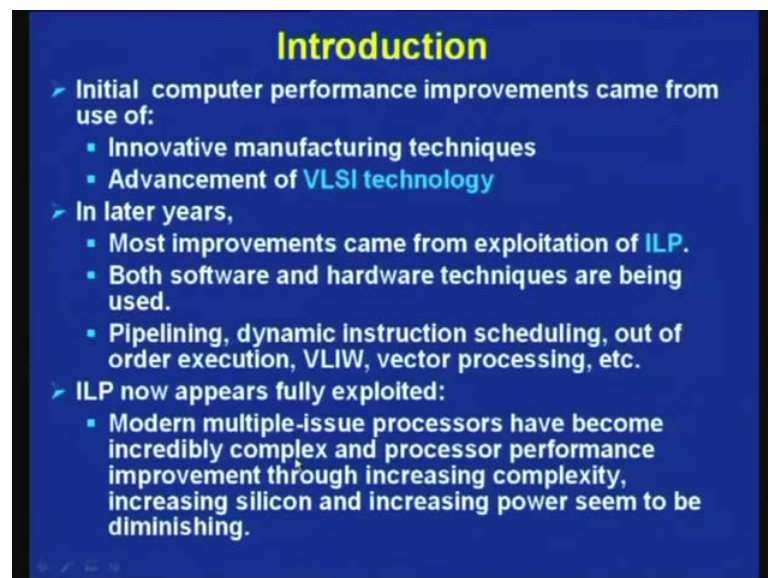


High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 37
Multithreading and Multiprocessing

Hello and welcome to today's lecture on Multi Threading and Multi Processing, here we shall discuss about thread level parallelism, and we shall also see how multiple processors can be used, for multi threading, multiple thread processing and multi processing.

(Refer Slide Time: 00:54)



We have seen that initial computer performance improvements came from the use of innovative manufacturing techniques and advancement of VLSI technology. The computers if you go to the early history of computers, you will see many innovative technology we have incorporated to improve the performance. Then of course, when the VLSI technology was available, the advancement of VLSI technology has reduce the size of the devices and increase the speed of processing.

So, as a consequence computer performance improvement took place based on these two few techniques, innovative manufacturing techniques and advancement of VLSI technology. Subsequently in later years most improvements came from exploitation of instruction level parallelism. And we has discuss instructional level parallelism in details,

and we have seen how hardware and software can be used to achieving instruction level parallelism.

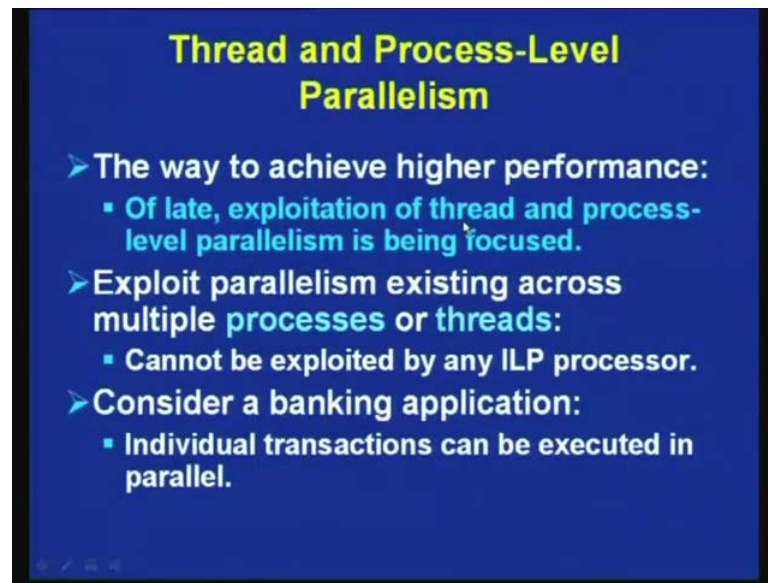
We have discussed about pipelining, dynamic instructions scheduling, out of order execution, we have discussed about VLIW and vector processing I shall discuss little bit later on. And these are essentially utilizes the instruction level parallelism available, and we have seen how different processors have been implemented using this instruction level parallelism.

Particularly, in the last three lectures we have discussed Intel series of processors, and how the instruction level parallelism have been incorporated starting with pipelining and different processors and leading to performance improvement. But, it has now been recognize that instruction level parallelism is now fully exploited; that means, whatever parallelism is possible, whatever performance gain is possible by exploiting instruction level parallelism is already done.

And modern multiple processors have become incredibly complex, we have seen that those super scalar processors for the multiple issues of I mean issue of multiple instructions are done. They require very large silicon real estate and they are very complex and processor performance to increasing complexity, increasing silicon and increasing power seem to be diminishing; that means, we have reached more or less a point of diminishing return.

