SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Unit – II
String Handling - I

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Unit - II

- **Multithreaded Programming:**
- Java Thread Model - The Main Thread - Creating a Thread - Creating Multiple Threads - Thread Priorities - Synchronization.

- **I/O:**
- I/O Basics - Reading Console Input - Writing Console Output.

- **String Handling:**
- String - String Buffer - String Builder

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Strings.
- String Operations.

- StringBuffer.
- StringBuffer Operations.

- StringBuilder
- StringBuilder Operations

Strings

- Java string is a sequence of characters.
- They are objects of type String.
  - String
  - StringBuffer
  - StringBuilder

~~char str[100];~~

- Once a String object is created it cannot be changed.
- Strings are Immutable.

Strings

- To get changeable strings use the class called StringBuffer.

- String and StringBuffer classes are declared final
  - so there cannot be subclasses of these classes.

```
class demo extends String
{
}
```

Creating Strings

- The default constructor creates an empty string.

```
String s = new String();
```

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

- If data array in the above example is modified
  - after the string object str is created,
  - then str remains unchanged.

- Construct a string object by passing another string object.

```
String str2 = new String(str);
```

Creating Strings

- String str="Java";

- String str=new String("Java");

- String a = "Java";
- String b = "Java";
- System.out.println(a == b); // true

- String c = new String("Java");
- String d = new String("Java");
- System.out.println(c == d); // False

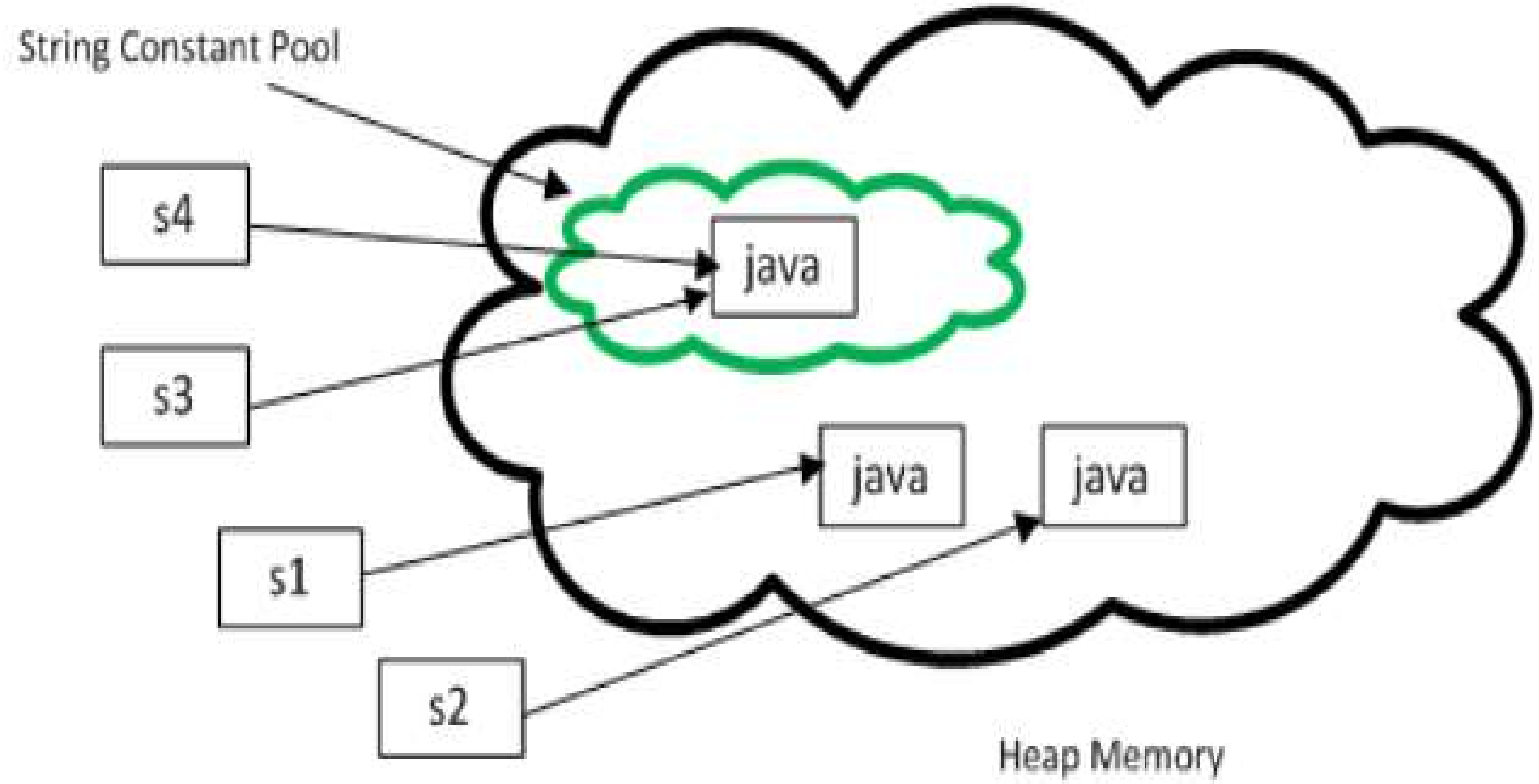
Creating Strings

- These double quoted literal is known as **String literal** and the cache which stored these String instances are known as **String pool**.
- In earlier version of Java,
  - String pool is located in **permgen area** of heap,
  - but in Java 1.7 updates its moved to main heap area.

- Earlier since it was in PermGen space,
  - it was always a risk to create too many String object,
  - because its a very limited space,
  - **default size 64 MB** and used to store class metadata e.g. .class files.
- Creating too many String literals can cause **java.lang.OutOfMemory: permgen space**.

Difference between String literal and String object

- Now because String pool is moved to a much larger memory space, it's much more safe.



String Operations

String Operations

- The **length()** method returns the length of the string.

  Eg: System.out.println("Hello".length()); // prints 5

- The **+ operator** is used to concatenate two or more strings.

  Eg: String myname = "Harry"

  String str = "My name is" + myname+ ".";

  My name is Harry.

String Operations

- For string concatenation
- the Java compiler converts an operand to a String
- whenever the other operand of the + is a String object.

```
String s="six"+3+3;
System.out.println(s);
```

```
E:\slides\Java - 2021\Unit - II>java string
six33
```

```
String s="six"+(3+3);
System.out.println(s);
```

```
E:\slides\Java - 2021\Unit - II>java string
six6
```

Character Extraction

- Characters in a string can be extracted in a number of ways.

- **public char charAt(int index)**
  - Returns the character at the specified index.
  - An index ranges from 0 to length() - 1.
  - The first character of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

```
char ch;
ch = "abc".charAt(1); // ch = "b"
```

Character Extraction

```
ch = "abc".charAt(10);
ch = "abc".charAt(-1);
```

```
E:\slides\Java - 2021\Unit - II>javac string1.java

E:\slides\Java - 2021\Unit - II>java string
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String
index out of range: 10
        at java.lang.String.charAt(Unknown Source)
        at string.main(string1.java:29)

E:\slides\Java - 2021\Unit - II>javac string1.java
E:\slides\Java - 2021\Unit - II>java string
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String
index out of range: -1
        at java.lang.String.charAt(Unknown Source)
        at string.main(string1.java:29)
```

Character Extraction

- **getChars()** - Copies characters from this string into the destination character array.

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

  - srcBegin - index of the first character in the string to copy.
  - srcEnd - index after the last character in the string to copy.
  - dst - the destination array.
  - dstBegin - the start offset in the destination array.

Character Extraction - Example

```
class string
{
public static void main(String args[])
{
String s1="SASTRA Deemed University";

char c[]=new char[10];

System.out.println(s1);
s1.getChars(2,8,c,0);
System.out.println(c);

}
}
```

```
E:\slides\Java - 2021\Unit - II>java string

SASTRA Deemed University
0 1 2345678
STRA D
```

Character Extraction - Example

```
class string
{
public static void main(String args[])
{
String s1="SASTRA Deemed University";

char c[]=new char[10];
System.out.println(s1);
s1.getChars(2,8,c,11);
System.out.println(c);

}
}
```

```
E:\slides\Java - 2021\Unit - II>javac string1.java

E:\slides\Java - 2021\Unit - II>java string
SASTRA Deemed University
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at java.lang.System.arraycopy(Native Method)
        at java.lang.String.getChars(Unknown Source)
        at string.main(string1.java:23)
```

String Comparison

- **equals() -** Compares the invoking string to the specified object.
- The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as the invoking object.

```
public boolean equals(Object anObject)
```

- **equalsIgnoreCase()-** Compares this String to another String, ignoring case considerations.
- Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

```
public boolean equalsIgnoreCase(String anotherString)
```

String Comparison

- **startsWith()** – Tests if this string starts with the specified prefix.

```
public boolean startsWith(String prefix)
"Figure".startsWith("Fig"); // true
```

- **endsWith() -** Tests if this string ends with the specified suffix.

```
public boolean endsWith(String suffix)
"Figure".endsWith("re"); // true
```

String Comparison

- **startsWith()** -Tests if this string starts with the specified prefix beginning at a specified index.

```
public boolean startsWith(String prefix,
                          int toffset)
```

prefix - the prefix.

toffset - where to begin looking in the string.

```
"figure".startsWith("gure", 2); // true
```

String Comparison

- **compareTo()** - Compares two strings lexicographically.
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically follows the argument string.
  - The result is zero if the strings are equal.
  - compareTo returns 0 exactly when the equals(Object) method would return true.

```
public int compareTo(String anotherString)
public int compareToIgnoreCase(String str)
```

String Comparison - Example

```
class string
{
public static void main(String args[])
{
String s1=new String("Abisheik");
String s2=new String("Bharath");
String s3=new String(s1);

int x=s1.compareTo(s2);
int y=s2.compareTo(s1);
int z=s1.compareTo(s3);

System.out.println(x);
System.out.println(y);
System.out.println(z);
}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
-1
1
0
```

Searching Strings

- **indexOf –** Searches for the first occurrence of a character or substring. Returns -1 if the character does not occur.

`public int indexOf(int ch)-` Returns the index within this string of the first occurrence of the specified character.

`public int indexOf(String str)` - Returns the index within this string of the first occurrence of the specified substring.

```
String str = "How was your day today?";
      str.indexOf('w'); //2
      str.indexOf("was"); //4
```

Searching Strings

`public int indexOf(int ch, int fromIndex)`

- Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

`public int indexOf(String str, int fromIndex)`

- Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

```
String str = "How was your day today?";
      str.indexOf('w',3); //4
      str.indexOf("day",17); //19
```

Searching Strings

lastIndexOf()

- Searches for the last occurrence of a character or substring.
- The methods are similar to indexOf().

```
String str = "How was your day today?";
str.lastIndexOf('y'); //21
System.out.println(s1.lastIndexOf("day",21));//19
System.out.println(s1.lastIndexOf("day",18));//13
```

Modifying a String

substring

- Returns a new string that is a substring of this string.
- The substring begins with the character at the specified index and extends to the end of this string.

- `public String substring(int beginIndex)`

  Eg: `"unhappy".substring(2) returns "happy"`

- `public String substring(int beginIndex,int endIndex)`

  Eg: `"smiles".substring(1, 5) returns "mile"`

Modifying a String

Concat

- Concatenates the specified string to the end of this string.

- If the length of the argument string is 0, then this String object is returned.

- Otherwise, a new String object is created, containing the invoking string with the contents of the str appended to it.

```
public String concat(String str)
"to".concat("get").concat("her")
        returns "together"
```

Modifying a String

- Replace
- Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

```
public String replace(char oldChar, char newChar)
```

```
"mesquite in your cellar".replace('e', 'o')
```

```
returns
```

```
"mosquito in your collar"
```

Modifying a String

- **trim() -** Returns a copy of the string, with leading and trailing whitespace omitted.

```
public String trim()
```

```
String s = "  Hi Mom!  ".trim();
S = "Hi Mom!"
```

Modifying a String

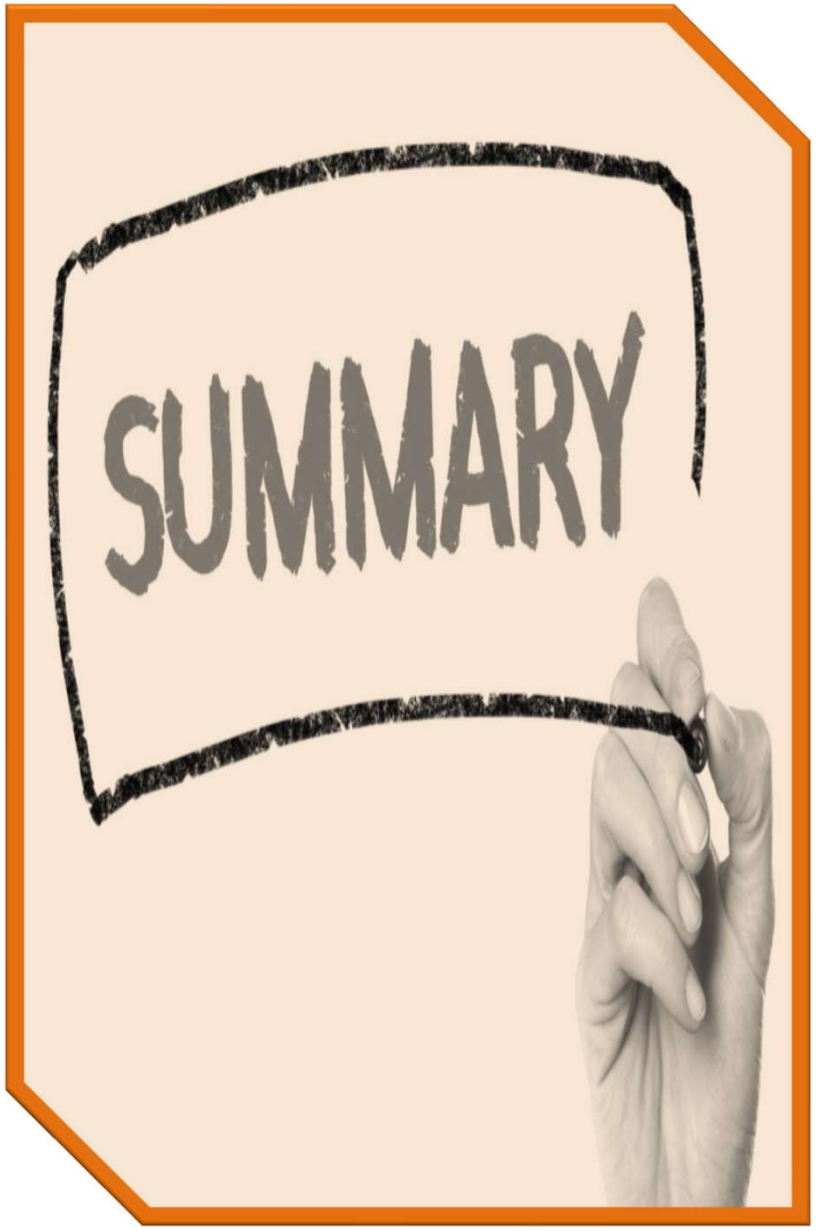
- **toLowerCase():**
  - Converts all of the characters in a String to lower case.
- **toUpperCase():**
  - Converts all of the characters in this String to upper case.

```
public String toLowerCase()
public String toUpperCase()
```

```
Eg: "HELLO THERE".toLowerCase();
    "hello there"
```

End

- String Definition
- String Class Constructors
- String Operations
  - Pattern Matching
  - Searching
  - Modification

