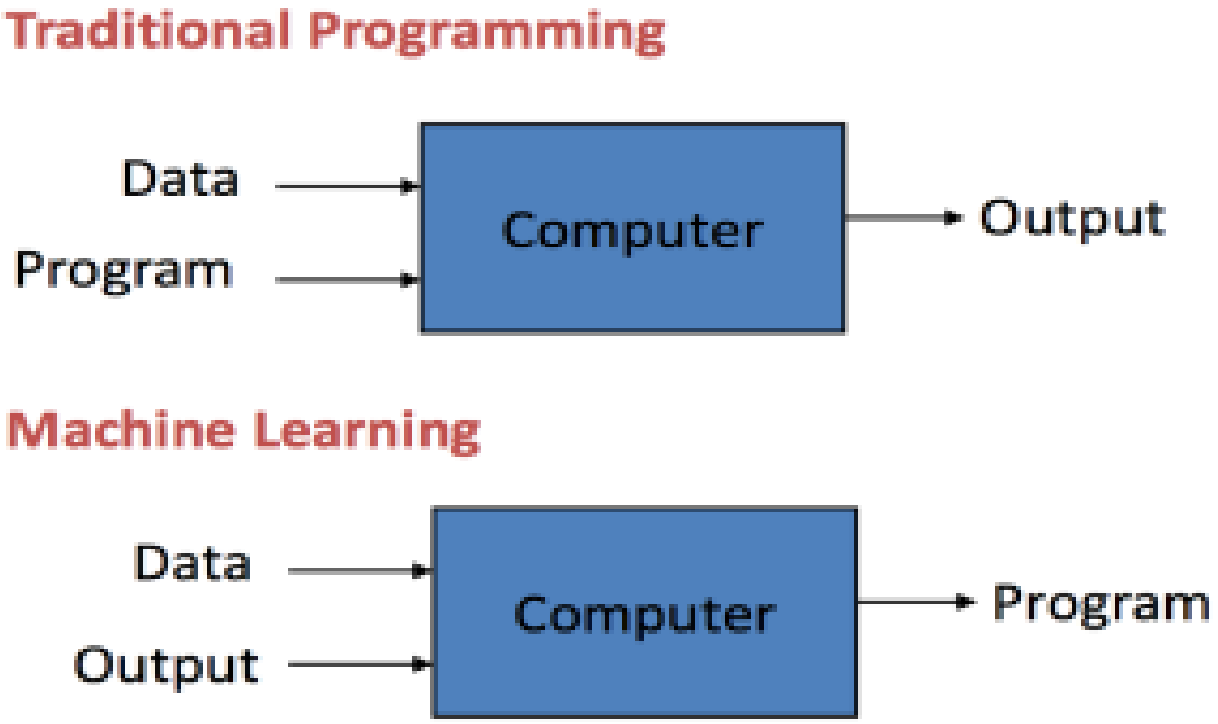
Problem soling by state space search



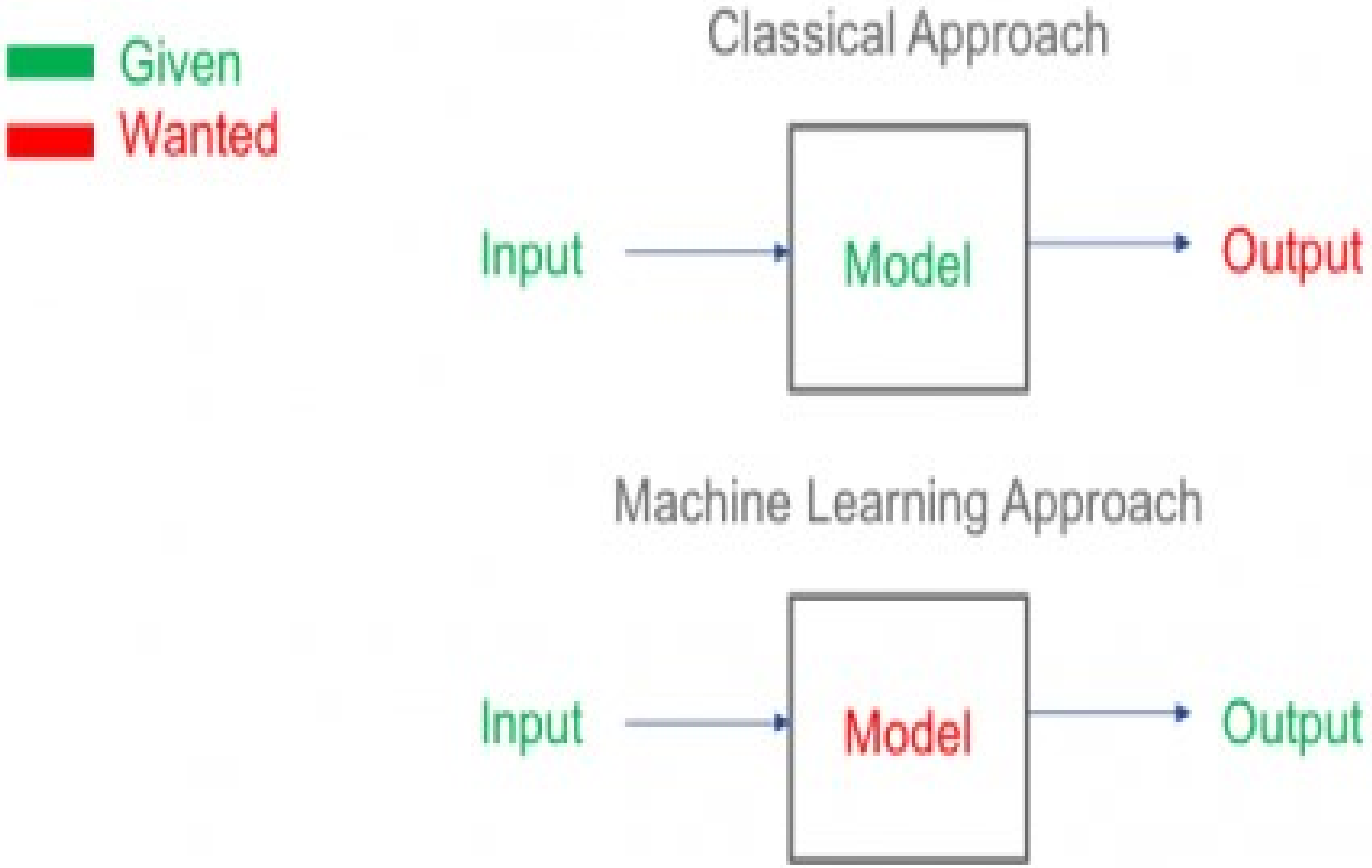
State space tree for 8 puzzle problem



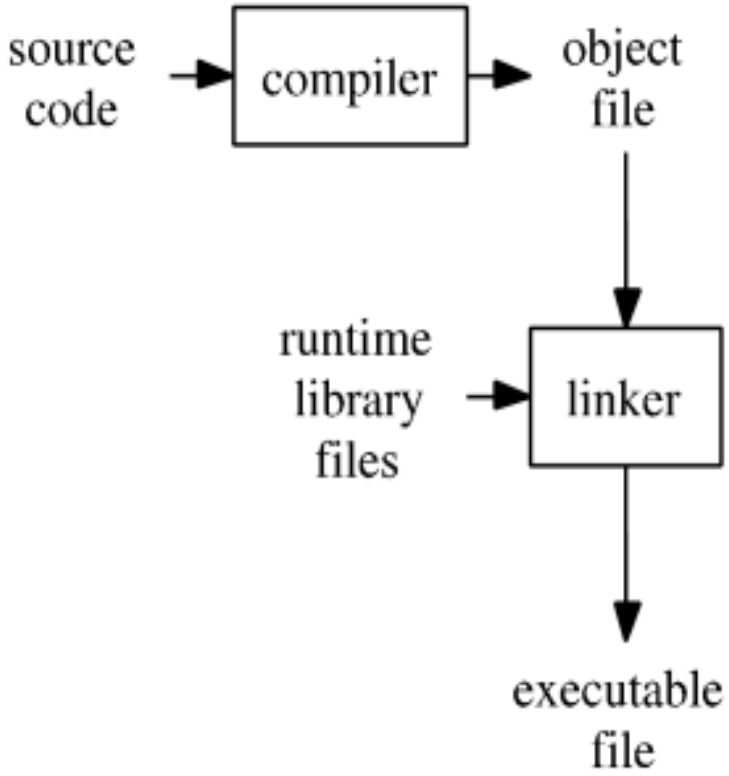
Traditional Programming vs Machine learning