That means, even after providing lot of hardware or using more sophisticated software, the performance gain that is achieved now utilizing or exploiting instruction level parallelism is minimal very small.

(Refer Slide Time: 04:12)



So, what to do, what is the way out, way out is we have to look for something else, so though the way to achieve higher performance of late by exploitation of thread and process level parallelism is being focused. So, that means, exploit parallelism existing across multiple processors and threads, so now, we are doing it at looking at parallelism at little higher level not at the instruction level, but at process level and thread level.

And it has been founded, this type of parallelism what you are is achieved by this cannot be done by instruction level parallelism, so for example, if you consider a banking applications. Now, it is you know centralize banking has become very popular, so there are many users which are which are communicating to a bank through internet, so and each user is a is a accessing the banks and doing transactions.

So, multiple transactions are taking place concurrently and it is done by different users, so individual transactions, which are taking place can be executed in parallel. So, if we if we consider now a day's core banking or centralize banking system, where all the transactions which are done initiated by multiple users can be executed in parallel; that means, here the parallelism is more or less inherent or built in the application itself.

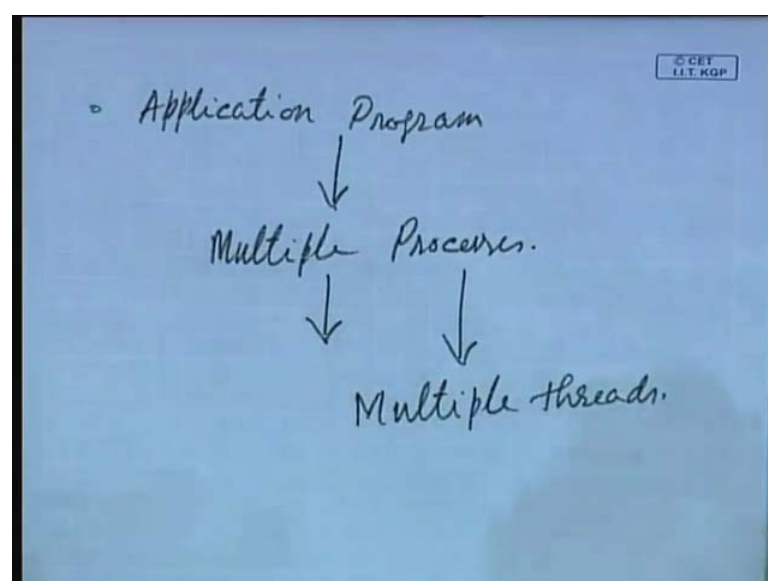
(Refer Slide Time: 06:00)

Processes Versus Threads

- **Processes:**
 - A process is a **program in execution**.
 - An application normally consists of multiple processes.
- **Threads:**
 - A process consists of **one or more threads**.
 - Threads belonging to the same process share **data**, and **code space**.

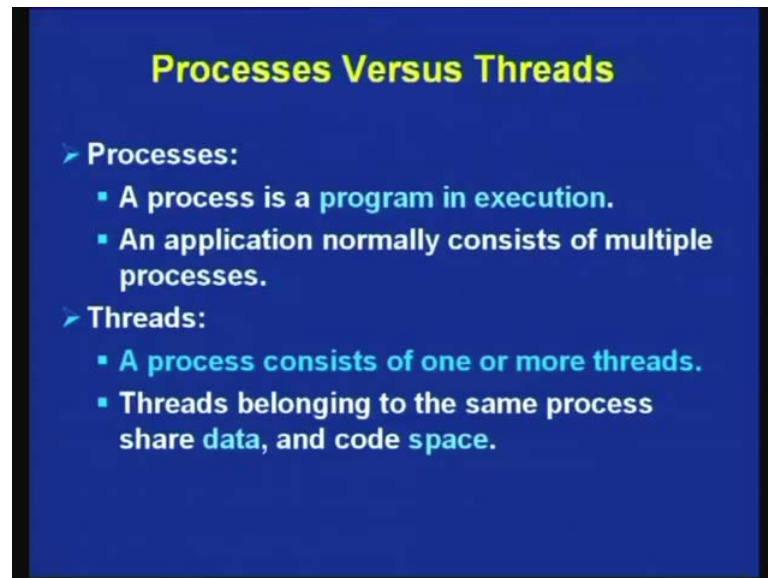
Now, let us consider the difference between process and thread, as we know a process is a program in execution, and as you are running different programs for different applications. A like what processing and various other applications and each application whenever a program is running, we call it a process, and as we you an application normally consists of multiple processors. So, normally a single application is broken down in to multiple processors, and then multiple processors can run concurrently, even for a single application. on the other hand a three a processor consists of one or more threads.