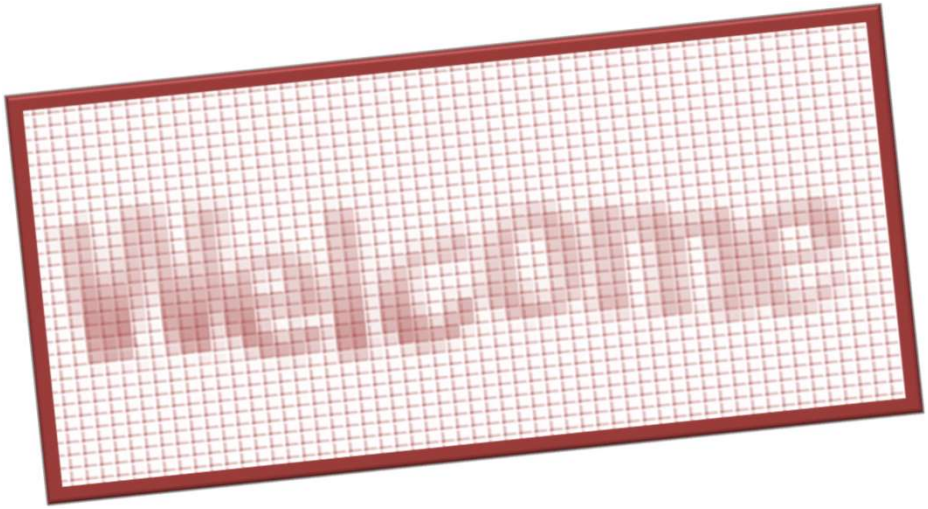


- String Buffer
- String Builder

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

String Handling - II



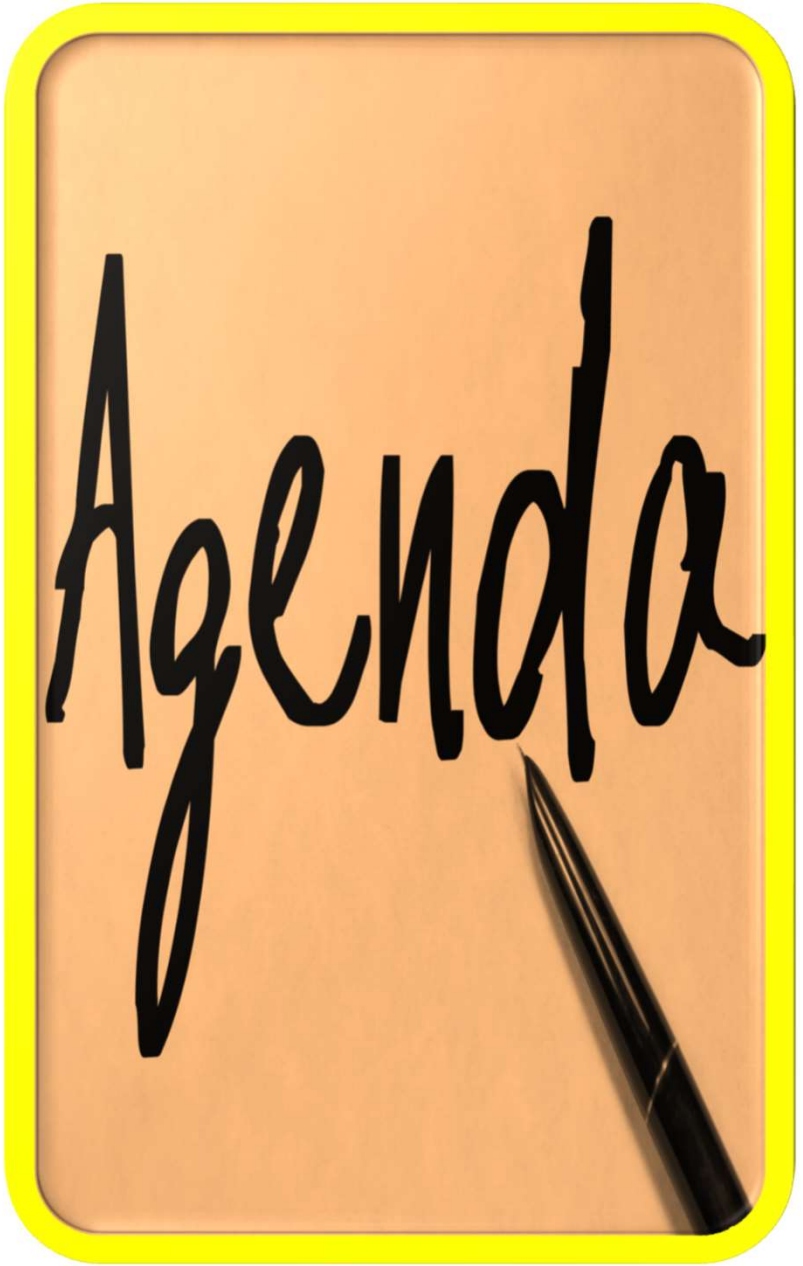
G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- StringBuffer.
- StringBuffer Operations.

- StringBuilder
- StringBuilder Operations

- String Tokenizer



Review Question - 1

```
class string
{
public static void main(String args[])
{
String s1=new String();
System.out.println(s1);
}
}
```

String Constructors - continued

```
String(byte chrs[ ])
String(byte chrs[ ], int startIndex, int numChars)

String(StringBuffer strBufObj)

String(StringBuilder strBuildObj)
```

```
class string
{
public static void main(String args[])
{
byte b[]={65,66,67};
String s=new String(b);
System.out.println(s);
}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
ABC
```

region Matches()

- **regionMatches( ) method**
  - **compares a specific region inside a string with** another specific region in another string.

```
boolean regionMatches(int startIndex, String str2,
                      int str2StartIndex, int numChars)
```

```
boolean regionMatches(boolean ignoreCase,
                      int startIndex, String str2,
                      int str2StartIndex, int numChars)
```

region Matches()

```
class string
{
public static void main(String args[])
{

boolean b= "SASTRA".regionMatches(1,"SASSAS",4,2);
System.out.println(b);

boolean c= "SASTRA".regionMatches(1,"SASTRA",4,2);
System.out.println(c);
}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
true
false
```

Data Conversion Using valueOf( )

- The **valueOf() method**
  - converts data from its internal format into a humanreadable form.
- It is a static method that is overloaded within **String for all of** Java's built-in types
- **valueOf( ) is also overloaded for type Object**
  - static String valueOf(double num)
  - static String valueOf(long num)
  - static String valueOf(Object ob)
  - static String valueOf(char chars[ ])
- **valueOf( ) can be called when a string representation** of some other type of data is needed

Data Conversion Using valueOf()

```
class string
{
public static void main(String args[])
{
int x=10;
double d=12.34;

String s=new String("10");

System.out.println(String.valueOf(x));
System.out.println(String.valueOf(d));

System.out.println(s.equals(String.valueOf(x)));

}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
10
12.34
true
```

Joining Strings

- JDK 8 added a new method to **String called join().**
- **It is used to concatenate** two or more strings, separating each string with a delimiter, such as a space or a comma.
- static String join(CharSequence delim, CharSequence . . . strs)

```
class string
{
public static void main(String args[])
{
String result=String.join("  ","SASTRA","Deemed","University");
System.out.println(result);
}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
SASTRA  Deemed  University
```

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

StringBuffer

StringBuffer

- A StringBuffer is like a String, **but can be modified.**
- The **length and content of the StringBuffer sequence** can be changed through certain method calls.
- **StringBuffer may have characters and substrings** inserted in the middle or appended to the end.
- StringBuffer defines four constructors:
  - **StringBuffer()**
  - **StringBuffer(int size)**
  - **StringBuffer(String str)**
  - **StringBuffer(CharSequence chars)**

StringBuffer Operations - append

- The principal operations on a StringBuffer are the append and insert methods, which are overloaded so as to accept data of any type.

Here are few append methods:

- `StringBuffer append(String str)`
- `StringBuffer append(int num)`

- The append method always adds these characters at the end of the buffer.

StringBuffer Operations - insert

- The insert method adds the characters at a specified point.

Here are few insert methods:

- `StringBuffer insert(int index, String str)`
- `StringBuffer insert(int index, char ch)`

- Index specifies at which point the string will be inserted into the invoking StringBuffer object.

StringBuffer – Example 1

```
class string
{
public static void main(String args[])
{
StringBuffer sb=new StringBuffer("SASTRA");
System.out.println(sb);

sb.append("University");
System.out.println(sb);

sb.insert(6,"Deemed");
System.out.println(sb);

}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
SASTRA
SASTRAUniversity
SASTRADeemedUniversity
```

StringBuffer Operations - delete

- **delete() / deleteCharAt()**
- Removes the characters in a substring of this StringBuffer.

- The substring begins at the specified **start and extends to the character at index end - 1** or to the end of the StringBuffer if no such character exists.

- If start is equal to end, no changes are made.

  - **StringBuffer delete(int start, int end)**
  - **StringBuffer deleteCharAt(int loc)**

StringBuffer Operations

- **replace()**

- Replaces the characters in a substring of this StringBuffer with characters in the specified String.

➢ `public StringBuffer replace(int start, int end, String str)`

StringBuffer Operations

- **reverse()**

- The character sequence contained in this string buffer is replaced by the reverse of the sequence.

➢ `public StringBuffer reverse()`

StringBuffer – Example 2

```
class string
{
public static void main(String args[])
{
System.out.println(sb);

sb.deleteCharAt(0);
System.out.println(sb);

sb.delete(0,5);
System.out.println(sb);

System.out.println(sb.reverse());
}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
SASTRADeemedUniversity
ASTRADeemedUniversity
DeemedUniversity
ytisrevinUdemeeD
```

StringBuffer Operations

- **length()**
- Returns the length of this string buffer.

```
public int length()
```

- **capacity()**
- Returns the current capacity of the String buffer.
- The capacity is the amount of storage available for newly inserted characters.

```
public int capacity()
```

- **setLength**()
- Sets the length of the StringBuffer.

```
public void setLength(int newLength)
```

StringBuffer – Example 3

```
StringBuffer sb = new StringBuffer("Hello");
  sb.length(); // 5
  sb.capacity(); // 21 (16 characters room is
                    added if no size is specified)
  sb.charAt(1); // e
  sb.setCharAt(1,'i'); // Hillo
  sb.setLength(2); // Hi
  sb.append("l").append("l"); // Hill
  sb.insert(0, "Big "); // Big Hill
  sb.replace(3, 11, ""); // Big
  sb.reverse(); // gib
```

End

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

String Builder

Introduction

- StringBuilder is identical to StringBuffer
  - except for one important difference
  - it is not synchronized,
  - which means it is not thread safe.

Introduction

- In other words,
  - if multiple threads are accessing a StringBuilder instance at the sametime,
  - its integrity cannot be guaranteed.

- However,
  - for a single-thread program (most commonly),
  - doing away with the overhead of synchronization makes the StringBuilder faster.

StringBuilder Constructors

StringBuilder( )

Creates an empty string builder with a capacity of 16 (16 empty elements).

StringBuilder(int size)

Creates an empty string builder with the specified initial capacity.

StringBuilder (String str)

Creates a string builder whose value is initialized by the specified string, plus an extra 16 empty elements trailing the string.

StringBuilder(CharSequence cs)

Constructs a string builder containing the same characters as the specified CharSequence, plus an extra 16 empty elements trailing the CharSequence.

StringBuilder – Example 1

```
class string
{
public static void main(String args[])
{
StringBuilder sb=new StringBuilder("SASTRA");
System.out.println(sb);

System.out.println(sb.capacity());

sb.setLength(3);
System.out.println(sb);

}
}
```

```
E:\slides\Java - 2021\Unit - II>java string
SASTRA
22
SAS
```

End

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

String Tokenizer class

StringTokenizer class

- The processing of text often consists of parsing a formatted input string.
- Parsing is the division of text into a set of discrete parts, or tokens, which in a certain sequence can convey a semantic meaning.
- The StringTokenizer class provides the first step in this parsing process, often called the lexer (lexical analyzer) or scanner.

StringTokenizer class

- To use StringTokenizer,
  - you specify an input string and
  - a string that contains delimiters.

- Delimiters are characters that separate tokens.

- Each character in the delimiters string is considered a valid delimiter—for example, ",;:" sets the delimiters to a comma, semicolon, and colon.

- The default set of delimiters consists of the whitespace characters: space, tab, newline,and carriage return.

StringTokenizer Constructors

StringTokenizer(String, String)

Constructs a StringTokenizer on the specified String, using the specified delimiter set.

StringTokenizer(String)

Constructs a StringTokenizer on the specified String, using the default delimiter set (which is " \t\n\r").

Methods

a.countTokens()

Returns the next number of tokens in the String using the current deliminter set.

b. hasMoreTokens()

Returns true if more tokens exist.

c. nextToken()

Returns the next token of the String.

Example usage

```
String s = "this is a test";
StringTokenizer st = new StringTokenizer(s);
while (st.hasMoreTokens()) {
println(st.nextToken());
}
```

Prints the following on the console:

this

is

a

test

- Additional constructors in String.
- Other methods – String
- StringBuffer – Constructors / Methods
- StringBuilder – Constructors
- Use of String Tokenizer class

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Files

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Java's basic I/O System

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

- Need for File
- Data Hierarchy
- Streams
- Predefined Streams
- Reading Console Input
- Writing Console Output
- Reading and Writing Files
- File Class

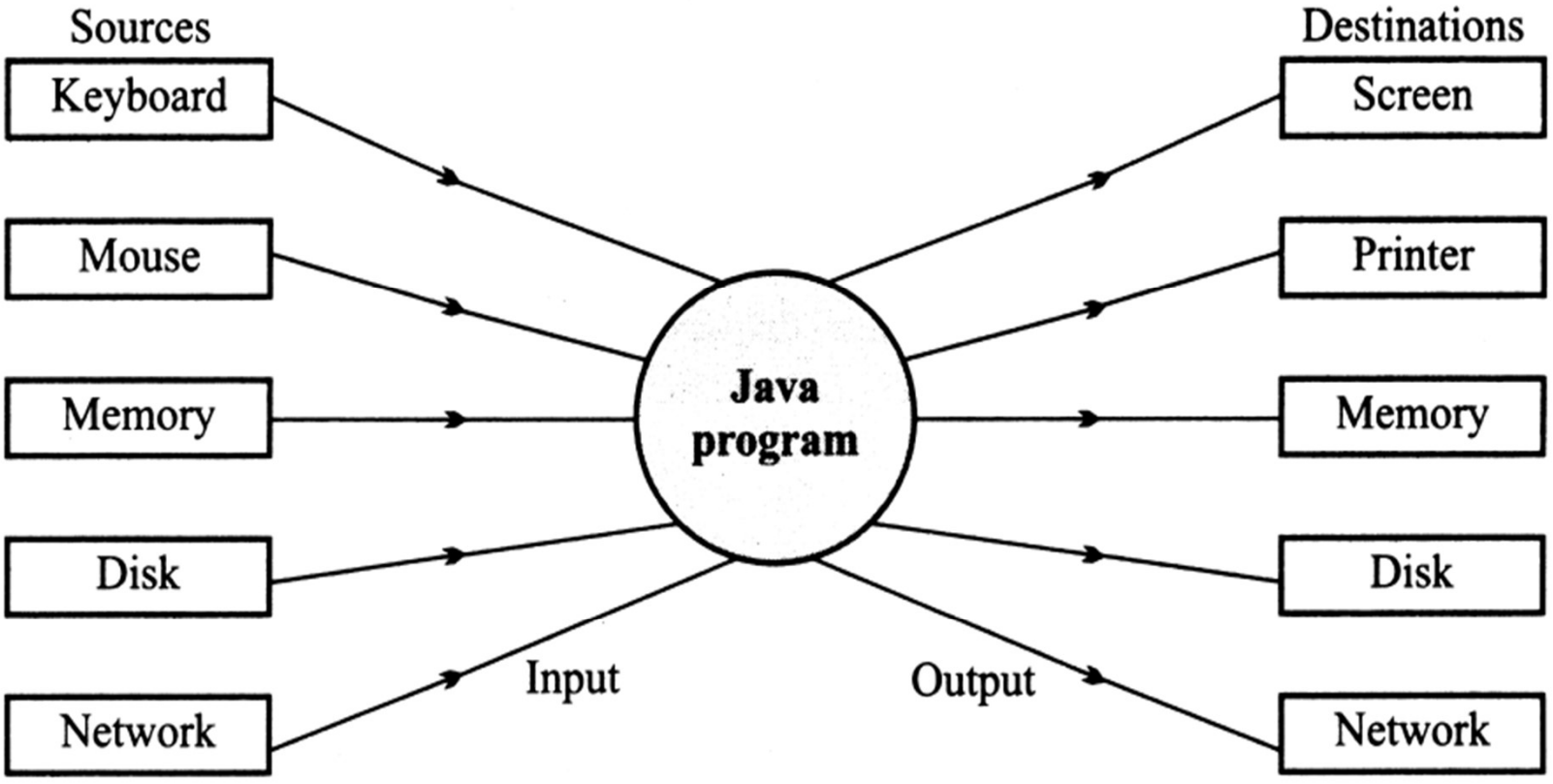
C - Input/Output Recap

- FILE* fp;
- fp = fopen(“In.file”, “rw”);

- fscanf(fp, ……);
- frpintf(fp, …..);

- fread(……., fp);
- fwrite(……..., fp);

I/O and Data Movement



Relationship of Java program with I/O devices

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Why do we need to store data in file?

