Chapter 5

Computer Components

Chapter Goals

- Describe how computer memory is organized and accessed
- Name and describe different auxiliary storage devices
- Define three alternative parallel computer configurations

Stored-Program Concept

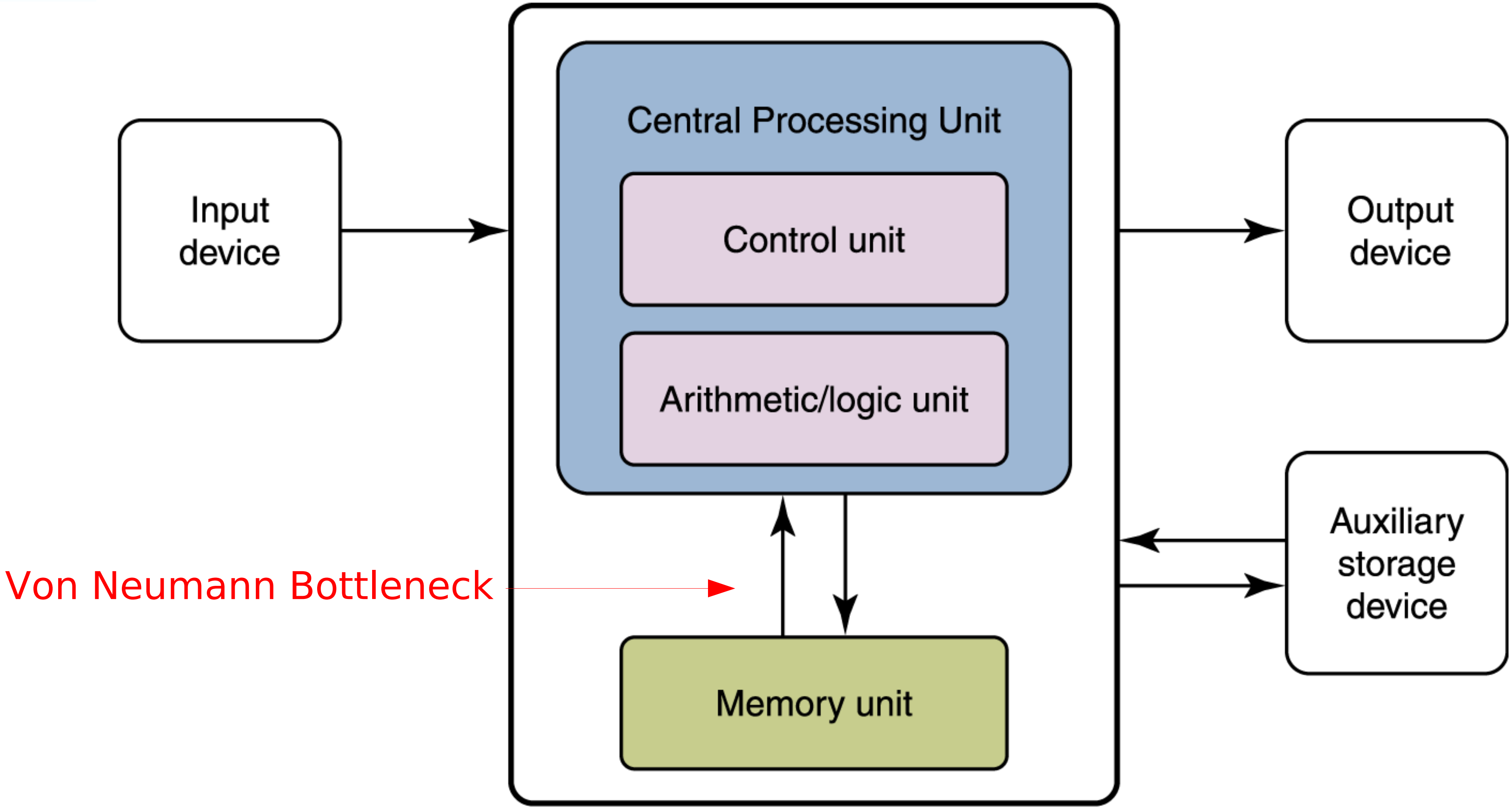


Figure 5.1 **The von Neumann architecture**

RAM and ROM

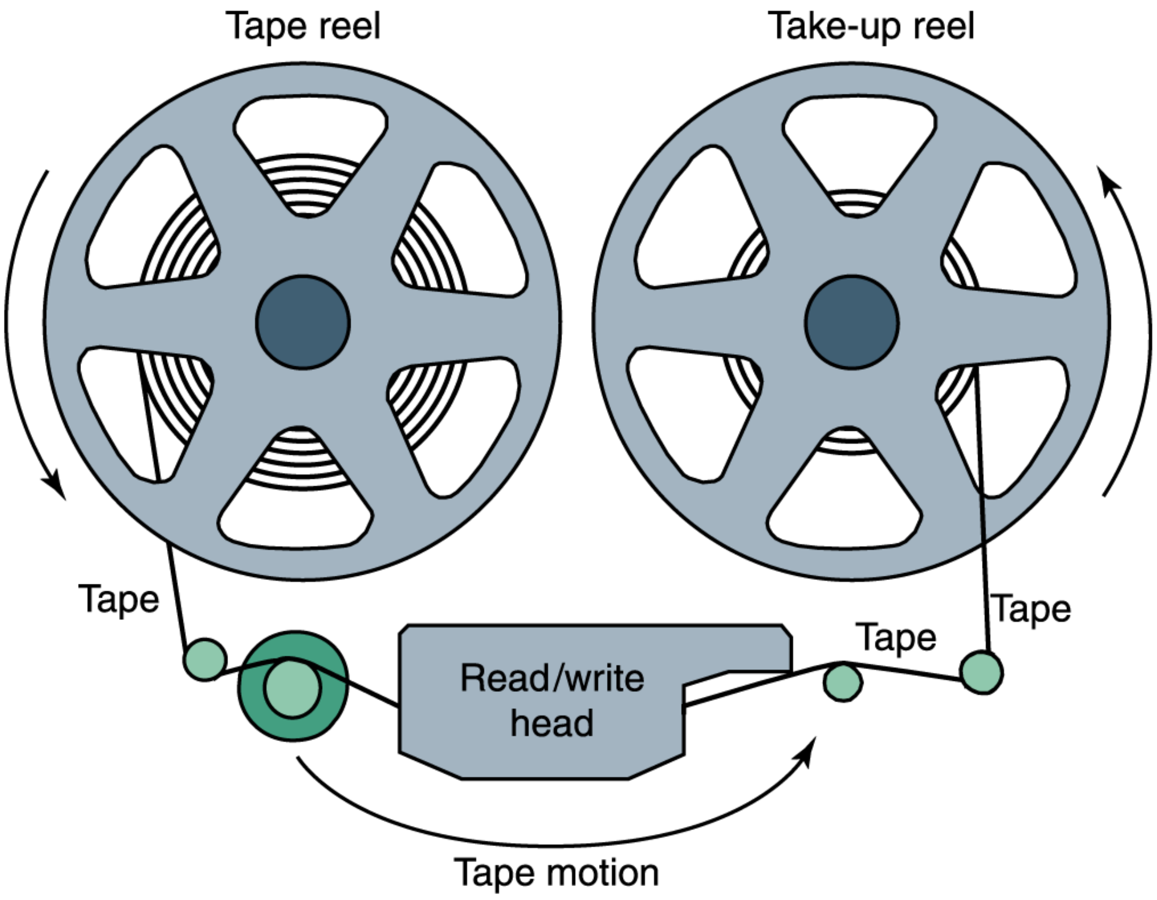
- **RAM** stands for **Random Access Memory**
  - Inherent in the idea of being able to access each location is the ability to change the contents of each location
- **ROM** stands for **Read Only Memory**
  - The contents in locations in ROM cannot be changed
- RAM is volatile, ROM is not
  - This means that RAM does not retain its bit configuration when the power is turned off, but ROM does

