SunilOS

Agenda

- Elements of Client Server Computing
- Network Basics
- Understanding Ports and Sockets
- Java Sockets
  - Implementing a Server
  - Implementing a Client
- Sample Examples
- Conclusion

Clip slide

SunilOS

JAVA Networking

www.sunilos.com

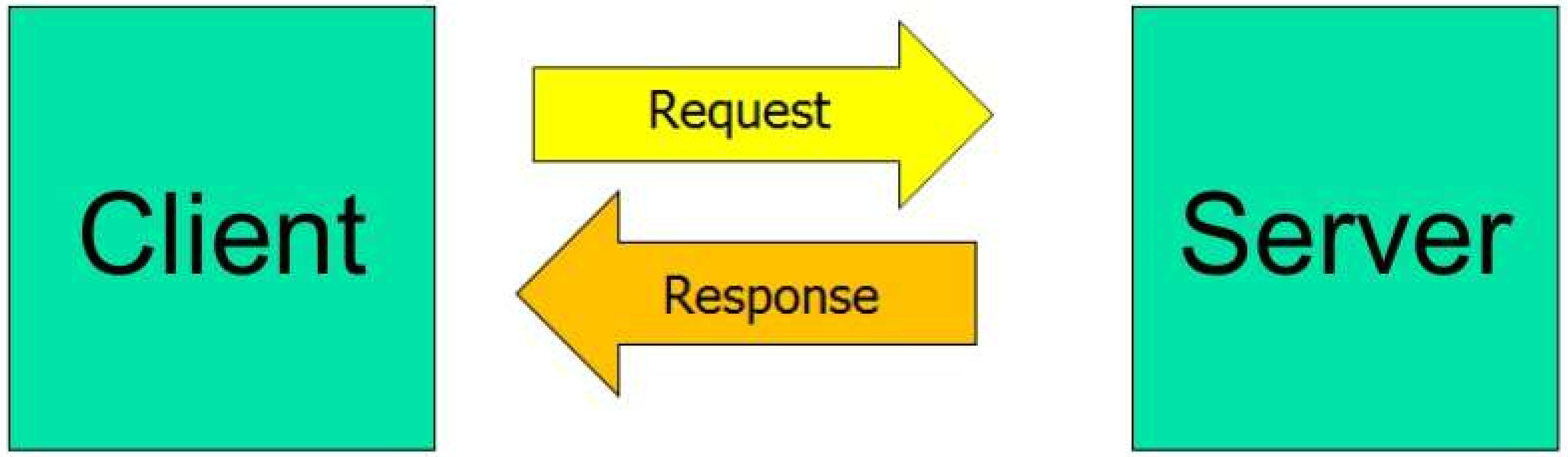
www.raystec.com

Think IT Think Us

RAYS

What is Networking?

- When two processes, lying on same or different machines are communicating over the network is called networking.



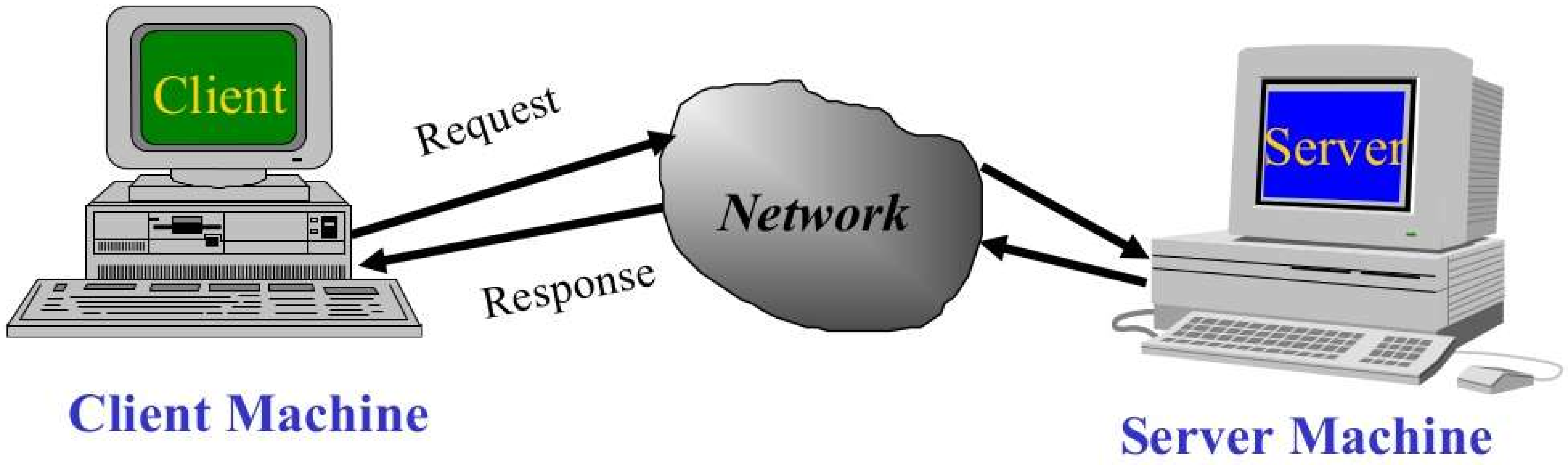
Server & Client

- Process providing services is called Server.

- Process consuming services is called Client

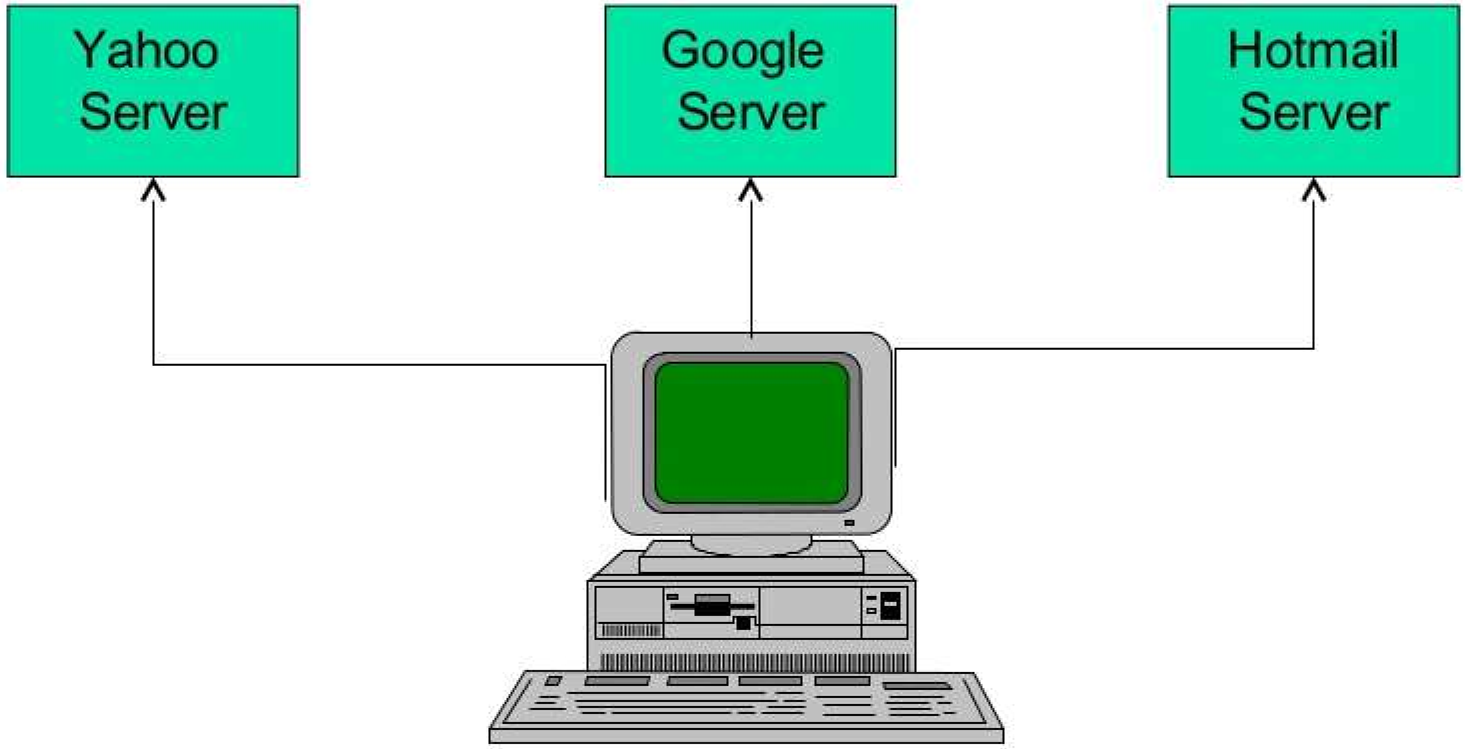
SunilOS

Elements of Client-Server computing



- There are three elements in networking
- Client : sends request for services
- Server : sends response
- Network : media of communication