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Need for File

- Storage of data in variables and arrays is temporary
- Files
  - used for long-term retention of large amounts of data
  - even after the programs that created the data terminate
- Persistent data
  - exists beyond the duration of program execution
- Files stored on secondary storage devices

Data Hierarchy

- Computers process all data items as combinations of zeros and ones
- Bit – smallest data item on a computer, can have values 0 or 1
- Byte – 8 bits
- Characters – larger data item
  - Consists of decimal digits, letters and special symbols
  - Character set – set of all characters used to write programs and represent data items
    - Unicode – characters composed of two bytes
    - ASCII

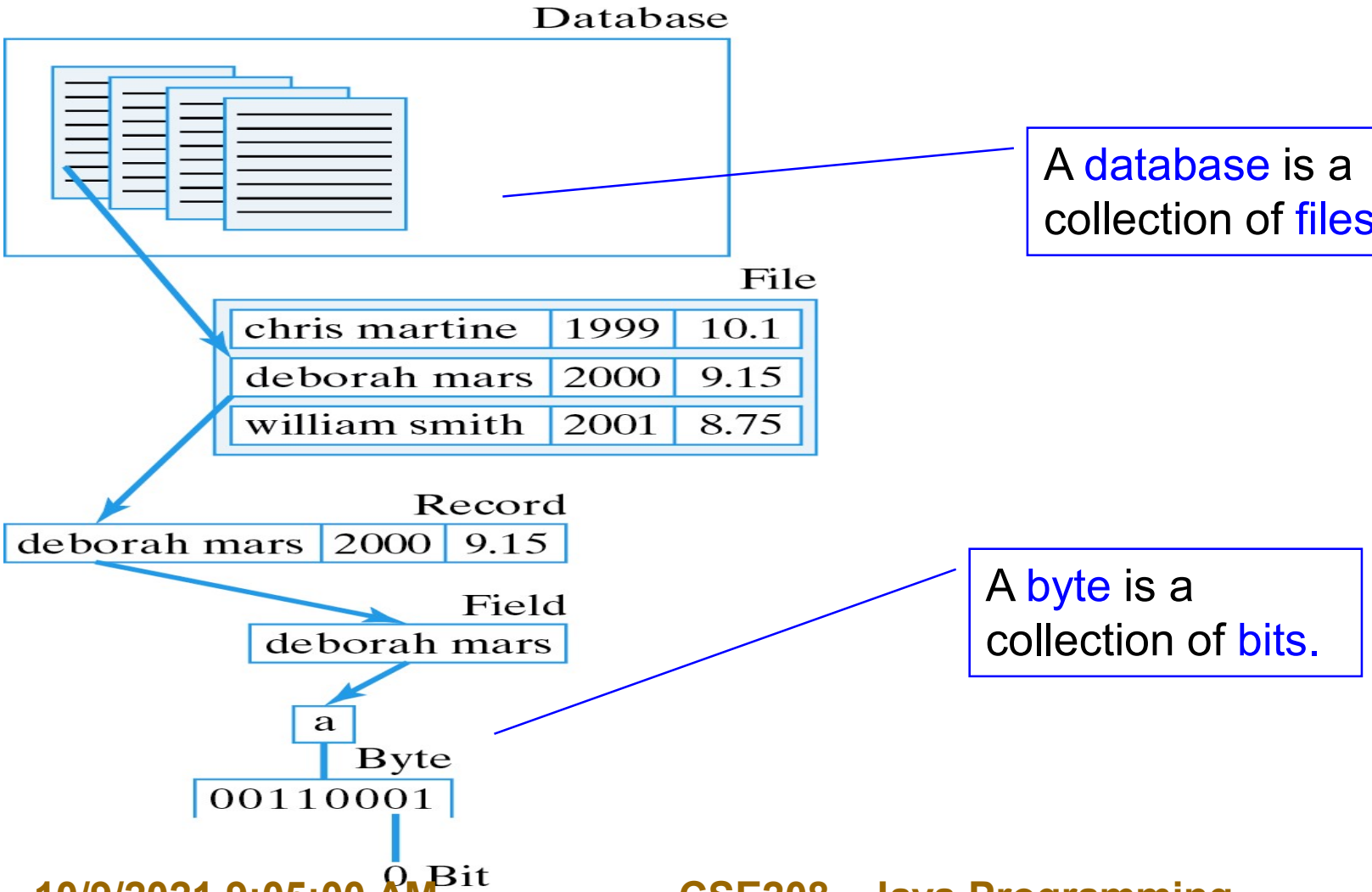
SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Data Hierarchy

- Fields – a group of characters or bytes that conveys meaning
- Record – a group of related fields
- File – a group of related records

Data Hierarchy

- Data can be arranged in a hierarchy.

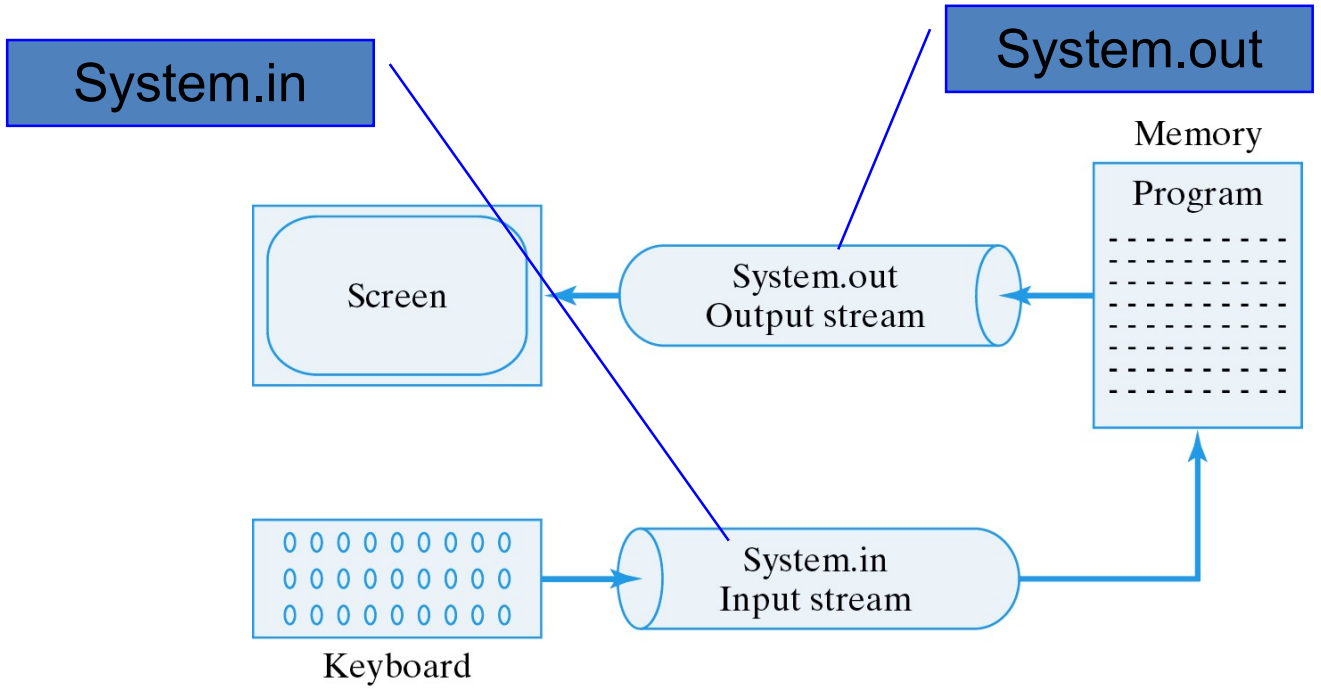


A database is a collection of files.

A byte is a collection of bits.

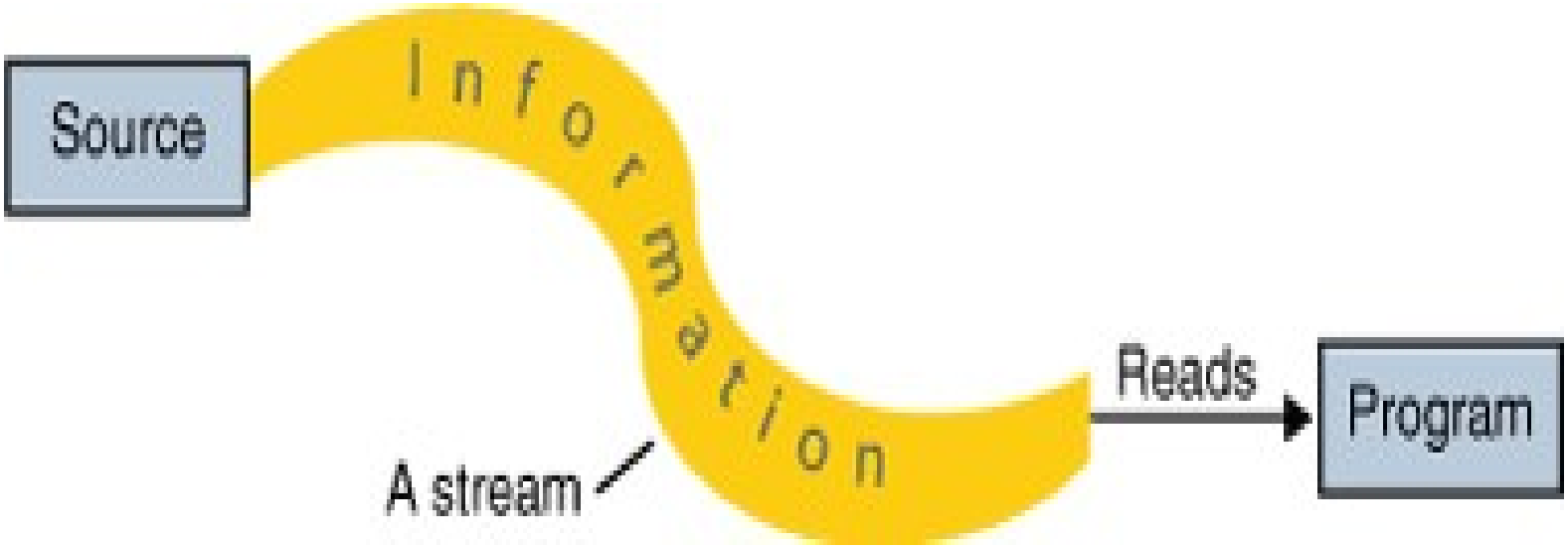
Streams and Files

- Java performs i/o – streams
- stream - abstraction - produces/consumes information
- logical entity
- stream - linked - physical device - same manner - file/keyboard/socket



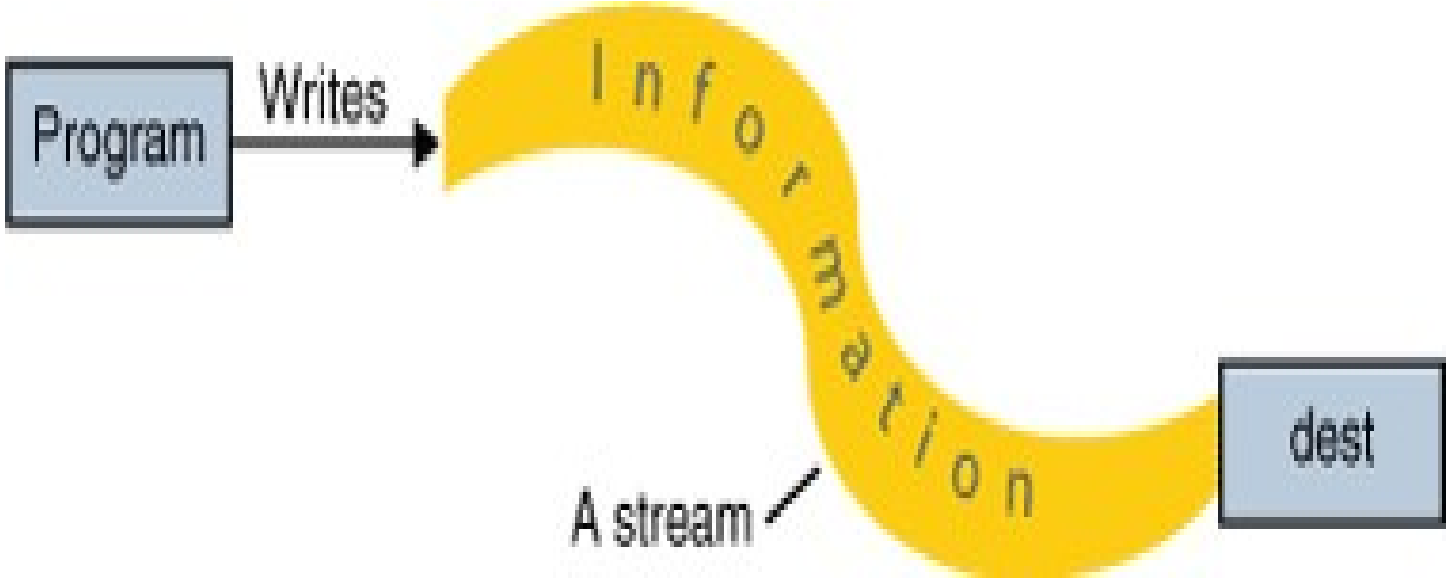
Reading Data

To bring in information, a program opens a stream on an information source (a file, memory, a socket) and reads the information sequentially, as shown in the following figure.