Secondary Storage Devices

- Because most of main memory is volatile and limited, it is essential that there be other types of storage devices where programs and data can be stored when they are no longer being processed
- Secondary storage devices can be installed within the computer box at the factory or added later as needed

Magnetic Tape

- The first truly mass auxiliary storage device was the magnetic tape drive

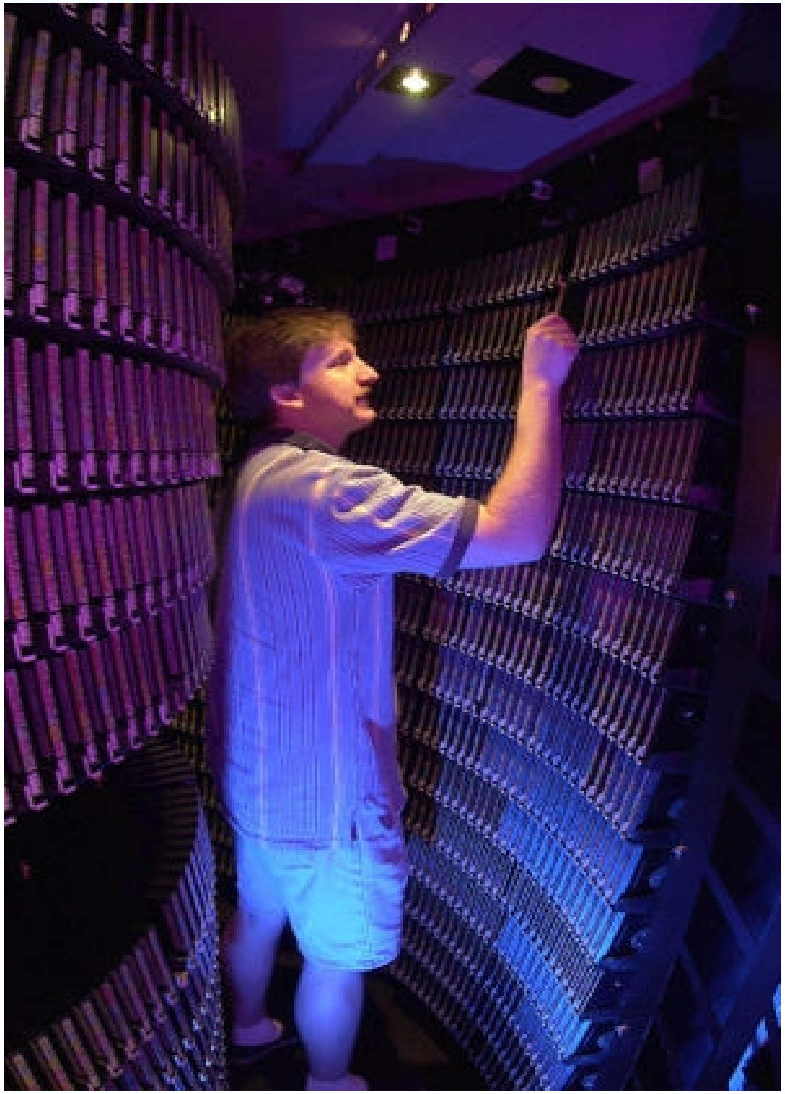


A magnetic tape storage mechanism

Figure 5.4 A magnetic tape

Tape Silo

- Handled by a robot
- Travels 80 mph



Magnetic Disks

- A read/write head travels across a spinning magnetic disk, retrieving or recording data