(Refer Slide Time: 07:02)



So, here we can say an application program can lead to multiple processors, a single application will give you multiple processors, and each process and in turn can give you multiple threads. So, this is the relationship between process and thread; that means, a process can be an application can be broken down, in multiple processors and a process can be broken down in to multiple threads.

(Refer Slide Time: 07:56)

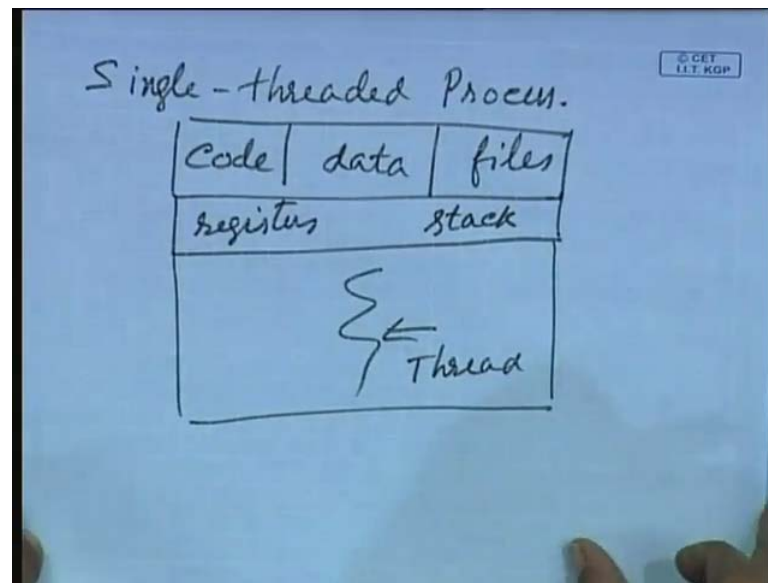


Processes Versus Threads

- **Processes:**
 - A process is a **program in execution**.
 - An application normally consists of multiple processes.
- **Threads:**
 - A process consists of **one or more threads**.
 - Threads belonging to the same process share **data**, and **code space**.

So, now in what way do they differ a process and thread differs, threads belonging to the same process share data and code space, so let us have a look at the difference between a process and thread.

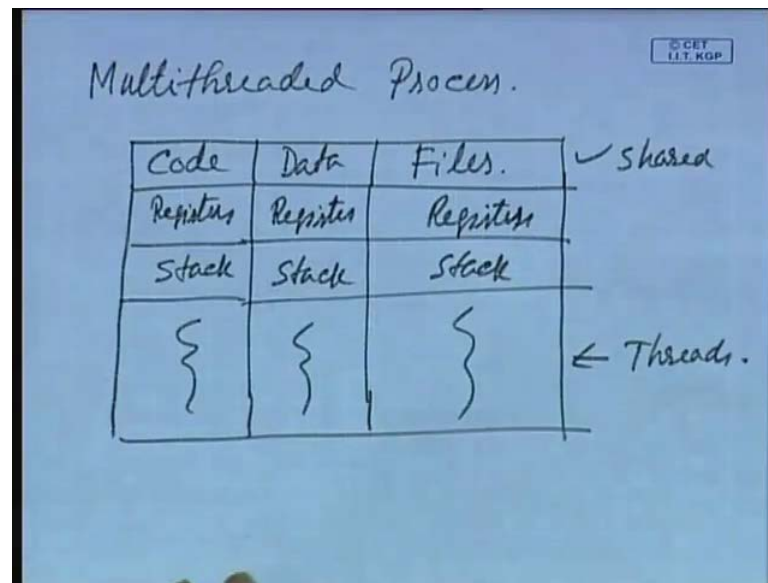
(Refer Slide Time: 08:14)



So, let us consider look at let us consider a single threaded process, in each case we will find that it will share, it will require a particular process will require code, data and files. So, you will require code, data and files, and then you will require to execute a particular program, you will require resistors to store the intermediate results, while processing a program.

You will require stack, whenever you do sub routine call, you have to store the you know that different various parameters on the stack, and also when you do context teaching you will require stack. So, you will require resistors and stack, and then you will be running a single thread like this a, so a thread is this is a single thread for a single process running, so this is a single threaded process.

(Refer Slide Time: 09:33)



Now, let us consider a multi threaded process, in a multi threaded process the code, data and files, these will be shared by multiple threads; however, you will require, suppose you are creating three threads, so you will require separate registers separate stack. So, you will require registers, stack, registers, stack, and then you have one thread here, this is another thread, this is another thread, this part is shared by all these threads. So, you can see here in a multi threaded, the you are sharing the same code space data and files; however, you will require separate set of separate registers, separate stack for different threads, so these are different threads.

(Refer Slide Time: 11:14)

How can Threads be Created?

- By using any of the popular thread libraries:
 - POSIX Pthreads
 - Win32 threads
 - Java threads, etc.