Writing Data

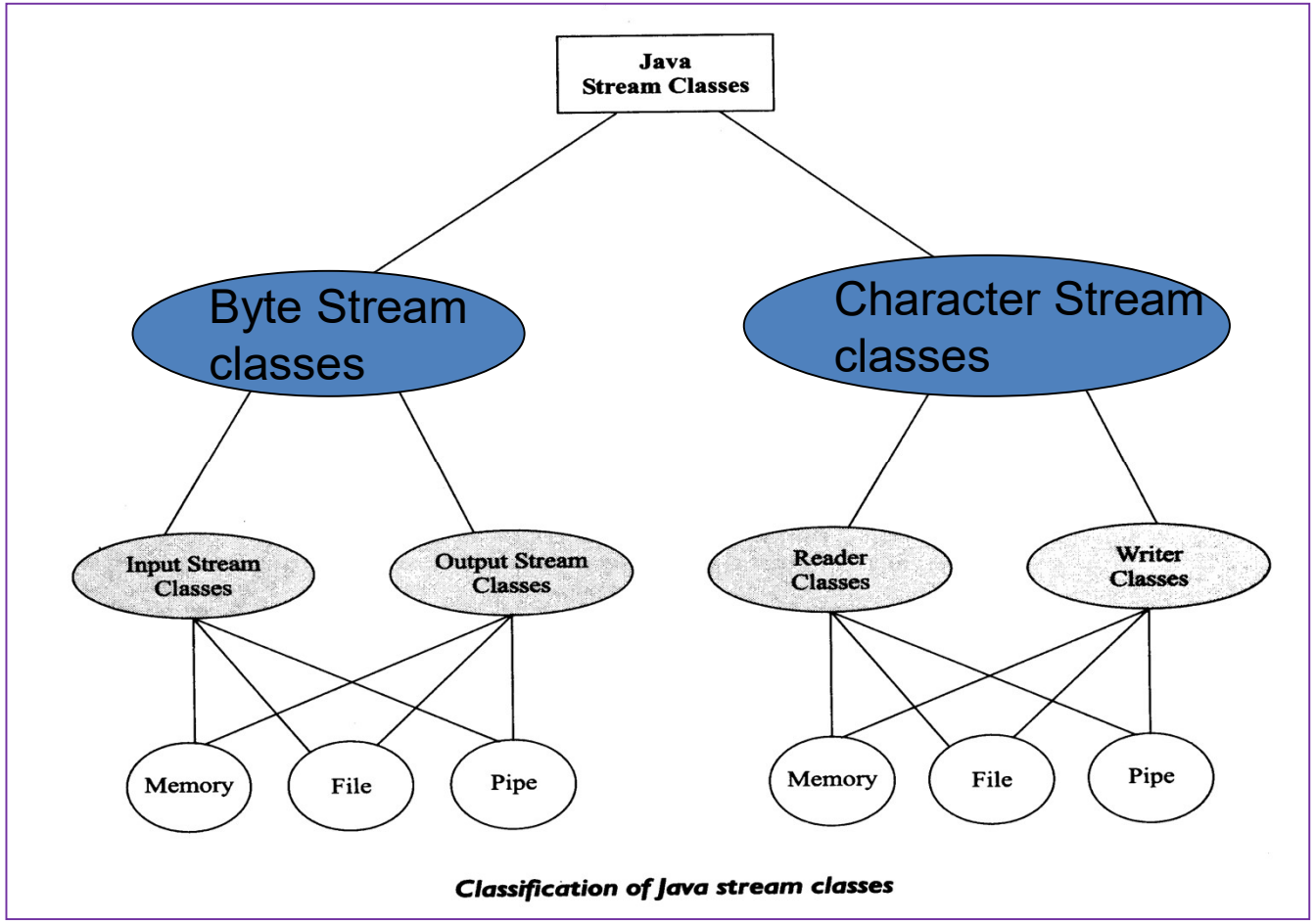
A program can send information to an external destination by opening a stream to a destination and writing the information out sequentially, as shown in the following figure.



Streams - Classification

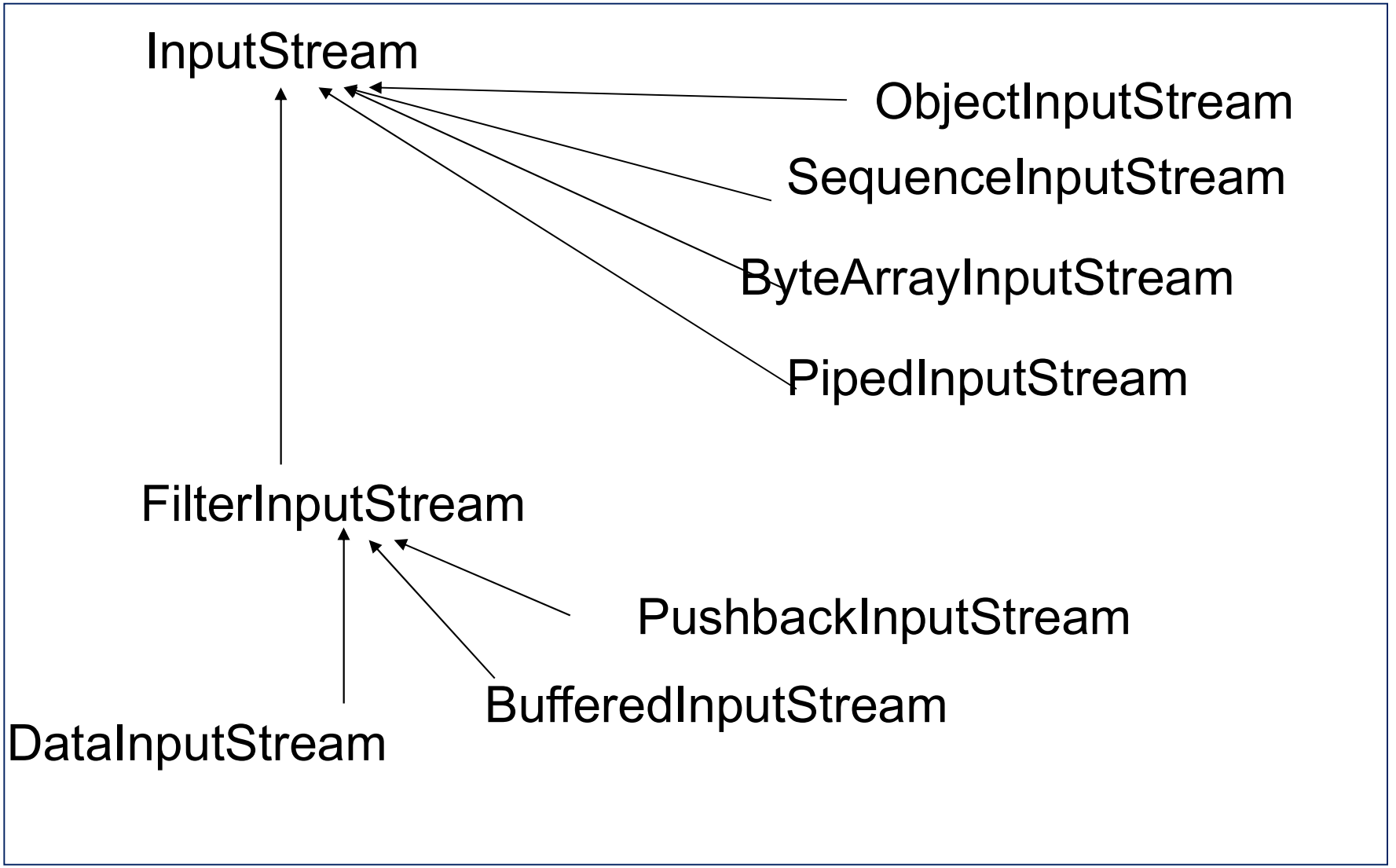
Byte Streams	Character streams (JDK 1.1)
when reading or writing binary data	input and output of characters
Operated on 8 bit (1 byte) data.	Operates on 16-bit (2 byte) unicode characters.
Input stream/Output stream	Reader/ Writer

Classification of Java Stream Classes



Classification of Java stream classes

Byte Input Streams



Byte Input Streams - operations

public abstract int read()	Reads a byte and returns as a integer 0-255
public long skip(long count)	Skips count bytes.
public int available()	Returns the number of bytes that can be read.
public void close()	Closes stream

The Predefined Streams

- All Java programs automatically import the **java.lang package.**
- This package defines a class called **System,**
  - which encapsulates several aspects of the run-time environment.
- For example,
  - using some of its methods,
  - you can obtain the current time
  - the settings of various properties associated with the system.
- **System also contains three predefined stream variables: in, out,** and **err.**

The Predefined Streams

- These fields are declared as public, static, and final within System.
- They can be used by any other part of your program and without reference to a specific System object.

- System.out refers to the standard output stream.
  - By default, this is the console.
- System.in refers to standard input,
  - keyboard by default.
- System.err refers to the standard error stream,
  - console by default.

The Predefined Streams

- System.in
  - is an object of type InputStream
- System.out and System.err
  - are objects of type PrintStream

Reading Console Input

```
import java.io.*;

class consoleread
{
public static void main(String args[]) throws Exception
{
char c;
DataInputStream din=new DataInputStream(System.in);
System.out.println("Enter Characters, q to quit");
do
{
c=(char) din.read();
System.out.println(c);
}while(c!='q');

}
}
```

int read( ) throws IOException

```
E:\slides\Java - 2021\Unit - II>java consoleread
Enter Characters, q to quit
sastraq
s
a
s
t
r
a
q
```

Reading Strings

```
import java.io.*;

class readingstrings
{
public static void main(String args[]) throws Exception
{
String str;
DataInputStream din=new DataInputStream(System.in);
System.out.println("Enter a String ");
str=din.readLine();
System.out.println(str);
}
}
```

String readLine( ) throws IOException

```
E:\slides\Java - 2021\Unit - II>java readingstrings
Enter a String
SASTRA
SASTRA
```

```
E:\slides\Java - 2021\Unit - II>javac readingstrings.java
Note: readingstrings.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

Writing Console Output

```
import java.io.*;

class writedemo
{
public static void main(String args[]) throws Exception
{
int ch;
ch='a';
System.out.write(ch);
System.out.write('\n');
}
}
```

void write(int byteval)

```
E:\slides\Java - 2021\Unit - II>java writedemo
a

E:\slides\Java - 2021\Unit - II>
```

Reading and Writing Files

```
import java.io.*;

class filedemo
{
public static void main(String args[]) throws Exception
{
FileInputStream f=new FileInputStream("filedemo.java");
int size=f.available();

for(int i=0;i<size;i++)
{
System.out.print((char)f.read());
}
}
}
```

```
E:\slides\Java - 2020\Unit - II>java filedemo
import java.io.*;

class filedemo
{
public static void main(String args[]) throws Exception
{
FileInputStream f=new FileInputStream("filedemo.java");
int size=f.available();

for(int i=0;i<size;i++)
{
System.out.print((char)f.read());
}

}
}
```

Reading and Writing Files

```
import java.io.*;

class filedemo1
{
public static void main(String args[]) throws Exception
{
FileInputStream fin=new FileInputStream("sample.txt");
FileOutputStream fout=new FileOutputStream("temp.txt");

int i;
do
{
i=fin.read();

if(i!=-1)
fout.write(i);

}while(i!=-1);
}
}
```

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

File Class

File class

- Does not operate on streams describes the properties of a file
  - File(String directorypath)
  - File(String directorypath,String filename)
- File Class is not symmetrical.
- Class File useful for retrieving information about files and directories from disk
- Objects of class File do not open files or provide any file-processing capabilities

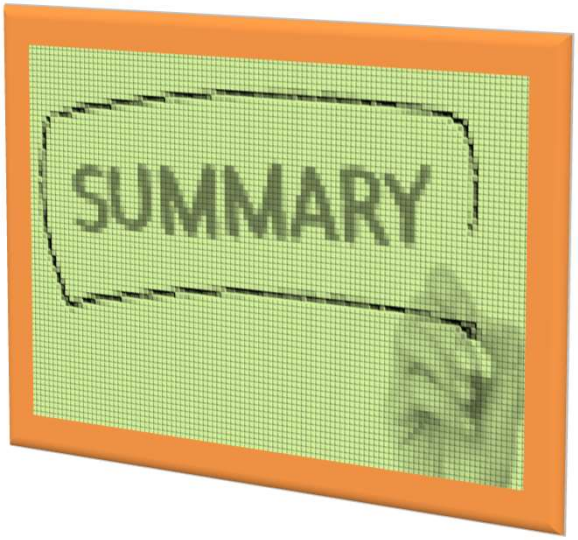
File class

fdemo.java

```
import java.io.*;

class fdemo
{
public static void main(String args[])
{
File f=new File("fdemo.java");
System.out.println(f.getName());
System.out.println(f.exists());
System.out.println(f.canWrite());
System.out.println(f.canRead());
System.out.println(f.isDirectory());
System.out.println(f.isFile());
System.out.println(f.isAbsolute());
System.out.println(f.length());
}
}
```