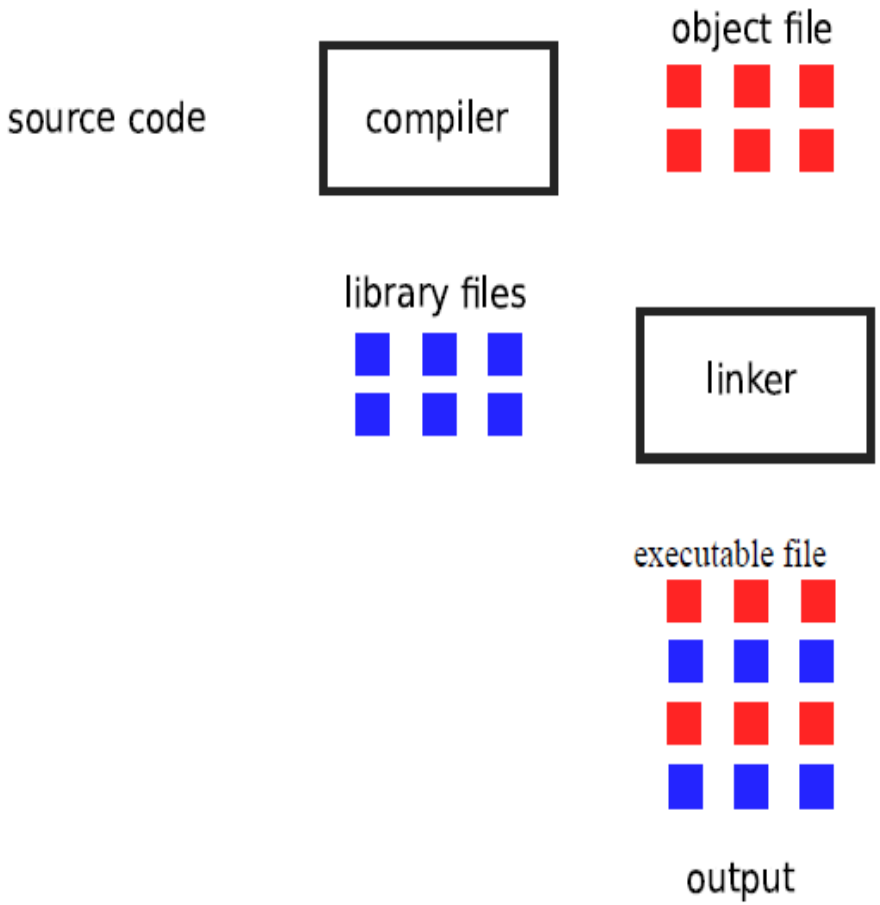


Machine Learning vs. Classical Approach



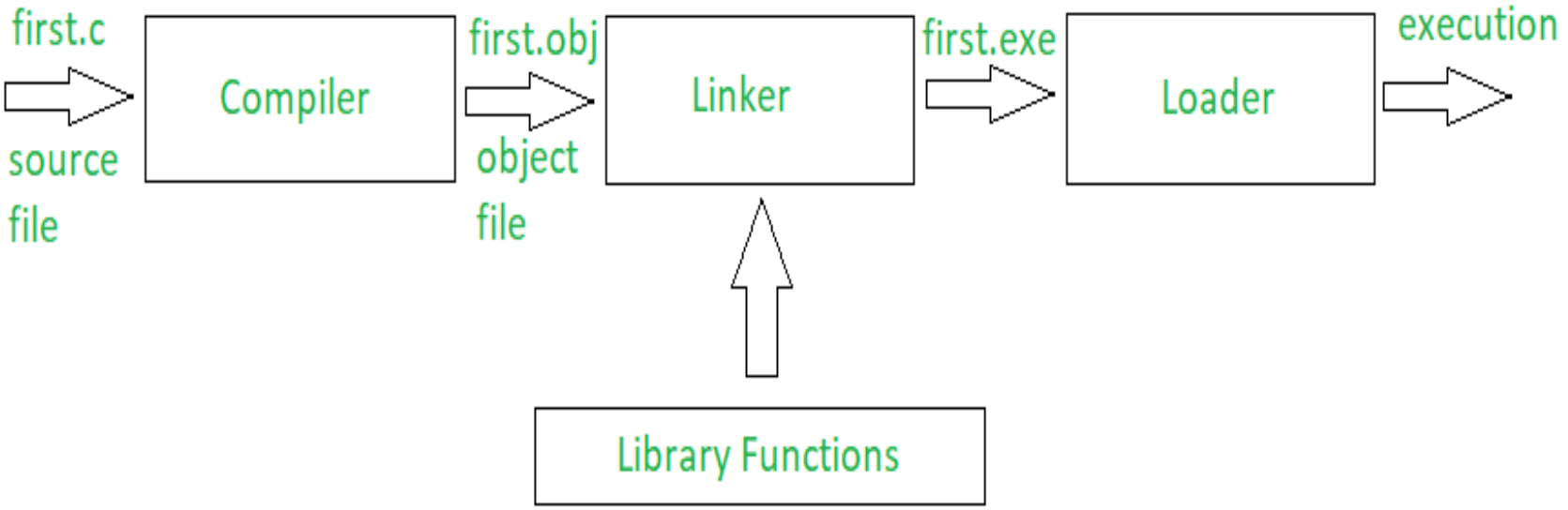
Complier


source code
compiler
object file
runtime library files
linker
executable file

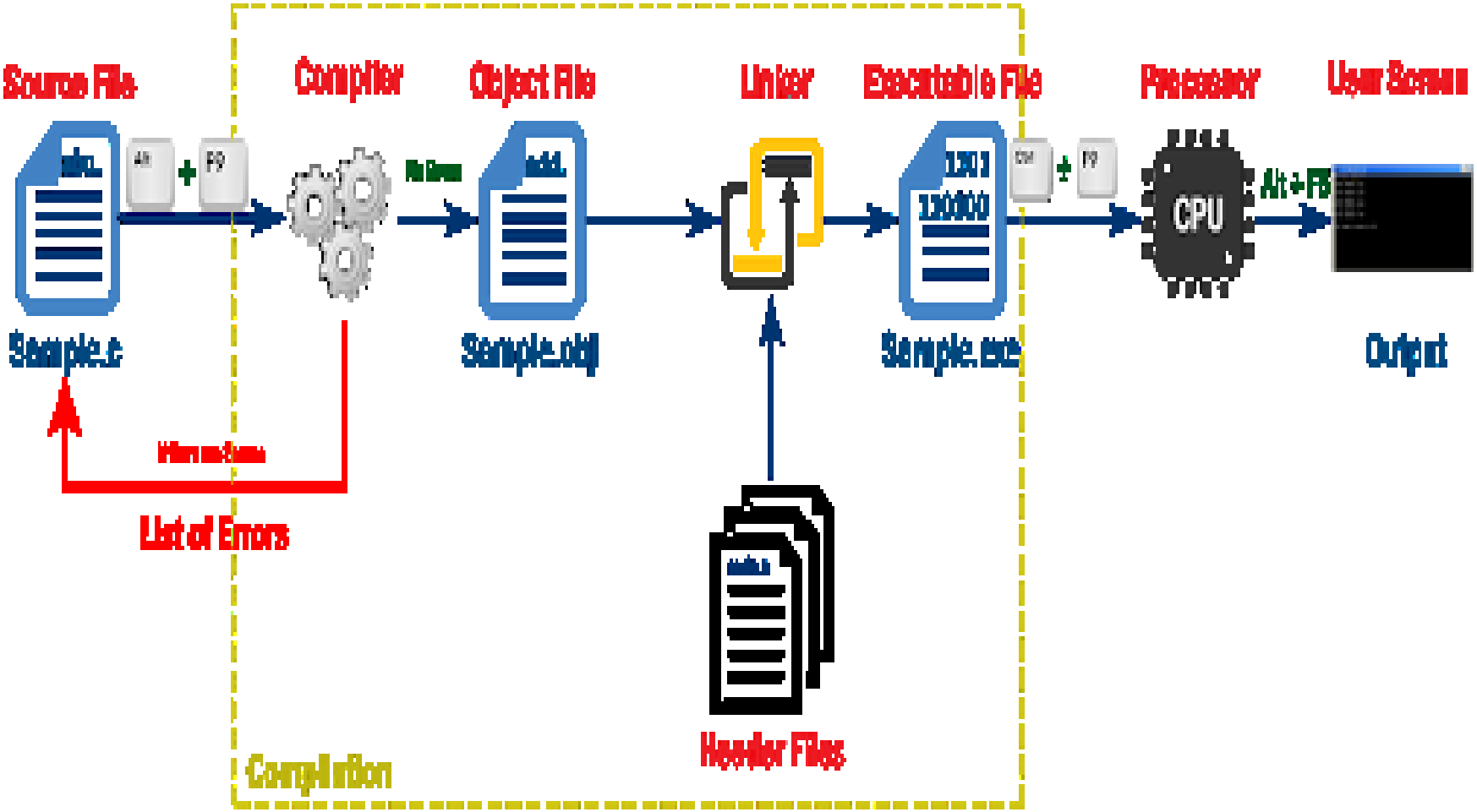

source code
compiler
object file
library files
linker
executable file
output

The compilation process

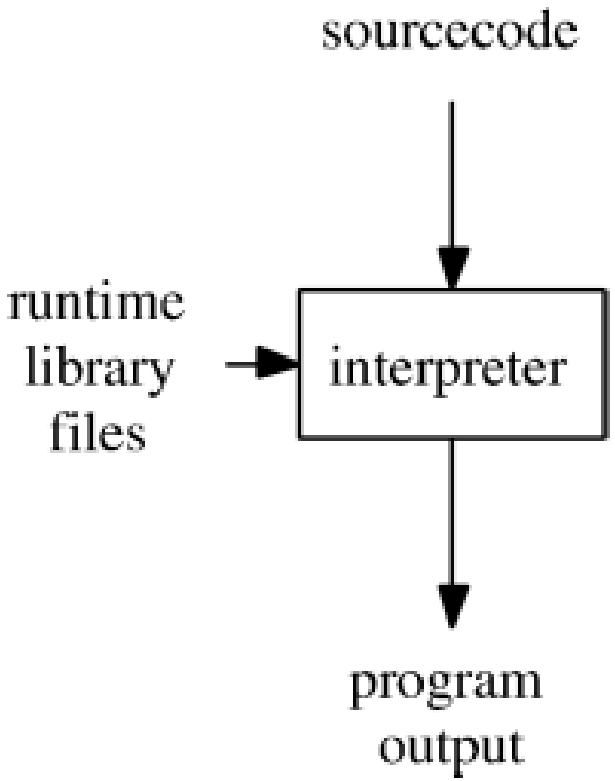
- The compilation process with the files created at each step of the compilation process