So, this is a multi threaded process, I will believe when your operating system course, we will learn about process threading more details, question naturally arises how do you create threads, how a thread can be created. Fortunately, by using a different I mean there are popular thread like very provided POSIX threads win 32 threads, java threads. So, they will facilitate you in implementing in creating threads, so with the help of this thread libraries you can create threads.

(Refer Slide Time: 11:43)



Now, the threads can be are two types, one is your user thread another is kernel thread, what is the difference user thread and kernel thread. So, first let us focus on user thread, in case of user thread the thread management is done in user space. We have already discussed about actual memory management, there we have seen the the kernel space users space we have mention about that.

And here whenever user threads are created the thread management is done in users space, and user threads are supported and manage without kernel support, and; obviously, since it is done without kernel support. This is in with this is user threads are invisible to the kernel, and if your thread gets block the entire process get blocked. So, and as a consequence, because of this limitation whenever one thread gets blocked and entire process gets block, it has got limited benefits, it provides you limited benefit of threading.

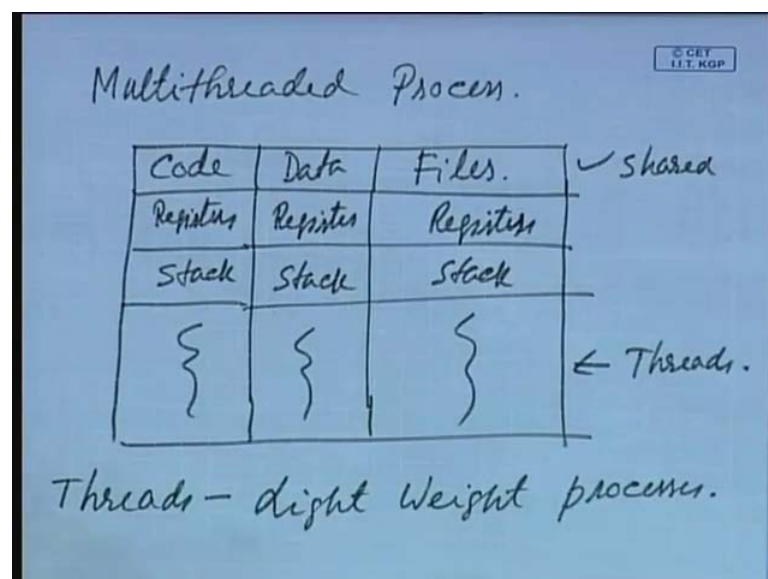
(Refer Slide Time: 12:54)

Kernel Threads

- Kernel threads supported and managed directly by the OS.
 - Kernel creates **Light Weight Processes (LWPs)**.
- Most modern OS support kernel threads:
 - Windows XP/2000
 - Solaris
 - Linux
 - Mac OS, etc.

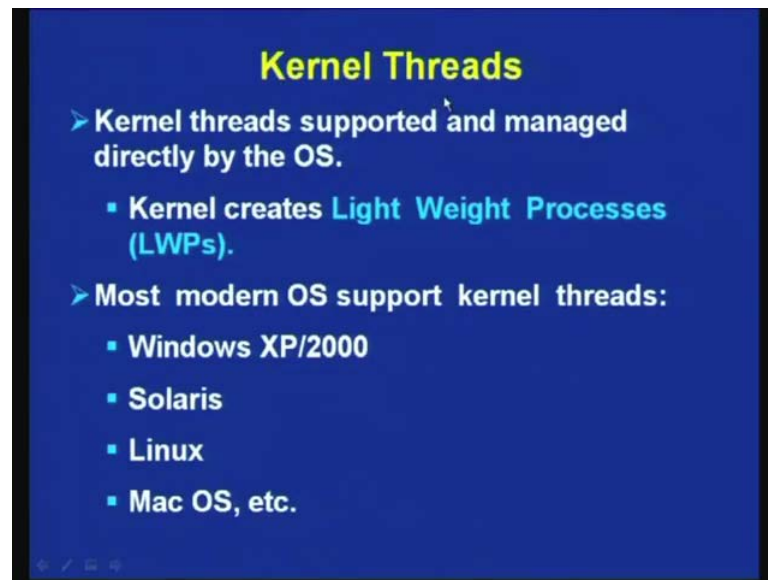
On the other hand whenever you go for kernel threads, kernel threads are supported and manage directly by the operating system. So, operating system at execute full control of kernel threads, and kernel creates light weight processes.

