CMPS1134
Fundamentals of Computing

Programming Languages 2

Computer Science: An Overview
Eleventh Edition

J. Glenn Brookshear

Chapter 6

Chapter 6: Programming Languages

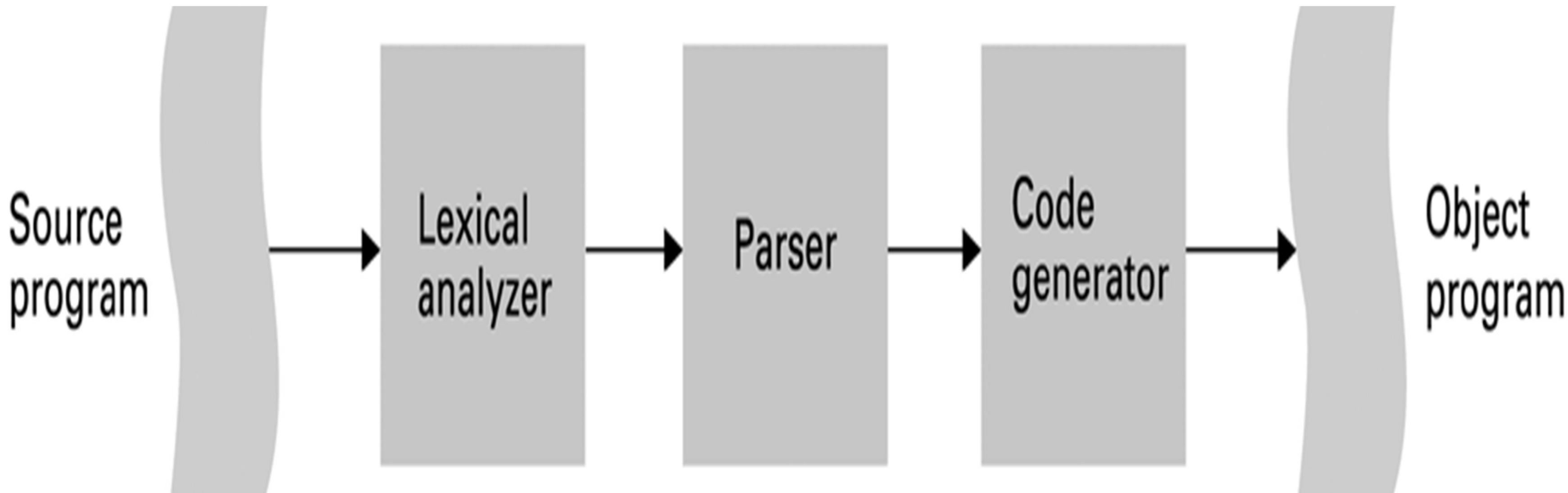
- Language Implementation
- Object Oriented Programming
- Programming Concurrent Activities
- Declarative Programming

Language Implementation

The process of converting a program in a high-level language into a machine-executable form.

The translation process (Fig 6.13)

- Process of converting a program from one language to another is called **translation**.
- Program in its original for is the **source program**.
- The translated version is the **object program**.
- The translation process comprises three units: the **lexical analyser**, **parser**, and **code generator**.

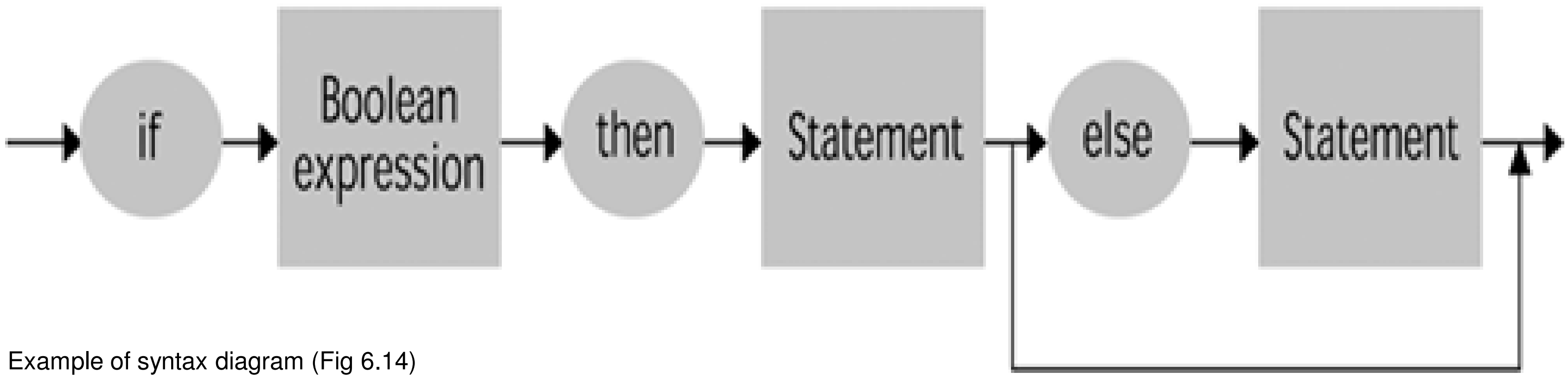


Lexical Analyzer

- Process of recognizing which strings of symbols from the source program represent a single entity, or **token**
  1. Reads the source program symbol by symbol
  2. Identifies which groups of symbols represent tokens
  3. Encodes the token with its classification and hands them to the Parser
- **Fixed-format** vs. **free-format** languages.
- **Key words** are used to identify the structure (syntax) of a program (e.g. **if** in Fortran – which has no reserved words)
- **Reserved words** cannot be used for other purposes within the program (e.g. **if** in C)