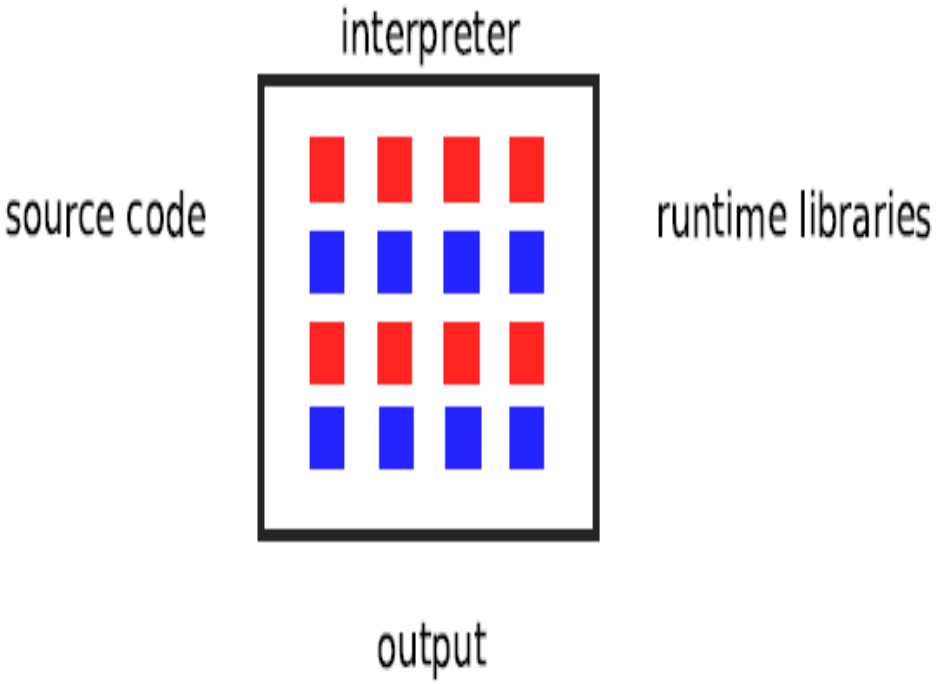


Execution Process of a C Program

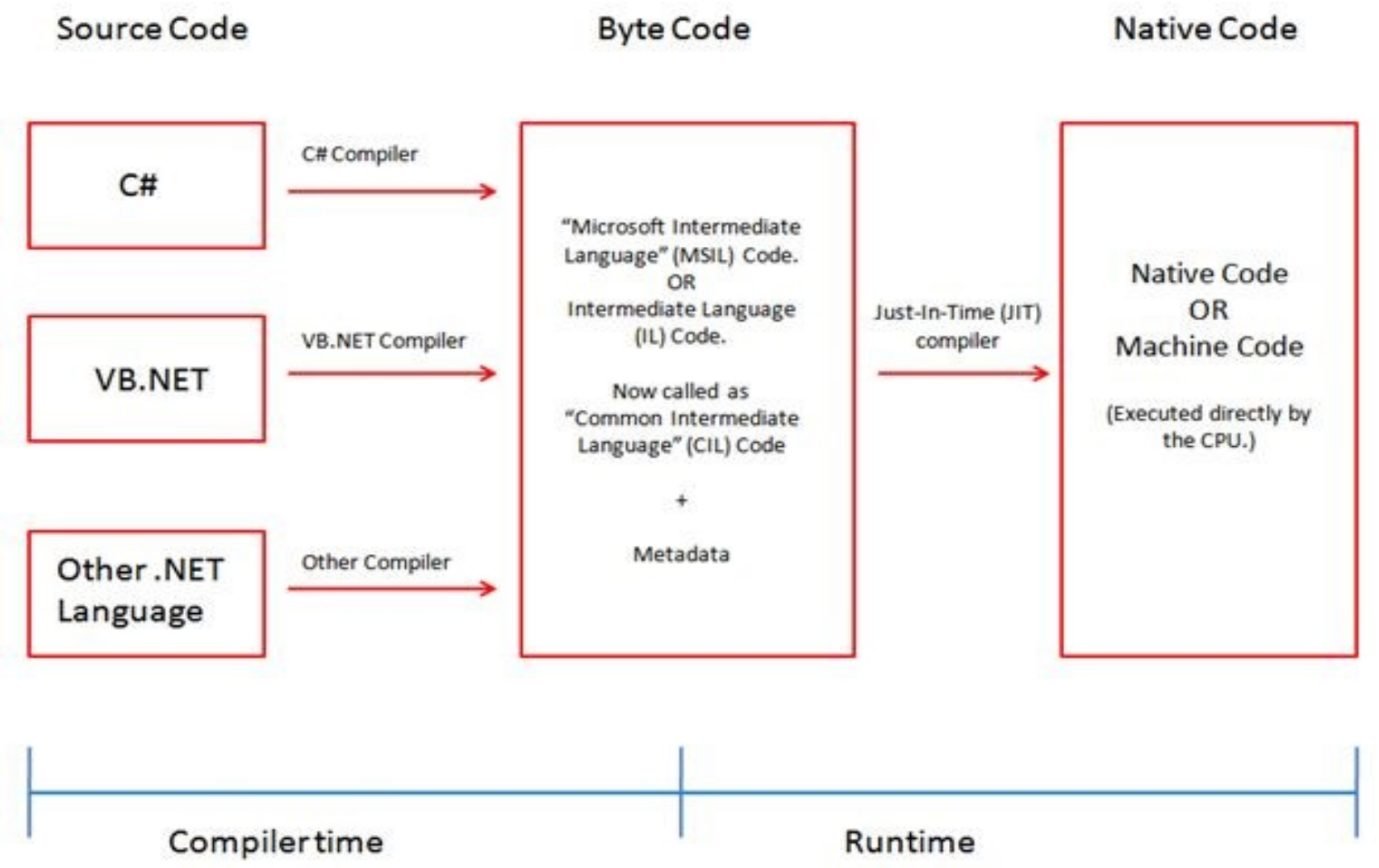


Interpreter

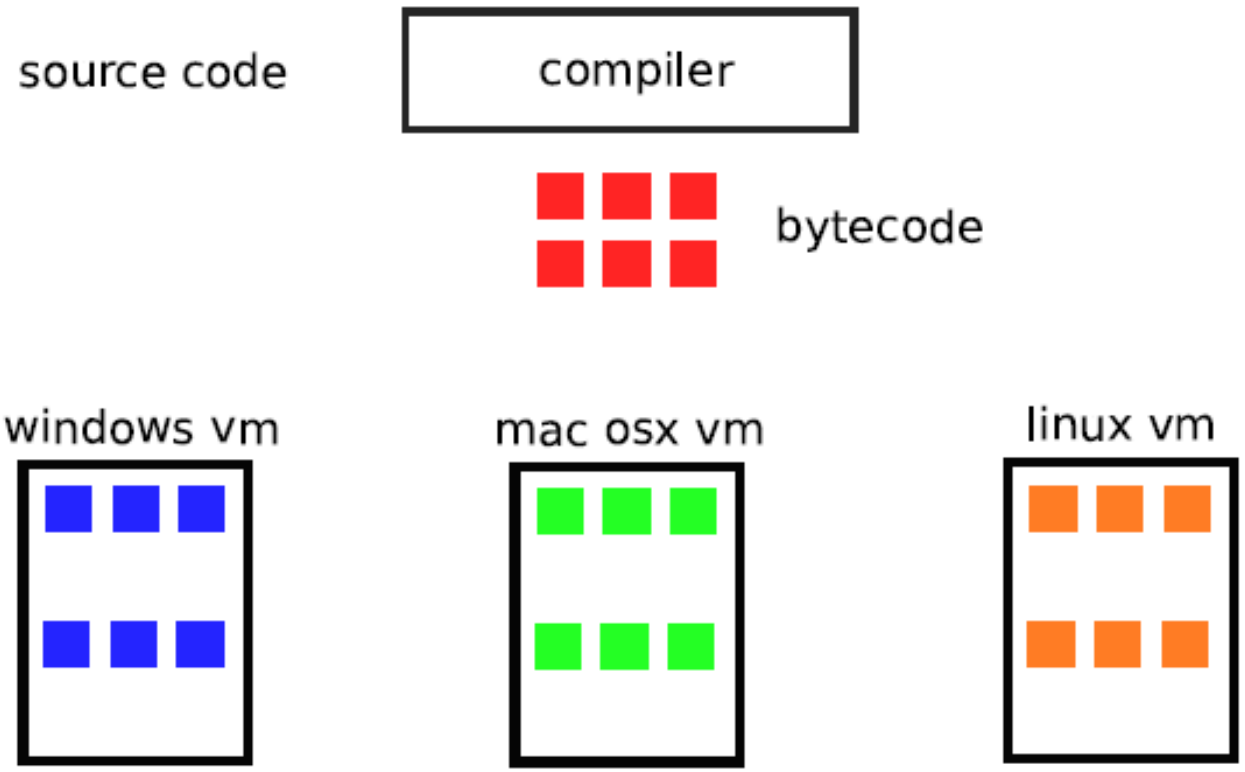




Code execution process



Compiler on virtual machine

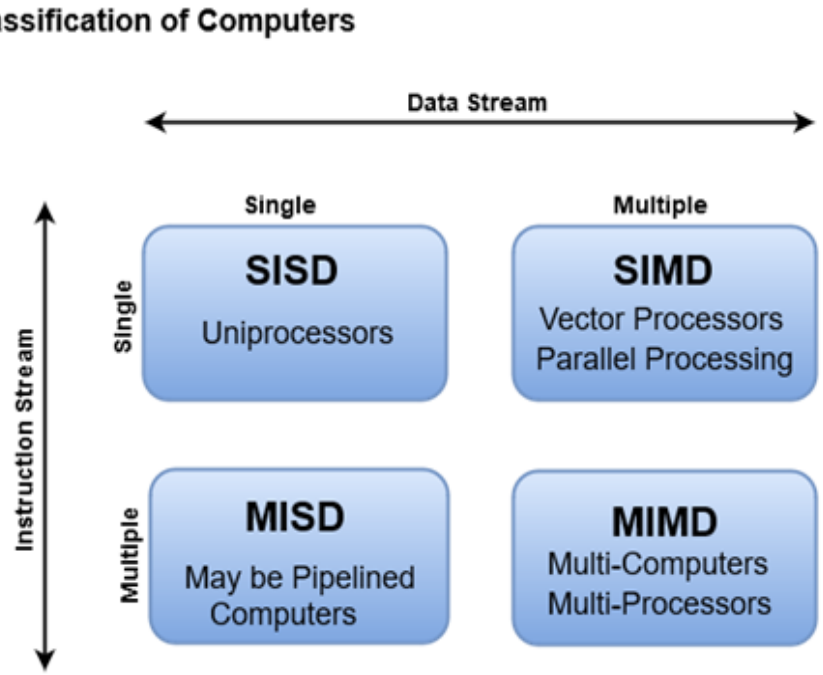


Model of Computation

Model of Computation [Flynn's classification divides]