(Refer Slide Time: 13:14)



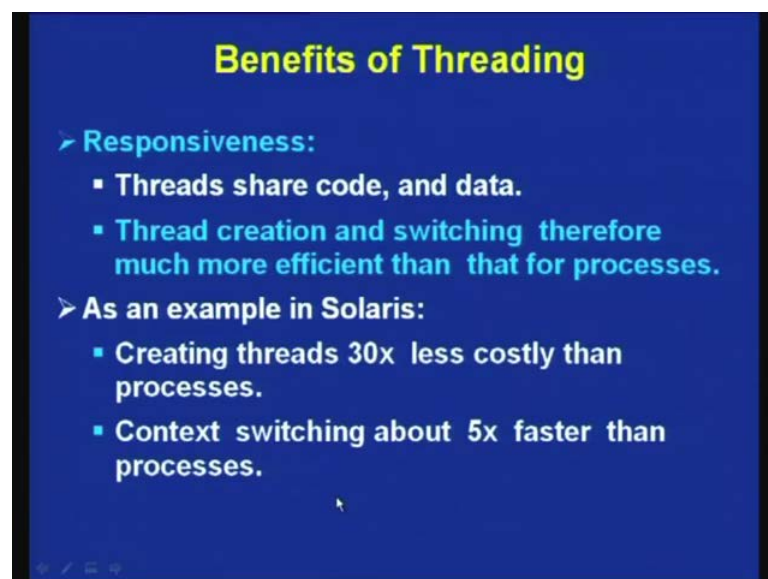
So, sometimes we called a threads as light weight processes.

(Refer Slide Time: 13:25)



And modern operating system support kernel thread, for example doing windows x p 2000, Solaris Linux Mac operating system etc, all this modern operating systems support this kernel threads.

(Refer Slide Time: 13:43)



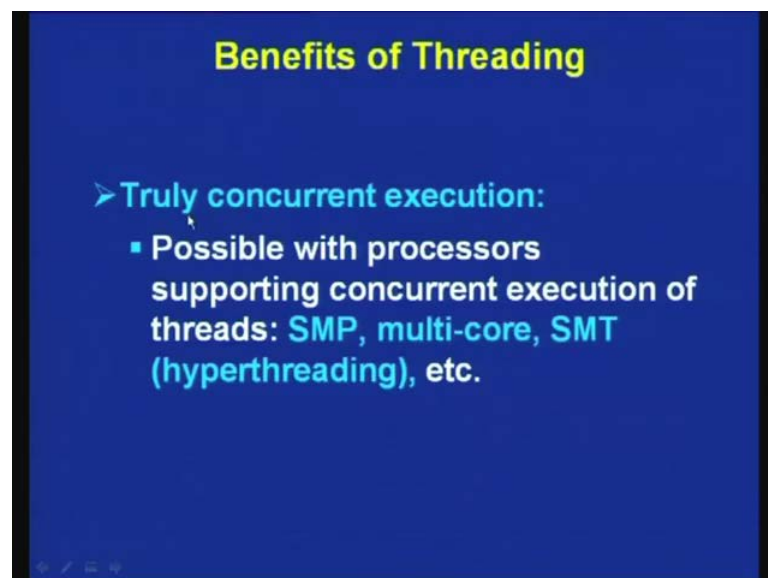
Now, let us have a look at, so we have now understood the differ, we have been defined what is thread, we have discussed about the relationship between process and thread. Now, let us have a look at the benefits of threading, what kind of benefit we get whenever we do the threading, first of all responsiveness. We have seen that threads

here, code and data, we have seen in this particular diagram that this code data this part is shared whenever we create threads.

So, thread creation and switching therefore, is much more efficient than that of processors, so whenever we switch from one process to another process. Suppose, let us consider a single processing system, so in the single processing system also you can have multiple process or multiple threads. So, whenever we switch from one process to another process, it will involve lot of over head, I mean the you have to store and de store various information. Particularly, since code data and files are separate, but since you are sharing the code, data and files, and for different threads the thread creation and switching is much more efficient.

So, as an example, if you look at the threads created by supported by Solaris operating system, it has been found that creating threads is 30 times less costly than processes, and frankly speaking is about 5 times faster than processes. That means, creation of thread is much more efficient switching of thread I mean one thread to another thread is also much more efficient. So, this tells you the when this gives you the benefit of thread multi threading compare to multi processing.