Parser: Syntax Analysis

- The parsing process is based on a set of rules that define the syntax of a programming language called a **grammar**
- **Syntax diagrams** is one way of expressing a grammar
  - **Nonterminals** are terms that require further description (rectangles)
  - **Terminals** are terms that do not require further description (ovals)



Example of syntax diagram (Fig 6.14)

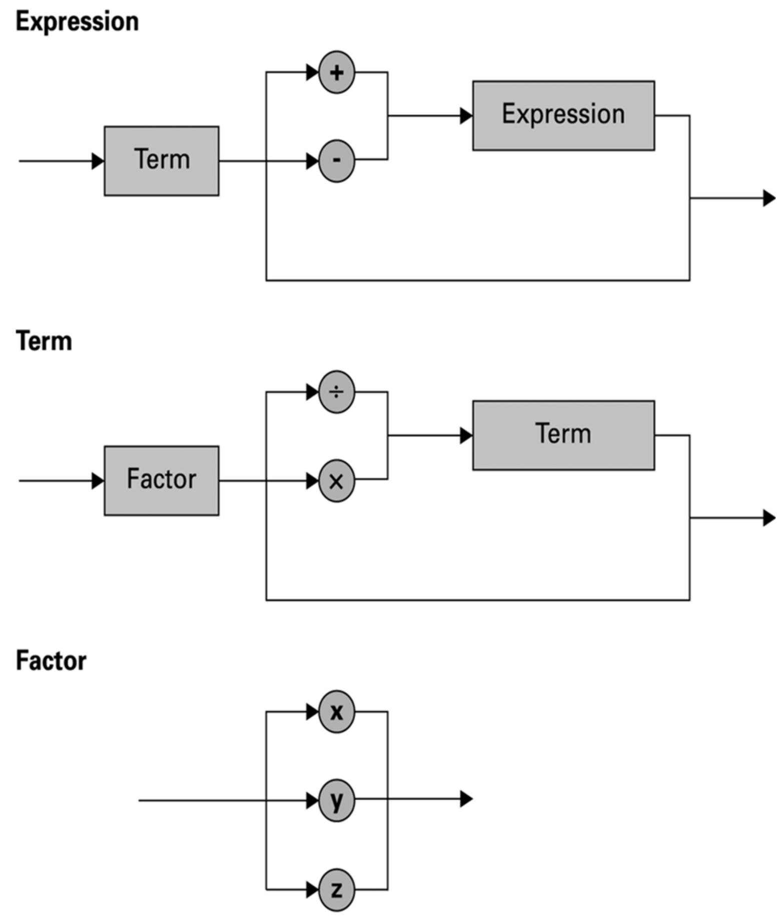
Syntax diagrams describing the structure of a simple algebraic expression (Fig 6.15)

Syntax Diagrams for the structure called Expression:

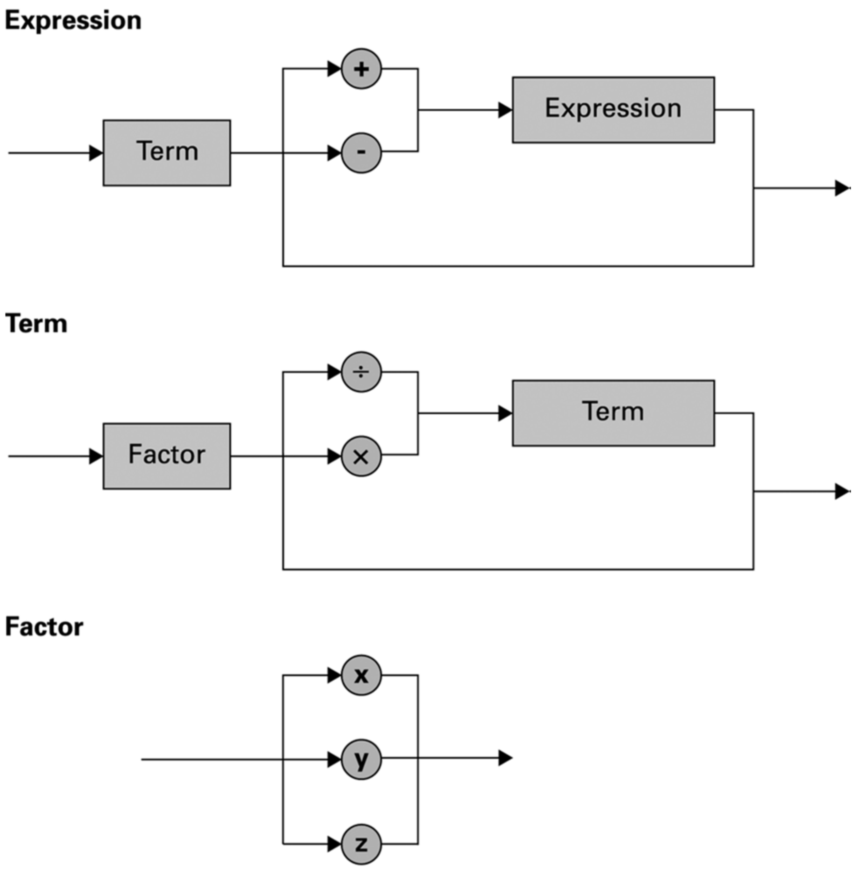
Expression = Term or
Term +|- Expression