```
E:\slides\Java - 2020\Unit - II>java fdemo
fdemo.java
true
true
true
false
true
False
434
```



We have discussed

- Streams
- Predefined Streams
- Reading Console Input
- Writing Console Output
- Reading and Writing Files
- File Class

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

➢Multithreading

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

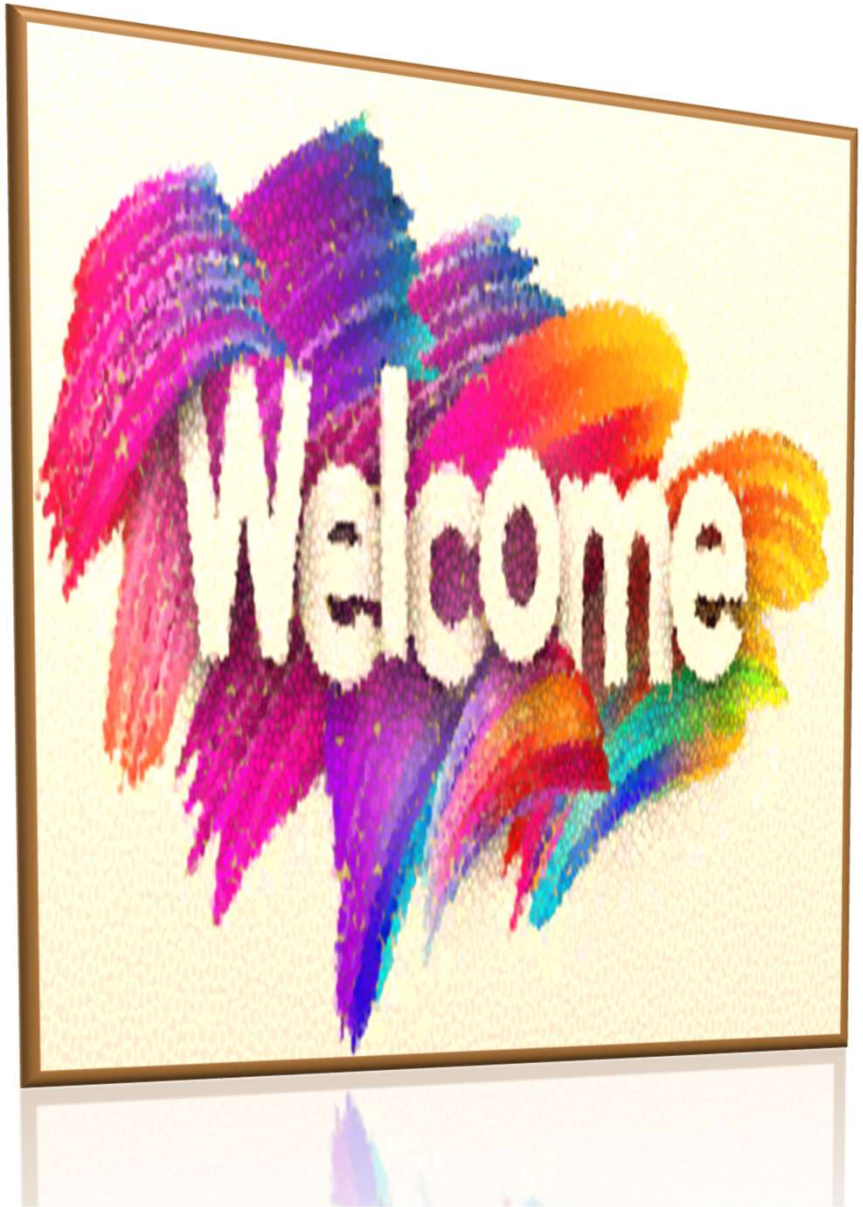
SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA



Multithreading - I

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Introduction
- Multitasking and Multithreading
- Thread Applications
- Defining Threads

Acknowledgement

- Prof. Rajkumar Buyya
- Cloud Computing and Distributed Systems (CLOUDS) Laboratory
- Dept. of Computer Science and Software Engineering
- University of Melbourne, Australia
- Slides – (8 to 15)

- **Why do we need threads?**

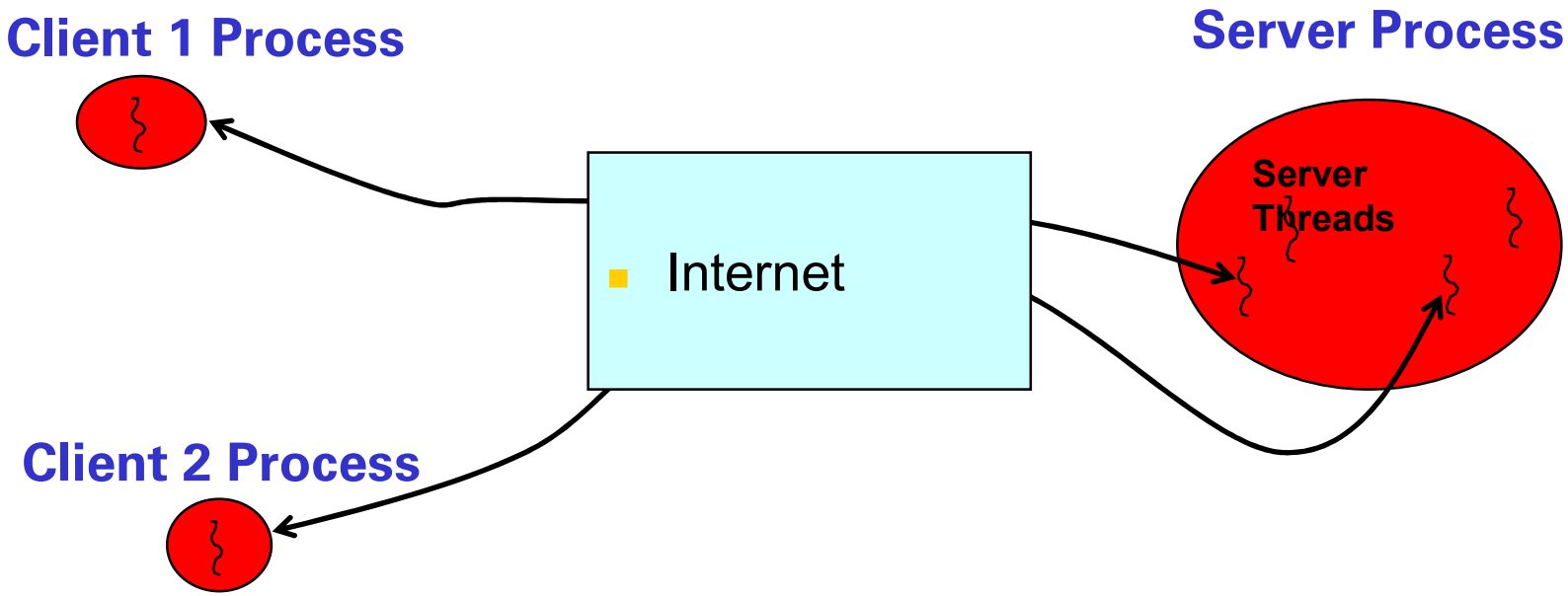
Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize your work depending on priority

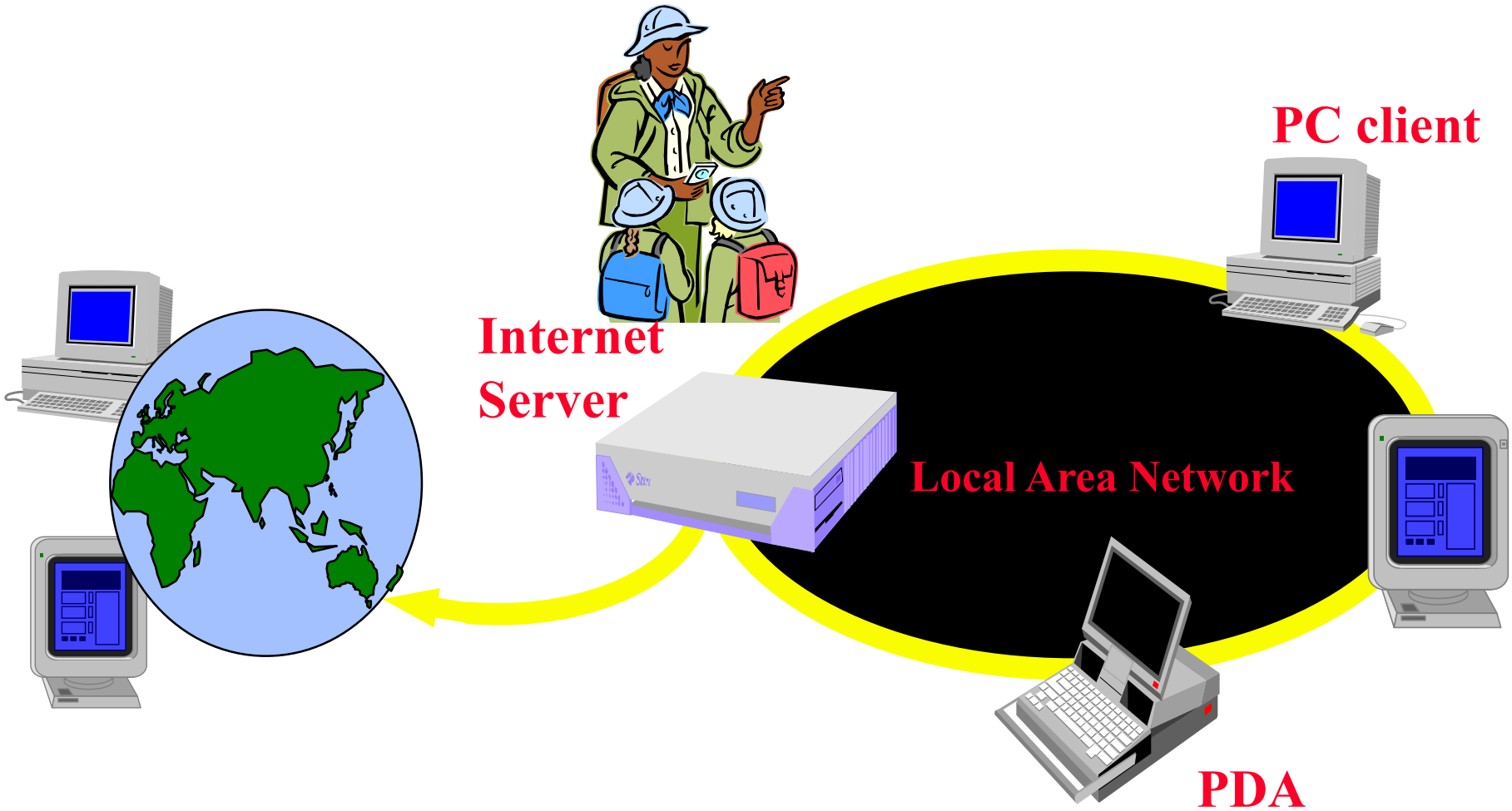
Example

- Consider a simple **web server**
- The web server listens for request and serves it
- If the web server was not multithreaded,
  - the requests processing would be in a queue
  - thus increasing the response time and also might hang the server if there was a bad request.
- By implementing in a multithreaded environment
  - the web server can serve multiple request simultaneously thus improving response time

Multithreaded Server: For Serving Multiple Clients Concurrently



Web/Internet Applications: Serving Many Users Simultaneously



Multitasking and Multithreading

Introduction

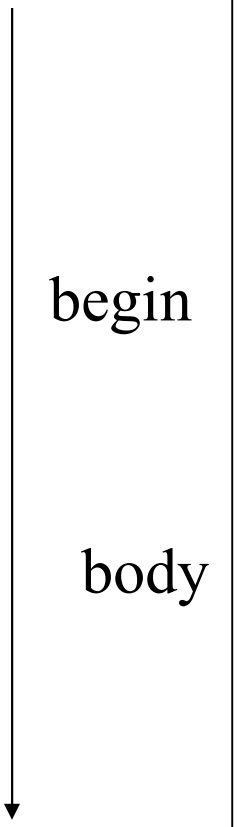
- Multi-tasking (as carried out by an operating system) and Multi-threading (as carried out by a single application)

- Multi-tasking refers to an operating system running several processes concurrently
  - Each process has its own completely independent data
  - Multi-tasking is difficult to incorporate in application programs requiring system programming primitives

- A thread is different from a process in that threads share the same data
  - Switching between threads involves much less overhead than switching between programs
  - Sharing data can lead to programming complications (for example in reading/writing databases)

A single threaded program

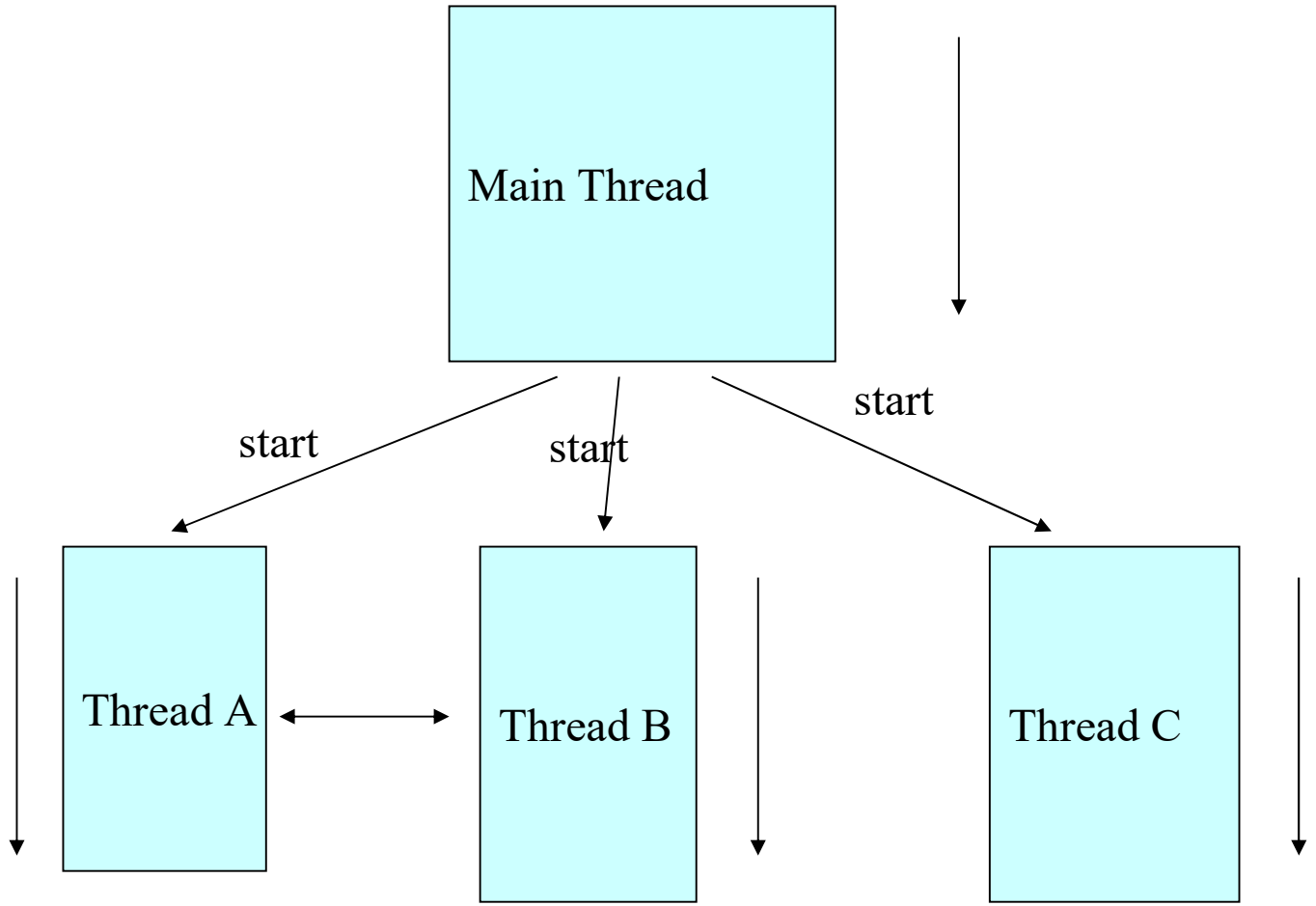
```
class ABC
{
....
public static void main(..)
    {
    …
    ..
    }
}
```




```
class XYZ
{
....