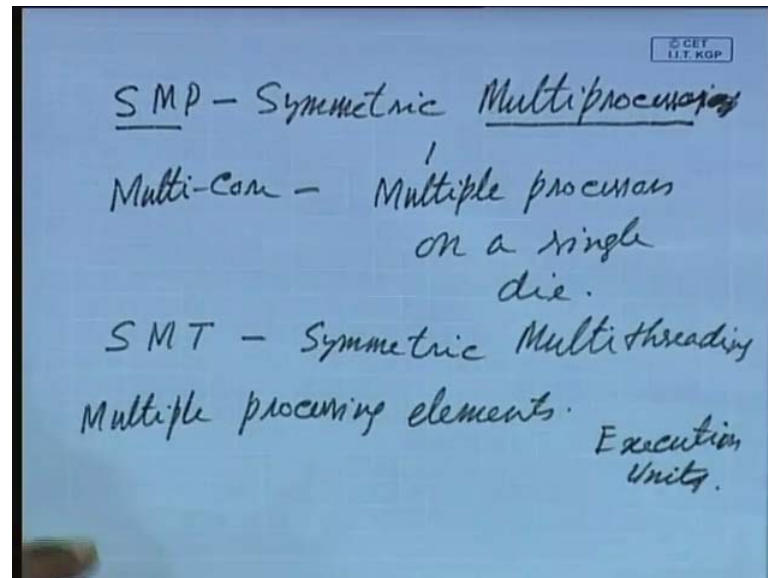
(Refer Slide Time: 16:12)



And it gives you truly concurrent execution and of course, this concurrent execution is possible, provided you have got enough support of the hardware; that means, if you want to do multiple threads to run concurrently. They are stood this comes support from the

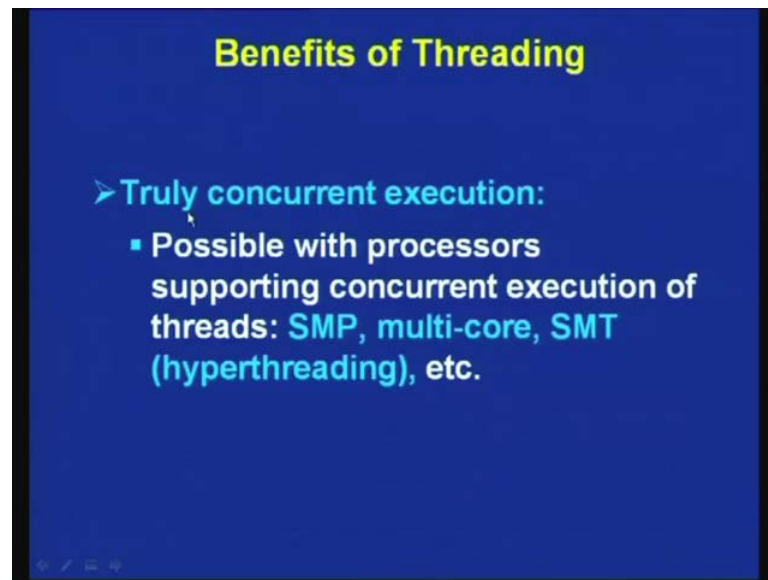
hardware, without the support from the hardware you cannot really do that. In what kind of supports are available in present day processors, so present day processors supports available in a form known s m p Symmetric Multi Processing Systems.

(Refer Slide Time: 16:51)



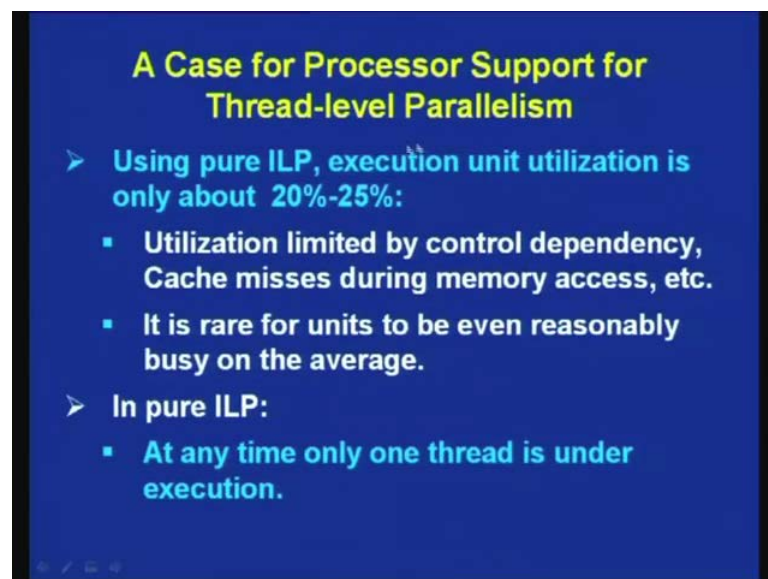
I shall going to the details of it later on, symmetric multi processing system, you have got multiple processors, symmetric multi processors, then you can have multi core. In case of symmetric multi processors, the processors may be on different codes or different chips, but in a multi core you have got multiple cores, multiple processors on a single die or we shall see the symmetric multi threading. I shall discuss in detail about in more details, which is also known as hyper threading.

(Refer Slide Time: 18:07)



So, with the help of this you can have concurrent execution of multiple threads.

(Refer Slide Time: 18:14)



Now, let us consider a case for processor support for thread level parallelism, using pure instruction level parallelism execution unit utilization is only about 20 to 25 percent. So, we have seen that whenever you are having multiple processing elements, as you have seen these are present in all modern processors, we have seen the (Refer Time: 18:54) processors square, there are 7 to 9 different types of execution units available.