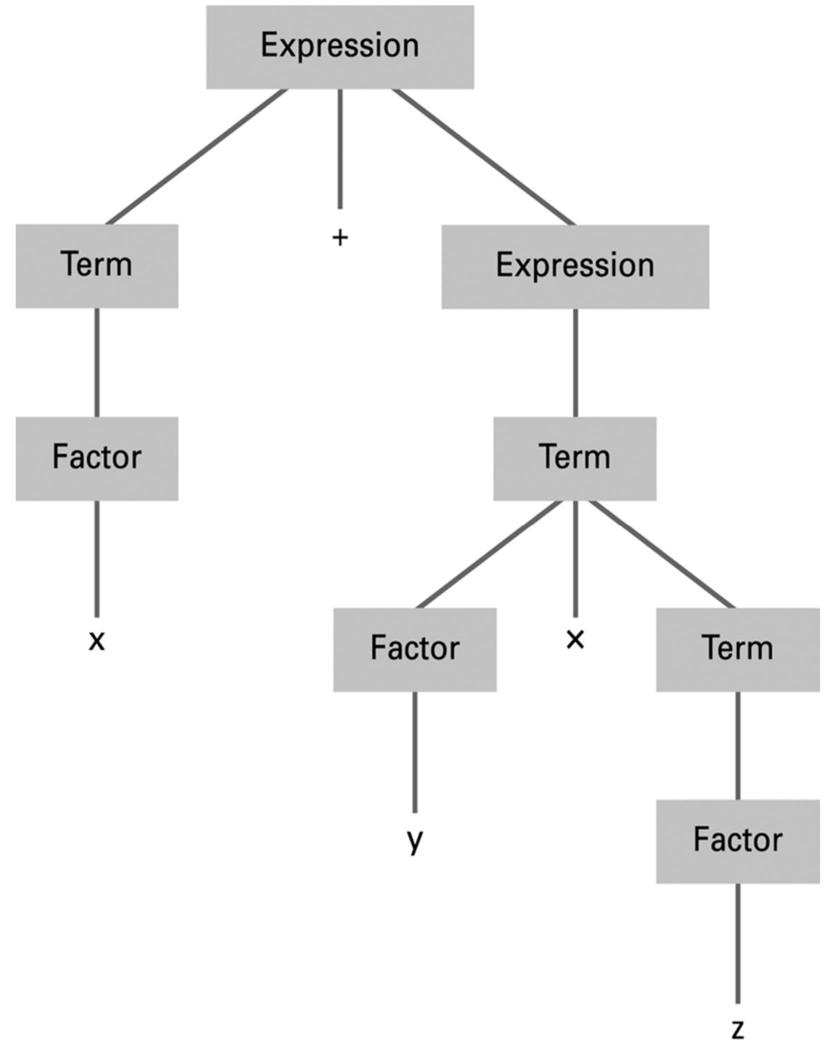
Term = Factor or
Factor ÷|× Term

Factor = x|y|z



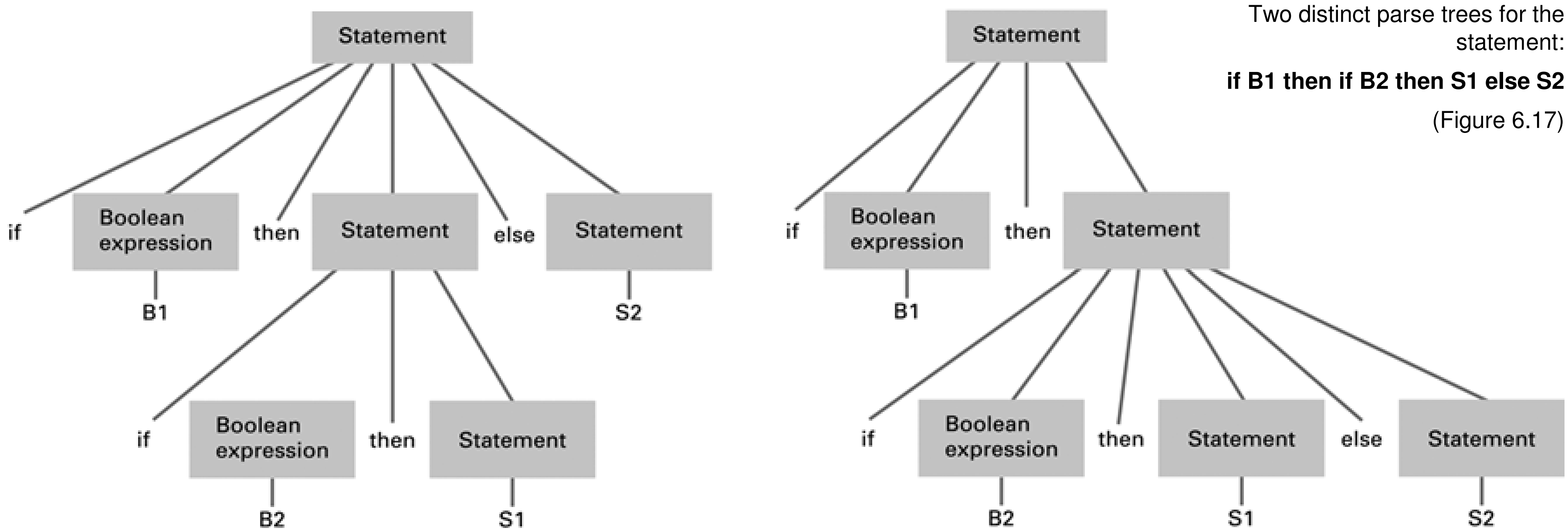
The parse tree for the string x + y x z based on the syntax in Figure 6.15 (Fig 6.16)





Importance of Parse Trees

- Process of parsing is essentially that of constructing a **parse tree** for the source program
- Grammar must not allow two distinct parse trees for one string