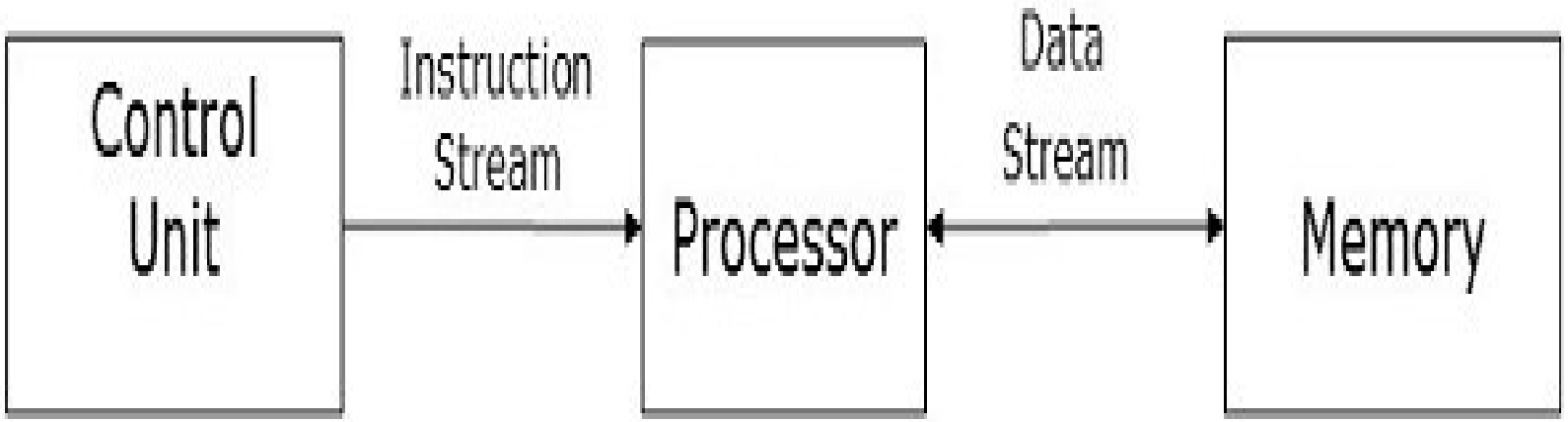
- Depending on the instruction stream and data stream, computers can be classified into four categories –

1. Single Instruction stream, Single Data stream (SISD) computers
2. Single Instruction stream, Multiple Data stream (SIMD) computers
3. Multiple Instruction stream, Single Data stream (MISD) computers
4. Multiple Instruction stream, Multiple Data stream (MIMD) computers



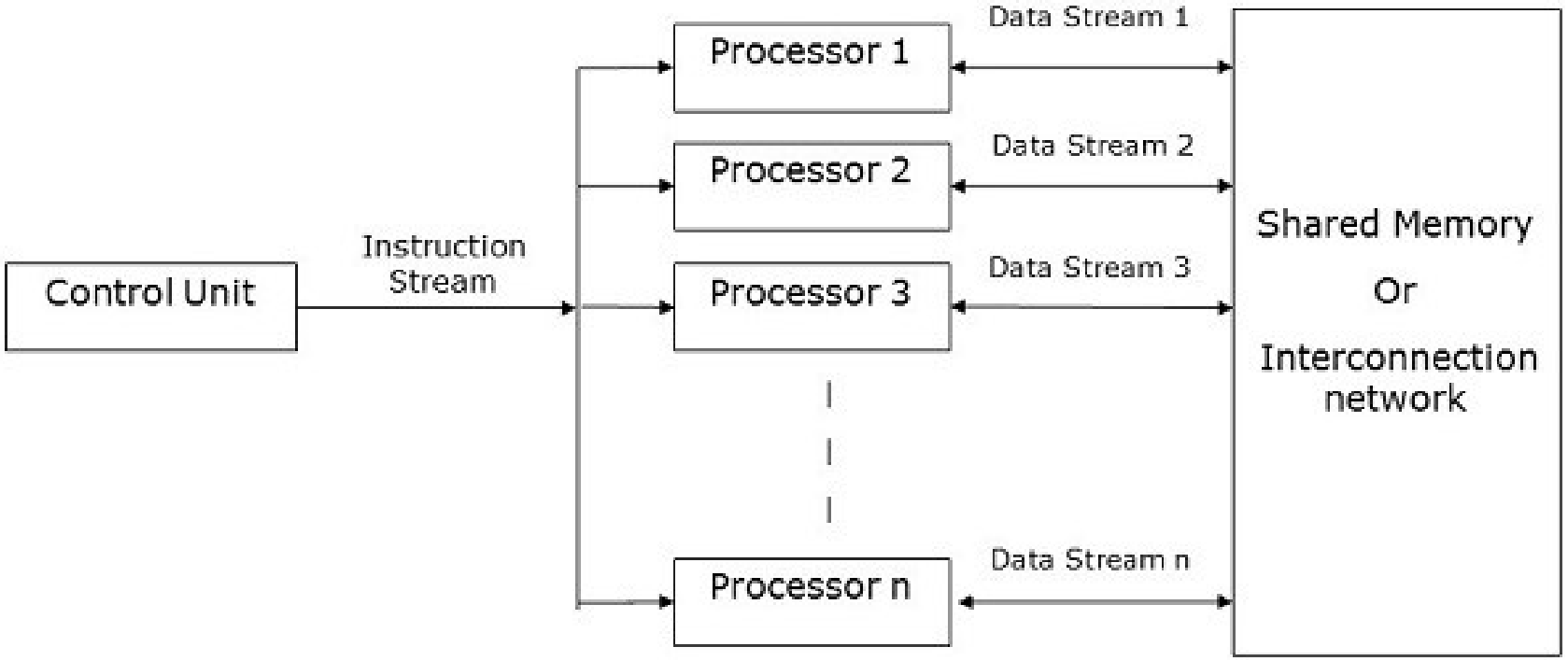
SISD Computer

- SISD computers contain **one control unit, one processing unit,** and **one memory unit**.

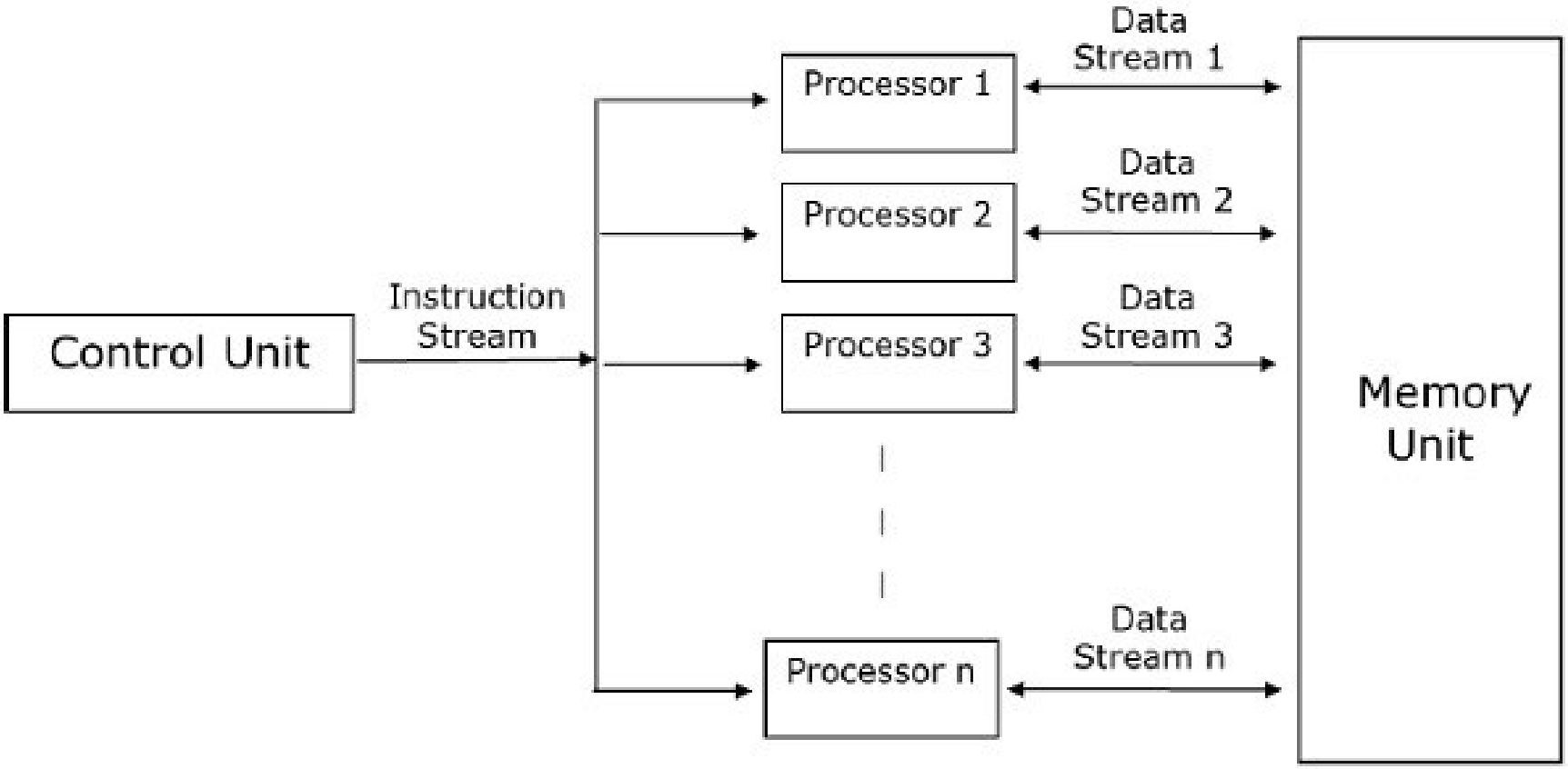


SIMD Computers

- SIMD computers contain **one control unit, multiple processing units,** and **shared memory or interconnection network.**