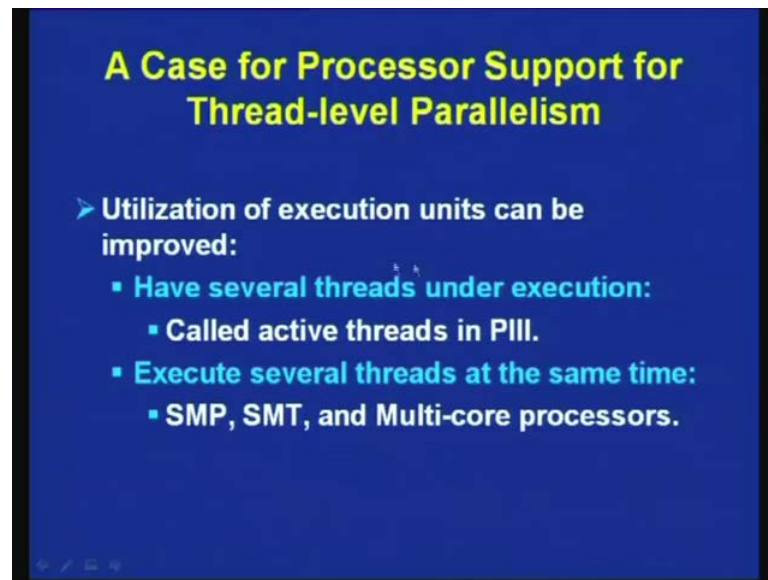
So, these execution units, how much they are utilized, utilization of these execution units are to be considered, in a hardware you have provided multiple execution units, can they be truthfully or meaningful utilized or they will remain idle in most of the situations. So, it has been found that using pure instruction level parallelism, execution unit utilization is only about 20 to 25 percent in modern processes, query have got a large essence say 8 9 execution units processing units.

So, utilization is limited by control dependency, cache misses during memory access and so on, so we have discussed about a different types of Hazzards, so whenever you do pipelining will be encountering different types of Hasserts. So, you have different types of dependencies, data dependency, control dependency, structural dependency and so on, so here by creating multiple execution units you can overcome the structural dependency.

That structural Hazzards will not be there, but the control dependency that will occur, because of you know whenever you are executing in a loops or decisions, then they control hazzards will be generated. So, because of this utilization of the instruction level parallelism is very limited, so it is rare for units to be even reasonably busy on the average.

And in pure instruction level parallelism at any time only one thread is under execution, so this is a limitation whenever we go for pure instruction level parallelism, only one thread is under execution at any point of 9. So, this demonstrates or this particularly tells you that it is very much essential to go for multi threading utilize thread level parallelism.

(Refer Slide Time: 21:13)



Now, utilization of the execution units can be improved, how you can have several threads under execution and for example, in Pentium III, these are called active threads. So, whenever you have multiple threads under execution, you can I mean you may give different names, but it is a you can concurrent execution of threads are taking place whatever may be the names. So, these are known as active threads in Pentium III and it executes several threads at the same time for example, by using as I have told SMP, SMT and multi core processors, so this clearly tells you the need for thread level parallelism.

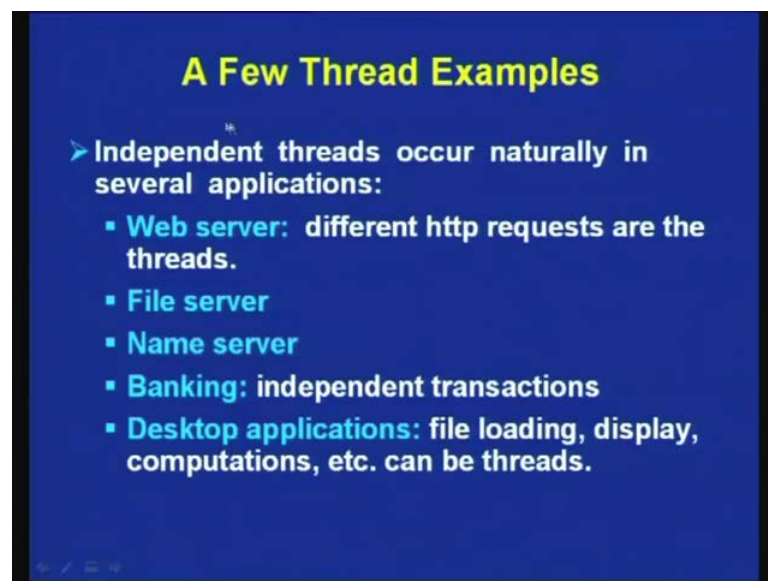
(Refer Slide Time: 22:05)



Now, threads in applications, where they are widely used threads are natural to a wide range in setup application, whenever one more or less independent often more or less independent. I mean for in many applications will find the different applications are independent, they do not dependent on one another, so completely independent applications can run and they can be naturally multi threaded.