Single Machine Multiple Clients



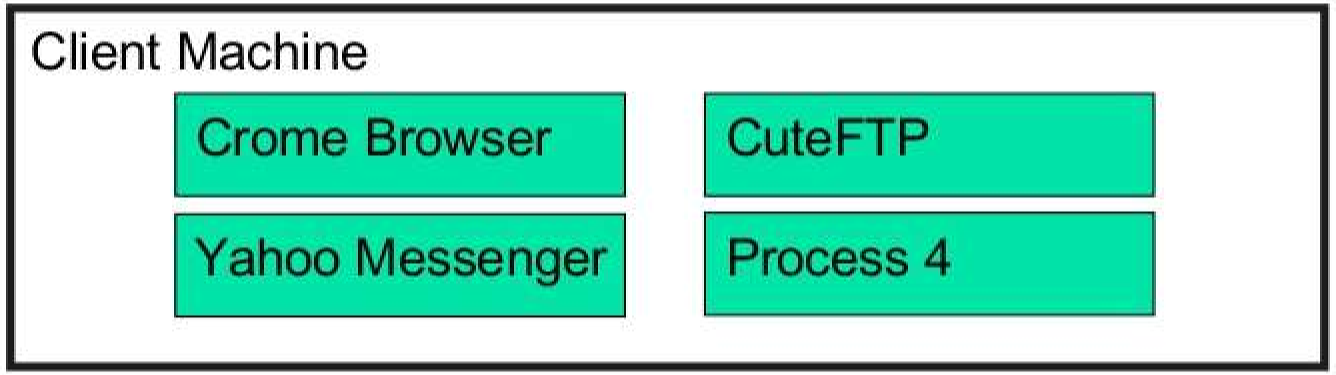
Client's Machine

One Machine can execute multiple clients' processes concurrently

SunilOS

Client is a Process

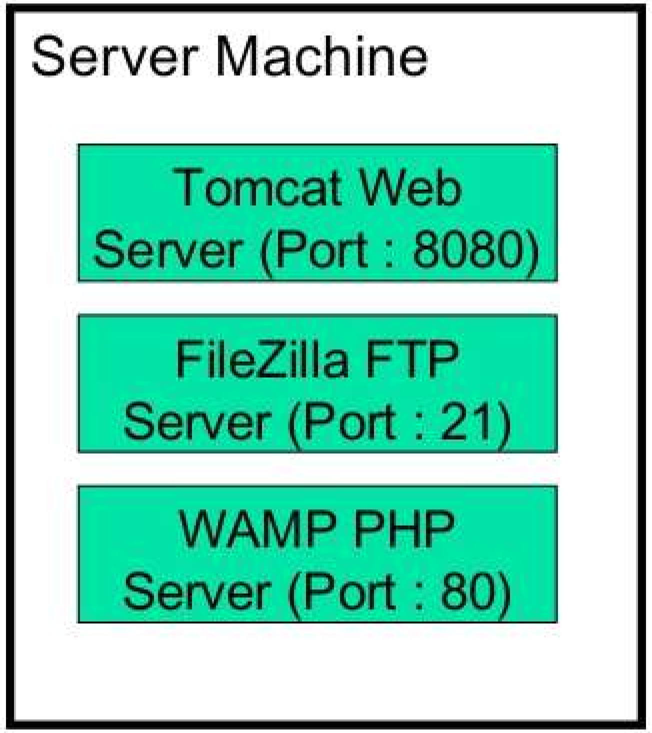
- One machine may run multiple clients (processes) at a time.
- Client is not a machine, it is a process.
  - Browser opens google.com is a Client.
  - Yahoo Messenger connected to chat server is a client.
  - CuteFTP uploading a file to FTP Server is a Client.
- All above mentioned clients (CuteFTP, Messenger, Browser) run together on a single Machine concurrently.



Server is a Process

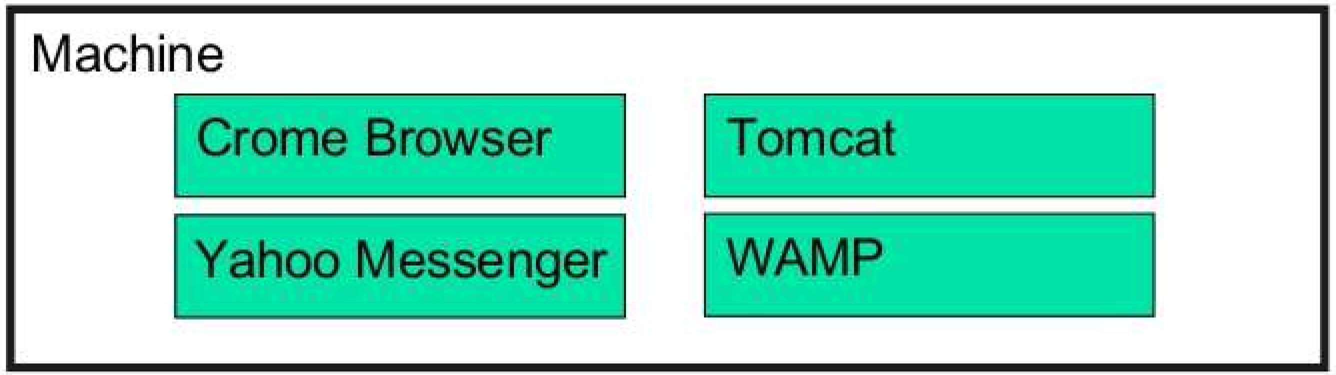
- One machine may run multiple servers (processes) at a time.
- Server is not a machine, it is a process.
  - Tomcat Web Server is a process.
  - WAMP PHP Server is a process.
  - Filezilla FTP Server is a process.
- All above mentioned servers run together on a single Machine concurrently.

Single Machine Multiple Servers


Server Machine
Tomcat Web Server (Port : 8080)
FileZilla FTP Server (Port : 21)
WAMP PHP Server (Port : 80)

Server & Client

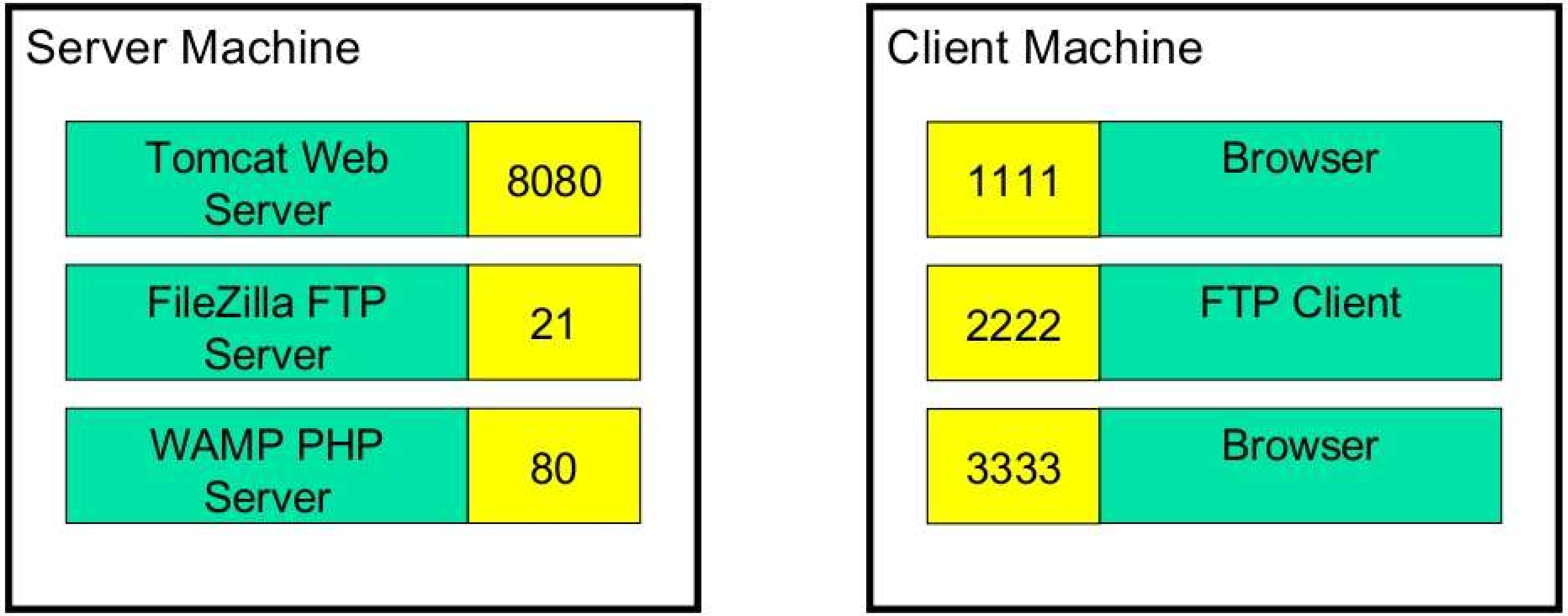
- Since Operating Systems can execute multiple processes concurrently using preemptive scheduling. Thus, one Machine may run multiple Servers and Clients together.
- Client and Server processes communicate over the network to exchange data in form of request and response.



How Clients and Servers are identified?

- ❑ Clients and Servers are uniquely identified on a single machine by unique Port Numbers.
- ❑ Unique Port Number is assigned by OS.
- ❑ Process can ask desired port number from OS, or OS will assign next available port number to a process.
- ❑ Port number is a two bytes unsigned number ranging from 0-65535.

Ports


Server Machine
Tomcat Web Server
8080
FileZilla FTP Server
21
WAMP PHP Server
80
Client Machine
1111
Browser
2222
FTP Client
3333
Browser

Communication Rules (Protocols)

Hello, I am Vijay, May I talk to Tisha →

← Yes, I m Tisha

↔ Blah, Blah, Blah .....
....
....

Bye →

← Have a Good Day

Certain rules are followed when you communicate over the network, are called **Protocols**

Protocol over Phone

- When you call someone over phone and start conversation, you follow protocol.
- First you greet and say “Hello” , then you tell your name “I am Vijay”, then you ask with whom you want to talk “May I talk to Tisha?” .
- Likewise when conversation is over you say “Bye” and other would respond “Have a Good Day”.

