INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

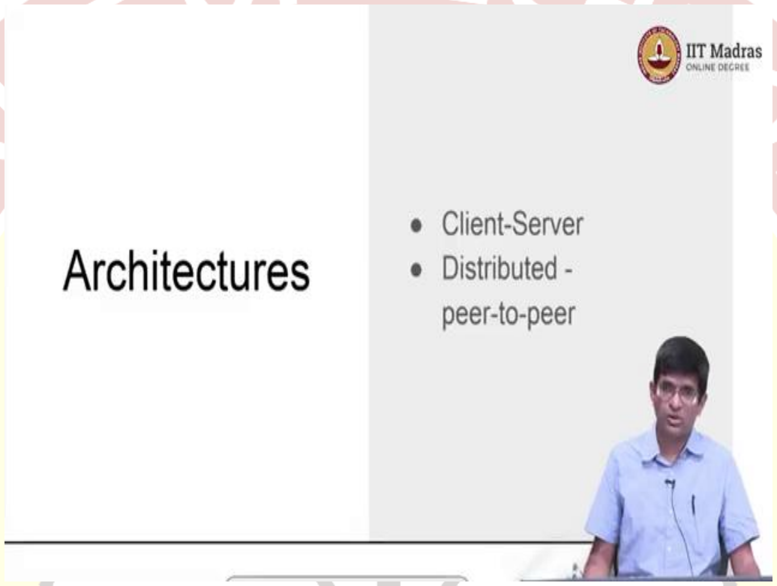
IIT Madras

ONLINE DEGREE

Modern Application Development - I
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Client-Server and Peer-to-Peer Architecture

Hello everyone and welcome to this course on Modern Application Development.

(Refer Slide Time: 0:16)



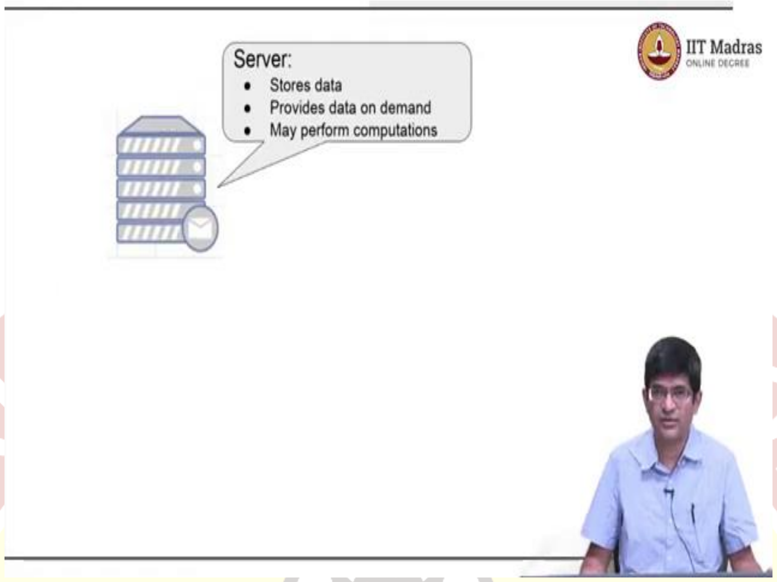
Architectures

- Client-Server
- Distributed - peer-to-peer

So, now that we have sort of decided that our platform is going to be the web, we also need to think a little bit about architectures and what I mean by architectures is how are devices connected to each other and what I mean by devices are you have one device which is going to be the actual phone or laptop or tablet that you use as the user. At the other end, there is a system that you are trying to connect to that actually runs the app that you are interested in.

Now, two of the most sort of well-known or popular paradigms again in this context or to architectures that you need to at least be familiar with roughly are the so called client server architecture and the distributed or peer to peer architecture.

(Refer Slide Time: 1:11)

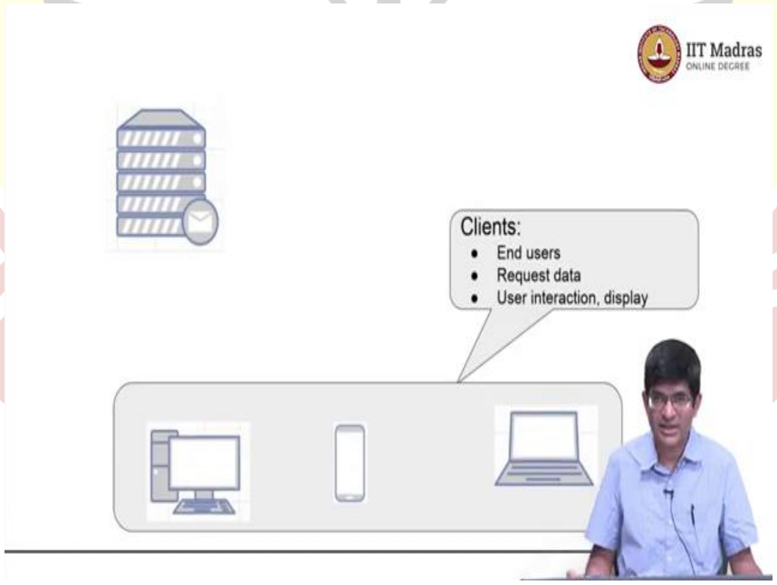


Server:

- Stores data
- Provides data on demand
- May perform computations

So, let us try and understand roughly what a client server architecture looks like. First we have a server whose job is basically to store data, so what it does is it provides data on demand, it may perform certain computations but its primary function is to store that data and respond to requests as and when they come in.

(Refer Slide Time: 1:36)

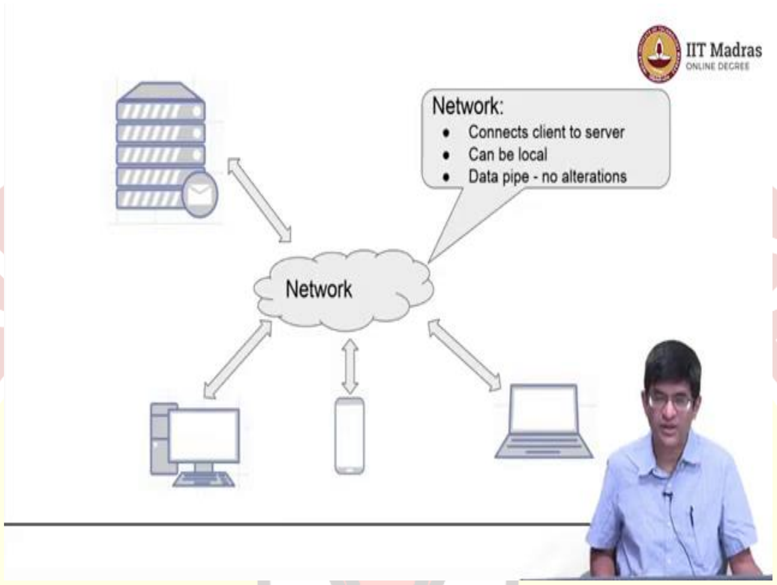


Clients:

- End users
- Request data
- User interaction, display

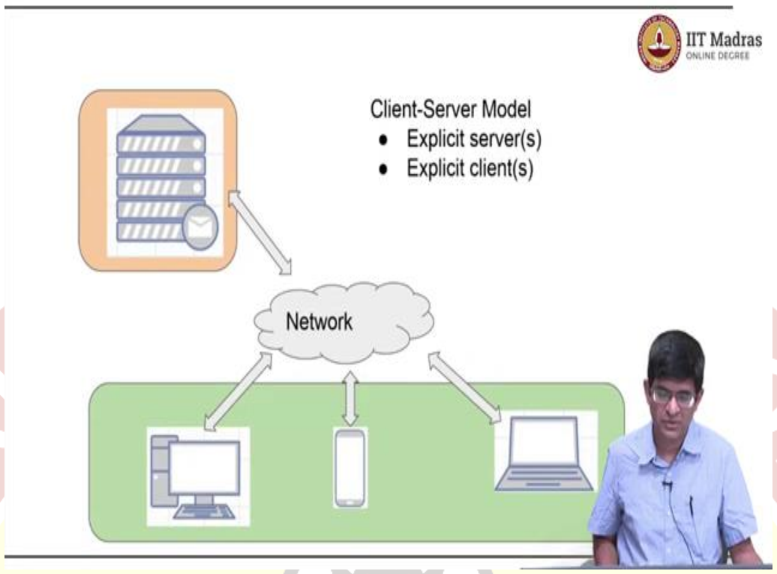
Now, you have a bunch of clients, these are end users, each of these end users is going to request data and what kind of requests are those they will have some kind of user interaction, they might want to display something on the screen, based on what they see on the screen they may ask for something else to be displayed.

(Refer Slide Time: 1:56)



And you have a network, the network's function is to connect the clients to the server. So, how does that happen? Basically the network sort of acts just like a data pipe, any request or any information that a client puts into the network is supposed to just get piped through to the server without really being modified in any way. It is the server that decides what to do with that information, and once the server responds that information should come back reliably to the client.

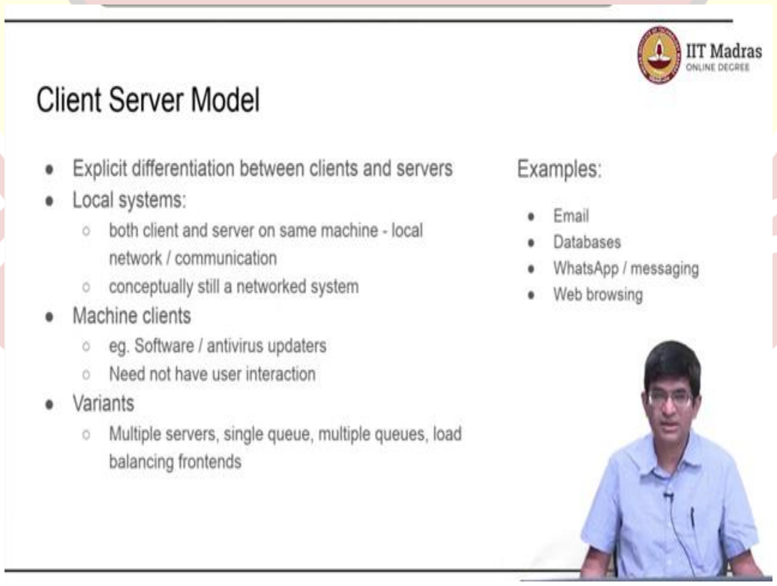
(Refer Slide Time: 2:35)



Client-Server Model
- Explicit server(s)
- Explicit client(s)

So, once we have these three components, there is a server a client and a network that connects the two together, we can have the full-blown client server model. And over here we have explicit servers, what I mean by explicit servers is there is a very well defined notion of what is a server on this, in this application, and similarly there are clients, they could be the end users. And there is some kind of a network that connects these two together.

(Refer Slide Time: 2:59)


IIT Madras
ONLINE DEGREE

Client Server Model

- Explicit differentiation between clients and servers
- Local systems:
  - both client and server on same machine - local network / communication
  - conceptually still a networked system
- Machine clients
  - eg. Software / antivirus updaters
  - Need not have user interaction
- Variants
  - Multiple servers, single queue, multiple queues, load balancing frontends

Examples:
- Email
- Databases
- WhatsApp / messaging
- Web browsing

So, to summarize, the client server model essentially has explicit differentiation between the clients and the servers. So, you clearly know at any given point which is the client, which is the server, you could potentially have both of these on the same system. So, there are client server applications that run entirely within your PC for example. So, the PC, one part of the operating system might act as a server, another part might act as a client which basically asks for something to be done.

So, even though I say that the network is a crucial part of this, when both are running on the same system where is the network involved? The point is that conceptually the same idea is used, we still think of it as a networked system, we still think that there is a server, there is a client and that separation of functionality helps to make it easier to go between the two.

Now, in many cases the client might not even be a user, the client might be another machine. An example would be, most of you might be running some kind of antivirus software and the antivirus software is probably updating in the background. So, every day or so it needs to connect to the servers and it needs to see are there any updates on the virus databases and if so, download them.

Now, the antivirus software is acting like a client on your behalf, it connects to the server, it pulls down data and it updates itself. Is there any user interaction involved? No. So, this is a case where you have for example a machine client. Now, examples of client server model email is probably the easiest to think off, you definitely need a server that stores your email and you have the clients which are actually used to display, read, delete and so on.

Databases are very often implemented as client server applications. You can think of it easily as there is a database server and there are a whole bunch of other client applications that are going to connect to it and read data from it. Now, in that case remember the client application is sort of working on your behalf, it is not that you are directly connecting to the database, you may not even know what database is being used, but there is part of the application that works on your behalf, connects to the server, pulls down the data and shows it to you.

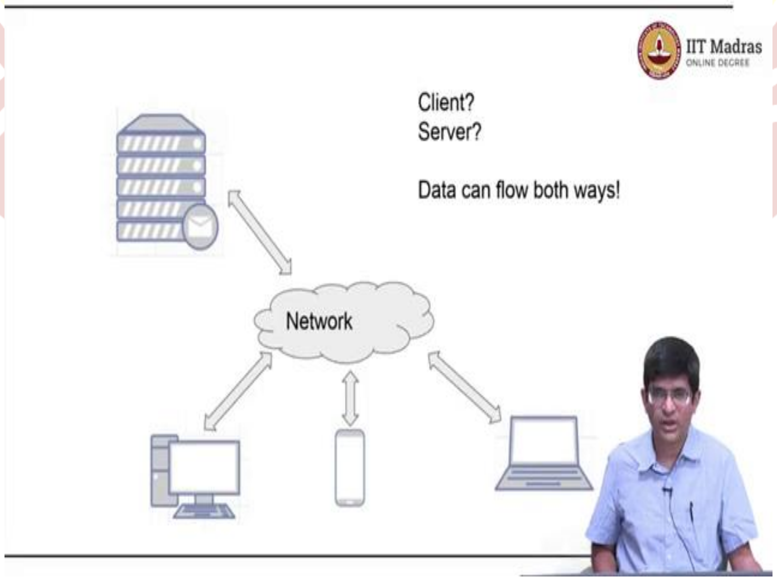
Now, even something like WhatsApp or any messaging service ultimately is a client server kind of application, there are servers, WhatsApp has servers that collect messages from individual users, see who are the target recipients and then sends it to them. So, anytime you send a WhatsApp message to your friends it is going through the WhatsApp servers, it first goes to the server, the server then connects to each of the other clients and in this case it has a way by which it can push information.

So, the other clients may not even be sort of continuously requesting the server, do I have new messages, do I have new messages. The server has some way by which it can push messages to them. Web browsing is sort of, at some level the simplest form of client server that we can think of, after all every web server just based on the name itself is a server. And in this case the client pretty much is you through your browser, you type in URL on your browser, it connects to the server, pulls down the data and displays it for you.

Now, having said all of this there are many variants on the client server model, you might find things where there are a single queue which is then split up among multiple servers, there might be things where there are multiple servers involved in a single application each one doing a different task, one might be a database, one might be a web server, one might be something for a key value cache storage.

You could have some kind of load balancing in the front of the servers, so all that I really want to convey over here is the underlying concept is that there is one or a limited set of servers whose task is actually storage and computation and there are clients whose job is to basically make the requests and process the final data that comes back to them, usually for the purpose of the end user interaction.

(Refer Slide Time: 7:18)

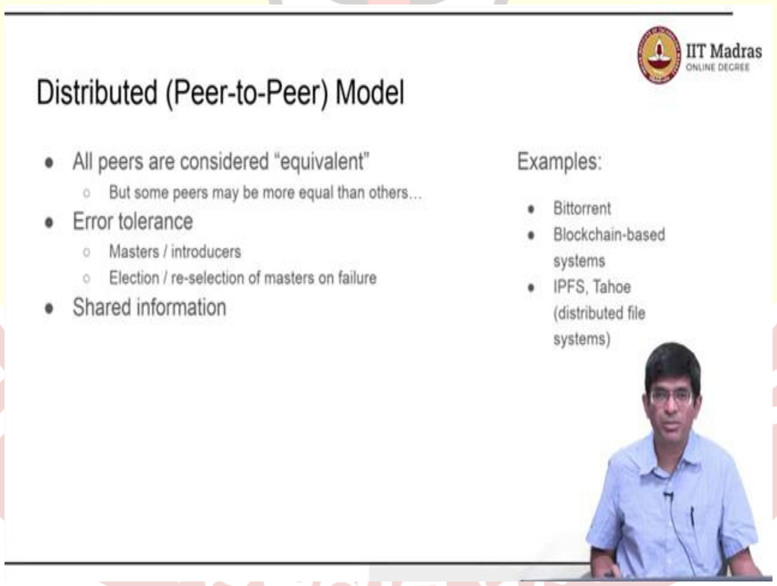


So, now what is the alternative? Let us look at a different kind of application with exactly the same structure, I still have one big server machine out there, a PC, a laptop or mobile phone and the network. But I am saying that I no longer now know which one is the client and which one is the server, data can flow in both ways and pretty much could be anything that you want. Where would we have something of this sort?

The simplest example that many of you may be familiar with is the so called BitTorrent. Now, what does BitTorrent do? It basically allows you to download files but you are not just downloading files, every time you are running BitTorrent on your PC, you are also in some sense acting as a server to others.

So, anytime a file is downloaded onto your PC other people could then start looking for those files and they would be able to download it from your machine. So, effectively if you ask the question, which one is the client and which one is the server over here, that is no longer a question with a clear answer, it is a distributed network.

(Refer Slide Time: 8:31)



Distributed (Peer-to-Peer) Model

- All peers are considered "equivalent"
  - But some peers may be more equal than others...
- Error tolerance
  - Masters / introducers
  - Election / re-selection of masters on failure
- Shared information

Examples:

- Bittorrent
- Blockchain-based systems
- IPFS, Tahoe (distributed file systems)

IIT Madras
ONLINE DEGREE

So, this is the so called peer to peer model, because all nodes on this network is it a laptop, a desktop, a mobile phone, a server, they are all sort of equivalent in some sense, they are all peers. And the peer to peer model is one where all these peers are considered equivalent. Of course some peers may be more equal than others and what I mean by that is servers are probably given higher weightage, servers with high bandwidth for example would be the ones that are primarily used for actually downloading the data.

Now, the good thing about this peer to peer network architecture is that you need to have some kind of notion of a master or an introducer but that is about it, you do not really have something which is the central controller of the network, you need some way by which a new node entering the network can get introduced to others.

But once you have that it is remarkably tolerant to failures. What if the big server in the data center crashes? It does not matter, the rest of the machines could now basically re-elect a new master whose job is only after all to introduce other nodes into the network and then say ok let us carry on I still have the files, you can download it from here, you do not need to go to the server in the data center.

Which basically means that these are all sort of sharing information among each other. The point is this has limited areas where you can apply it, I mean email for example is not a good idea to do in a peer-to-peer fashion. I mean you do not really want your emails to be sitting on somebody else's machine. You probably decided that you trust the Google mail servers at some point but you do not really want your email to be sitting on your competitor’s machine if you are in a business environment.

So, peer to peer has specific applications. Examples are, like I said BitTorrent, Blockchain based systems are another buzzword of the day, things you keep hearing about Blockchain this Blockchain that, they are also distributed systems, they rely on the fact that there are sufficient number of nodes in the network and based on that a certain kind of trust is built up.

And there are various kinds of distributed file systems IPFS and Tahoe are examples of that I am not going to get into the details, the whole point over here is that now your data could actually be stored across multiple devices. The point is it is still in some encrypted format. So, if there is a file that you did not want others to share, the data might still be on somebody else's machine but it is not that they can actually read it and understand what is going on, it is still encrypted in some form so you could potentially say that ok only those who are allowed to access it, can actually see it.