- A grammar that allows two distinct trees for a string is an **ambiguous grammar**



Two distinct parse trees for the statement:

if B1 then if B2 then S1 else S2

(Figure 6.17)

Code Generator

- Process of constructing the machine-language instructions to implement the statements recognized by the parser
- Involves **code optimization** that produces efficient machine-language versions of the program

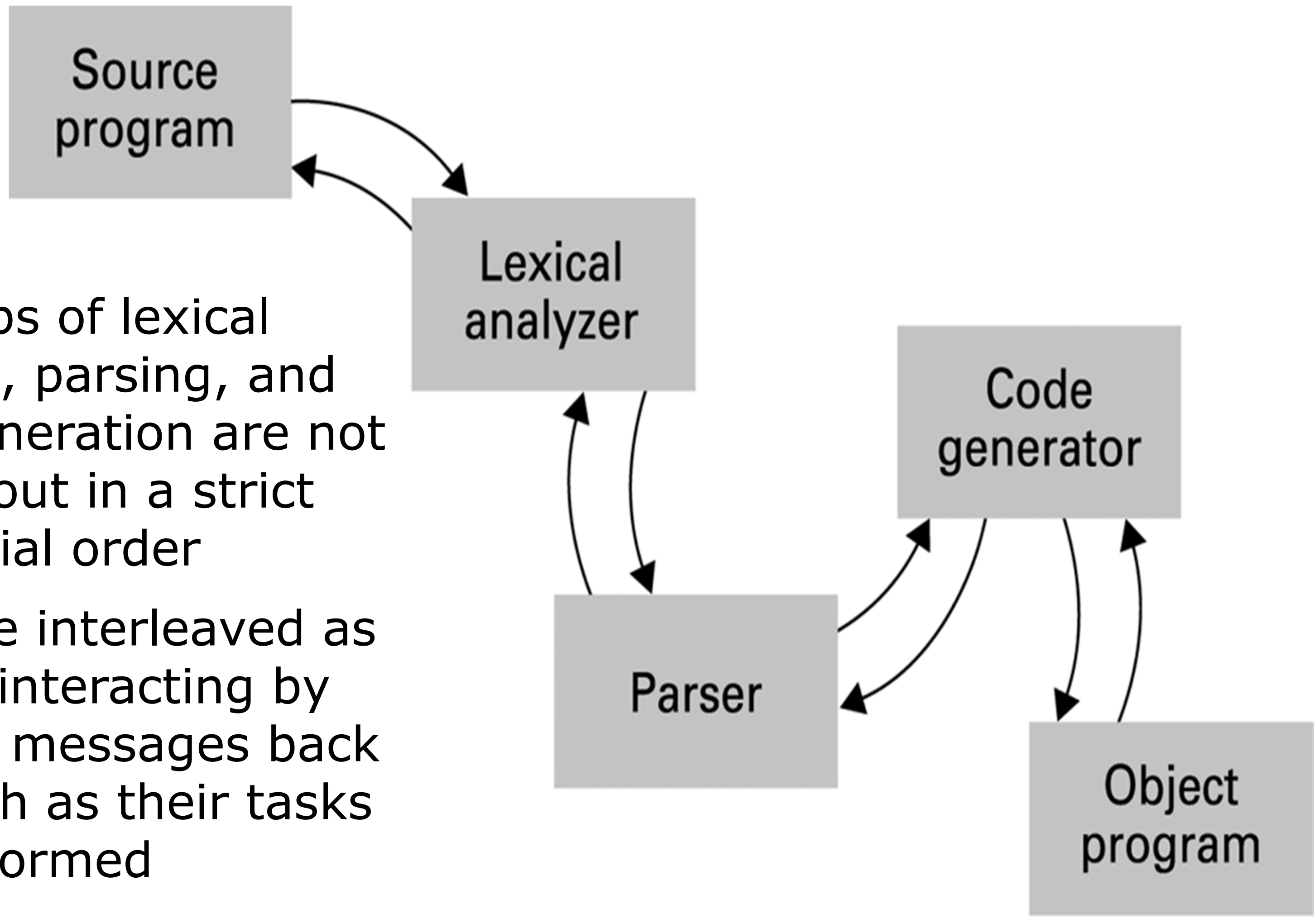
  Example:

  $\mathbf{x} \leftarrow y + \mathbf{z};$

  $w \leftarrow \mathbf{x} + \mathbf{z};$

  - Translation into individual statements requires that data is transferred from main memory to the CPU
  - Efficiency can be gained by recognizing that after the first statement is executed, **x** and **z** <u>are in the CPU general purpose registers</u> and do not need to be loaded for the second statement

An object-oriented approach to the translation process (Fig 6.18)



- The steps of lexical analysis, parsing, and code generation are not carried out in a strict sequential order
- They are interleaved as objects interacting by sending messages back and forth as their tasks are performed

Object-oriented Programming

Entails the development of active program units called objects.

- **Object:** Active program unit containing both data and procedures
- **Class:** A template from which objects are constructed
- An object is called an **instance** of the class
- Components of an object:
  - **Instance Variable:** Variable within an object
    - Holds information within the object
  - **Method:** Procedure within an object
    - Describes the actions that the object can perform
  - **Constructor:** Special method used to initialize a new object when it is first constructed

Structure of a class describing a laser weapon in a computer game (Fig 6.19)

```
class LaserClass

{  int RemainingPower = 100;

   void turnRight (   )
   { ... }

   void turnLeft (   )
   { ... }

   void fire (   )
   { ... }

}
```

Instance variable

Description of the data that will reside inside of each object of this “type.”

Methods describing how an object of this “type” should respond to various messages

Object instance of Class

C++ : LaserClass Laser1, Laser 2;

Java / C# : LaserClass Laser1 = new LaserClass();
LaserClass Laser2 = new LaserClass();

A class with a constructor (Fig 6.21)

```
class  LaserClass
{ int RemainingPower;

{ LaserClass (InitialPower)
  { RemainingPower = InitialPower;
  }

  void turnRight (   )
  { ... }

  void turnLeft (   )
  { ... }

  void fire (   )
  { ... }

}
```

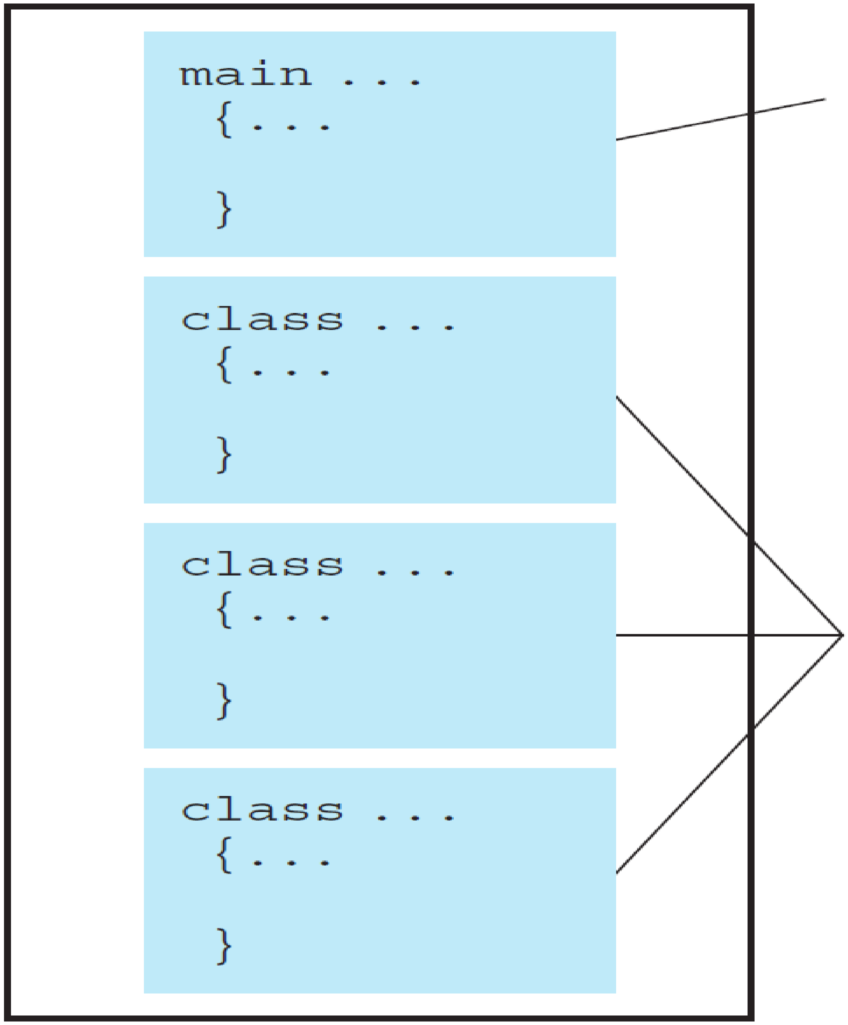
Constructor assigns a value to Remaining Power when an object is created.

C++ : LaserClass Laser1 (50), Laser 2 (100);

Java / C# : LaserClass Laser1 = new LaserClass(50);
LaserClass Laser2 = new LaserClass(100);

Structure of a typical object-oriented program (Fig 6.20)

Program



Procedural unit (often called main) that directs the construction of the objects and makes appropriate calls to their methods

C++:
```
LaserClass Laser1(50), Laser 2 (100);
   :
Laser1.fire();
   :
Laser2.turnleft;
   :
```

Class descriptions

Additional Object-oriented Concepts

- **Inheritance:** Allows new classes to be defined in terms of previously defined classes

```
RechargeableLaser Laser3, Laser 4;

Class RechargeableLaser extends LaserClass
{       :
        :
}
```

- **Polymorphism:** Allows method calls to be interpreted by the object that receives the call
  - e.g. Laser1.fire() vs. Laser3.fire()

Object Integrity

- **Encapsulation:** A way of restricting access to the internal components of an object
  - **Private**

    Features of an object that can only be accessed by the object
  - **Public**

    Features of an object that are accessible from outside the object

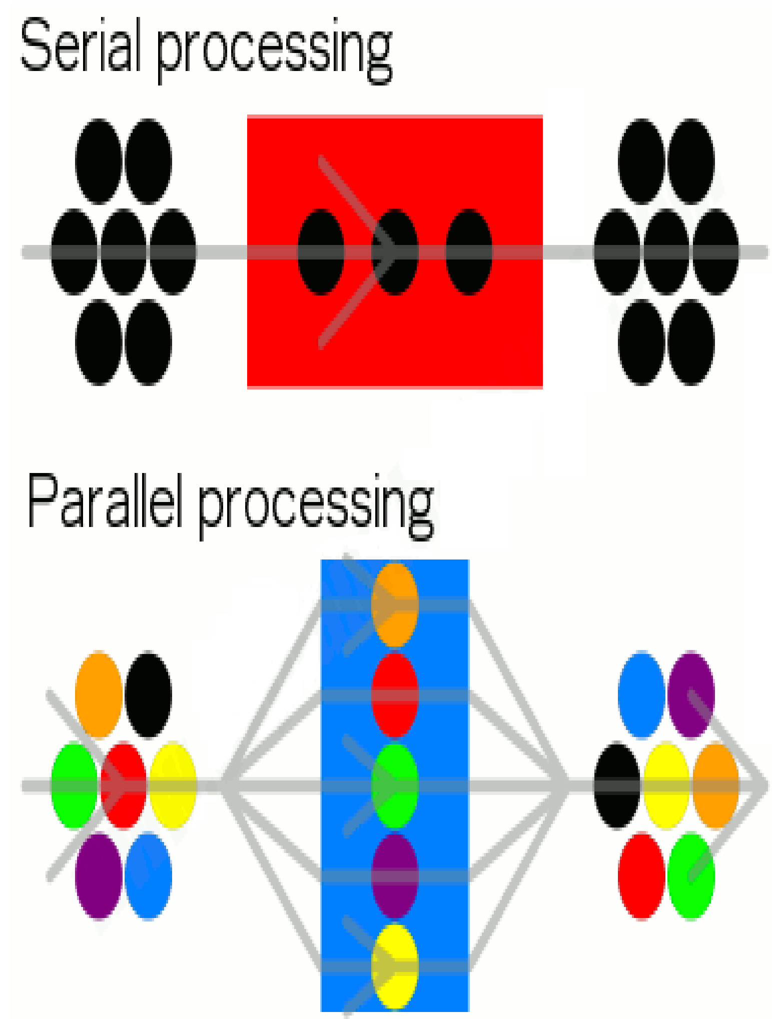
LaserClass definition using encapsulation as it would appear in Java or C# (Figure 6.22)

Components in the class are designated public or private depending on whether they should be accessible from other program units.

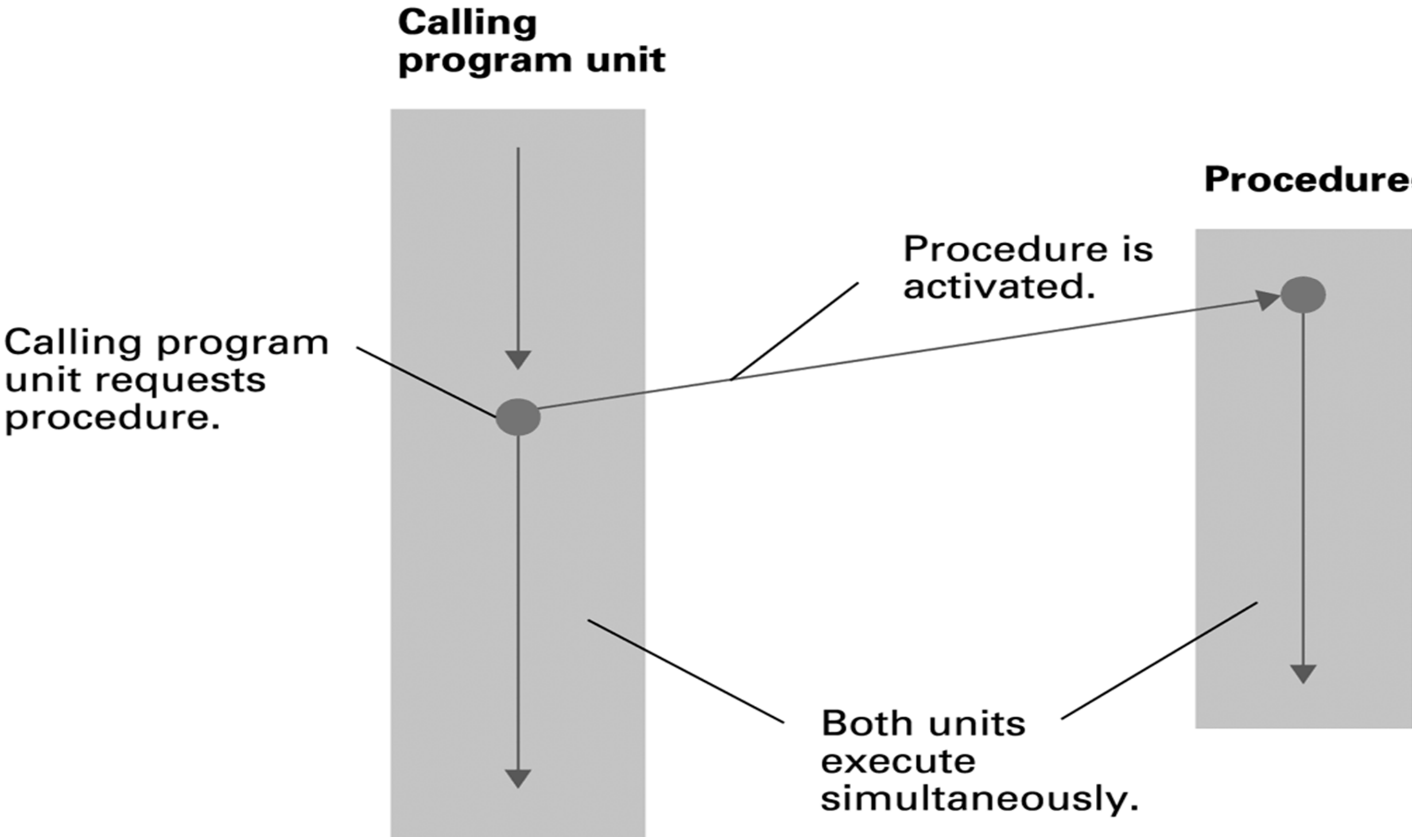
```
class  LaserClass
{private int RemainingPower;
 public LaserClass (InitialPower)
 {RemainingPower = InitialPower;
 }
 public void turnRight (  )
 { ... }
 public void turnLeft (  )
 { ... }
 public void fire (  )
 { ... }
}
```


Programming Concurrent Activities

- **Parallel** or **concurrent processing:** simultaneous execution of multiple processes
  - True concurrent processing requires multiple CPUs
  - Can be simulated using time-sharing with a single CPU