This is Protocol

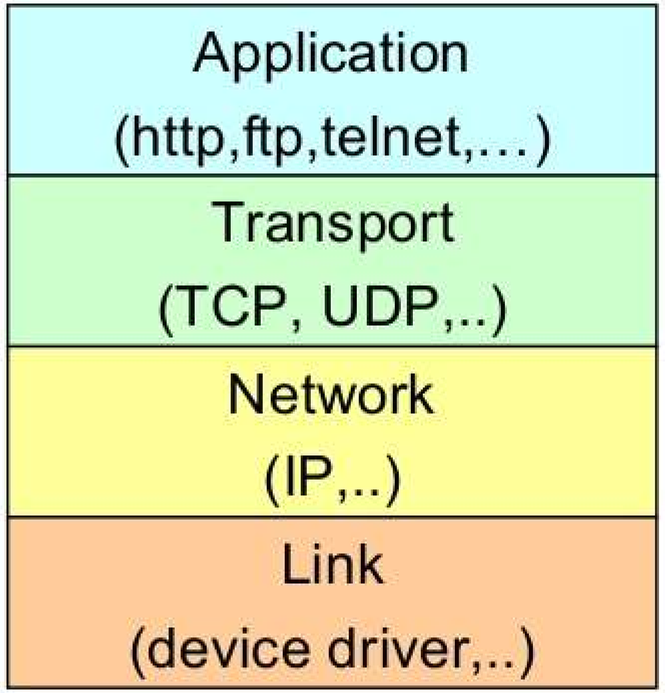
Protocol Responsibilities

- ❑ Applies a set of Rules.
- ❑ Converts your application data into byte stream and vice versa.
- ❑ Breaks data into packets and sends over network.
- ❑ Receives acknowledgements of sent packets.
- ❑ Decides network route to send data packets.

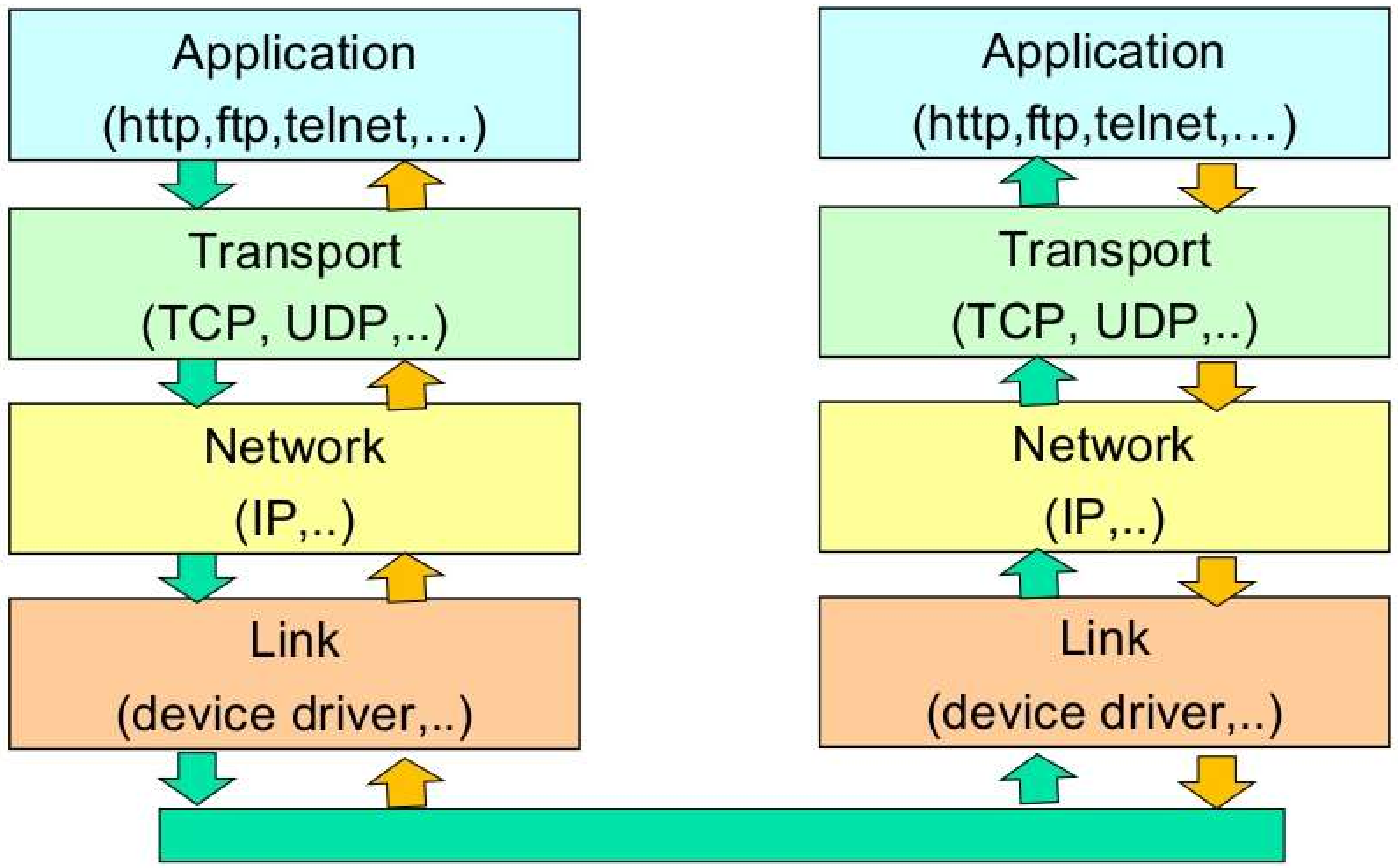
SunilOS

Protocol Stack

- A group of network protocols that work together to send and receive your data over the network, is called a **Protocol Stack**.
- The **TCP/IP** protocol stack uses four layers that map to the OSI model:



Protocol Stack Communication



TCP/IP Protocol Stack

- Applications Layer:
  - contains user custom and high level application protocols like HTTP, FTP, SMTP, Telnet etc.
- Transport Layer:
  - Contains TCP or UDP protocols, responsible for making data packets and send or receive across network.
  - Your custom application will communicate to this layer.
- Network Layer:
  - contains IP Protocol that uses routing information to decide route of data packet to send it to the destination.
- Link Layer:
  - converts data into signals.

SunilOS

TCP Protocols

- TCP (Transport Control Protocol) is a *connection-oriented* protocol that provides a *reliable* flow of data between two computers.
- Example applications:
  - HTTP
  - FTP
  - Telnet

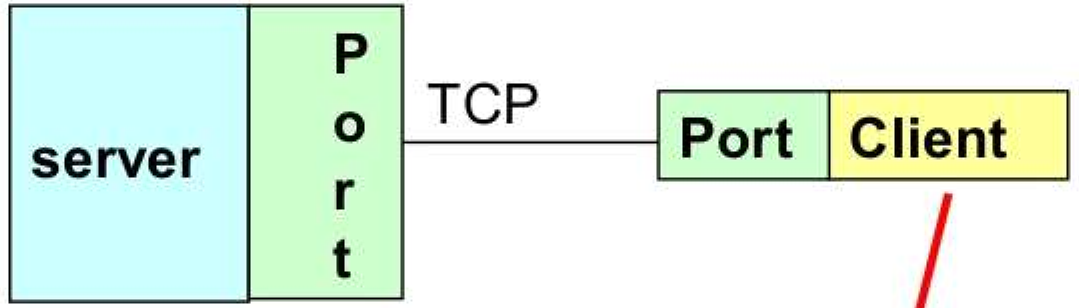
UDP Protocols

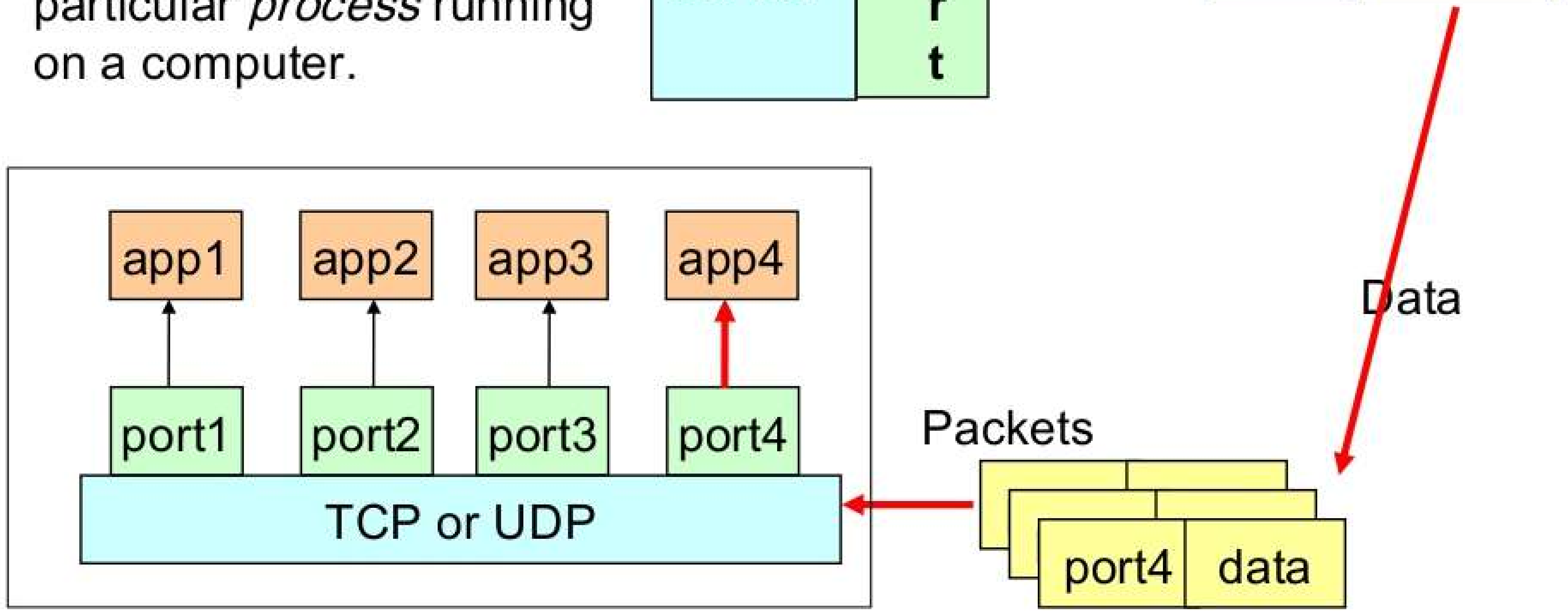
- UDP (User Datagram Protocol): It is a protocol that sends independent packets of data (called *datagrams*) from one computer to another with NO guarantee about arrival.
- Example applications:
  - Clock server
  - Ping

SunilOS

Network Ports

- The TCP and UDP protocols use *ports* to map incoming data to a particular *process* running on a computer.