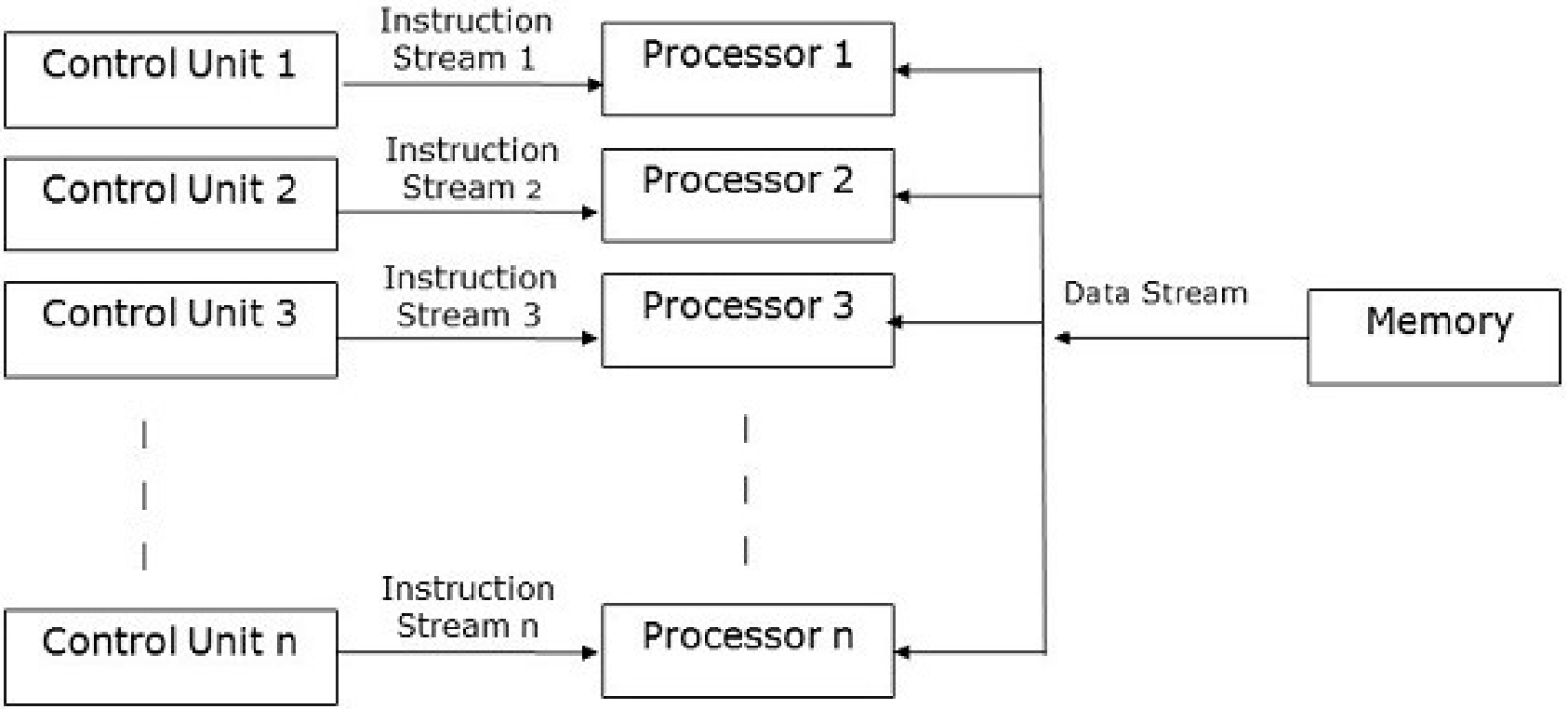


SIMD Computers



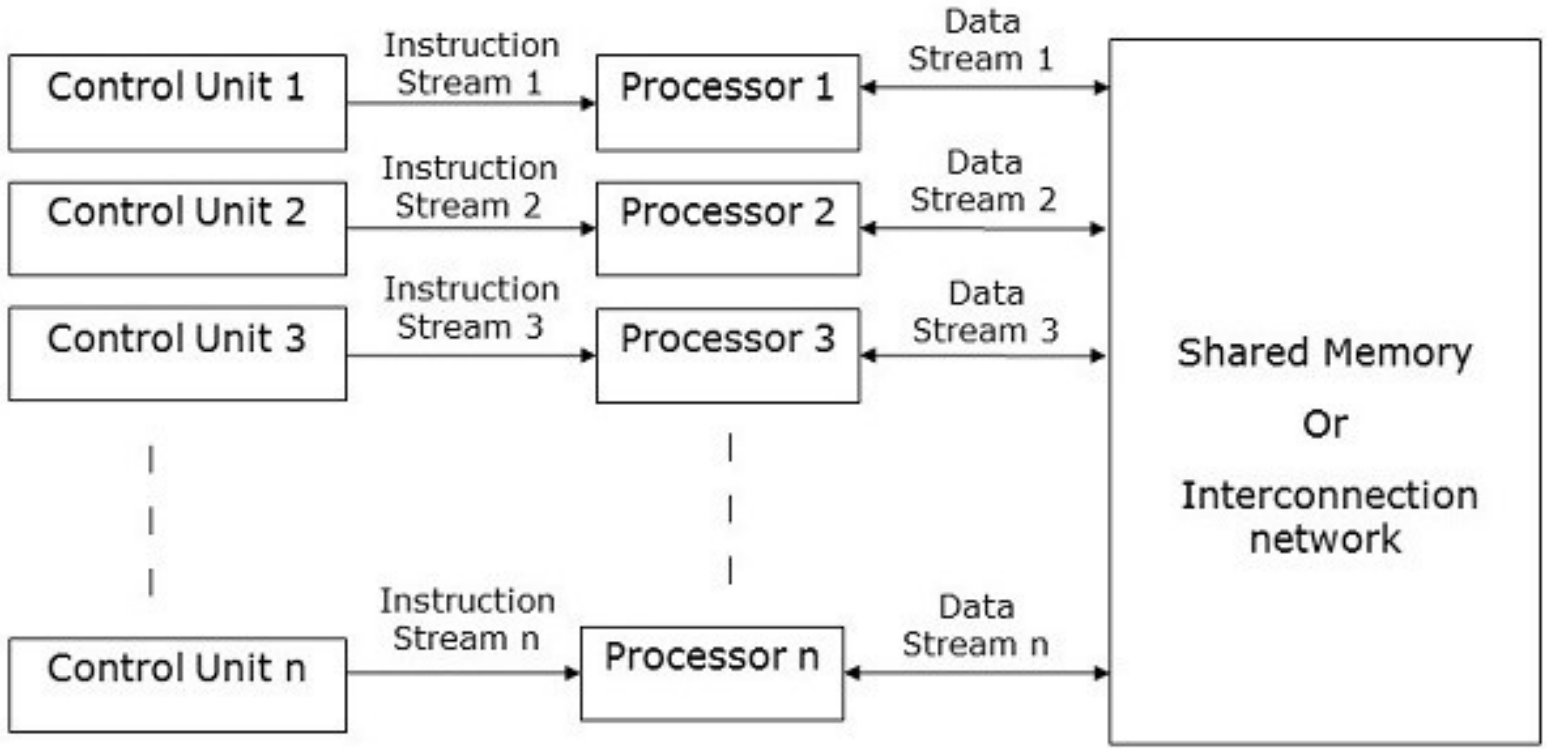
MISD Computers

- MISD computers contain **multiple control units, multiple processing units,** and **one common memory unit**.



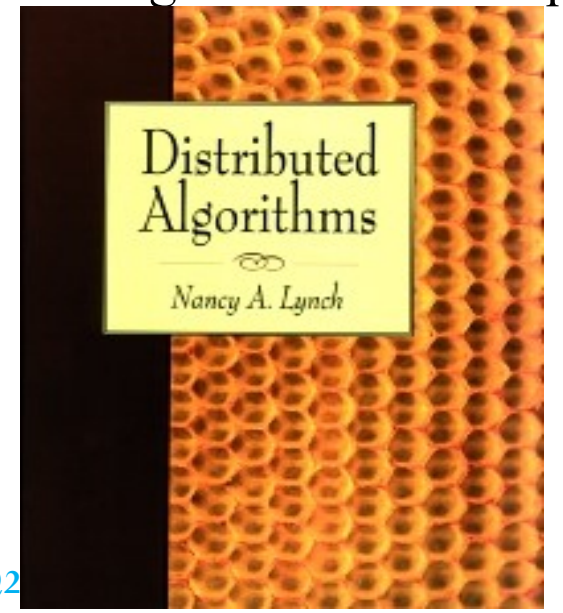
MIMD Computers

MIMD computers have **multiple control units, multiple processing units,** and a **shared memory** or **interconnection network**.

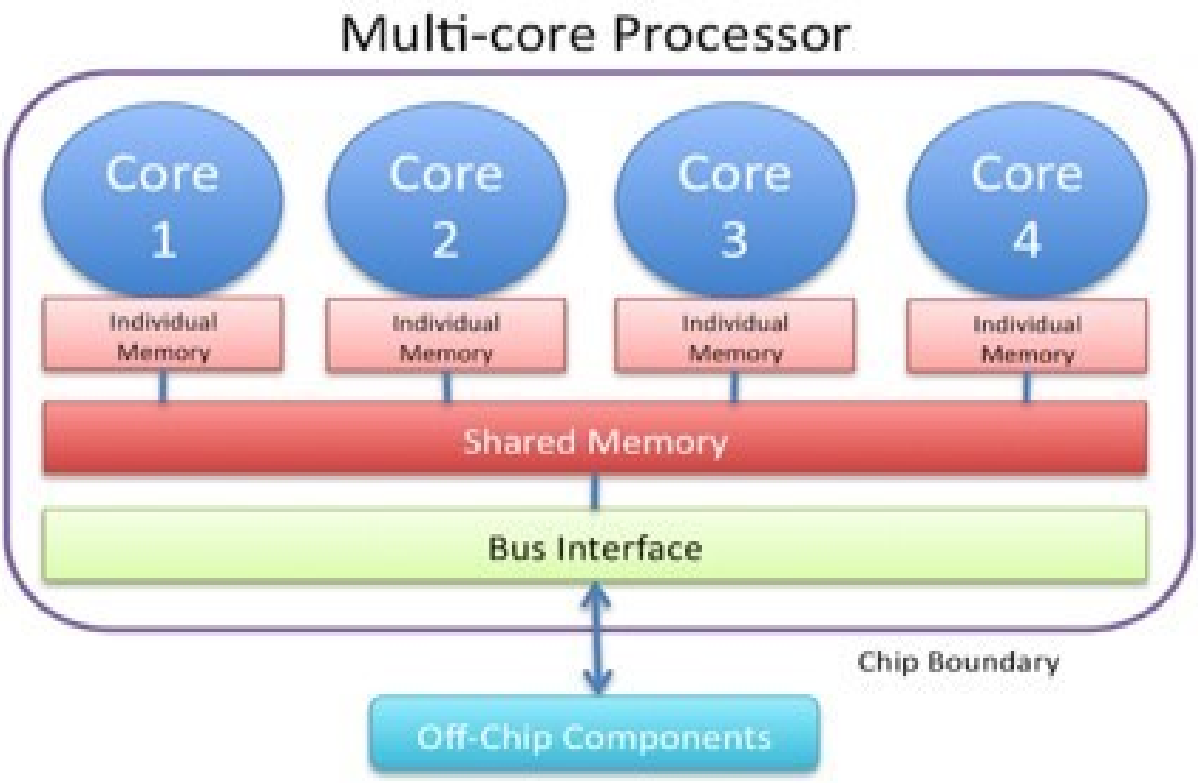


Distributed algorithm

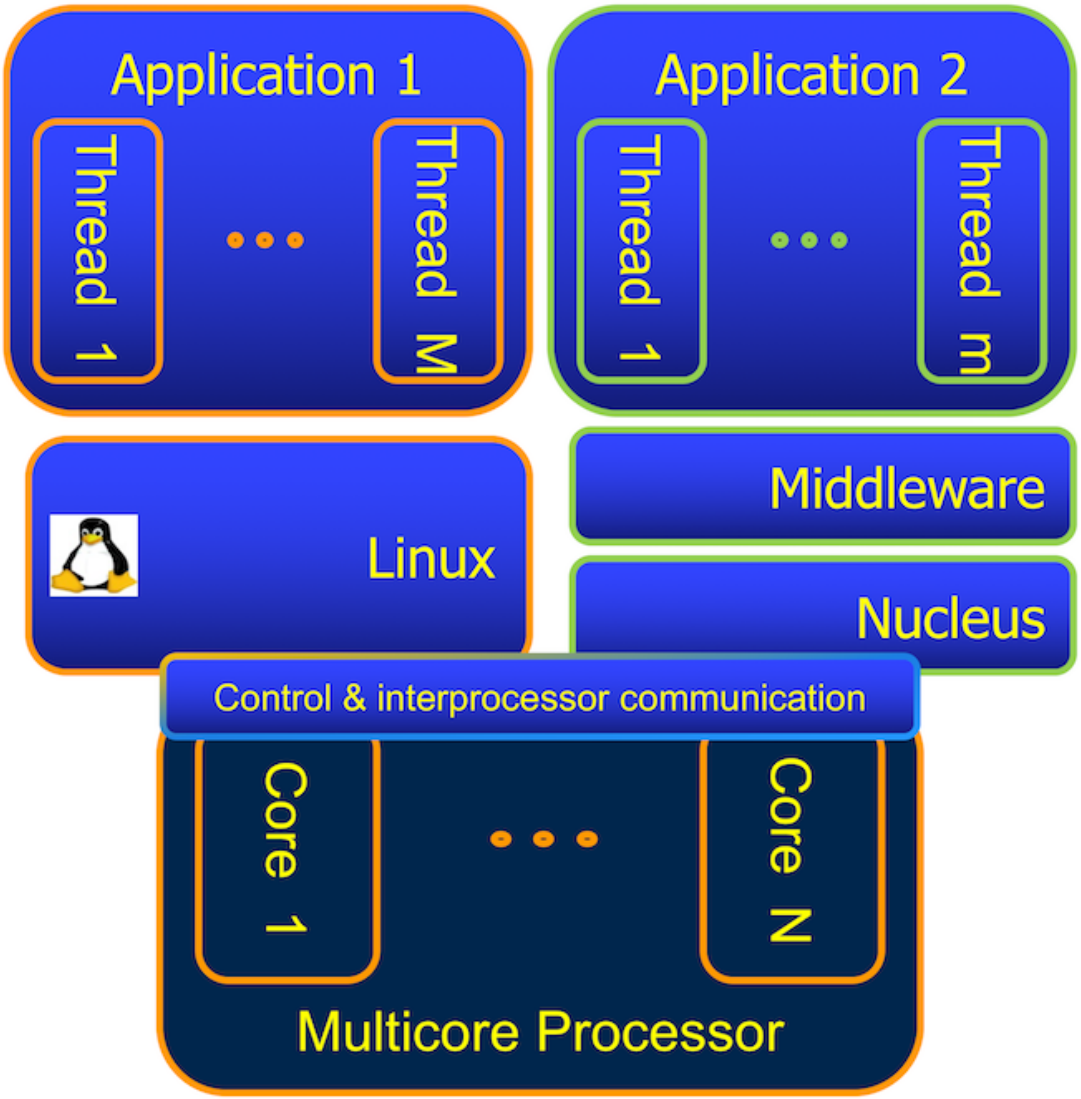
- A **distributed algorithm** is an algorithm, run on a distributed system.
- A **distributed algorithm** is an algorithm designed to run on computer hardware constructed from interconnected processors.
- Distributed algorithms are used in different application areas of distributed computing, such as telecommunications, scientific computing, distributed information processing, and real-time process control.



Multicore Processor



Multicore Processor



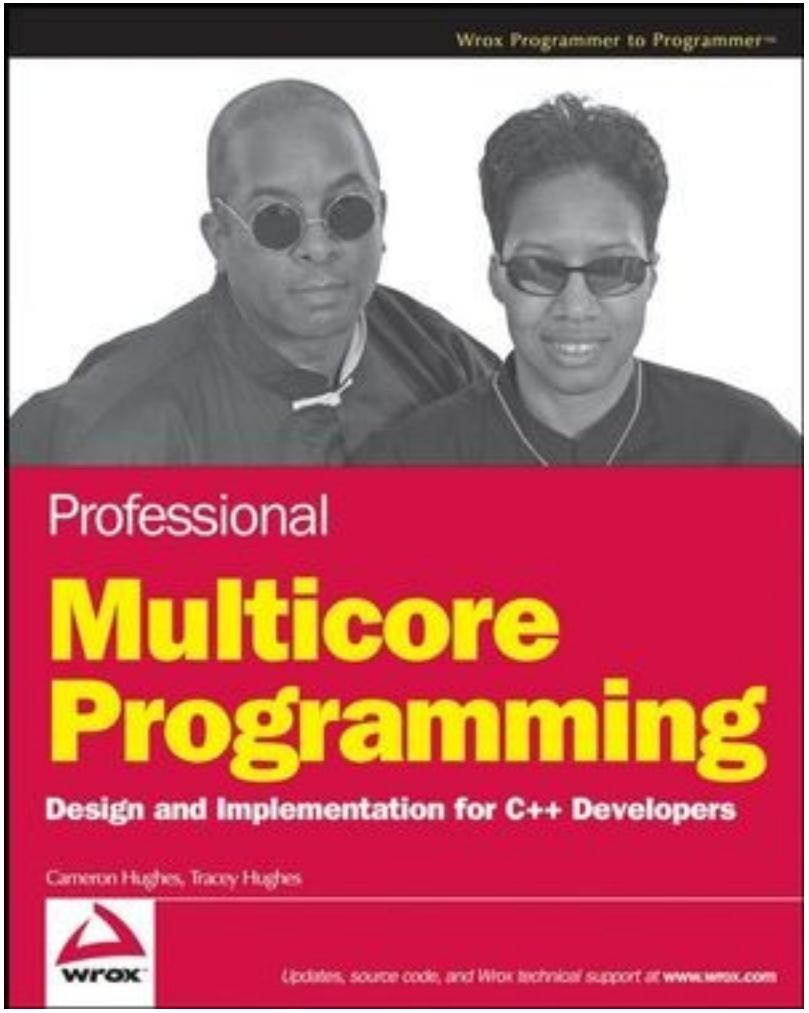
Three embedded multicore architectures:

- Broadly speaking, there are

1. **Homogeneous multicore**: all the cores are identical.
2. **Heterogeneous multicore**: all the cores are different.
3. **Hybrid:** multiple identical cores, but some different.

[This may be a selection of regular CPUs and specialized cores. It could also be pairs of CPUs, offering a low- and high-power option for each.]

Multicore programming



Algorithmic Thinking
(Computational thinking)

Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9937324437, 2462358

Why Study Algorithms?

- **Algorithmic thinking** skills support the development of general reasoning, problem-solving and communication skills by giving students the skills to fluently interpret and design structured procedures and rule systems.

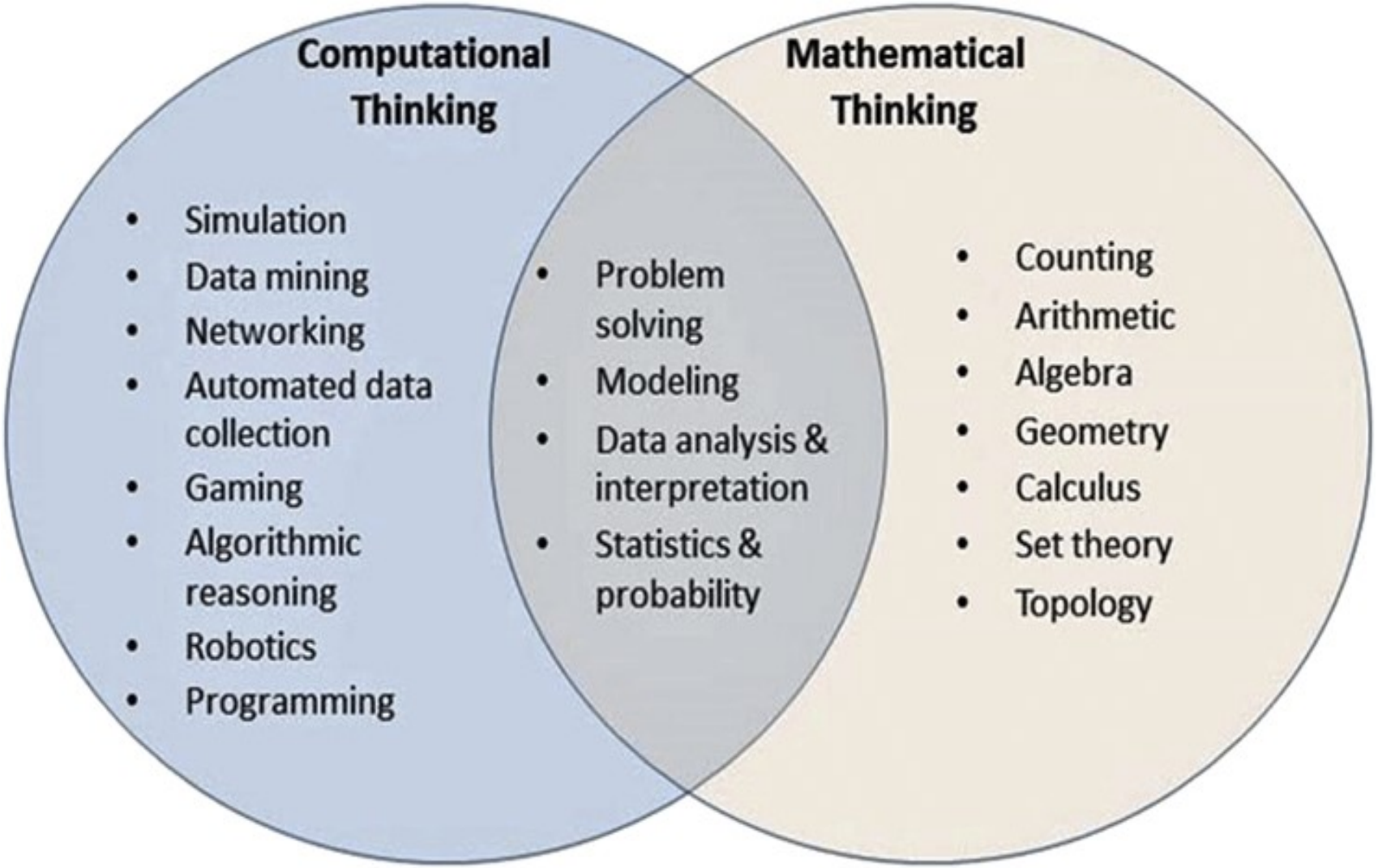
Algorithmic thinking?

- **Algorithmic thinking** skills support the development of general reasoning, problem-solving and communication skills by giving students the skills to fluently interpret and design structured procedures and rule systems.

Algorithmic thinking

- **Algorithmic thinking**, or **computational thinking**, refers to thinking about these processes for solving problems.
- **Algorithmic thinking** is a way of getting to a solution through the clear definition of the steps needed.
- **Algorithmic thinking** is the use of algorithms, or step-by-step sets of instructions, to complete a task.
- **Algorithmic thinking** has close ties to computer science and mathematics, as algorithms are the key to completing sequences of code or chunking big problems into smaller, more solvable parts.
- **Computational thinking** refers to the thought processes involved in expressing solutions as computational steps or algorithms that can be carried out by a computer. (Cuny, Snyder, & Wing, 2010; Aho, 2011; Lee, 2016).

Computational Thinking vs Mathematical Thinking



Algorithmic thinking

- Thinking computationally is not programming. It is not even thinking like a computer! Simply put, **programming** tells a computer what to do and how to do it.
- **Computational thinking** enables us to work out exactly what to tell the computer to do.

Algorithmic Thinking

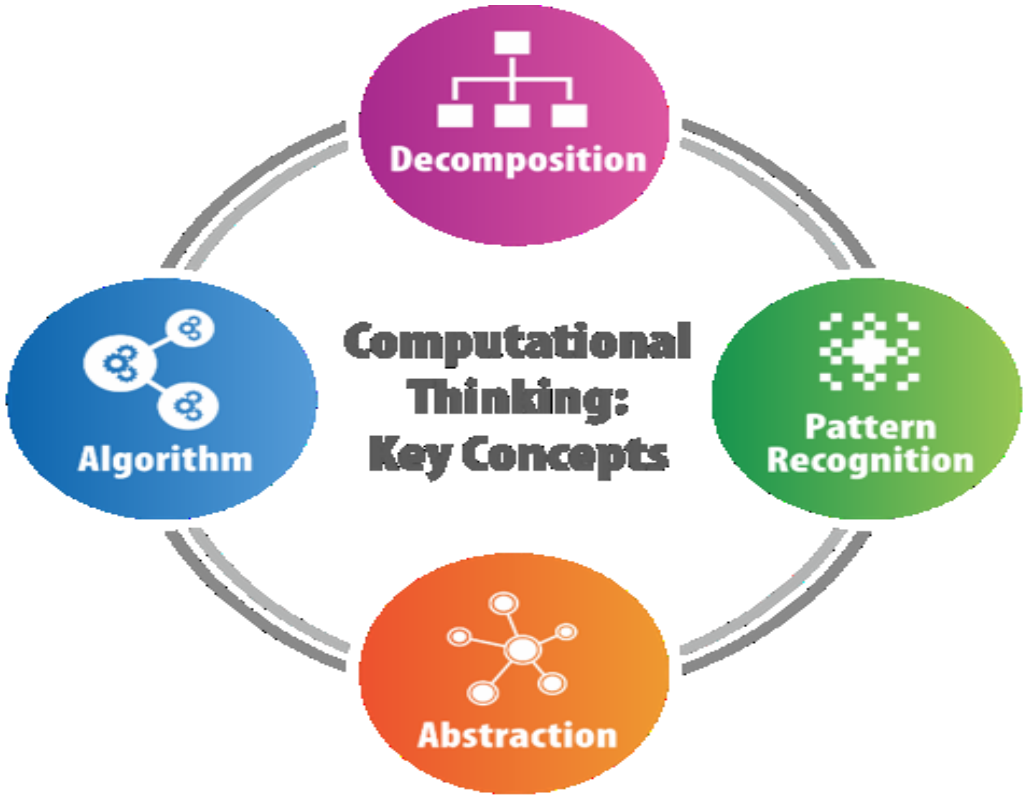
Algorithmic thinking is somehow a pool of abilities that are connected to constructing and understanding algorithms:

- - the ability to analyze given **problems**
- - the ability to specify a problem precisely
- - the ability to find the basic actions that are adequate to the given problem
- - the ability to construct a correct algorithm to a given problem using the basic actions
- - the ability to think about all possible special and normal cases of a problem
- - the ability to improve the efficiency of an algorithm

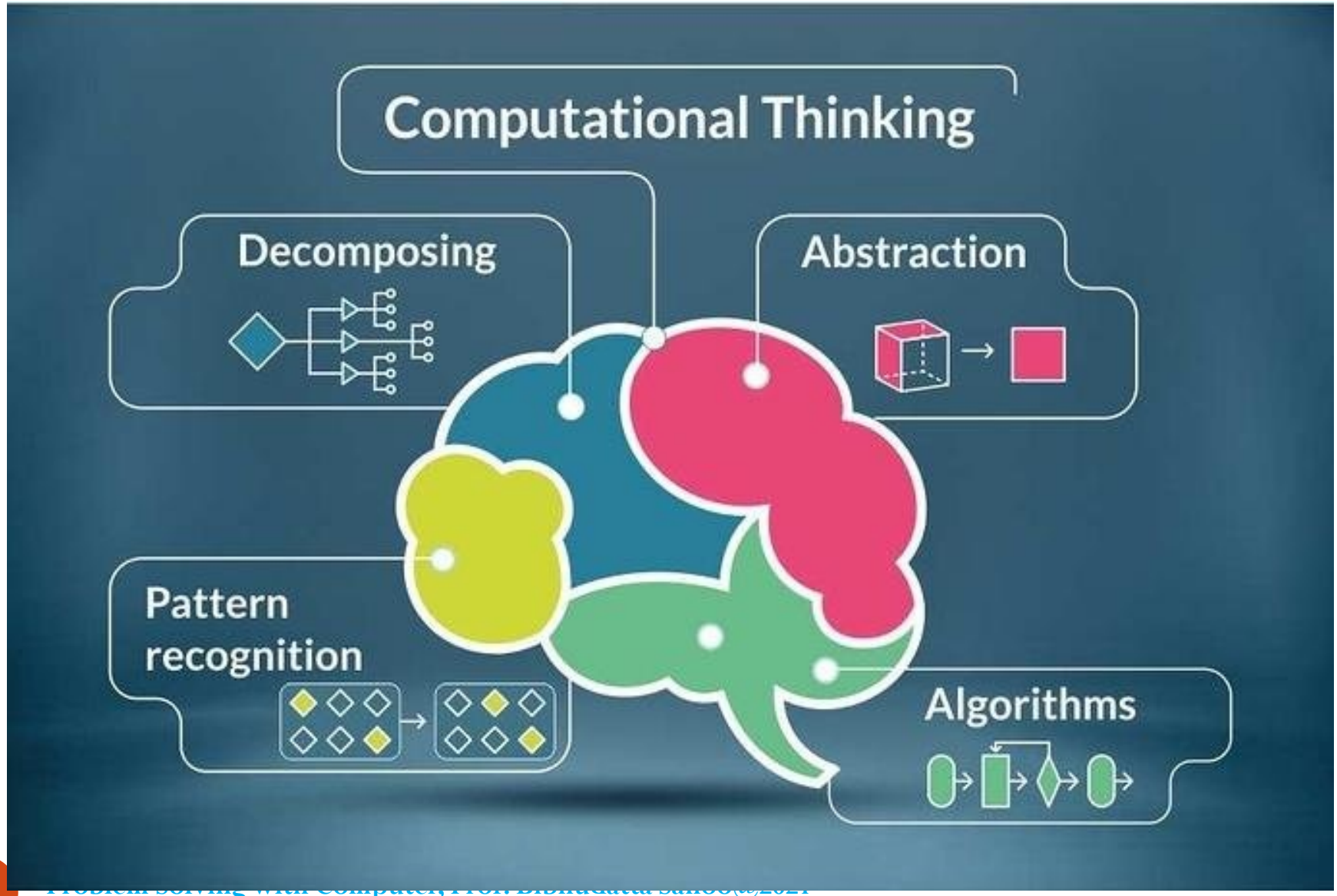
Algorithmic thinking has a strong creative aspect: the construction of new algorithms that solve given problems. If someone wants to do this he needs the ability of algorithmic thinking.

Computational Thinking

- **Computational Thinking** is an approach to problem solving with four key thinking processes; **decomposition**- taking ideas and problems apart, **pattern recognition**- looking for similarities or trends, **abstraction**- focusing on what's most important, and **algorithm design**- creating step-by-step instructions to solve a problem.



Computational thinking



Four key techniques of computational thinking:

- **Decomposition** - breaking down a complex problem or system into smaller, more manageable parts (e.g. where to go, how to complete the level)
- **Abstraction** - focusing on the important information only, ignoring irrelevant detail (e.g. weather, location of exit)
- **Pattern recognition** – looking for similarities among and within problems (e.g. knowledge of previous similar problems used)
- **Algorithms** - developing a step-by-step solution to the problem, or the rules to follow to solve the problem (e.g. to work out a step-by-step plan of action)

Computational thinking to solve a complex problem

STEP 1: SCOPE

- What would you like to do?
- What's the problem you have identified?
- Define problem statement

STEP 2: GENERATE

- Look for various solutions for your problem.
- Generate multiple scenarios – ideas

STEP 3: ANALYZE

- Go through each Idea/scenario and analyze

STEP 4: OPTIMIZE

- Fine tune the idea

Problem Solving and Algorithmic Thinking

Unit 1

- Problem Solving and Algorithmic Thinking Overview – problem definition, logical reasoning; Algorithm – definition, practical examples, properties, representation, algorithms vs programs.

Unit 2

- Algorithmic thinking – Constituents of algorithms – Sequence, Selection and Repetition, input-output; Computation – expressions, logic; algorithms vs programs, Problem Understanding and Analysis – problem definition, input-output, variables, name binding, data organization: lists, arrays etc. algorithms to programs.