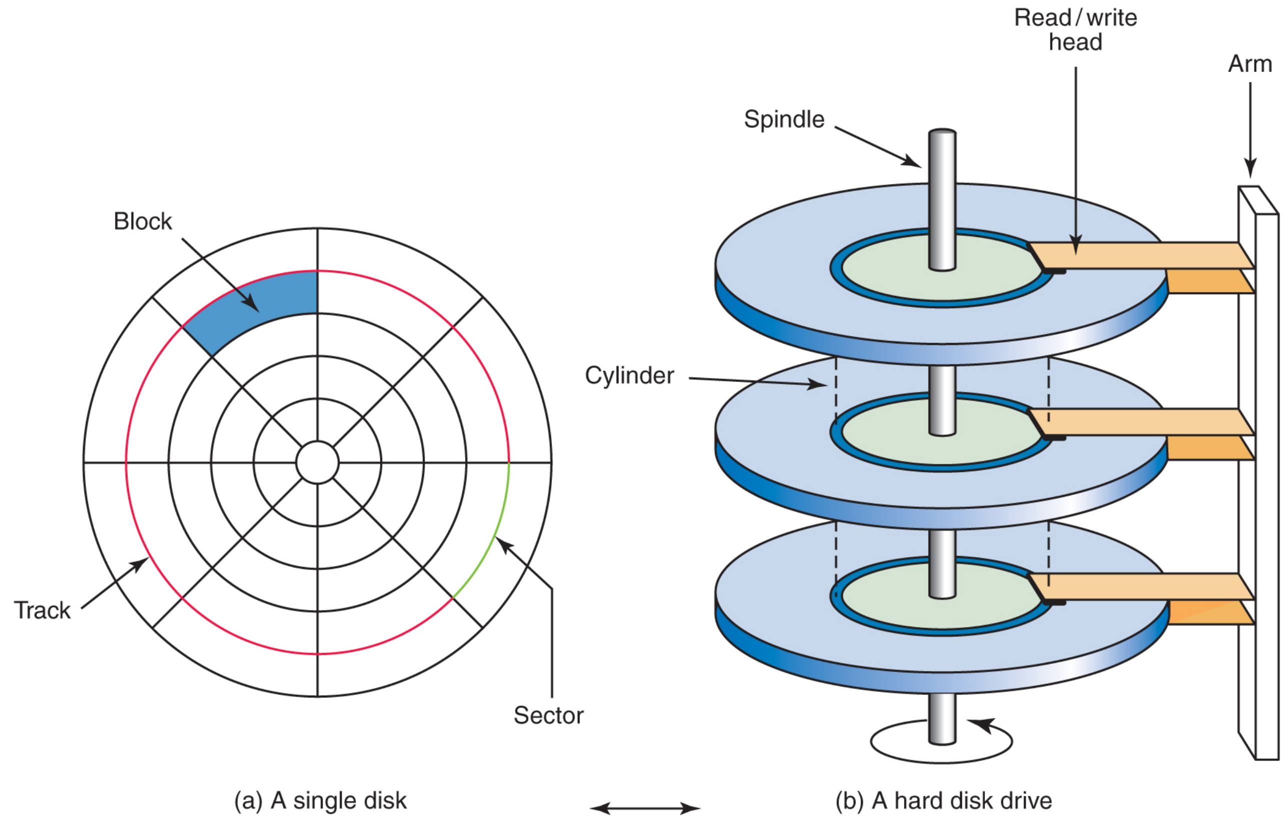


Figure 5.5
The organization of a magnetic disk

A Hard Drive In Action

- Inside A Hard Drive - Simple operations performed by a hard drive with no cover, so that you can see what it looks like inside. This experiment was performed on an old hard drive, do not try this with newer expensive hard drives, it is a bit risky. -from digg.com

- What Happens When You Drop A DIsk - Or Two!

Compact Disks

- A CD drive uses a laser to read information stored optically on a plastic disk
- CD-ROM is Read-Only Memory
- DVD stands for Digital Versatile Disk

Touch Screens

- **Touch screen** A computer monitor that can respond to the user touching the screen with a stylus or finger
- There are four types
  - Resistive
  - Capacitive
  - Infrared
  - Surface acoustic wave (SAW)

Touch Screens

- **Resistive touch screen** A screen made up of two layers of electrically conductive material.
  - One layer has vertical lines, the other has horizontal lines.
  - When the top layer is pressed, it comes in contact with the second layer which allows electrical current to flow.
  - The specific vertical and horizontal lines that make contact dictate the location on the screen that was touched.

Touch Screens

- **Capacitive touch screen** A screen made up of a laminate applied over a glass screen.
  - The laminate conducts electricity in all directions, and a very small current is applied equally on the four corners.
  - When the screen is touched, current flows to the finger or stylus.
  - The location of the touch on the screen is determined by comparing how strong the flow of electricity is from each corner.

Touch Screens

- **Infrared touch screen** A screen with crisscrossing horizontal and vertical beams of infrared light
  - Sensors on opposite sides of the screen detect the beams.
  - When the user breaks the beams by touching the screen, the location of the break can be determined.

Touch Screens

- **Surface acoustic wave** (SAW) A screen with crisscrossing high frequency sound waves across the horizontal and vertical axes.
  - When a finger touches the surface, the corresponding sensors detect the interruption and determine the location of the touch.

Synchronous processing

- One approach to parallelism is to have multiple processors apply the same program to multiple data sets

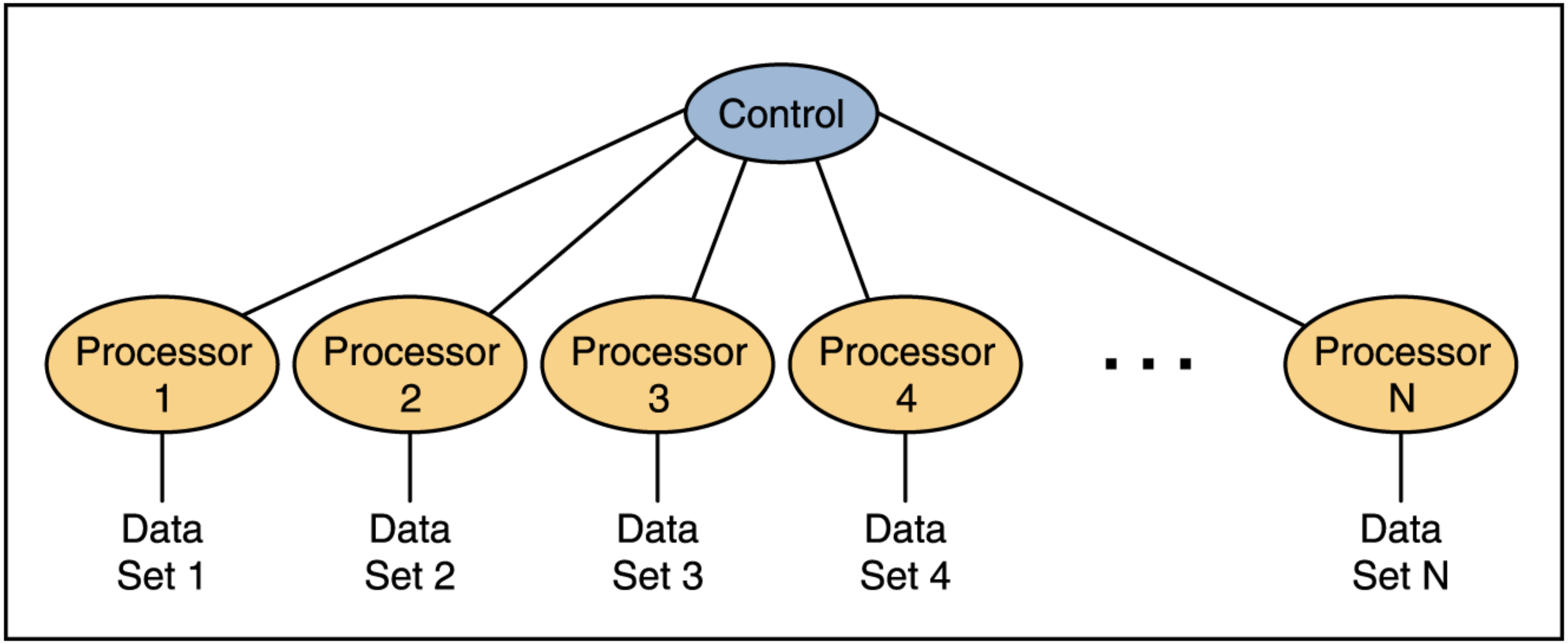


Figure 5.7 **Processors in a synchronous computing environmen - SIMD**

Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Pipelining

- Arranges processors in tandem, where each processor contributes one part to an overall computation