public static void main(..)
    {
    …
    ..
    }
}
```

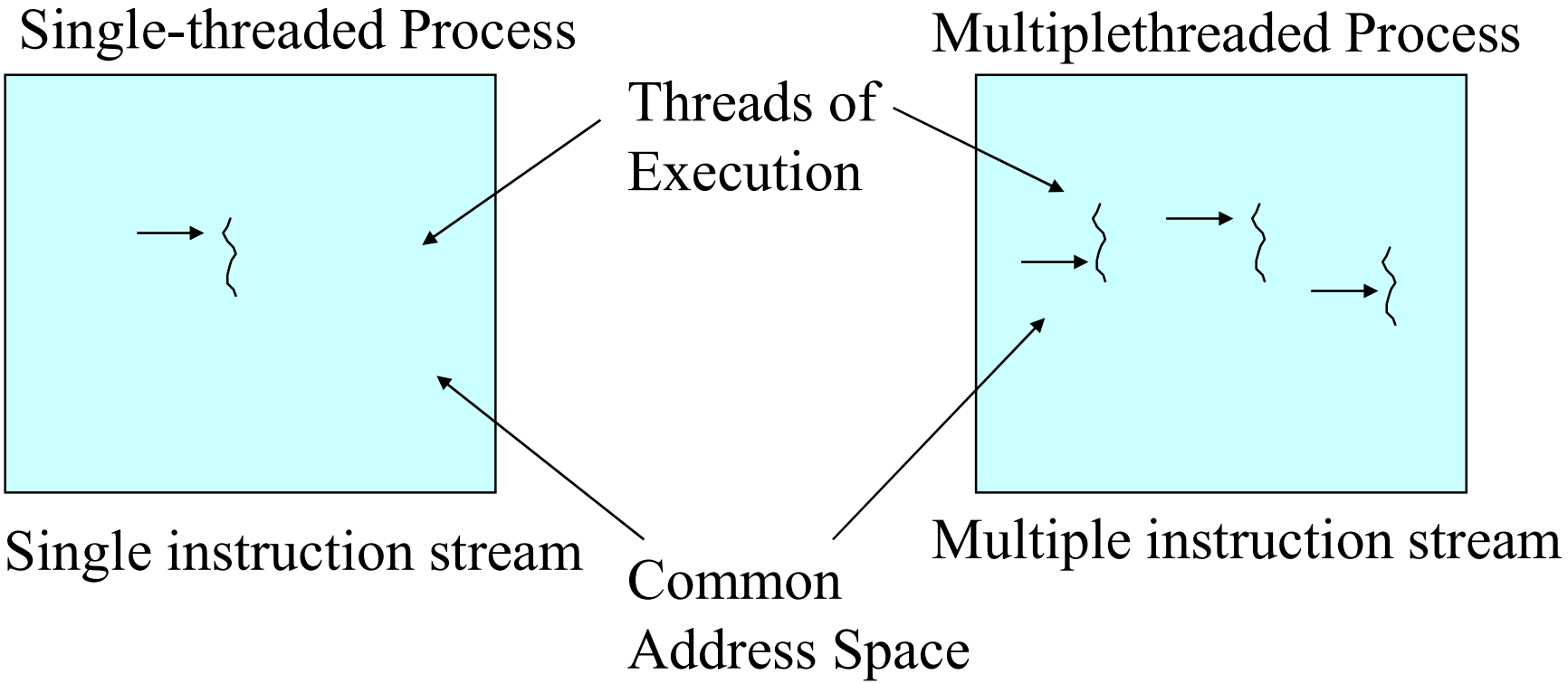
A Multithreaded Program



Threads may switch or exchange data/results

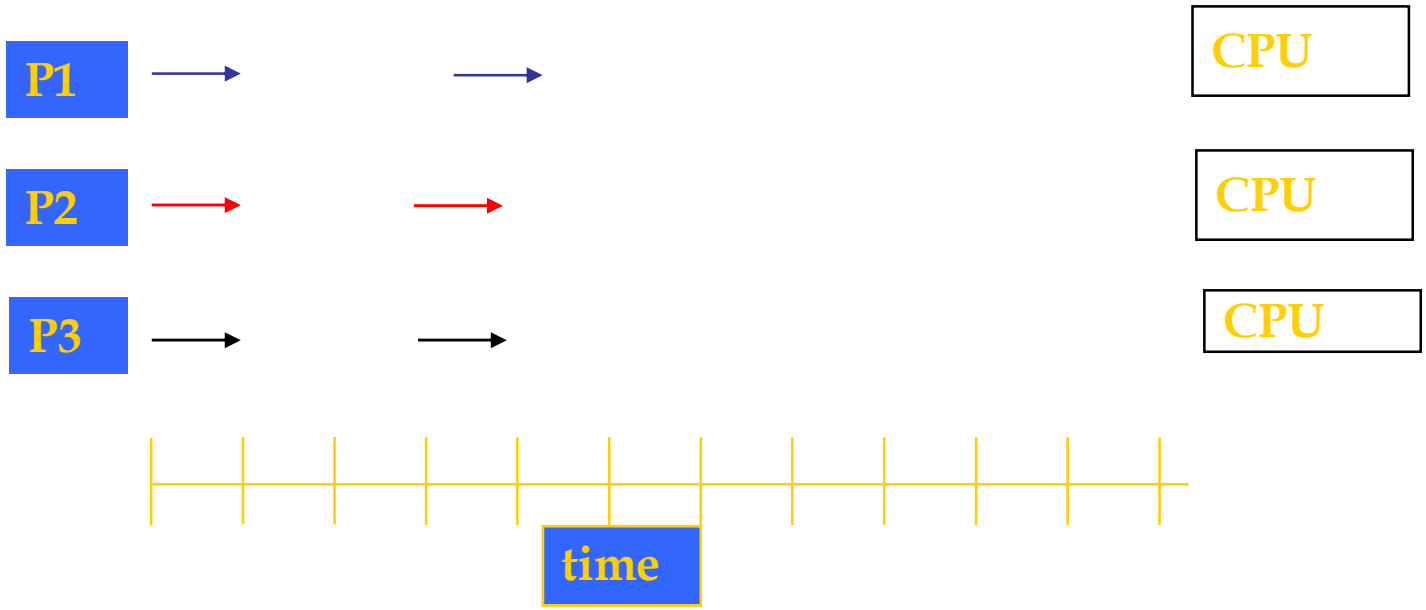
Single and Multithreaded Processes

Threads are light-weight processes within a process



Multithreading - Multiprocessors

Process Parallelism

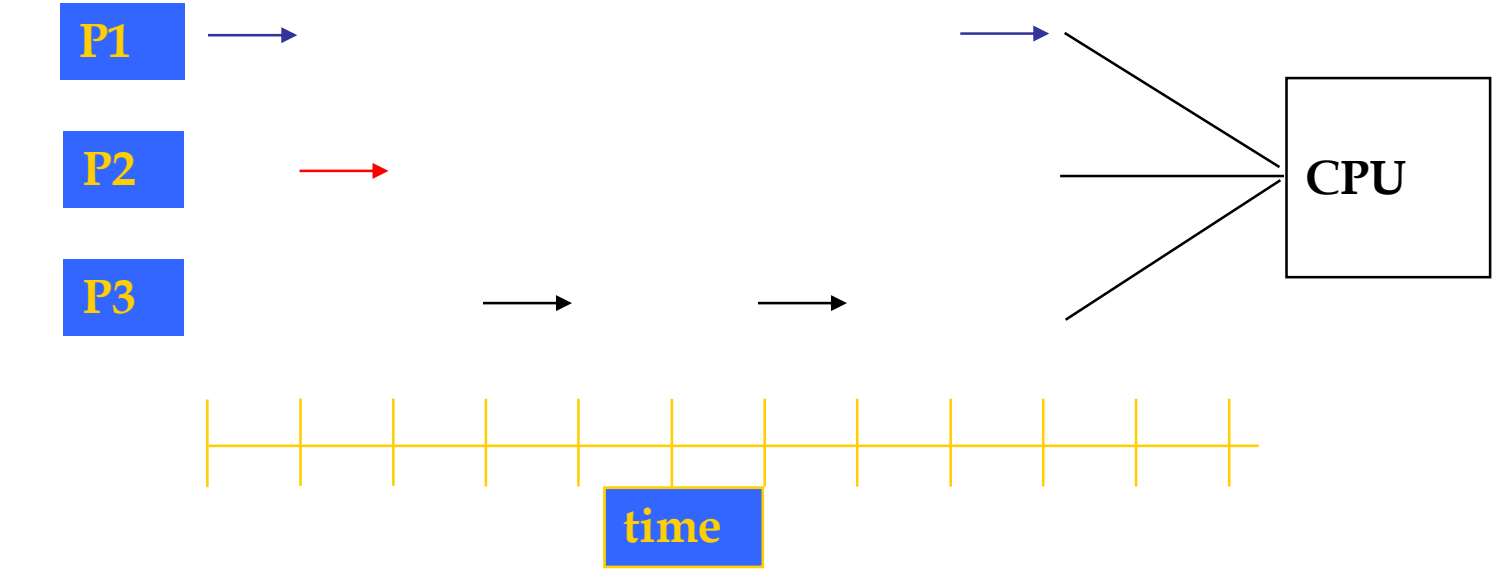


No of execution processes <= the number of CPUs

Multithreading on Uni-processor

Concurrency Vs Parallelism

- Process Concurrency



Number of Simultaneous execution units > number of CPUs

Thread - Definition

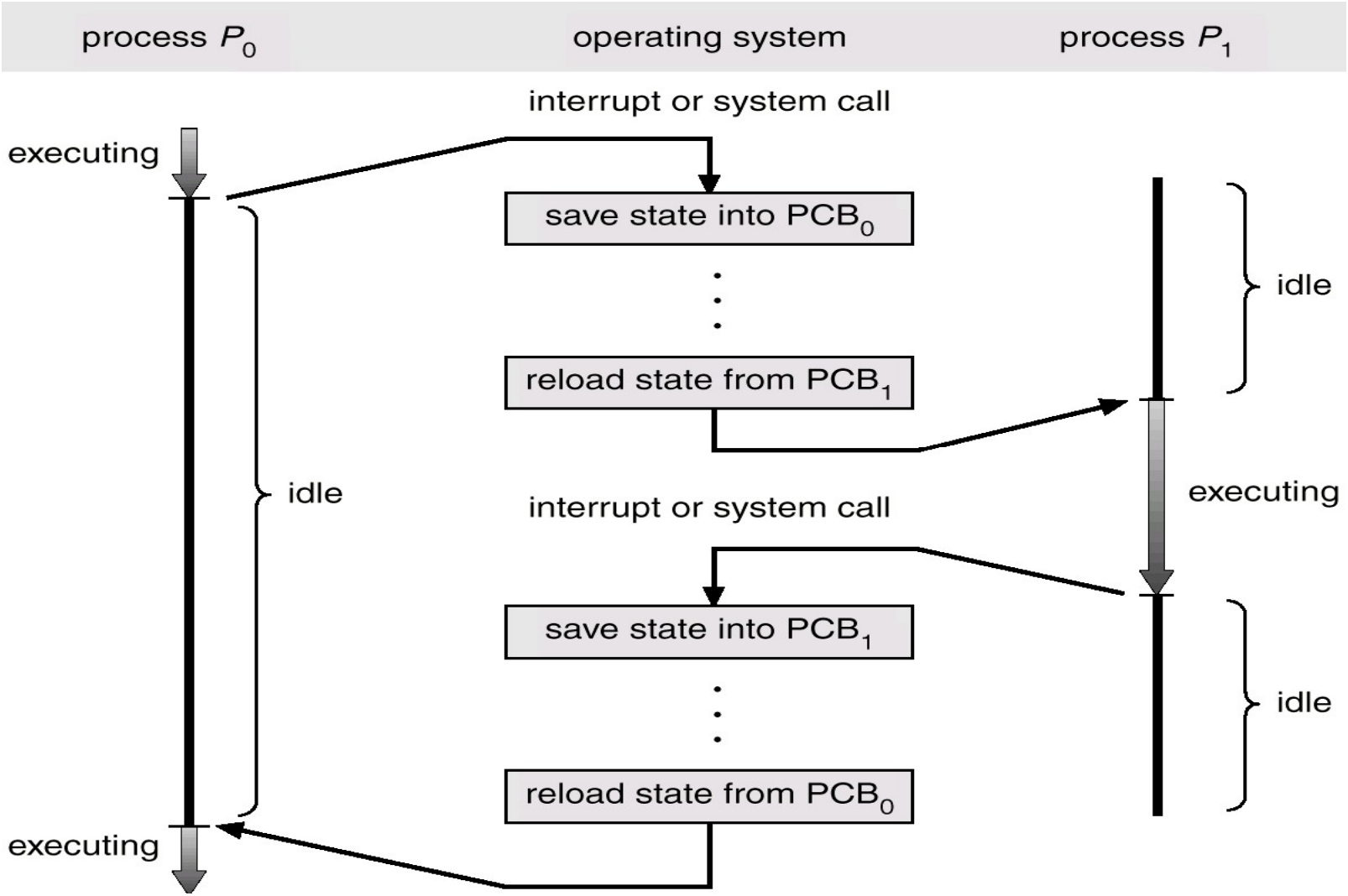
- Java provides built-in support for multithreaded programming.
- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a thread, and each thread defines a separate path of execution.
- In a thread-based multitasking environment, the thread is the smallest unit of dispatchable code.
- This means that a single program can perform two or more tasks simultaneously.
- For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads.

Thread - Definition

- Multitasking threads require less overhead than multitasking processes.
- Processes are heavyweight tasks that require their own separate address spaces.
  - Interprocess communication is expensive and limited.
  - Context switching from one process to another is also costly.
- Threads, on the other hand, are lighter weight.
  - They share the same address space and cooperatively share the same heavyweight process.
  - Interthread communication is inexpensive, and context switching from one thread to the next is lower in cost.

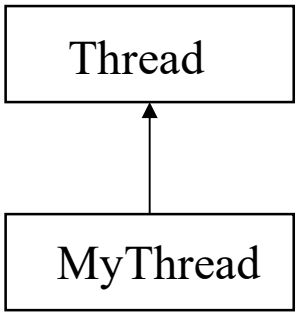
Context Switch

OS – Silberschatz and Galvin



Thread – Creation 1

- Create a class that extends the Thread class

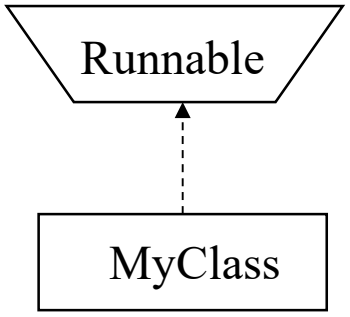



```
class MyThread extends Thread
{
      public void run()
    {
     System.out.println(" this thread is running ... ");
      }
}

class ThreadEx1
{
 public static void main(String [] args  )
 {
    MyThread t = new MyThread();
     t.start();
      }
}
```

Thread – Creation 2

- Create a class that implements the Runnable interface




```
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                 t.start();
        }
}
```

Example - 1

```
class one extends Thread
{
public void run()
{
for(int i=1;i<=5;i++)
{
System.out.println("From one:--->"+i);
}
System.out.println("Exit From one");
}
}
```

```
class two extends Thread
{
public void run()
{
for(int j=1;j<=5;j++)
{
System.out.println("From two:--->"+j);
}
System.out.println("Exit From two");
}
}
```

Example - 1

```
class threaddemo
{
public static void main(String args[])
{
one t1=new one();
two t2=new two();

t1.start();
t2.start();

System.out.println("main thread exiting");
}
}
```

Output- First Run

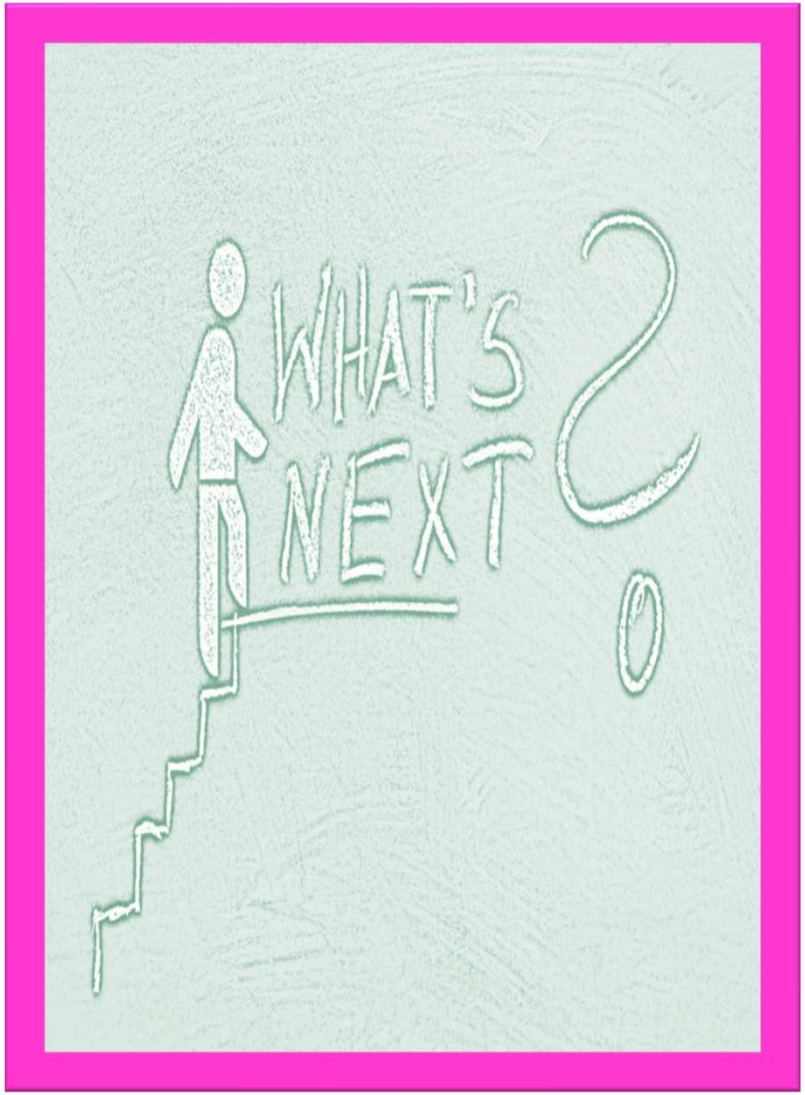
```
E:\slides\Java - 2021\Unit - II>java threaddemo
main thread exiting
From two:--->1
From one:--->1
From one:--->2
From one:--->3
From one:--->4
From one:--->5
Exit From one
From two:--->2
From two:--->3
From two:--->4
From two:--->5
Exit From two
```

Output- Second Run