Ports

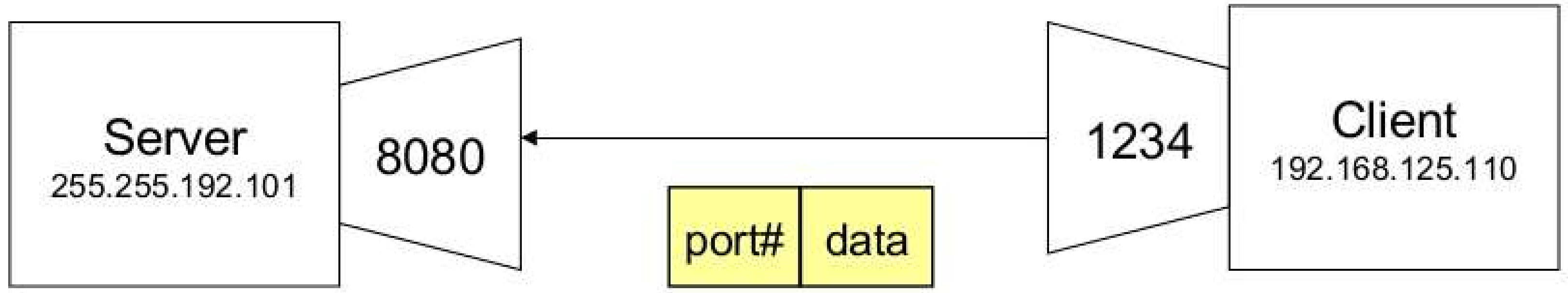
- Port number is a two bytes unsigned number ranging from 0-65535.
- Some ports have been reserved to support common/well known services:

ftp	21/tcp
telnet	23/tcp
smtp	25/tcp
http	80/tcp

- Developer defined processes/services are advised to use port numbers >= 1024 because port numbers <1024 are reserved for special services.

Sockets

Server and Client have network end points called sockets. Sockets are bound to a specific port.



Socket = IP+ Port

Server Socket = 255.255.192.101:8080

Client Socket = 192.168.125.110:1234

Java Socket Classes

- Java provides socket classes to make Server and Client.
- Package `java.net` contains socket classes.
- There are separate classes for TCP and UDP connections.

Make TCP Connection

- Following classes are used in Java to make TCP connection.
  - java.net.Socket – for client implementation
  - java.net.ServerSocket – for server implementation

Make UDP Connection

- Following classes are used in Java to make UDP connection.
  - java.net.DatagramSocket – for client and server implementation
  - java.net.DatagramPacket – for making data packets

Sockets IO

- Sockets provide an interface for programming networks at the transport layer.
- Network communication using Sockets is very much similar to performing file I/O
  - The streams used in file I/O operation are also applicable to socket-based I/O
- Socket-based communication is programming language independent.
  - That means, a socket program written in Java language can also communicate to a program written in Java or non-Java socket program.

SunilOS

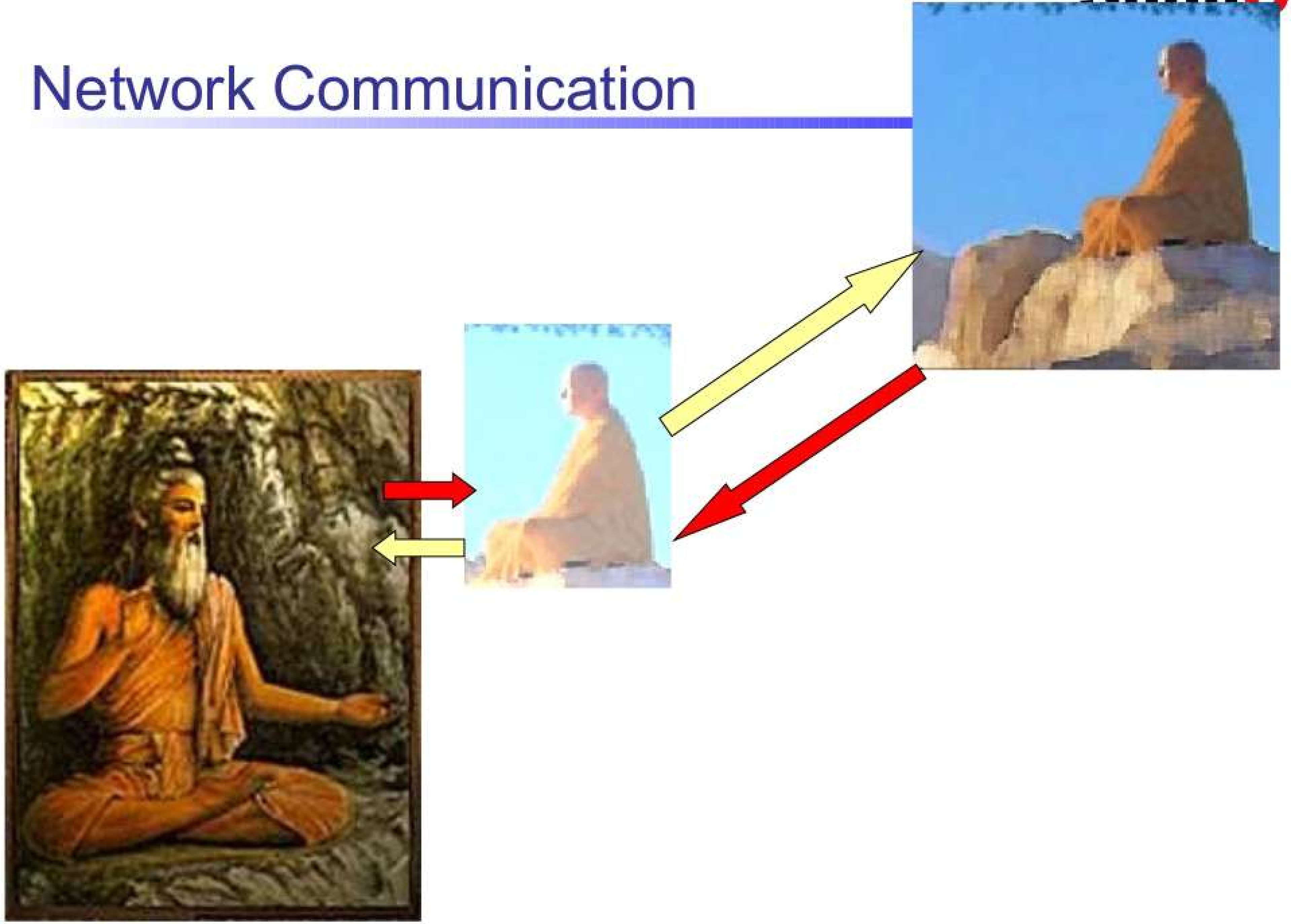
Read and Write from Socket

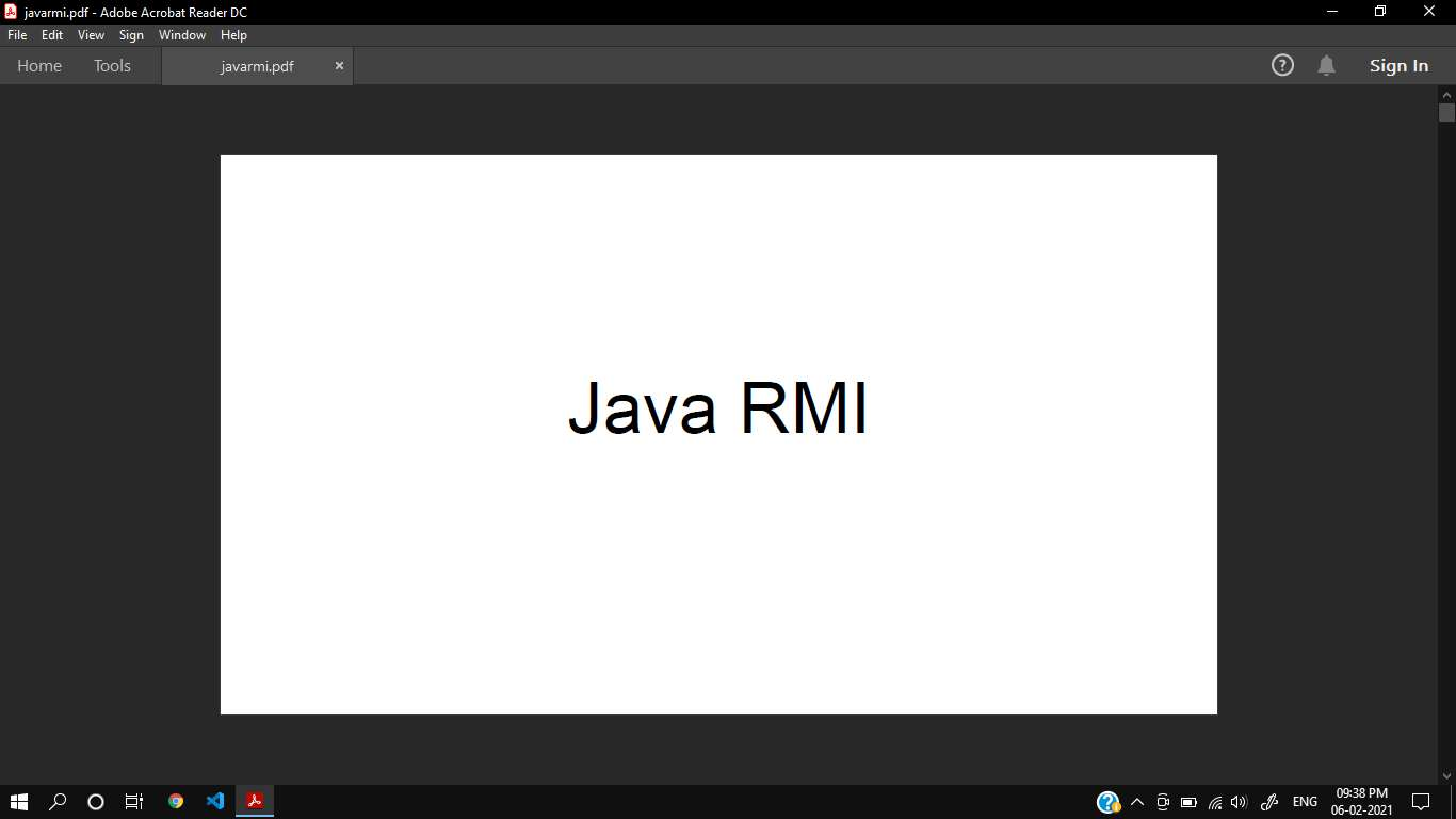
- IO classes are used to read and write byte streams from the Socket.
- `Socket s=new Socket(“127.0.0.1”,4444);//Client Side`
- `Or`
- `Socket s=ss.accept();//Server Side`

- `DataOutputStream os;`
- `DataInputStream is;`
- `is=new DataInputStream(`**`s.getInputStream()`**`);`
- `os=new DataOutputStream(`**`s.getOutputStream()`**`);`

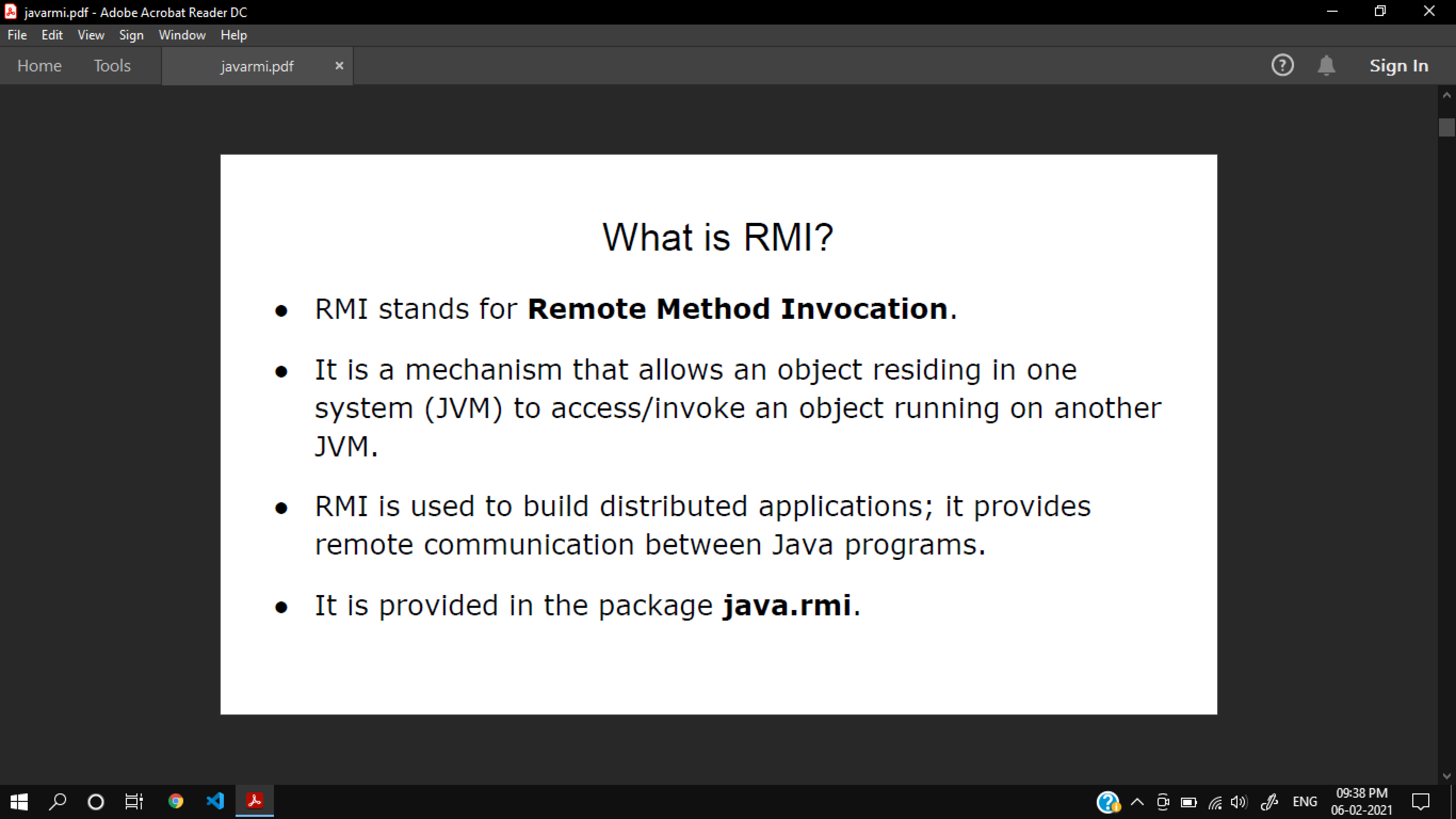
- `String line = `**`is.readLine()`**`; //Read from Socket`
- **`os.writeBytes("Hello\n");`**`  //Write to Socket`

Network Communication



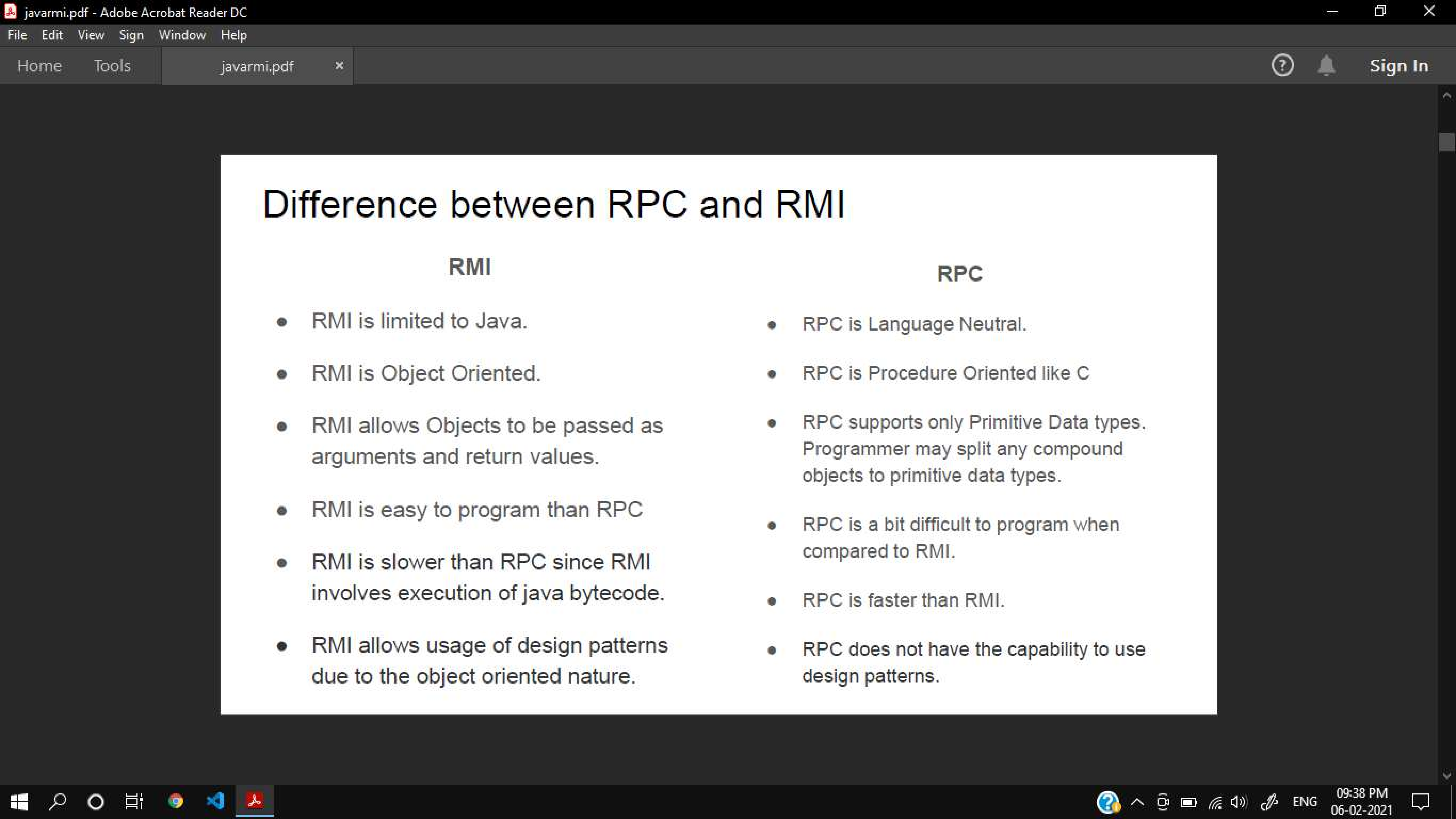

javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG
09:38 PM
06-02-2021

Java RMI


javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG
09:38 PM
06-02-2021

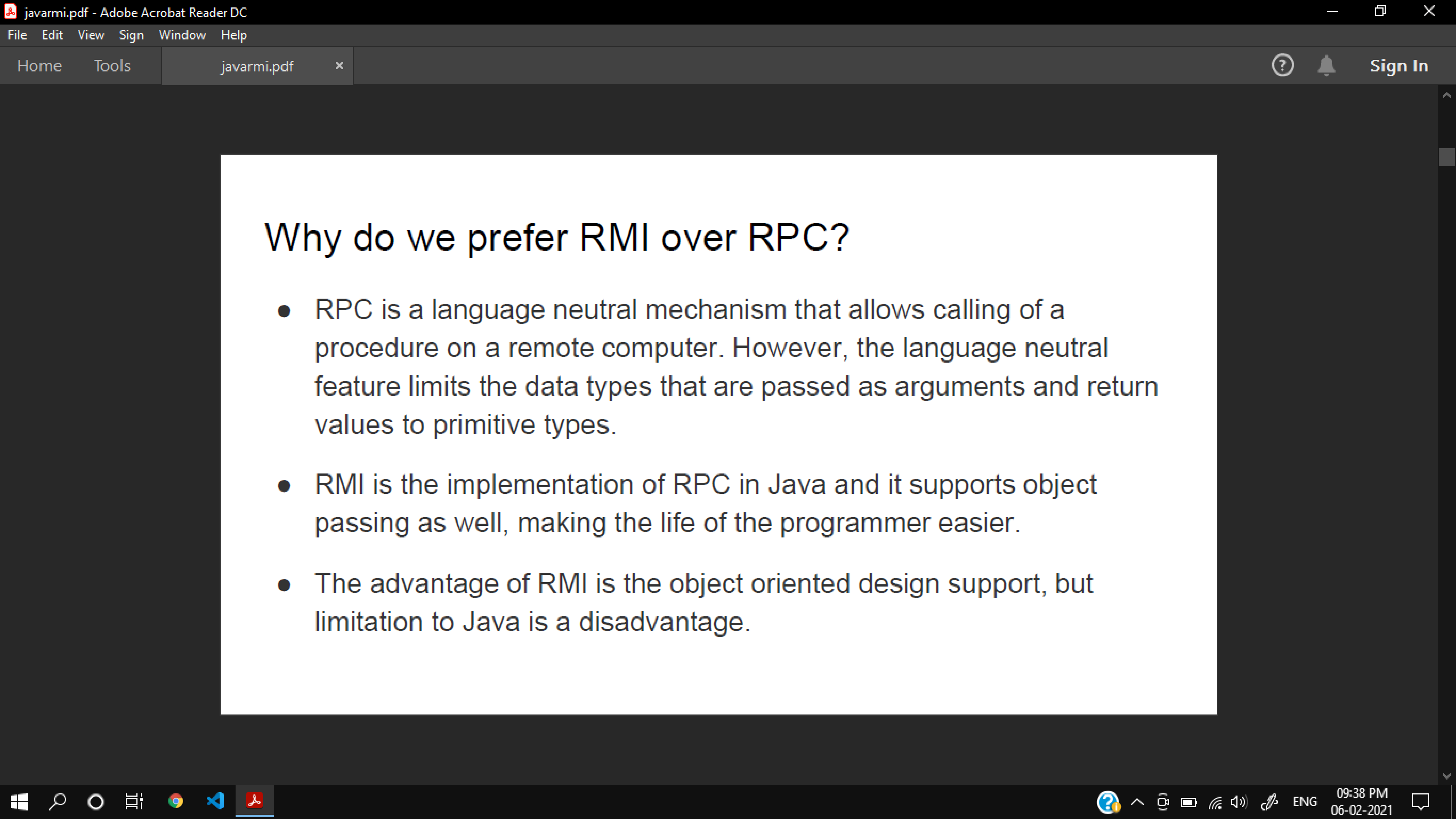
What is RMI?

- RMI stands for **Remote Method Invocation**.
- It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.
- RMI is used to build distributed applications; it provides remote communication between Java programs.
- It is provided in the package **java.rmi**.


javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG 09:38 PM 06-02-2021

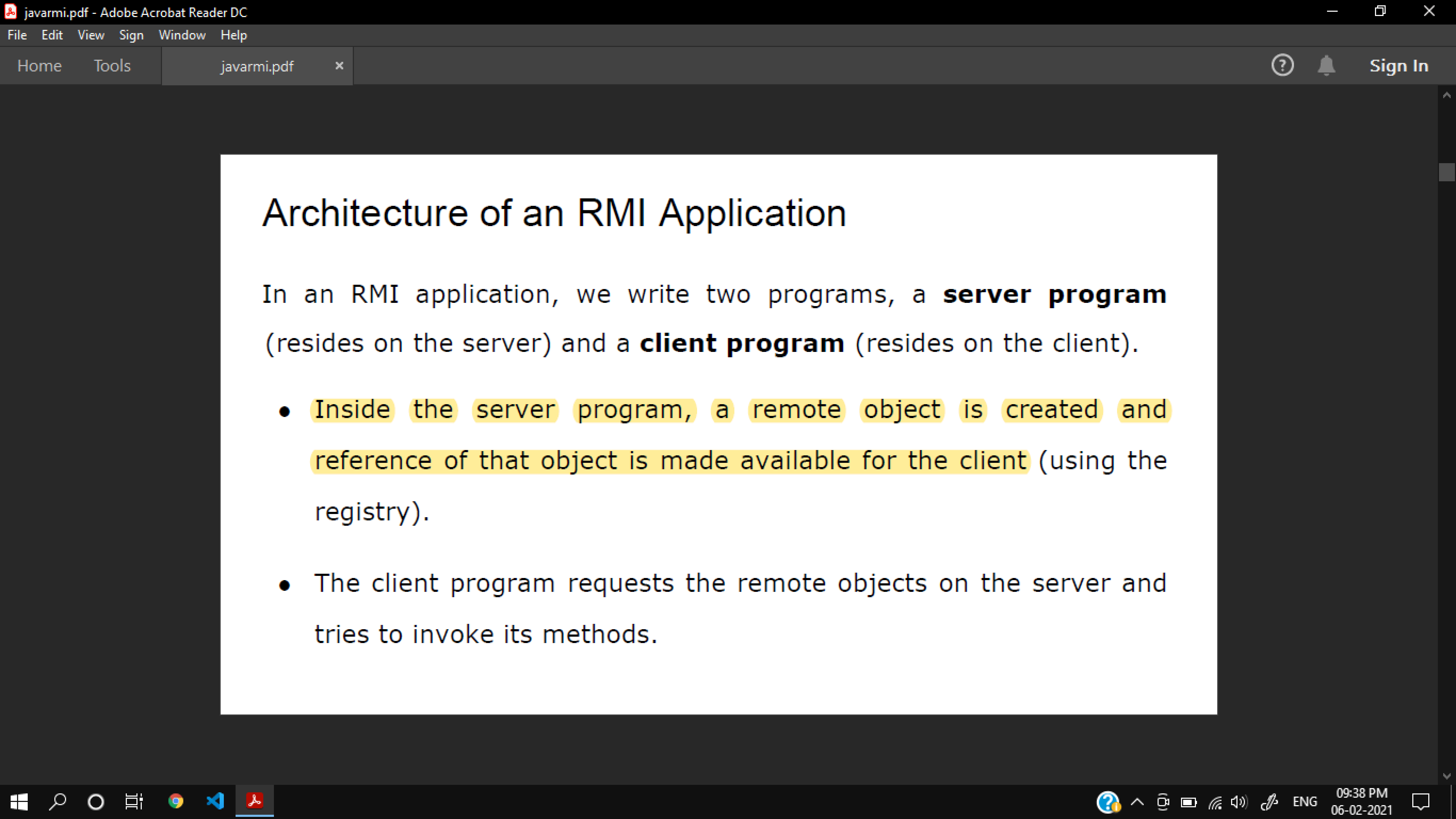
Difference between RPC and RMI

RMI	RPC
RMI is limited to Java.	RPC is Language Neutral.
RMI is Object Oriented.	RPC is Procedure Oriented like C
RMI allows Objects to be passed as arguments and return values.	RPC supports only Primitive Data types. Programmer may split any compound objects to primitive data types.
RMI is easy to program than RPC	RPC is a bit difficult to program when compared to RMI.
RMI is slower than RPC since RMI involves execution of java bytecode.	RPC is faster than RMI.
RMI allows usage of design patterns due to the object oriented nature.	RPC does not have the capability to use design patterns.


javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG
09:38 PM
06-02-2021

Why do we prefer RMI over RPC?

- RPC is a language neutral mechanism that allows calling of a procedure on a remote computer. However, the language neutral feature limits the data types that are passed as arguments and return values to primitive types.
- RMI is the implementation of RPC in Java and it supports object passing as well, making the life of the programmer easier.
- The advantage of RMI is the object oriented design support, but limitation to Java is a disadvantage.

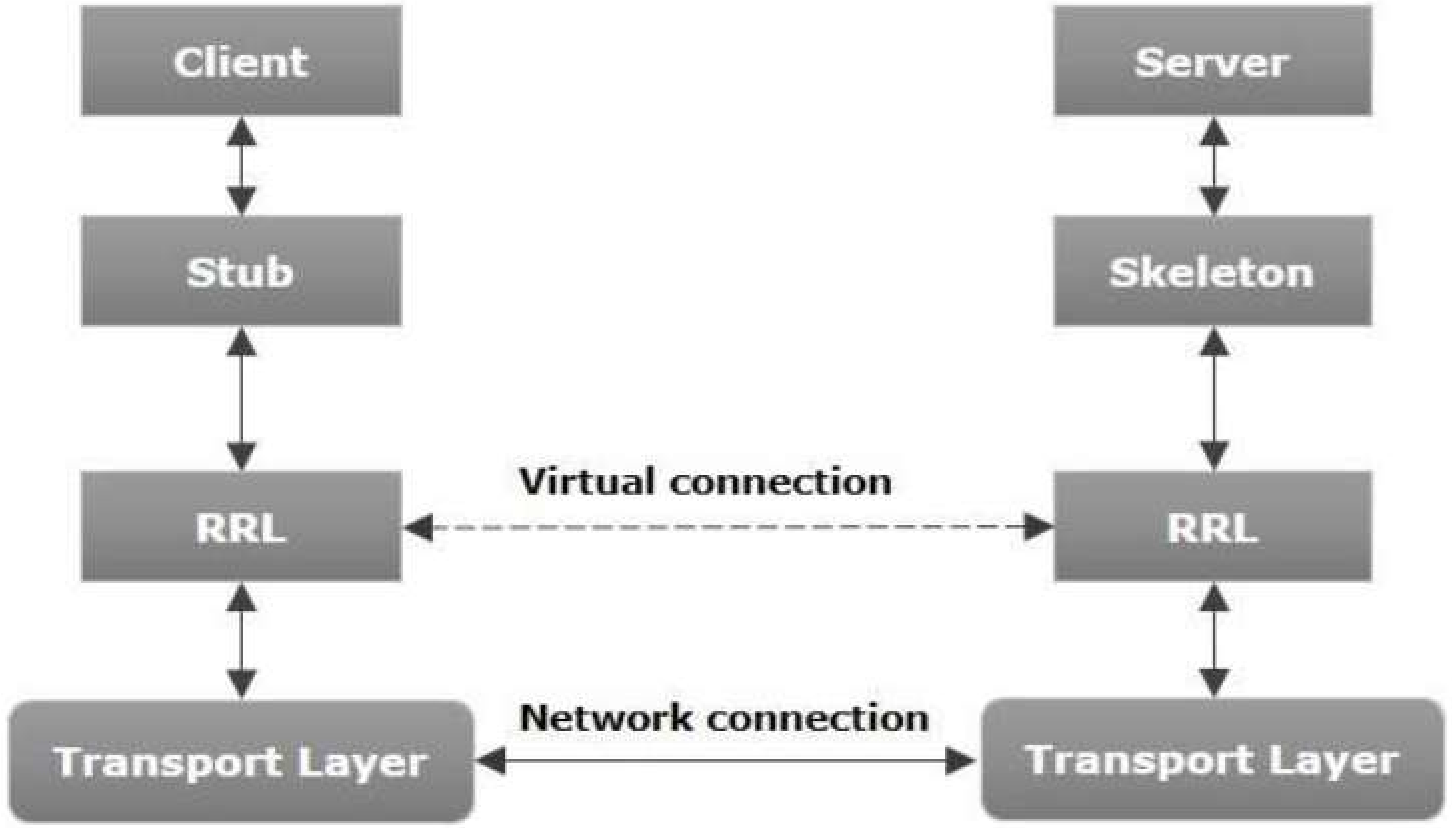

javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG
09:38 PM
06-02-2021

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



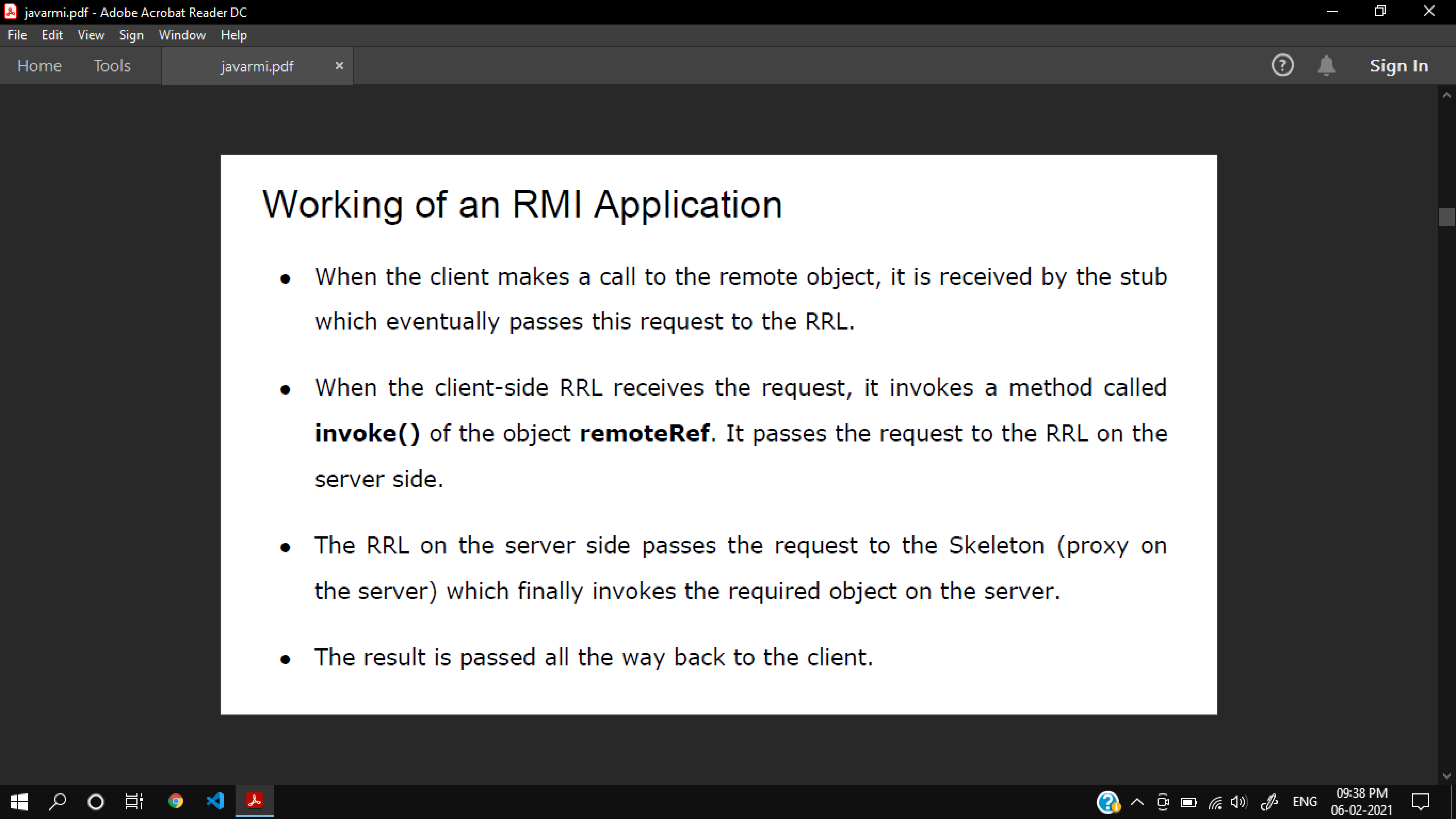
Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.

- **Stub** – A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

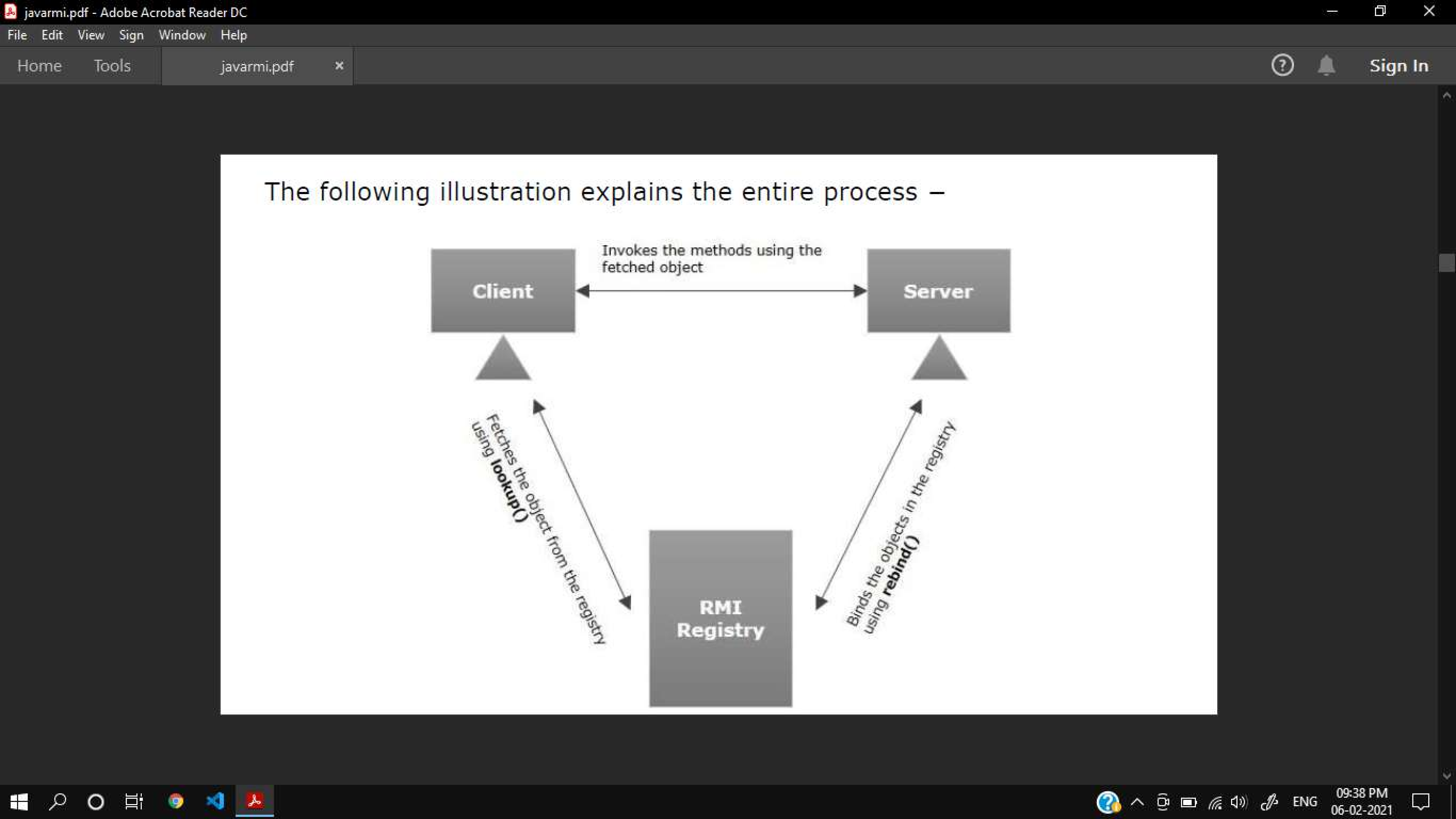
- **Skeleton** – This is the object which resides on the server side. **Stub** communicates with this skeleton to pass request to the remote object.

- **RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

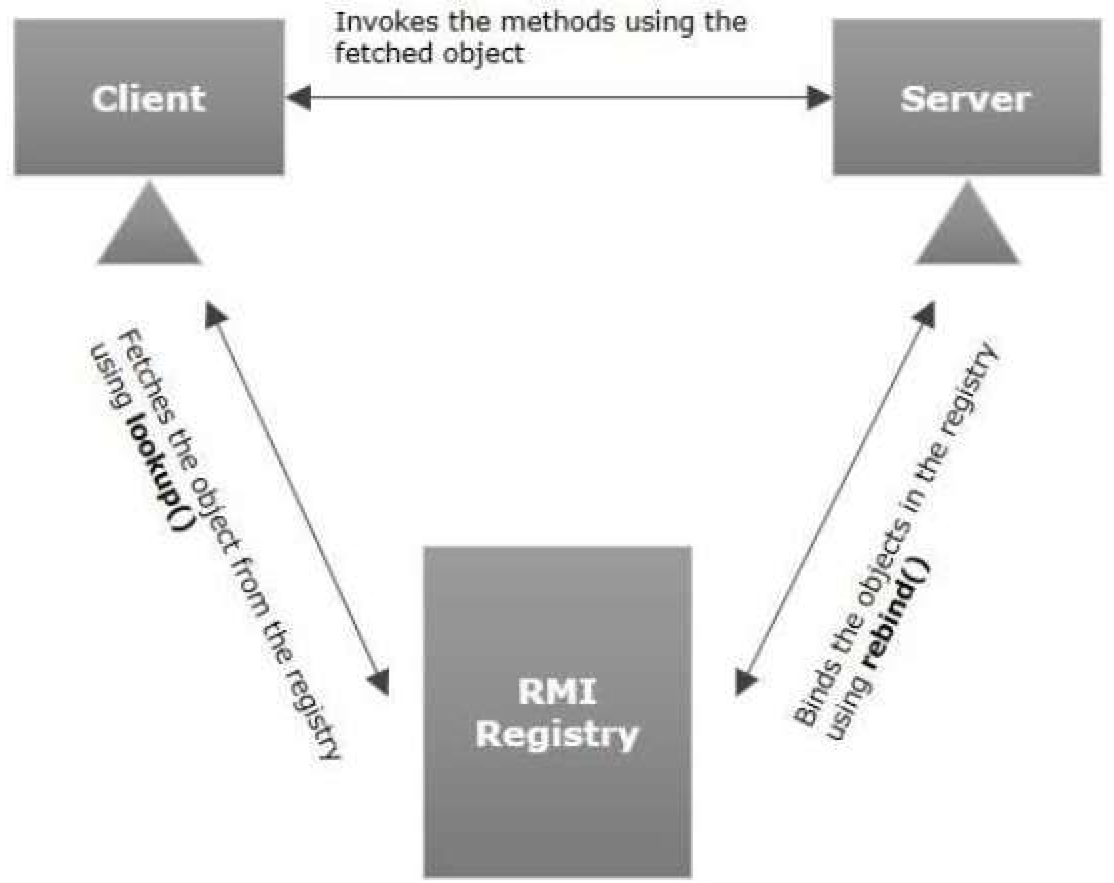

javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG 09:38 PM 06-02-2021

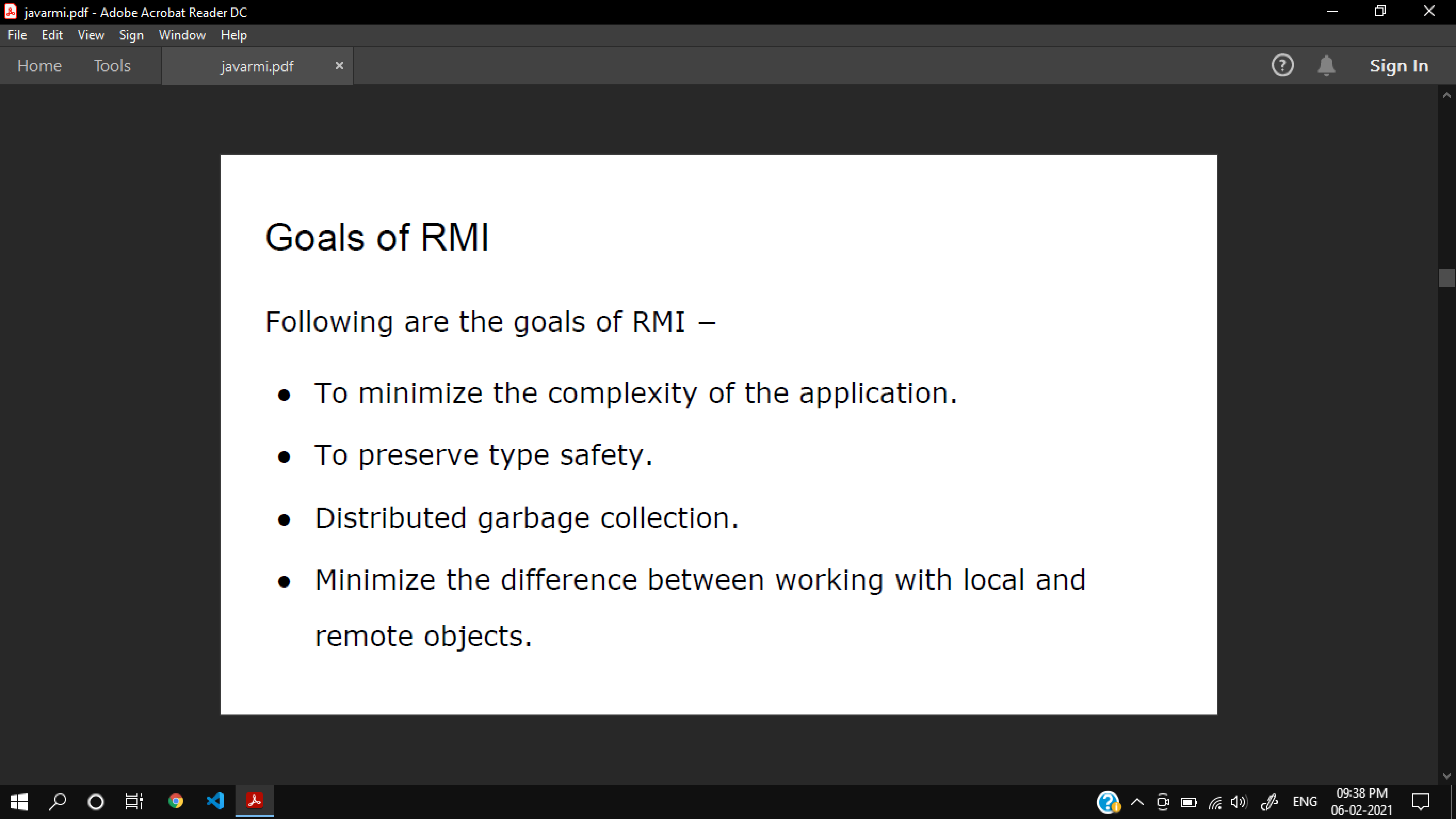
Working of an RMI Application

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.


javarmi.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools javarmi.pdf
Sign In
ENG
09:38 PM
06-02-2021

The following illustration explains the entire process –


Client
Server
RMI Registry
Invokes the methods using the fetched object
Fetches the object from the registry using **lookup()**
Binds the objects in the registry using **rebind()**



Goals of RMI

Following are the goals of RMI –

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.