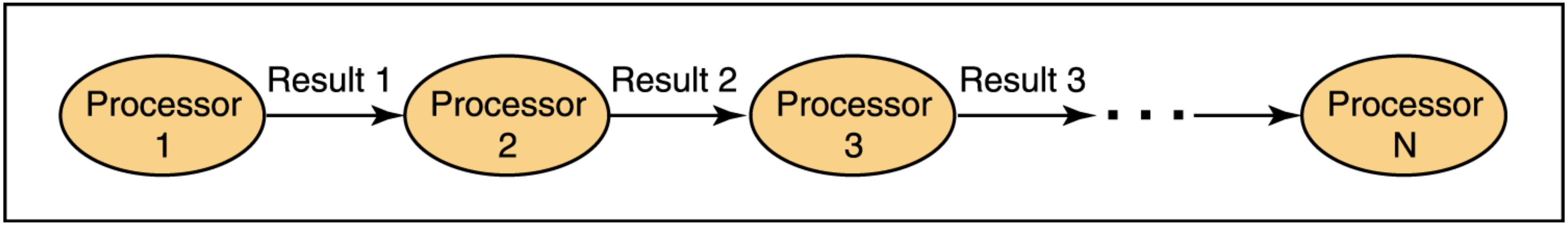
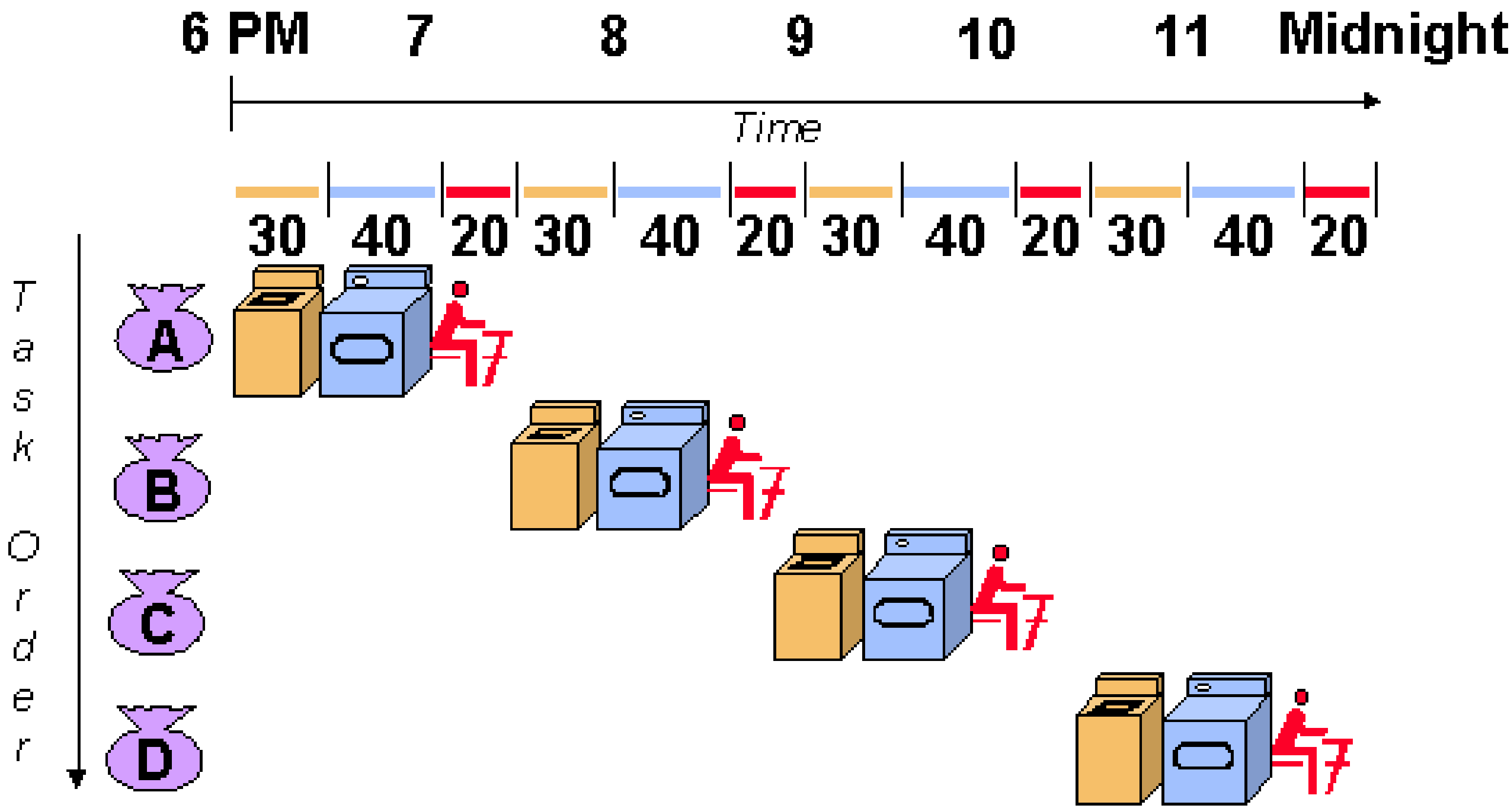
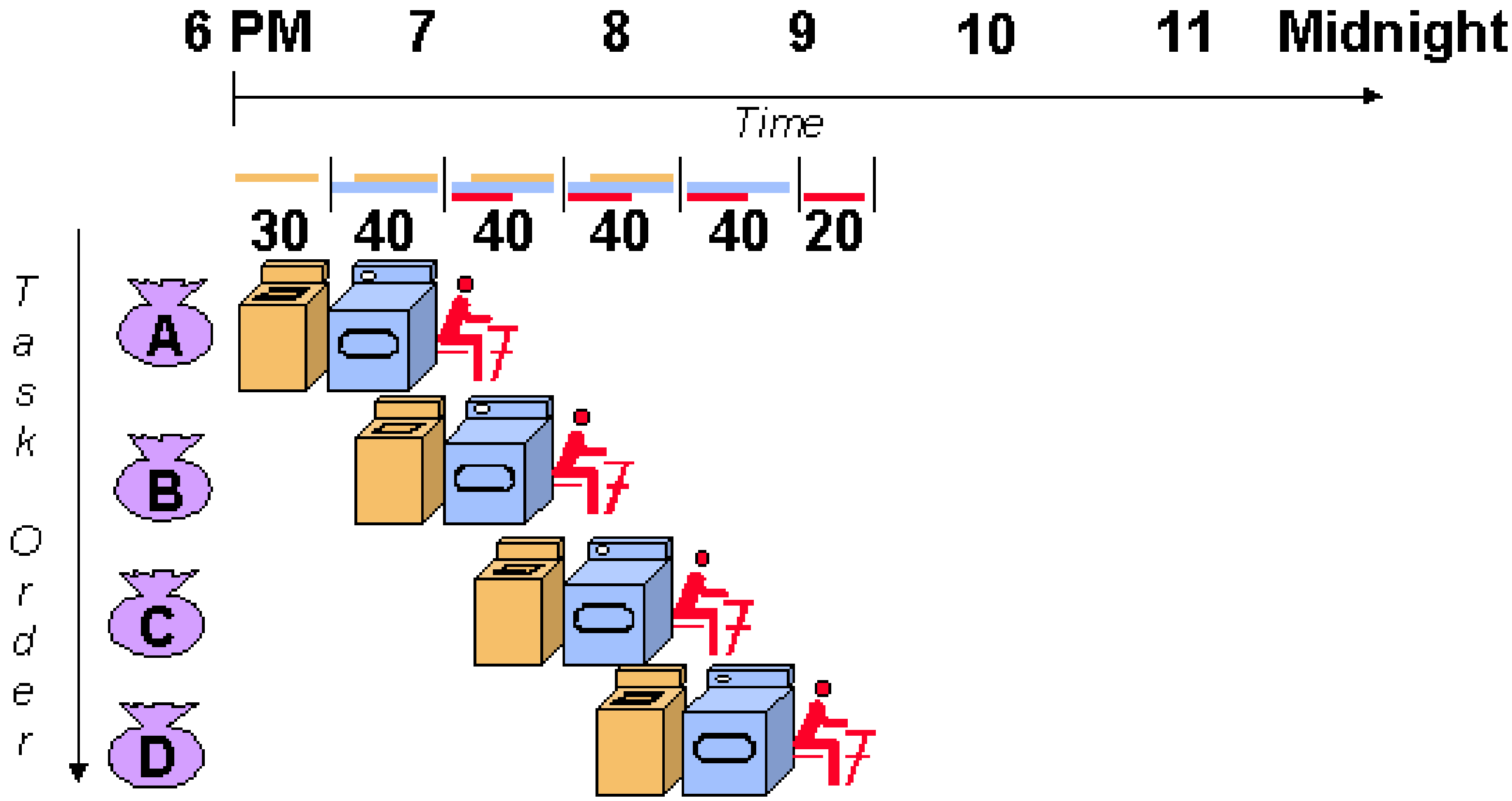