```
E:\slides\Java - 2021\Unit - II>java threaddemo
From one:--->1
From one:--->2
From one:--->3
From one:--->4
From one:--->5
main thread exiting
Exit From one
From two:--->1
From two:--->2
From two:--->3
From two:--->4
From two:--->5
Exit From two
```

- **We have discussed :**
  - Need for Multithreading
  - Multitasking vs Multithreading
  - Options for creating a thread

- Life Cycle of Thread
- Thread Methods



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Multithreading - II

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Round Robin Algorithm
- Life Cycle of Thread
- Thread Methods
- Thread Priorities
- Using isAlive() and join()

Acknowledgement

- Prof. Rajkumar Buyya
- Cloud Computing and Distributed Systems (CLOUDS) Laboratory
- Dept. of Computer Science and Software Engineering
- University of Melbourne, Australia
- Slide -6

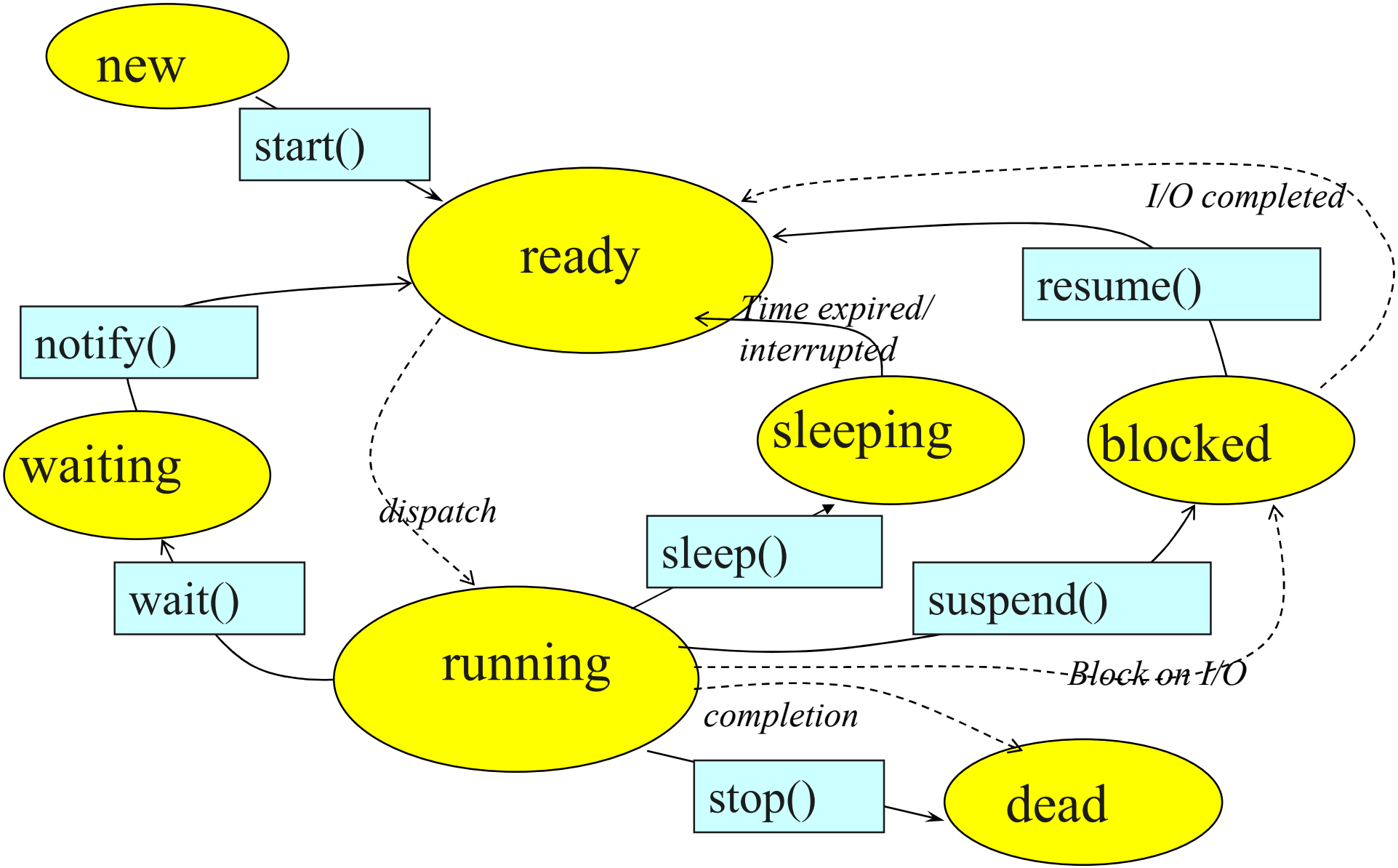
Basic Scheme

- Two threads with the same priority are competing for CPU cycles, the situation is a bit complicated.
- operating systems,
  - threads of equal priority are time-sliced automatically in round-robin fashion.
- For other types of operating systems,
  - threads of equal priority must voluntarily yield control to their peers.
  - If they don't, the other threads will not run.

Basic Scheme

- A thread can voluntarily relinquish control.
  - This occurs when explicitly yielding, sleeping, or when blocked.
  - In this scenario, all other threads are examined, and the highest-priority thread that is ready to run is given the CPU.
- A thread can be preempted by a higher-priority thread.
  - In this case, a lower-priority thread that does not yield the processor is simply preempted—no matter what it is doing—by a higher-priority thread.
- Basically, as soon as a higher-priority thread wants to run, it does.
- This is called preemptive multitasking.

Life Cycle of Thread



Thread Methods

- static void sleep(long milliseconds) throws InterruptedException
- static void sleep(long milliseconds, int nanoseconds) throws InterruptedException

- public void yield()
- public void start()
- public void stop()

Threadmethodsdemo.java

- final void setName(String *threadName)*
- final String getName()

- We have discussed :
  - RR algorithm
  - Thread Methods

- Thread Priorities
- Using isAlive() and join()


Thank you!

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Multithreading - III

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Thread Priorities
- Using isAlive() and join()

Thread Priorities

- Java assigns to each thread a priority
  - determines how that thread should be treated with respect to the others.
- Thread priorities are integers that specify the relative priority of one thread to another.
- As an absolute value, a priority is meaningless;
  - a higher-priority thread doesn't run any faster than a lowerpriority thread if it is the only thread running.
- Instead, a thread's priority is used to decide when to switch from one running thread to the next.

Thread Priorities

- final void setPriority(int *level)*
- final int getPriority( )

- values are 1 and 10

Threadpriority.java

- MIN_PRIORITY - 1
- NORM_PRIORITY – 5 (Default)
- MAX_PRIORITY - 10

Using isAlive() and join()

- final boolean isAlive( )
- final void join( ) throws InterruptedException

Threadjoin.java

The Main Thread

- When a Java program starts up
  - one thread begins running immediately.
- This is usually called the main thread of your program
  - it is the one that is executed when your program begins.
- The main thread is important for two reasons:
  - It is the thread from which other "child" threads will be spawned.
  - Often, it must be the last thread to finish execution because it performs various shutdown actions.

The Main Thread

- Although the main thread is created automatically
  - when your program is started
  - it can be controlled through a Thread object.
- obtain a reference to it
- by calling the method currentThread()
- which is a public static member of Thread.
- Its general form is shown here:

Current Thread

```
static Thread currentThread( )
```

The Main Thread

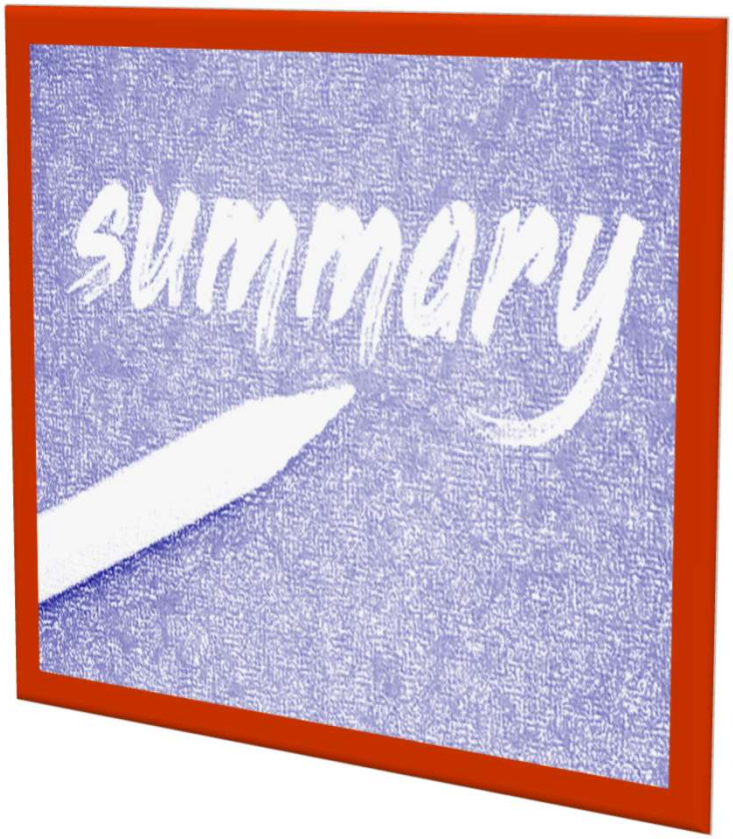
```
E:\slides\Java - 2021\Unit - II>java currentthread
RUNNABLE

 current thread:Thread[main,5,main]

 After Name Change:Thread[ICT,5,main]

 After Priority Change:Thread[ICT,10,main]
```

- We have discussed :
  - Thread Priorities
  - Using isAlive() and join()



- Synchronization


Thank You

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

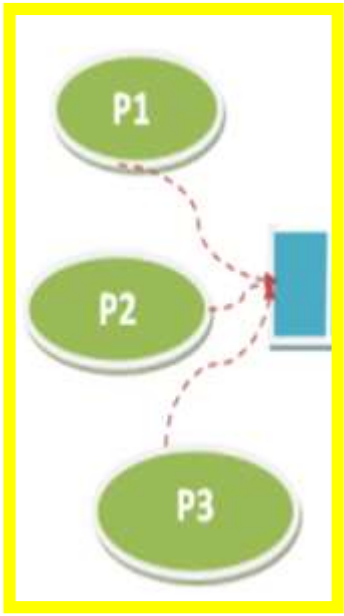
Multithreading - IV

G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Limitations of Multithreading
- Need for Synchronization
- Race Condition
- Producer-Consumer Problem

Limitations of Multithreading

- Multiple Threads namely p1,p2 and p3
- Concurrent execution – Data Inconsistency
- Sharable data – (x)



- **Solution:**
  - **Need to Synchronize the action between the threads**

Non Shared Data

- Two Independent Threads
- No need to Synchronize.

Thread 1:

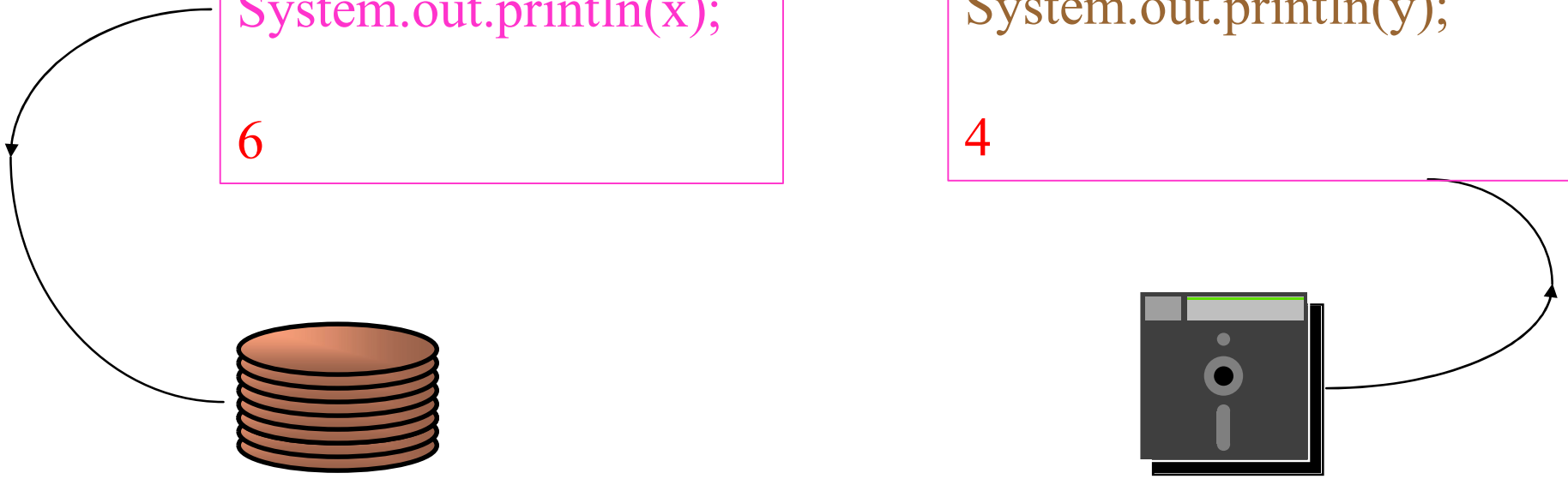
```
int x=5;
x=x+1;
System.out.println(x);
```

6

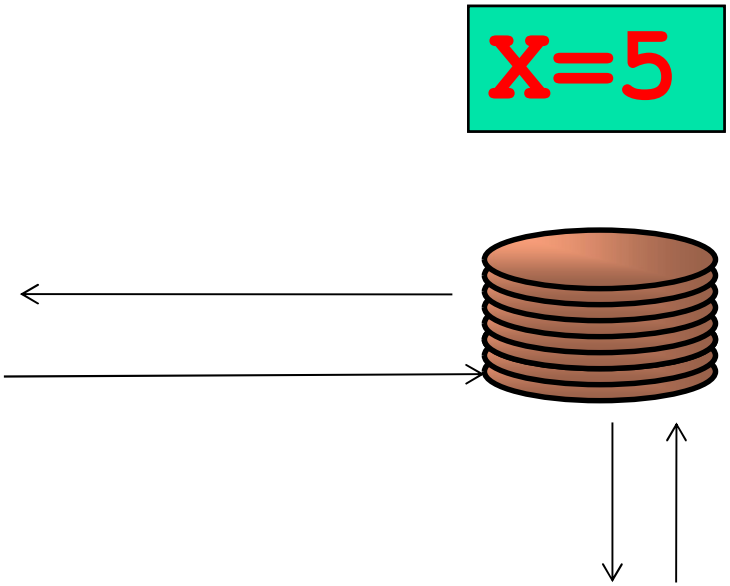
Thread 2:

```
int y=5;
y=y-1;
System.out.println(y);
```

4



Cooperative Synchronized Threads

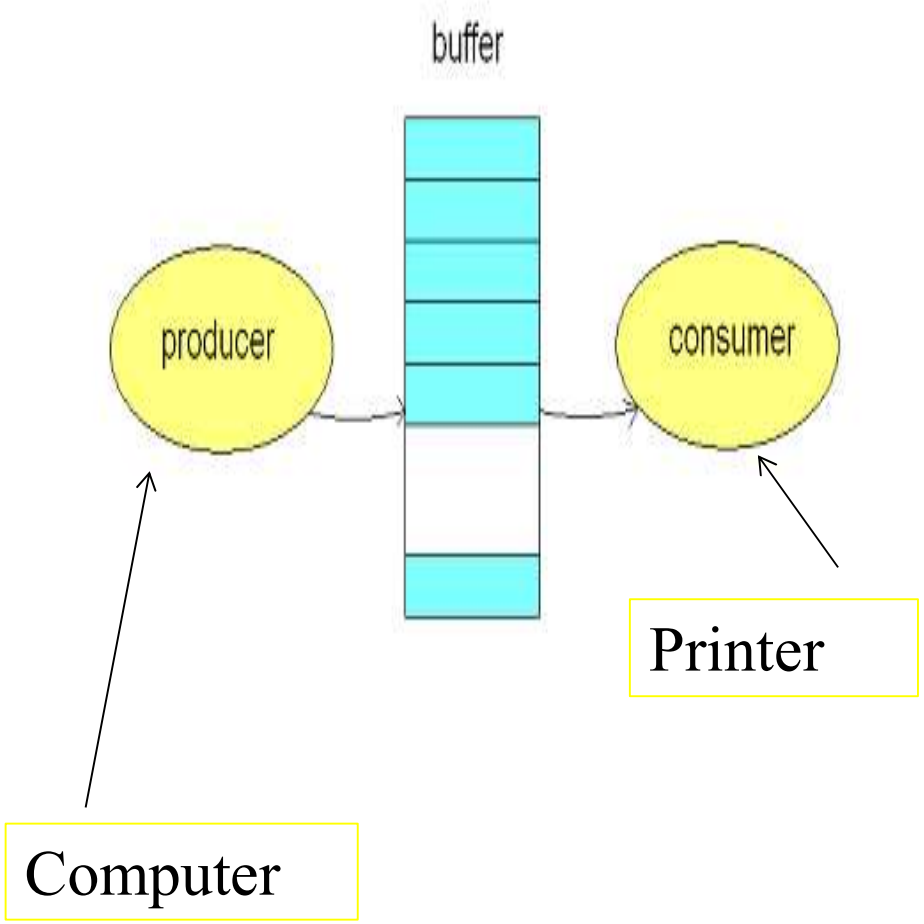


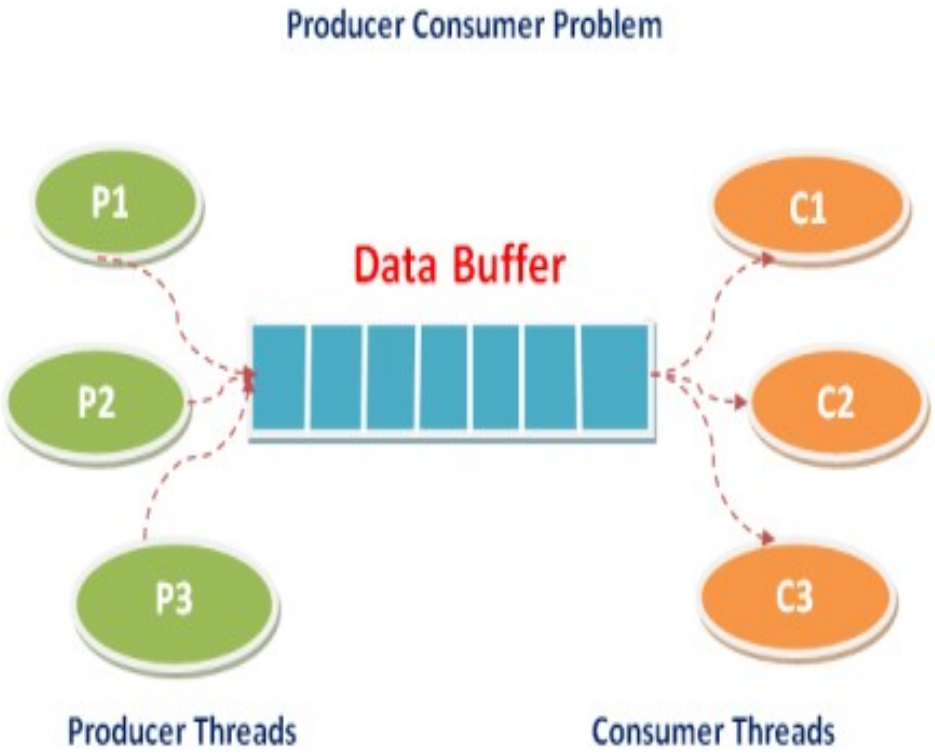
Thread 1:6
Thread 2:5

Thread 2:4
Thread 1:5

- Final value of **x=5**

Producer / Consumer problem





Bounded Buffer

- Both routines are correct separately
- May not function correctly when executed concurrently
- Counter (x) = 5
- Producer (Thread 1) – counter=counter+1
- Consumer (Thread 2) - counter=counter-1
- The value of counter is **4,6**
- Correct value is **5** – executed separately

Bounded Buffer

- **counter = counter + 1;**
  counter = counter - 1;

  must be performed **atomically.**

- Atomic operation means an operation that completes in its entirety without interruption.

Bounded Buffer

- The statement "**counter=counter+1**" may be implemented in machine language as:

```
register1 = counter
register1 = register1 + 1
counter = register1
```

- The statement "**counter=counter-1**" may be implemented as:

```
register2 = counter
register2 = register2 – 1
counter = register2
```

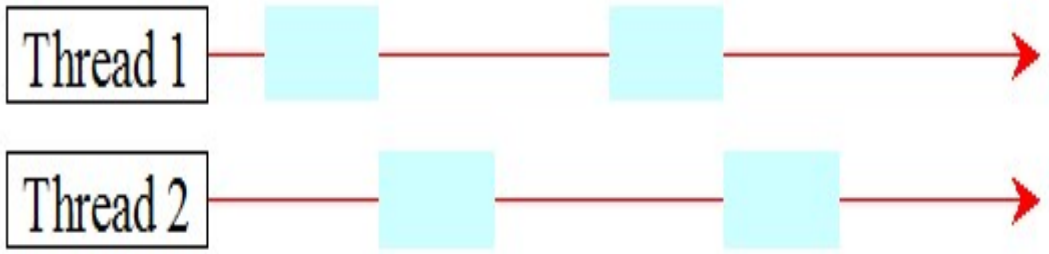
- **Register1,register2 are local CPU registers**

Bounded Buffer

- If both the producer and consumer attempt to update the buffer concurrently, the assembly language statements may get interleaved.

- Interleaving depends upon how the producer and consumer processes are scheduled.

Multiple threads sharing a single CPU



Bounded Buffer

- Assume **counter** is initially 5. One interleaving of statements is:

  T0:producer: **register1 = counter** (register1 = 5)
  T1:producer: **register1 = register1 + 1** (register1 = 6)

  T2:consumer: **register2 = counter** (register2 = 5)
  T3:consumer: **register2 = register2 – 1** (register2 = 4)

  T4:producer: **counter = register1** (**counter = 6**)
  T5:consumer: **counter = register2** (**counter = 4**)

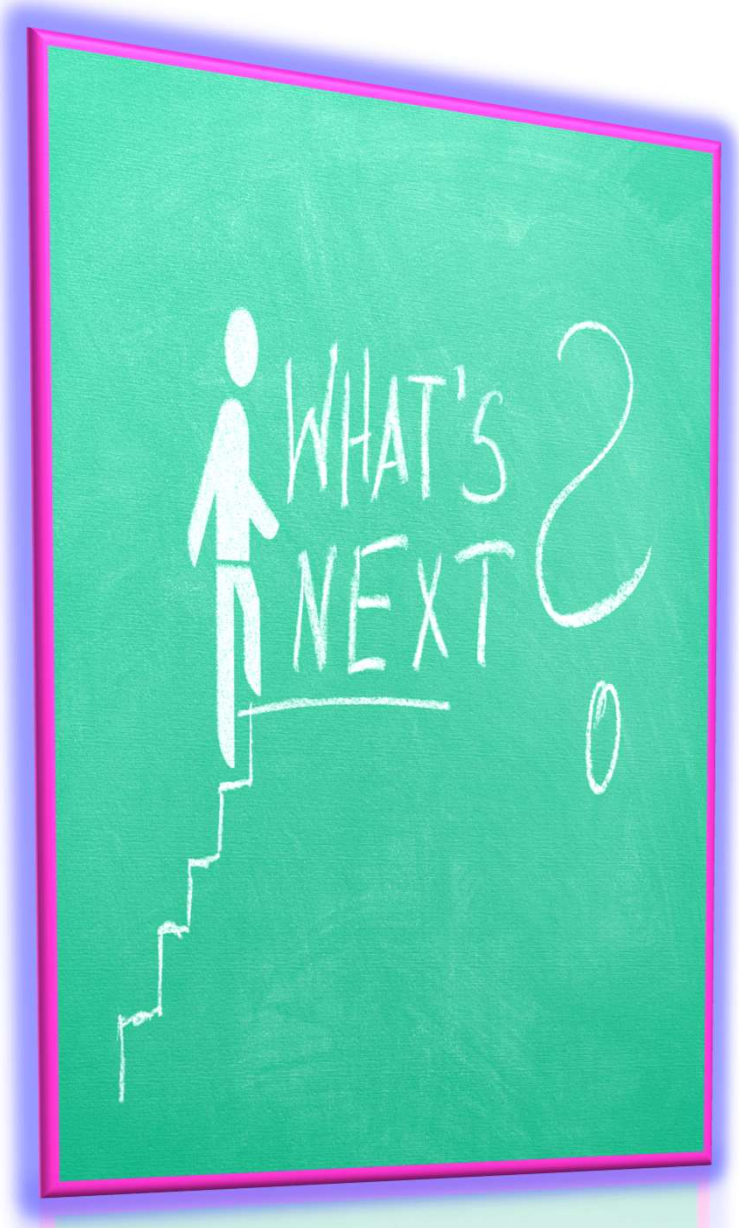
- The value of **count** may be either **4 or 6**, where the correct result should be **5.**

Race Condition

- **Race condition**:
  - The situation where several processes access – and manipulate shared data concurrently.
  - **The final value of the shared data depends upon which process finishes last.**

- To prevent race conditions,
  - concurrent processes must be **synchronized.**

- We have discussed :
  - Need for Synchronization
  - Producer-Consumer Problem
  - Race Condition



- Interthread Communication

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

Multithreading - V

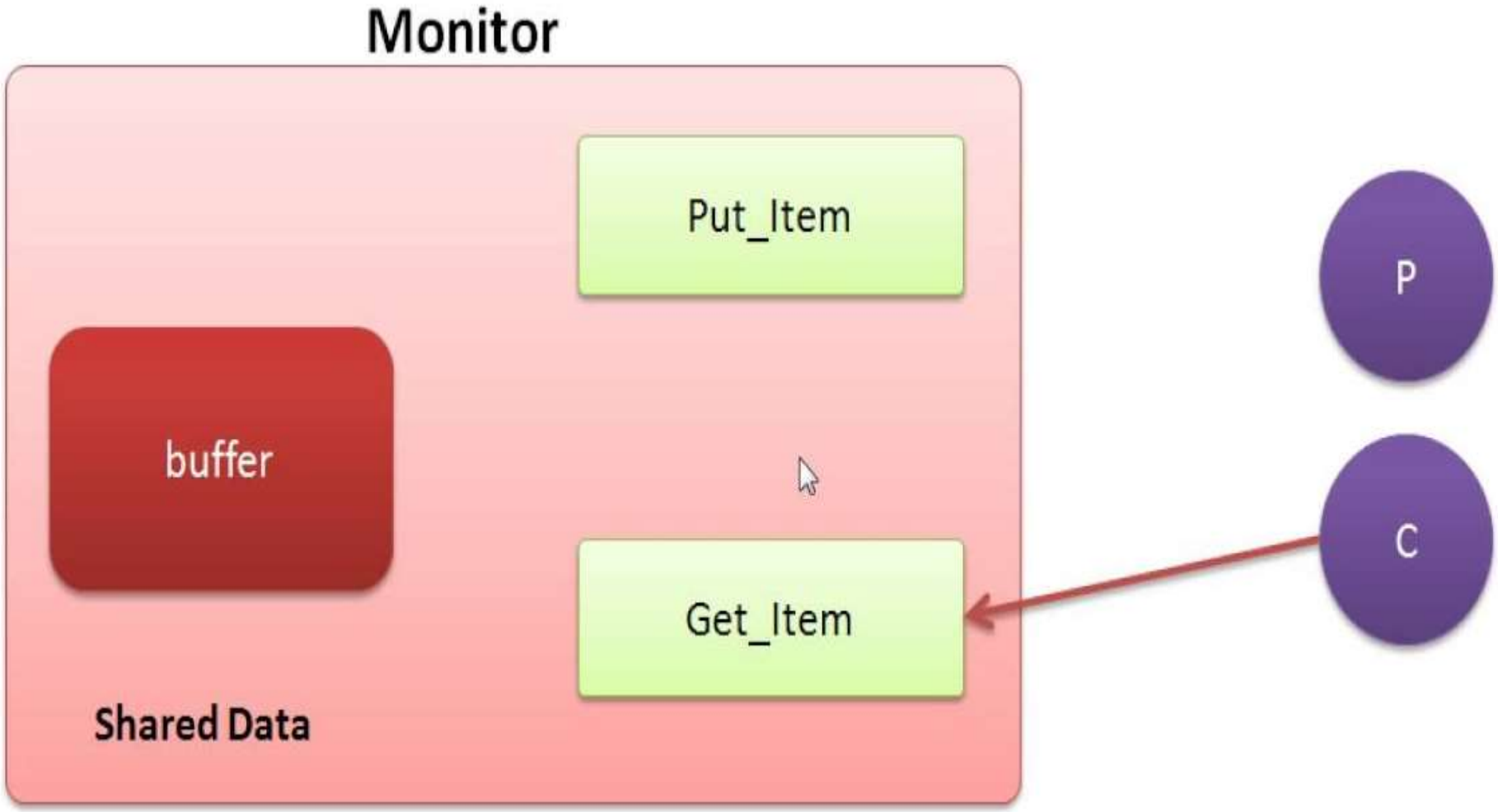
G.MANIKANDAN
SAP / ICT / SOC
SASTRA

- Synchronized Method
- Synchronized Block
- Interthread Communication

Synchronization

- When two or more threads need access to a shared resource
  - they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called synchronization.
- Key to synchronization is the concept of the monitor.
- A monitor is an object that is used as a mutually exclusive lock.

Monitor



Synchronization

- Only one thread can own a monitor at a given time.
- When a thread acquires a lock, it is said to have entered the monitor.
- All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.
- These other threads are said to be waiting for the monitor.
- A thread that owns a monitor can reenter the same monitor if it so desires.

Synchronized Method

- To synchronize your code
  - use the **synchronized keyword**

```
synchronized void show()
{
    }
```

Interthread Communication

- **wait( ) tells the calling thread to give up the monitor and go to sleep until** some other thread enters the same monitor and calls **notify( ) or notifyAll( ).**
- **notify( ) wakes up a thread that called wait( ) on the same object.**
- **notifyAll( ) wakes up all the threads that called wait( ) on the same** object. One of the threads will be granted access.

- **final void wait( ) throws InterruptedException**
- **final void notify( )**
- **final void notify All( )**

Producer/Consumer Problem

Synchronized Block

- Suppose you have multiple lines of code in your method
- But you want to synchronize only few lines
- you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

```
synchronized (object reference expression)
 {
  //code block
}
```

Without Synchronized Block

With Synchronized Block

Suspending and Resuming Threads

- **public void suspend():**
  - is used to suspend the thread(depricated).
- **public void resume():**
  - is used to resume the suspended thread(depricated).

Program

- We have discussed :
  - Synchronized Method
  - Synchronized Block
  - Interthread Communication
  - Suspend and Resume Methods

- **Unit III - Collections**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 OF THE UGC ACT, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA