Data Structures and Algorithms

Lecture 01 – Introduction

（Basics of Algo and Data Structure)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

Course Administration

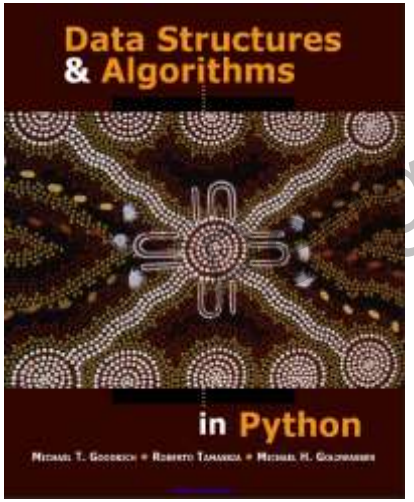
Instructor: Pengju Ren

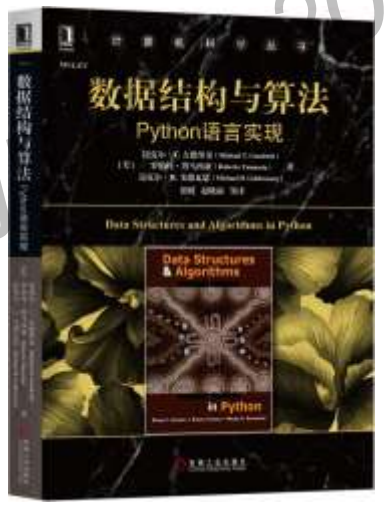
TA: Chenyu Ma（Ph.D Candidate)

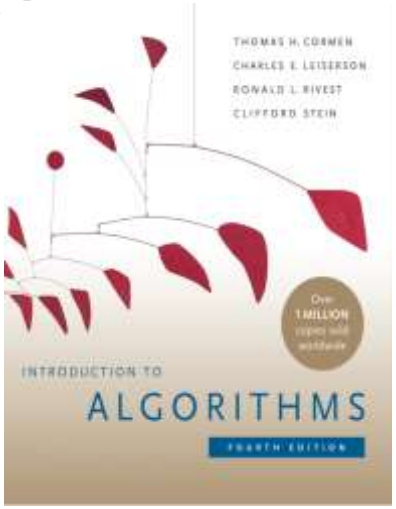
Lectures: Two 100-minutes lectures a week

Textbook: Data Structures & Algorithms

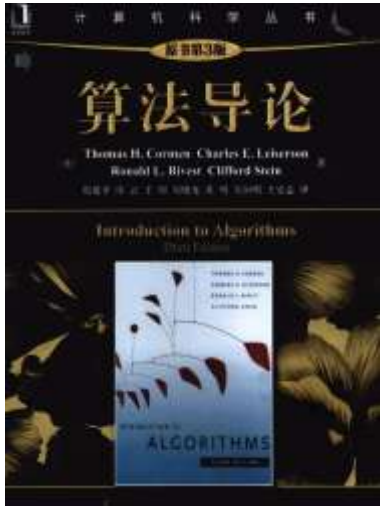
Prerequisite: Computer Programming







4th Edition



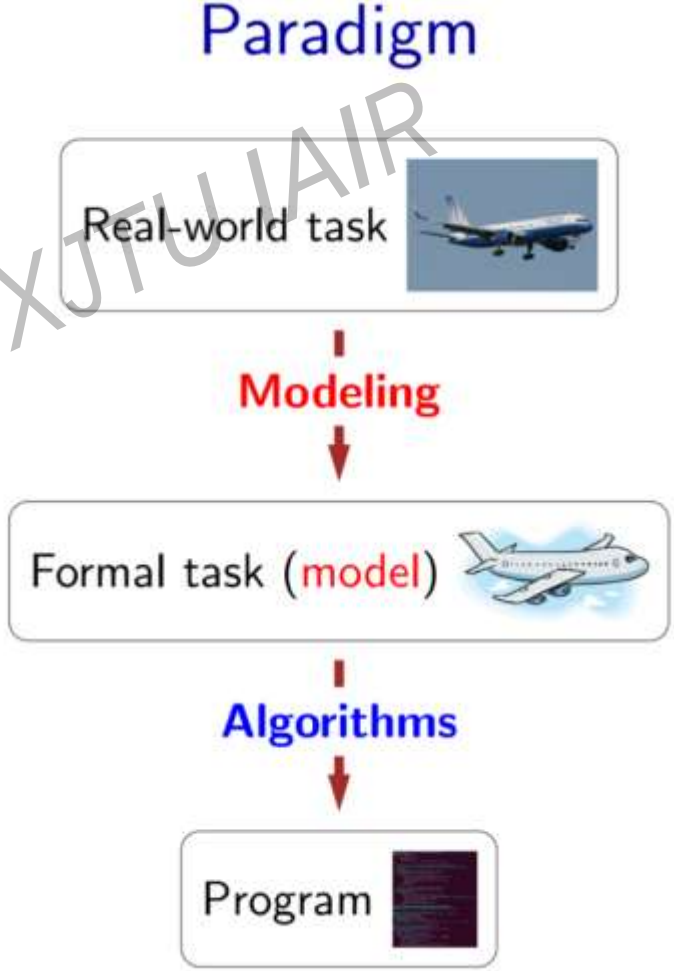
3th Edition

Preface

Solve Computational Problems

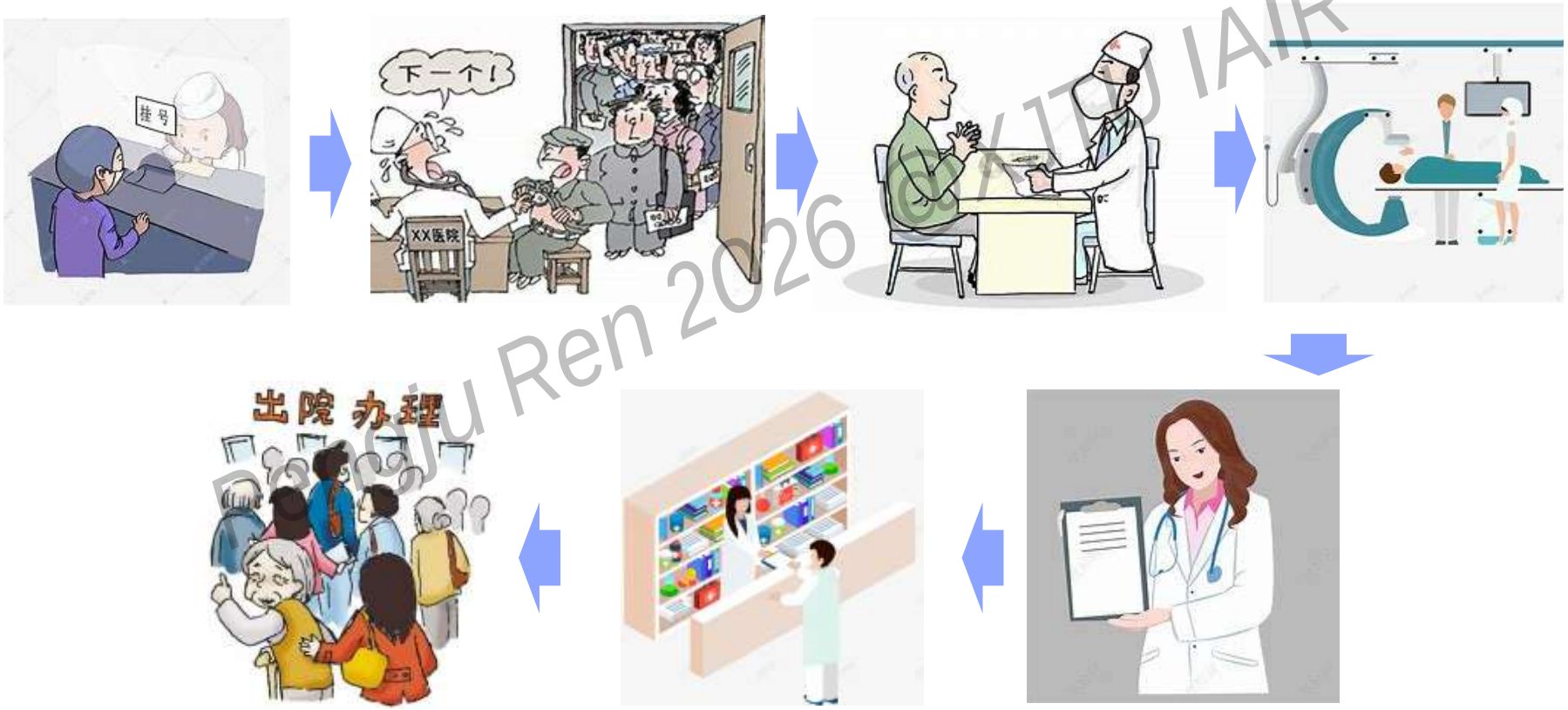
The entire course is driven by *"Problem Solving"*, gradually leading students to discover the relevant knowledge

AI Tools are encouraged but always remember *who is in control*



What is an Algorithm?

Set of instructions to perform a Task



What is an Algorithm?

Algorithms in Computer Science: Set of rules for a computer program to accomplish a specific task

	Algorithm	Program
Essence	Logical steps to solve a problem ("what" and "how"). Domain Knowledge is necessary.	Concrete implementation of an algorithm (executable code written in a programming language).
Form	Abstract, language-independent (can be described in natural language, pseudocode, flowcharts).	Concrete, dependent on a specific programming language (e.g., Python, C++).
Executable	❌ Not directly executable (needs to be translated into a program).	✅ Directly executable on a computer.
Focus	Correctness, efficiency (time/space complexity), generality.	Runnability, robustness, user experience, platform compatibility.
Hardware	Somehow overlook HW and OS	HW and OS matters

Common Types of Algorithms

Search	To find a specific value in a data set
Sorting	To sort data in ascending or descending order
Graph/Tree Related	To work with data that can be represented as a graph/tree
Divide and Conquer	To solve problems by breaking them down into smaller sub-problems, *solving each independently and then combining the results*
Recursive	To solve problems by breaking them down into smaller sub-problems that are *similar in nature*
Dynamic Programming	To solve problems by breaking them down into smaller sub-problems, *store the results of subproblems to reduce time complexities*

How to Analysis Algorithms?

Experimental Analysis: Run-time measured on various inputs

easier than theoretical analysis, but difficult to predict precise running time. Also, it is hardware/software and OS dependent

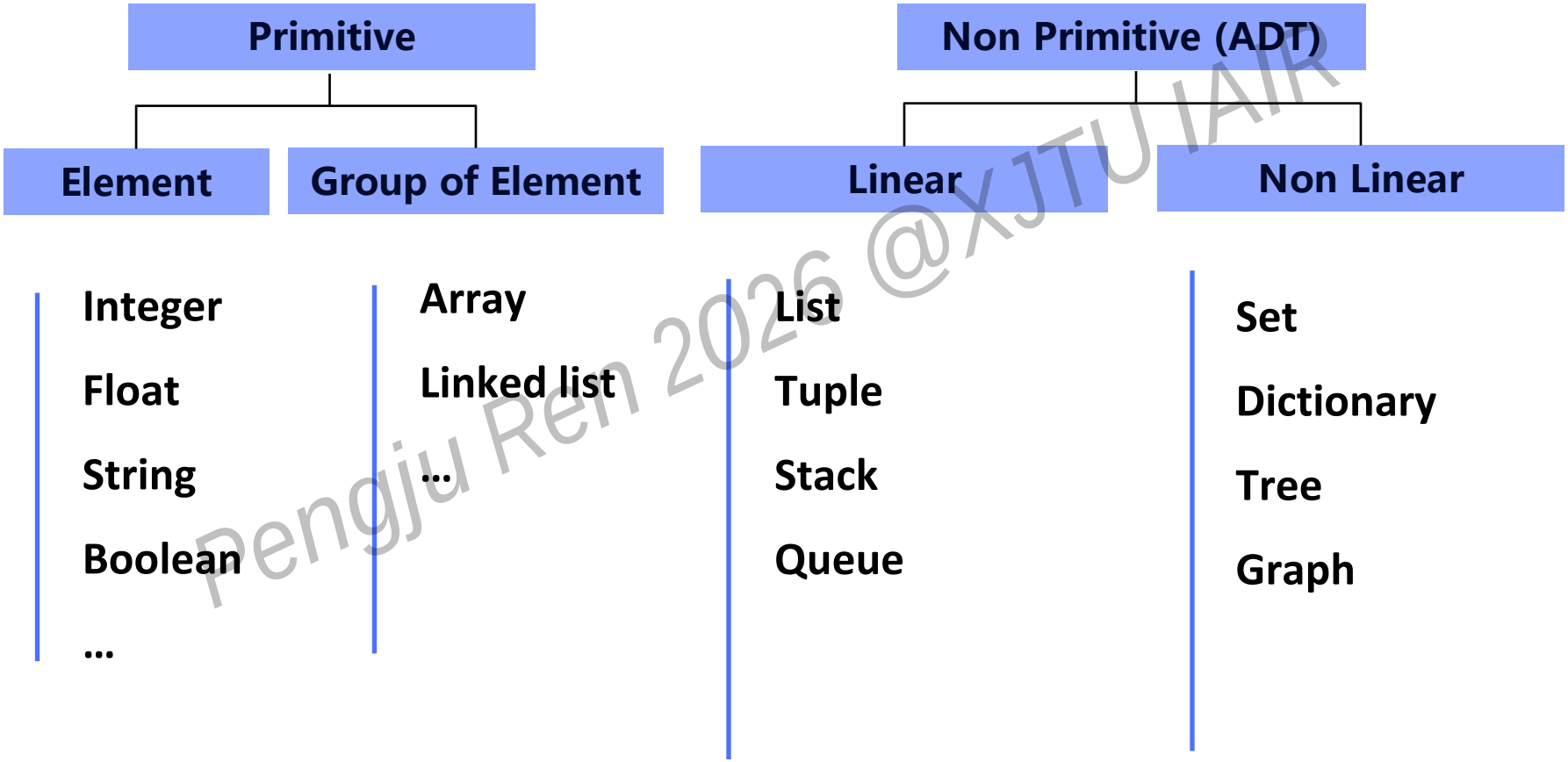
Theoretical Analysis: Considering all possible inputs and it is performed on description of Algorithm, and is independent of hardware and software

What is a Data Structure ?

Data Structures are *different ways* of organizing data on computer, that can be used *effectively* (store, arrange, access and manipulation).

Data processing & storing

Types of Data Structures (in Python)



Why need abstract data type?

Data Structure:

- ☐ Representation
- ☐ How to store data
- ☐ Algorithms to support operations

Interface (API/ADT):

- ☐ Specification
- ☐ What data can store
- ☐ What operations are supported and what parameters are needed

E.g. class stack K[10] **Representation:** Array or Linked-list

Operations:

push()

pop()

peek()

isEmpty()

size()

del()

```
def push(self, item): """将元素压入栈顶"""
    try:
        self._stack.append(item)
    except TypeError:
        raise TypeError(f"元素类型必须为{self._stack.typecode}")
def pop(self): """弹出栈顶元素（若栈空则抛出IndexError）"""
    if self.isEmpty():
        raise IndexError("栈为空，无法执行pop操作")
    return self._stack.pop()  # array的pop方法实现高效出栈
def peek(self): """查看栈顶元素（不弹出）"""
    if self.isEmpty():
        raise IndexError("栈为空，无法执行peek操作")
    return self._stack[-1]  # 直接访问最后一个元素
def isEmpty(self): """检查栈是否为空"""
    return len(self._stack) == 0
def size(self):  """返回栈中元素数量"""
    return len(self._stack)
```

Why need abstract data type?

Data Structure:

- ☐ Representation
- ☐ How to store data
- ☐ Algorithms to support operations

Interface (API/ADT):

- ☐ Specification
- ☐ What data can store
- ☐ What operations are supported and what parameters are needed

E.g. class Student

Operations:

getName()

getAge()

enrollCourse（String CourseID)

dropCourse(String CourseID)

setScore(String CourseID, score)

getScore(String CouseID)

getGPA()

E.g. class Course

Operations:

addStudent(Student s)

removeStudent(Student s)

recordScore（Student s, double score)

getClassAvg()

prtRoster()

getStudentCnt()

Why need abstract data type?

Main reasons:

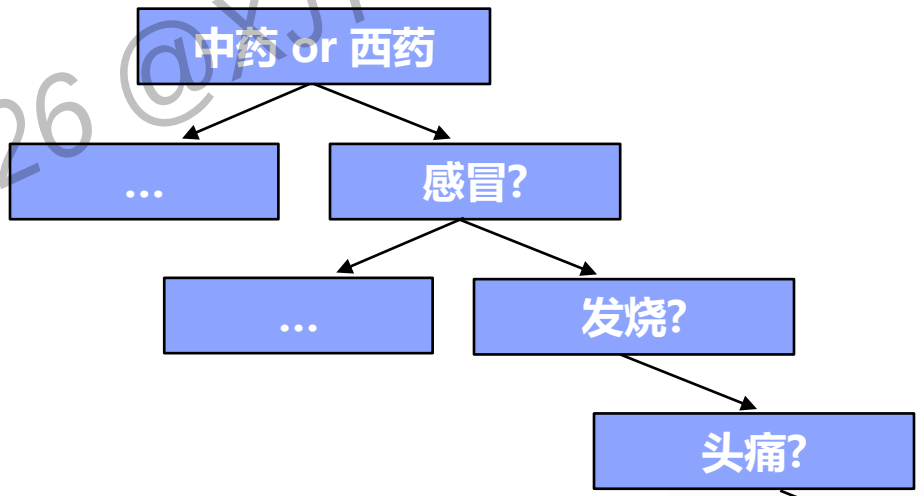
- **Encapsulation and Information Hiding: Protect data integrity, allow changes without affecting users**
- **Modularity: Break complex systems into manageable pieces**
- **Reusability: Well-defined ADTs can be used in multiple programs**
- **Abstraction: Focus on what operations do, not how they are implemented**
- **Ease of maintenance and debugging**

In object-oriented languages like *Java*, *C++*, or *Python*, classes are used to implement ADTs. ADTs are the foundation of modern software engineering—they help us manage complexity, collaborate in teams, and build robust, scalable systems.

Why are Data Structures and Algo important ?

Data Structure

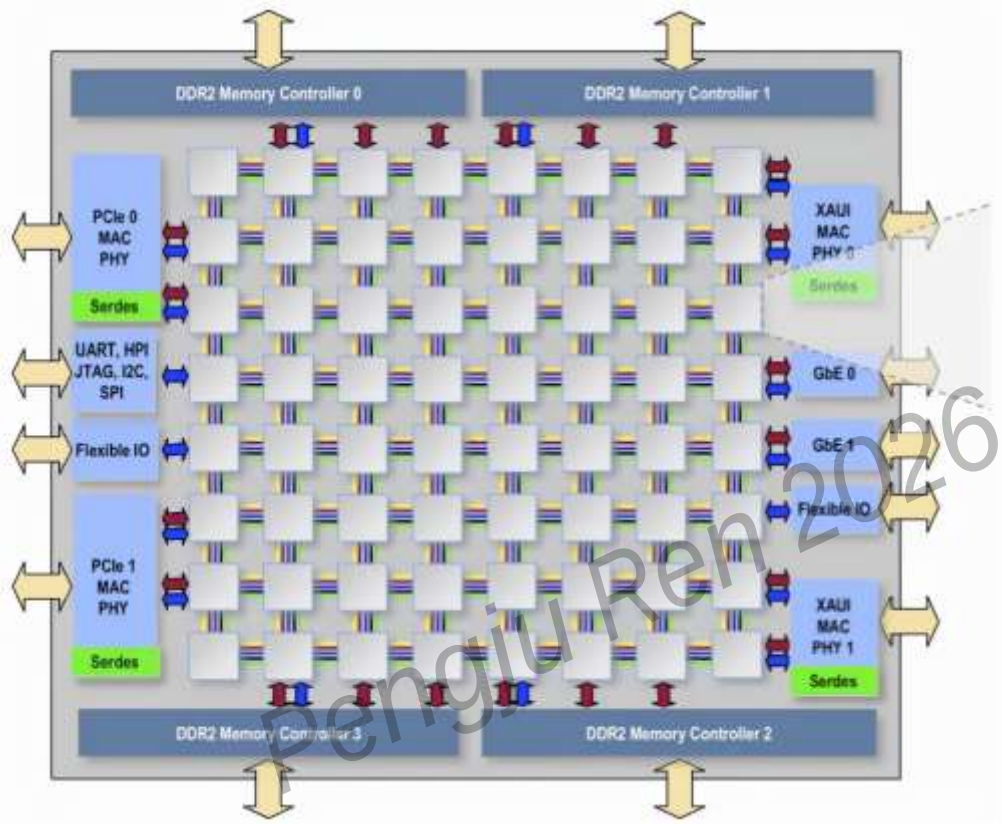
Algorithm

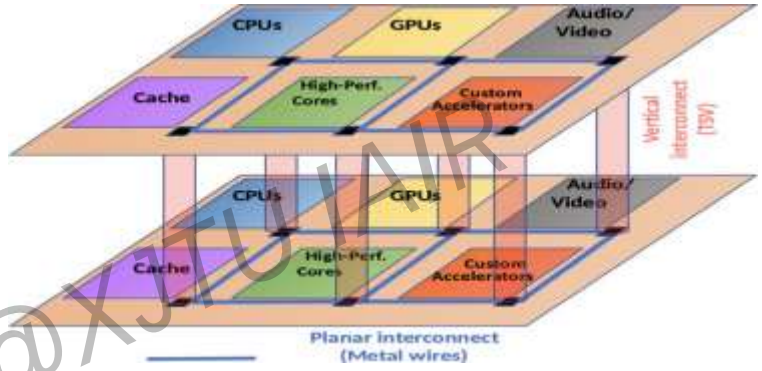


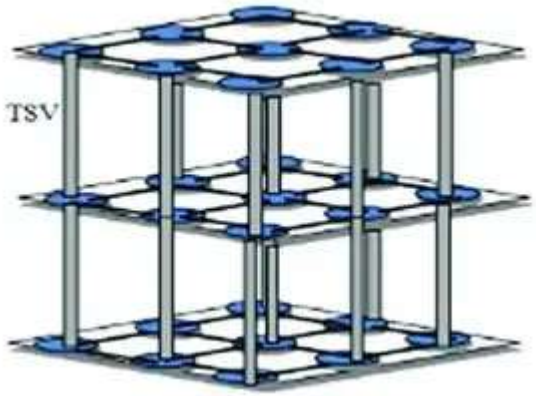

```
for (i=0, i<N; i++) {
    compare(M[i],Ibuprofen)?
}
```

```
If (){
  elesif(){
    elesif(){
          ...}
        }
    }
```

Problem Solving







Cycle Detection to avoid Deadlock in Manycore Processor

（e.g. Intel Xeon Chip or 3D Manycore System）

Summary

- The most crucial capability lies in formally modeling real-world problems, performing solvability analysis, and then finding suitable algorithms and data structures
- **Algorithm:** a Step-by-Step procedure for performing some task in a finite amount of time
- **Data Structure:** is a systematic way of organizing and accessing data
- **Abstract Data Types:** separation of implementation and usage, support for reuse of code, and the ability to use modular programming to solve complex problems