Figure 5.8 **Processors in a pipeline**

Without PipeLining



With PipeLining



View Pipelining In Action

Independent Processing with Shared Memory

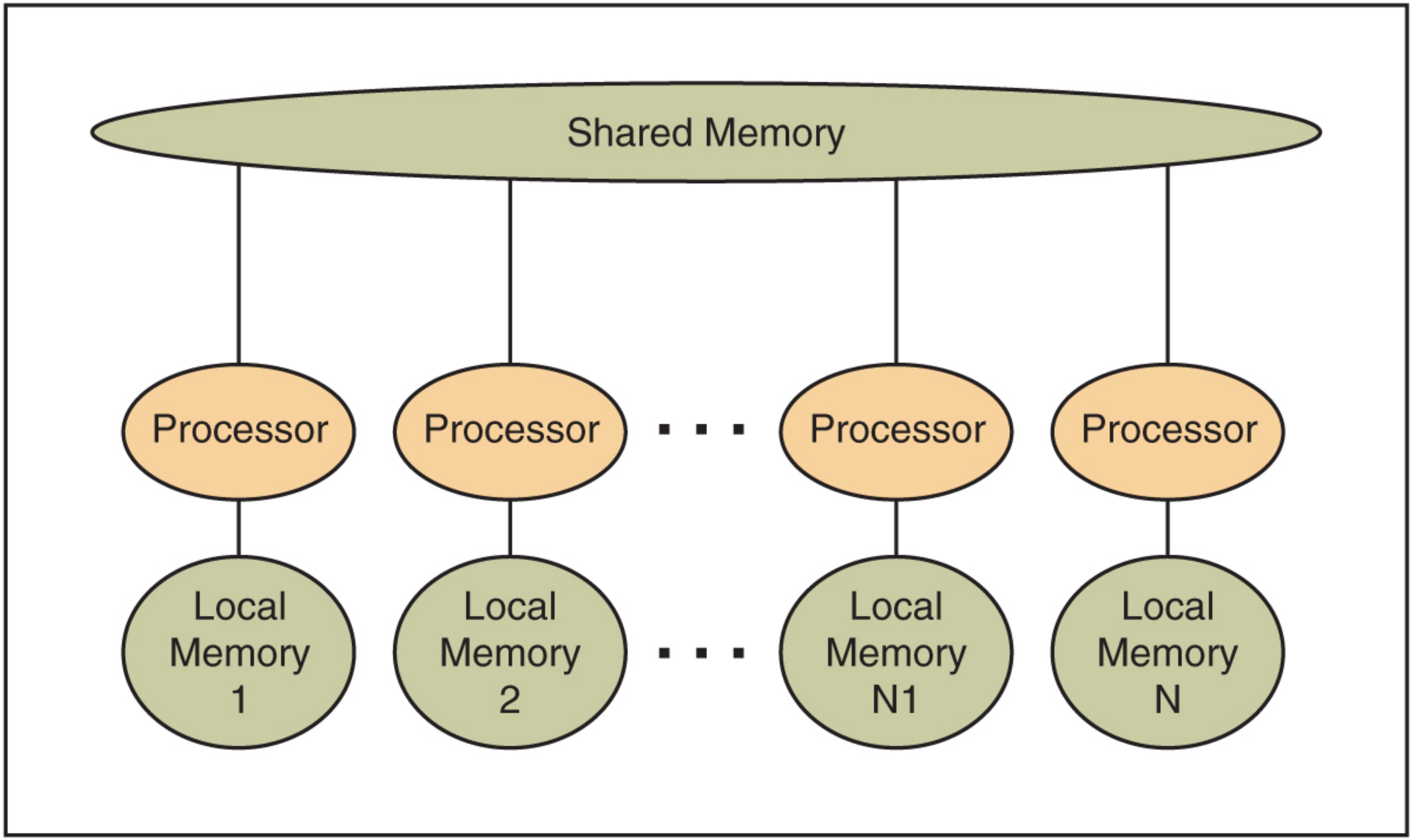
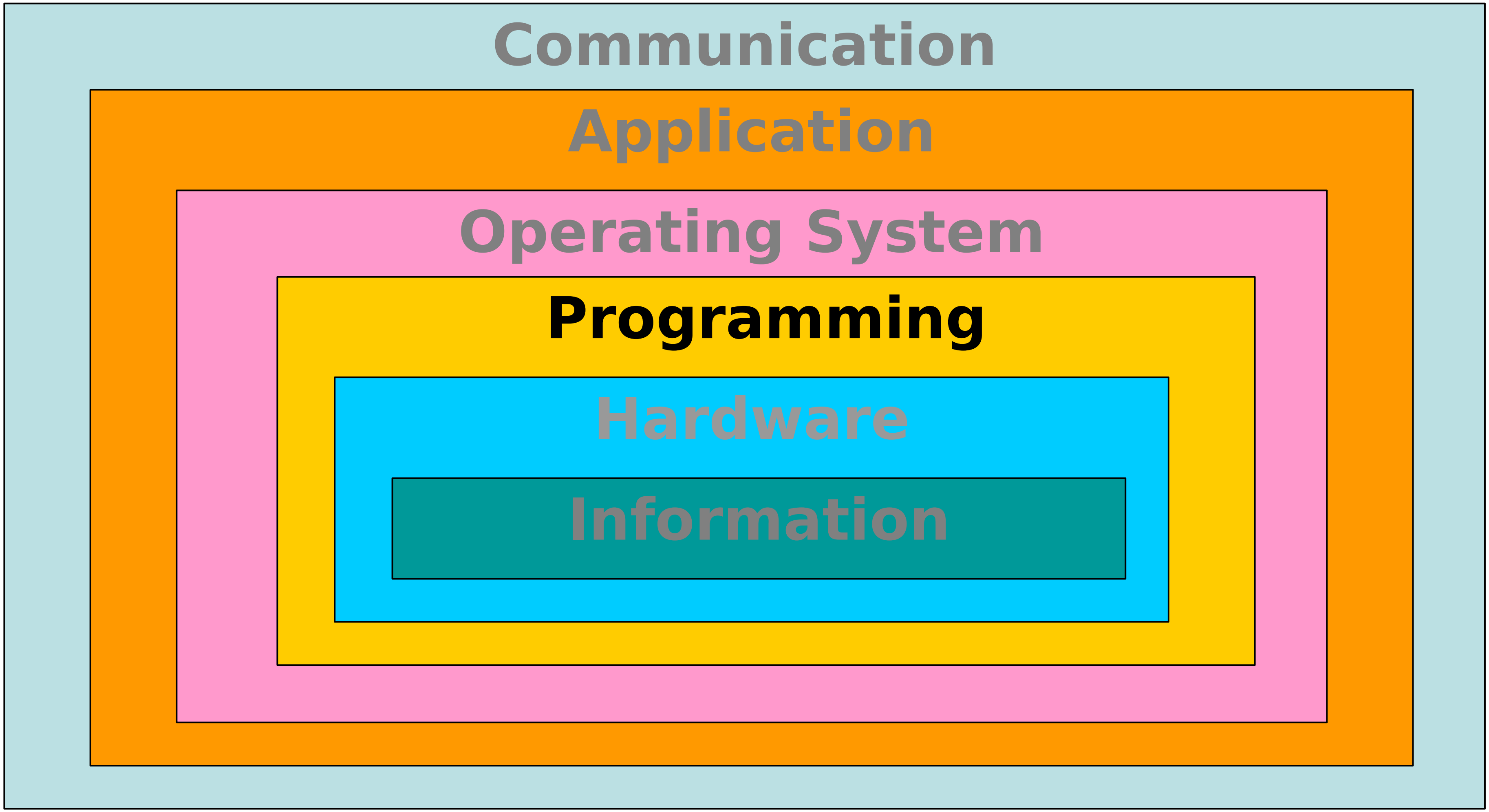


Figure 5.9 **A shared-memory configuration of processors**

Chapter 6

Problem Solving and Algorithm Design

Layers of a Computing System



Chapter Goals

- Determine whether a problem is suitable for a computer solution
- Describe the computer problem-solving process and relate it to Polya's How to Solve It list
- Distinguish between following an algorithm and developing one
- Apply top-down design methodology to develop an algorithm to solve a problem

Problem Solving

- **Problem solving** The act of finding a solution to a perplexing, distressing, vexing, or unsettled question

Problem Solving

- G. Polya wrote *How to Solve It: A New Aspect of Mathematical Method*
- His How to Solve It list is quite general
  - Written in the context of solving mathematical problems
  - The list becomes applicable to all types of problems

Ask Questions…

- …to understand the problem
  - *What do I know about the problem?*
  - *What is the information that I have to process in order the find the solution?*
  - *What does the solution look like?*
  - *What sort of special cases exist?*
  - *How will I recognize that I have found the solution?*

Look for Familiar Things

- You should never reinvent the wheel
- In computing, you see certain problems again and again in different guises
- A good programmer sees a task, or perhaps part of a task (a subtask), that has been solved before and plugs in the solution

Divide and Conquer

- Break up a large problem into smaller units that we can handle
  - Applies the concept of abstraction
  - The divide-and-conquer approach can be applied over and over again until each subtask is manageable

Algorithms

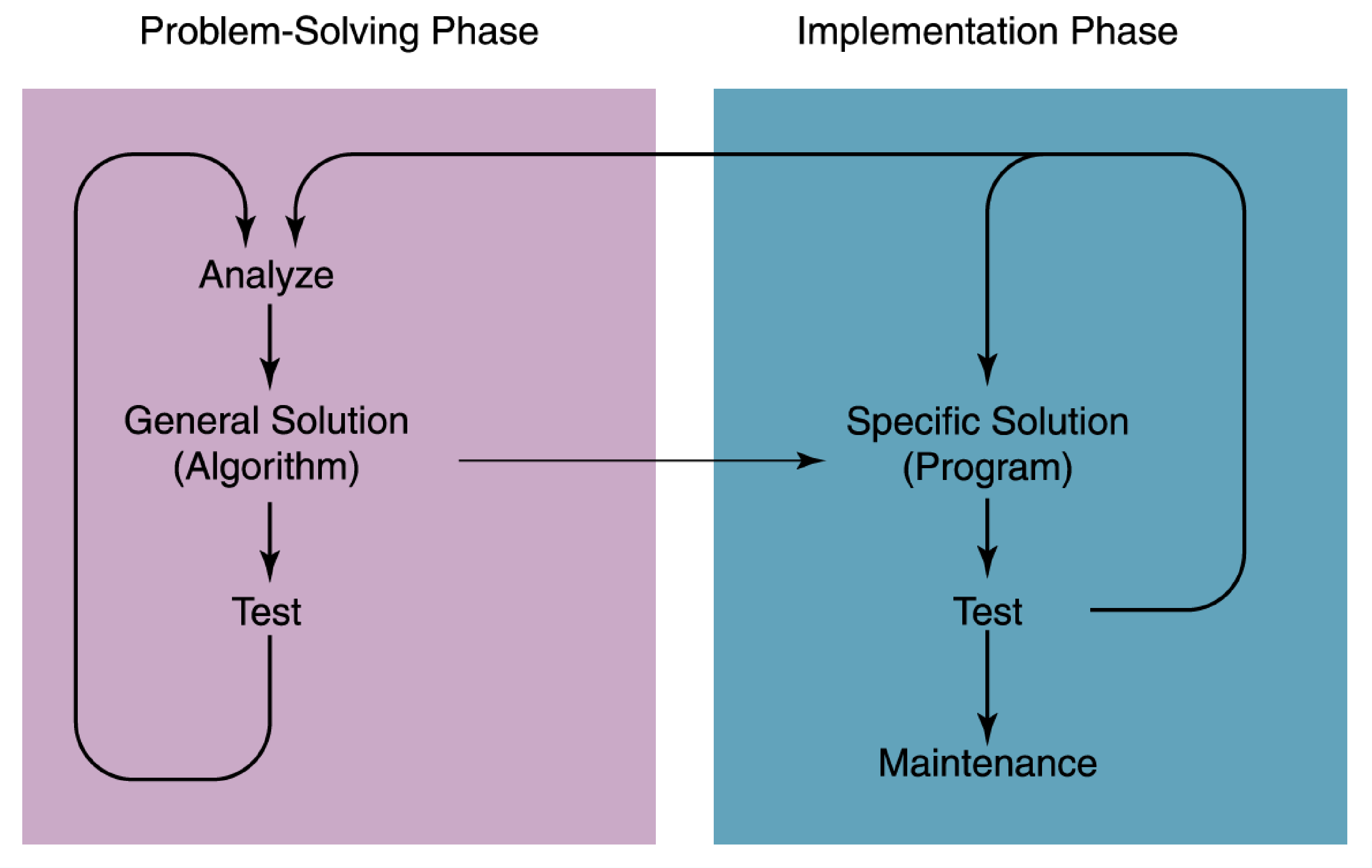
- **Algorithm** A set of instructions for solving a problem or subproblem in a finite amount of time using a finite amount of data
- The instructions must be unambiguous

Computer Problem-Solving

Algorithm Development Phase		
	Analyze	Understand (define) the problem.
	Propose algorithm	Develop a logical sequence of steps to be used to solve the problem.
	Test algorithm	Follow the steps as outlined to see if the solution truly solves the problem.
Implementation Phase		
	Code	Translate the algorithm (the general solution) into a programming language.
	Test	Have the computer follow the instructions. Check the results and make corrections until the answers are correct.
Maintenance Phase		
	Use	Use the program.
	Maintain	Modify the program to meet chaining requirements or to correct any errors.

Figure 6.2 **The computer problem-solving process**

The Interactions Between Problem-Solving Phases



Pseudocode

- Uses a mixture of English and formatting to make the steps in the solution explicit

```
While (the quotient is not zero)
    Divide the decimal number by the new base
    Make the remainder the next digit to the left in the answer
    Replace the original decimal number with the quotient
```

Following an Algorithm

Never-Fail Blender Hollandaise

1 cup butter
4 egg yolks
1/4 teaspoon salt
1/4 teaspoon sugar
1/4 teaspoon Tabasco
1/4 teaspoon dry mustard
2 tablespoons lemon juice

Heat butter until bubbling. Combine all other ingredients in blender. With blender turned on, pour butter into yolk mixture in slow stream until all is added. Turn blender off. Keeps well in refrigerator for several days. When reheating, heat over hot, not boiling, water in double boiler. Makes about 1-1/4 cups sauce.

Following an Algorithm

- Preparing a Hollandaise sauce

```
Put butter in a pot
Turn on burner
Put pot on the burner
While (NOT bubbling)
    Leave pot on the burner
Put other ingredients in the blender
Turn on blender
While (more butter)
    Pour butter into blender in slow stream
Turn off blender
```

Developing an Algorithm

- The plan must be suitable in a suitable form
- Two methodologies that currently used
  - Top-down design
  - Object-oriented design

Top-Down Design

- Breaking the problem into a set of subproblems called **modules**
- Creating a hierarchical structure of problems and subproblems (modules)