Unit 3

- Problem solving with algorithms – Searching and Sorting, Evaluating algorithms, modularization, recursion. C for problem solving – Introduction, structure of C programs, data types, data input, output statements, control structures.

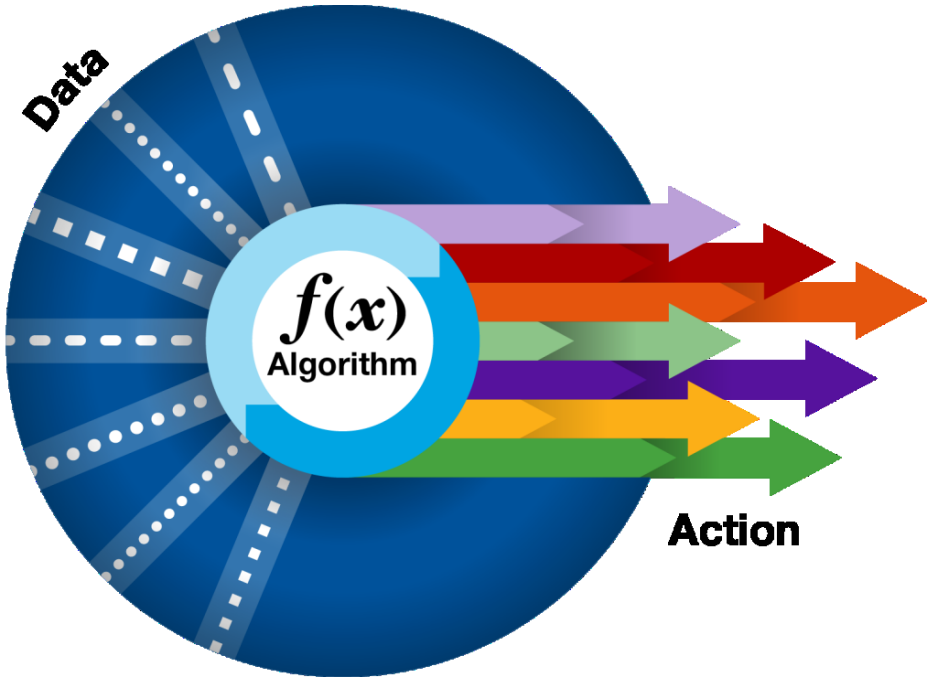
Reference: Problem Solving and Algorithmic Thinking

- Riley DD, Hunt KA. Computational Thinking for the Modern Problem Solver. CRC press; 2014 Mar 27.
- Ferragina P, Luccio Computational Thinking: First Algorithms, Then Code. Springer; 2018.
- Beecher Computational Thinking: A beginner's guide to Problem-solving and Programming.BCS Learning & Development Limited; 2017.
- Curzon P, McOwan The Power of Computational Thinking: Games, Magic and Puzzles to help you become a computational thinker. World Scientific Publishing Company; 2017.

Algorithmic business

Algorithmic business

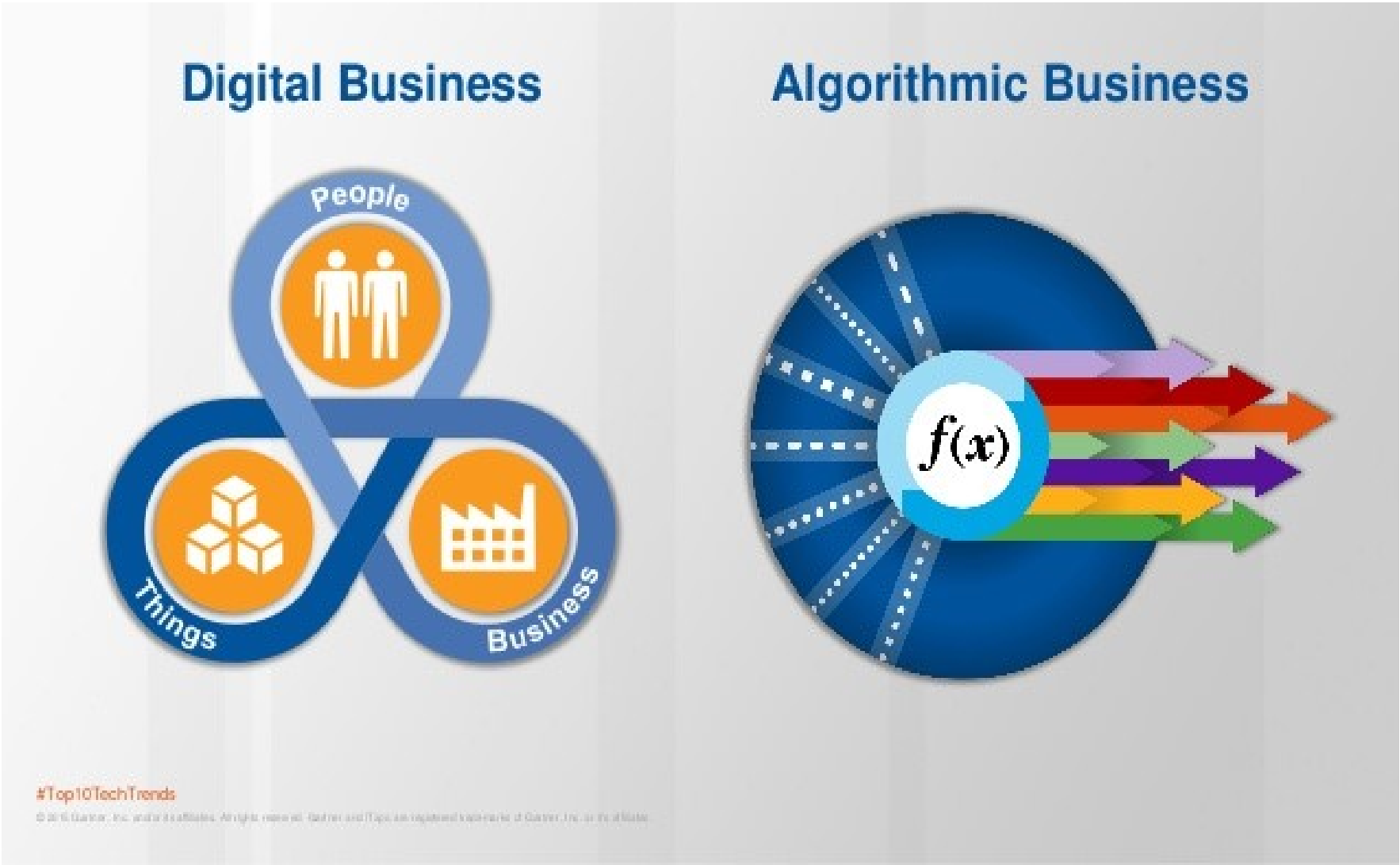
- **Algorithmic business** is the industrialized use of complex mathematical algorithms pivotal to driving improved business decisions or process automation for competitive differentiation.
- Algorithmic business provides the speed and scale to accelerate digital business to deliver even greater impact.



Algorithm application in business

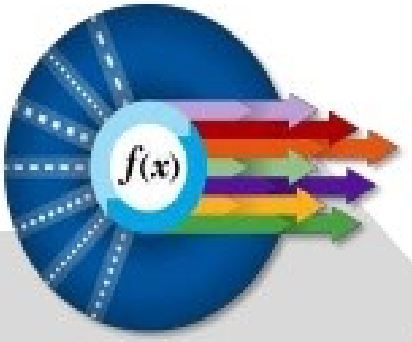
- Algorithms have actually been at the heart of some of the most successful corporate empires in the world.
- Google was born from an algorithm helping internet users find the information they need; while Coca-Cola boasts of its "secret recipe" – its own version of an algorithm.
- Meanwhile, the likes of the traders from Wall Street, Amazon and more have all relied on algorithms in the way they work.
- With organisations now generating more data than ever, collecting data is no longer a problem – instead, it's all about how that data can be used: and processing data is done best when it's achieved through algorithmic software.

Digital business vs Algorithmic business



What Is Changed?

The Internet of Things creates a digital version of physical world devices.



Digital business creates new business designs through merging the physical with the digital.

Algorithmic business applies the knowledge encapsulated in algorithms — business gets smarter and more powerful!

© 2016 Gartner, Inc. and/or its affiliates. All rights reserved.

Gartner

Algorithmic business: An Introduction

- Algorithmic business is the industrialized use of complex mathematical algorithms pivotal to driving improved business decisions or process automation for competitive differentiation.
- Algorithmic business provides the speed and scale to accelerate digital business to deliver even greater impact.
- With sufficient data sources and enough data to ensure algorithms are capable of offering real-time and actionable results that benefit the enterprise.

Example: Netflix

- Netflix is a classic example of an algorithmic business model. In fact, the success of their streaming video service is owed almost entirely to algorithms.
- How does Netflix learn your preferences and make quality viewing recommendations? Algorithms.
- How does viewer behavior inform Netflix on the shows to make and the viewing demographics to cater their content toward? Once again, algorithms.
- Netflix as we know it would not exist but for sophisticated algorithms that allow Netflix executives to take meaningful action based on an algorithm's ability to provide clear insight into the big picture of big data.

Example: Shipping company

- Consider a shipping company such as UPS. Dropping off packages efficiently is a critical component of their business model, but there are a number of variables to consider.
- Truck maintenance, employee work hours, traffic and more all impact the ability to follow through on delivery times. A quality algorithm could sift through vast amounts of applicable data to determine the best route and delivery strategies for all the packages on each truck.
- Companies cannot afford to ignore this level of advanced efficiency. Before business algorithms transformed enterprise, companies were forced to rely on teams of analysts to find patterns that were used to create data trends to use for better decision-making. This advanced work may take weeks to complete and calculate, whereas business algorithms are equipped to perform those same calculations in mere seconds.

Thanks for Your Attention!

Questions

- How does one calculate the running time of an algorithm?
- How can we compare two different algorithms?
- How do we know if an algorithm is `optimal'?
- You are given list of numbers, obtained by rotating a sorted list an unknown number of times. Write a function to determine the minimum number of times the original sorted list was rotated to obtain the given list. Your function should have the worst-case complexity of O(log N), where N is the length of the list. You can assume that all the numbers in the list are unique.

  Example: The list [5, 6, 9, 0, 2, 3, 4] was obtained by rotating the sorted list [0, 2, 3, 4, 5, 6, 9] 3 times.

Reference text