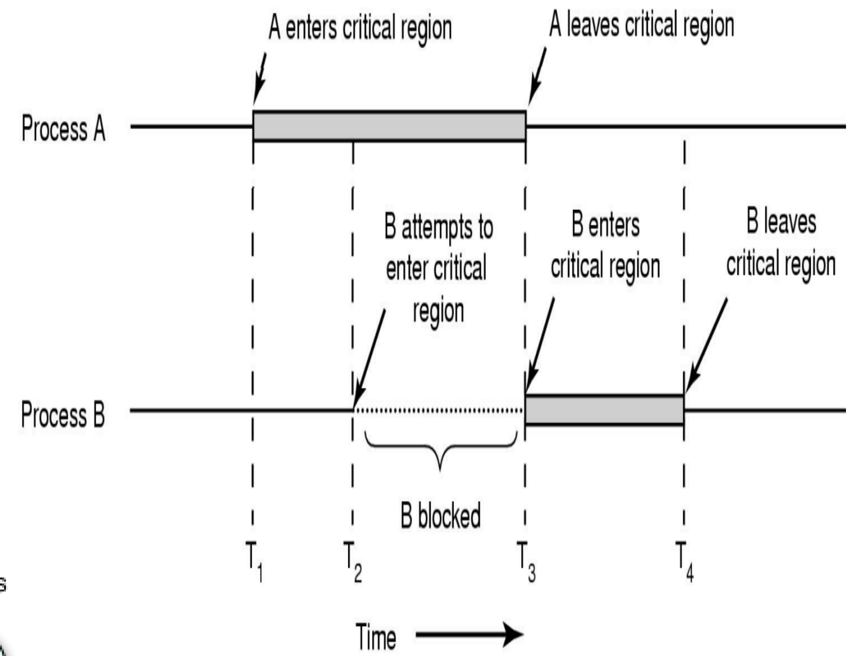


Spawning threads (Fig 6.23)

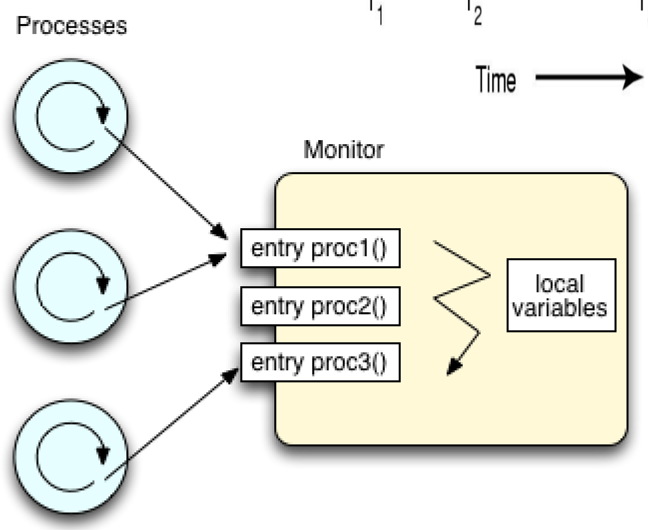


Controlling Access to Data

- **Mutual Exclusion:** A method for ensuring that data can be accessed by only one process at a time

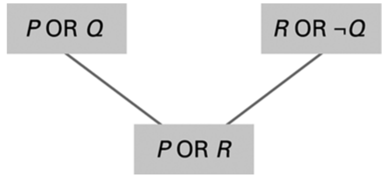


- **Monitor:** A data item augmented with the ability to control access to itself



Declarative Programming

- **Resolution:** Combining two or more statements to produce a new statement (that is a logical consequence of the originals).
  - Example: (P OR Q) AND (R OR ¬Q) resolves to (P OR R)
  - **Resolvent:** A new statement deduced by resolution
  - **Clause form:** A statement whose elementary components are connected by the Boolean operation OR



- A collection of statements is **inconsistent** if it is impossible for all statements to be true at the same time.
- **Unification:** Assigning a value to a variable so that resolution can be performed and two statements become "compatible."

Prolog

- **Fact:** A Prolog statement establishing a fact
  - Consists of a single predicate
  - Form: *predicateName*(*arguments*).
    - Example: `parent(bill, mary).`

- **Rule:** A Prolog statement establishing a general rule
  - Form: *conclusion* :- *premise.*
    - :- means "if"
  - Example: `wise(X) :- old(X).`
  - Example: `faster(X,Z) :- faster(X,Y), faster(Y,Z).`

Prolog Resolution Example

Rules:

```
has (X, A) :- votes (X, B).
older (X, C) :- has (X, A).
```

Resolution through unification

~~has (steve, voter-id)~~ :- votes (steve, elections).

older (steve, seventeen) :- ~~has (steve, voter-id).~~

Resolvent: new inferred rule through resolution

```
older (steve, seventeen) :- votes (steve, elections).
```

Prolog Goal through Unification

Facts:

```
faster(turtle, snail).
faster(rabbit, turtle).
faster(rabbit, snail).
```

Goals and responses

`faster (W, snail).` yields `faster (turtle, snail).`

If asked for more it yields:

`faster (rabbit, snail).`

Which animals are faster than the snail?

`faster (rabbit, W).` yields `faster (rabbit, turtle).`

If asked for more it yields:

`faster (rabbit, snail).`

Which animals is the rabbit faster than?

What about:

`faster (V, W).`