Top-Down Design

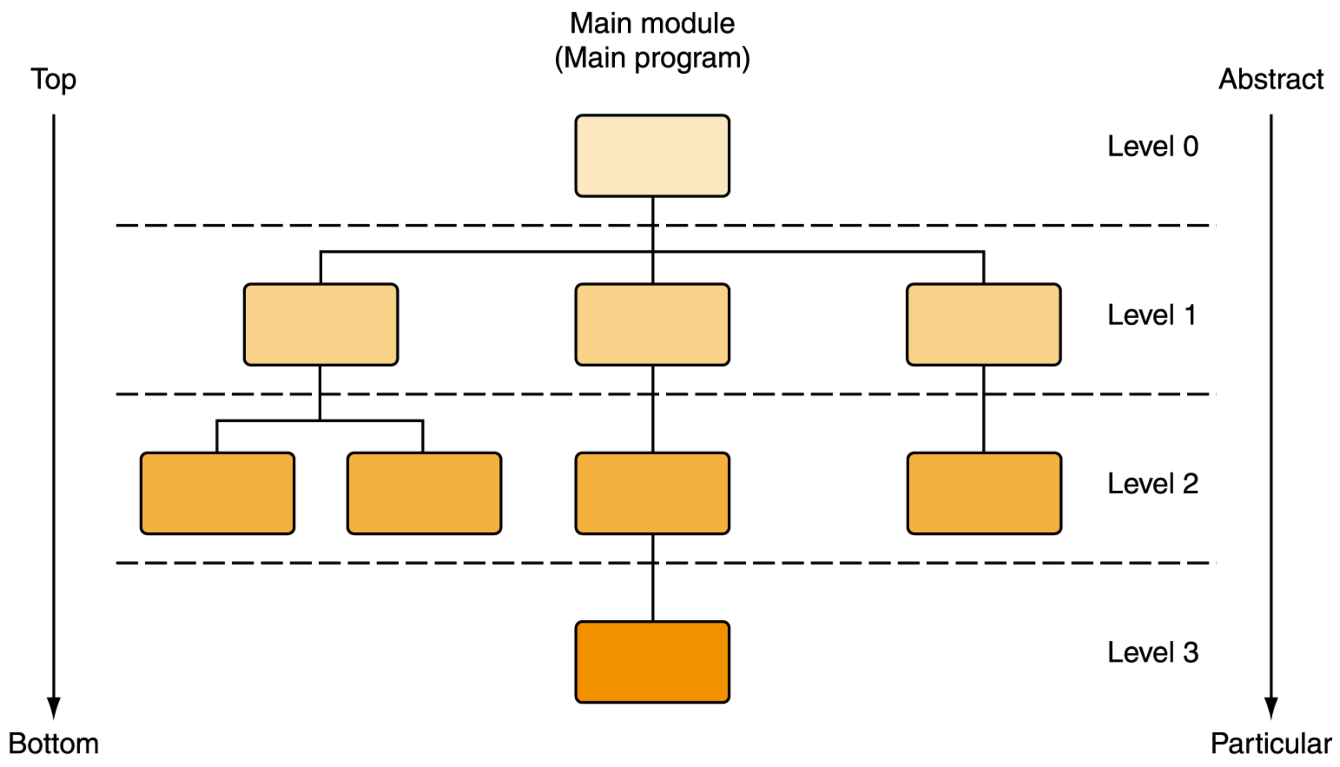
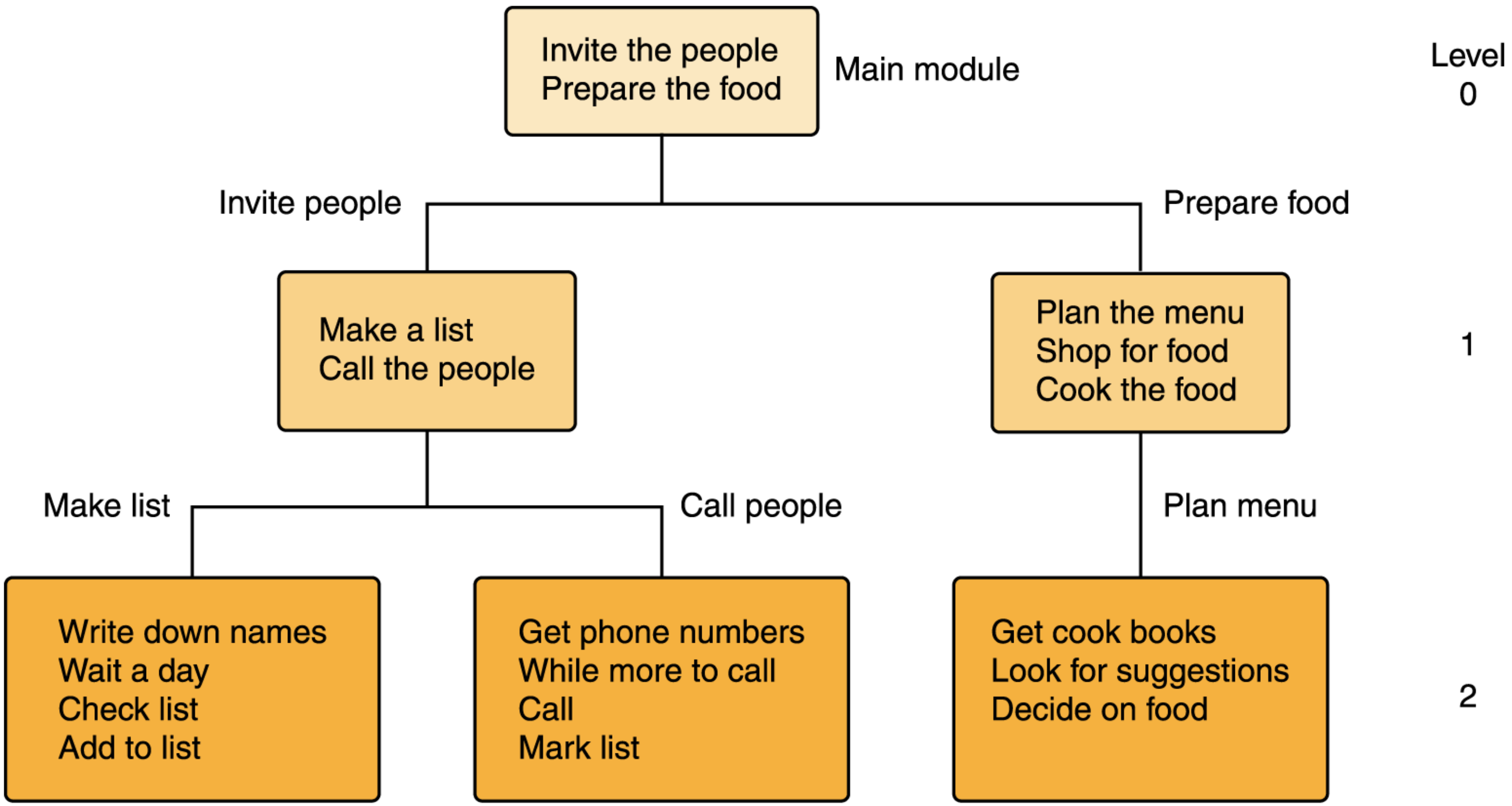


Figure 6.5
An example of top-down design

- This process continues for as many levels as it takes to expand every task to the smallest details
- A step that needs to be expanded is an abstract step

A General Example

- Planning a large party



A Computer Example

- Problem
  - Create an address list that includes each person's name, address, telephone number, and e-mail address
  - This list should then be printed in alphabetical order
  - The names to be included in the list are on scraps of paper and business cards

A Computer Example

Main — Level 0

Enter names into list

Fill in missing data

Put list into alphabetical order

Print the list

Enter names into list — Level 1

Prompt for and enter names — includes other data as well

Insert names into list

A Computer Example

Prompt for and enter names — **Level 2**

Write "To any of the prompts below, if the information is not known, just
press return."
While (more names)
Write "Enter the last name, a comma, a blank, and the first name;
press return."
Read lastFirst
Write "Enter street number and name; press return."
Read street
Write "Enter city, a comma, a blank, and state; press return."
Read cityState
Write "Enter area code and 7-digit number; press return."
Read telephone
Write "Enter e-mail; press return."
Read eMail

A Computer Example

Fill in missing data **Level 1**

```
Write "To any of the prompts below, if the information is still not known,
      just press return."
Get a name from the list
While there are more names
   Get a lastFirst
   Write lastFirst
   If (street is missing)
      Write "Enter street number and name; press return."
      Read street
   If (telephone is missing)
      Write "Enter area code and 7-digit number; press return."
      Read telephone
   If (eMail is missing)
      Write "Enter e-mail; press return."
   Get a name from the list
```

A Computer Example

Put list in alphabetical order **Level 3**

Sort list on lastFirst field

Print the list

Write "The list of names, addresses, telephone numbers, and e-mail addresses follows:"

Get a name from the list

While (there are more names)

    Write lastFirst

    Write street

    Write cityState

    Write e-Mail

    Write a blank line

    Get a name from the list

Testing the Algorithm

- The process itself must be tested
- Testing at the algorithm development phase involves looking at each level of the top-down design

Testing the Algorithm

- **Desk checking** Working through a design at a desk with a pencil and paper
- **Walk-through** Manual simulation of the design by the team members, taking sample data values and simulating the design using the sample data
- **Inspection** One person (not the designer) reads the design (handed out in advance) line by line while the others point out errors

Assignment One

- **Let Me Know If I Can Publish On Web Site**
- **There No Obligation**

Homework

- **Read Chapter Five, Secs 5.2-5.3**
- **Read Chapter Six, Secs 6.1-6.2**

Mid-Term

- Take Home Exam – Non-Trivial (think!)
- Cover Chapters 1-5 & 16 & Anything Covered In Class
- Given Out: Oct 16
- Due Back: Oct 23
- No Lateness!!!

Have A Good Weekend!