And they are may be some data sharing though data sharing can take place among them to some extent, but that will not I mean that can be provided without much of difficulty. So, with limited amount of data sharing, these multiple applications can run independently and also sometimes, they are may be need for synchronization. Sometimes, also synchronization among themselves will be needed, and for synchronization there are techniques available and with the help of that at synchronal synchronization, processing synchronization and thread level synchronization can be achieved.

(Refer Slide Time: 23:25)

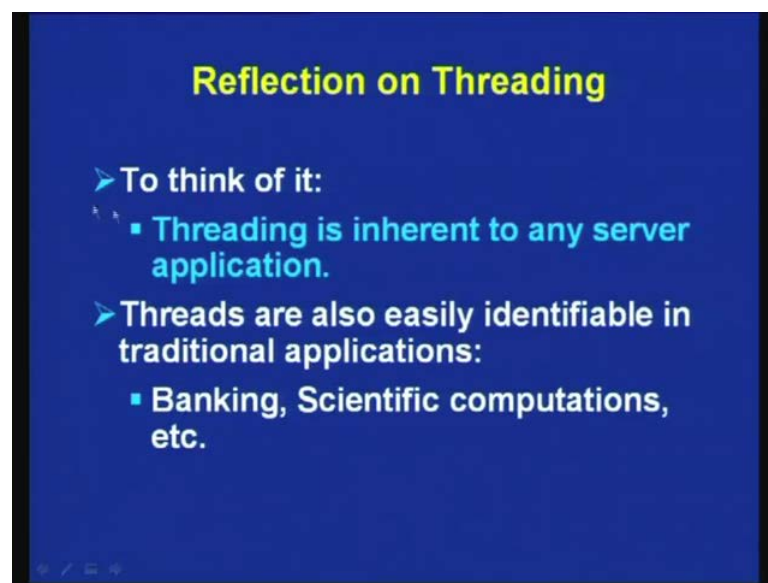


So, here are few thread examples given, so independent threads occur naturally in several applications as I was telling, and these are some of the applications, where these thread level parallelism we can be we lockers, and you can utilize them. Number 1 is web server, so different http request are threads, so a web server is giving a service to a large number of people and different each http request can be considered as a separate threads.

Similarly, a file server is giving service to a large number of users, and they are data is being stored file service is stored in the server, and from the server it is I mean it is access by multiple users locally or remotely whatever it may be. Then each of them can be considered as separate threads, similarly any or whenever you go for re implementing internet service you will require name server. So, name servers also receiving multiple request from different sources, and in such cases also each of these requests can be sub by using multiple threads.

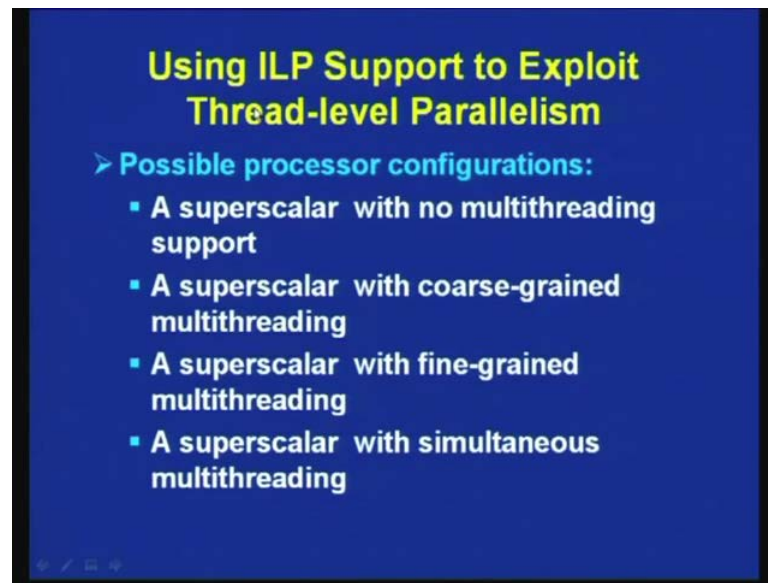
So, similarly as I have already told in banking applications, it is possible to have independent transactions, this independent transactions can be independently threaded and I can be can be taking care of by independent threads. So, not only in servers in rest of applications also it is possible to have independent threads, because in even in rest of applications different types of functions have been performed, like file loading, display of data on the screen, computation etc, can be different threads. So, if in a simple desktop you can have a multiple threads, so this gives you examples of different threads.

(Refer Slide Time: 25:42)



Now, so we can say, we are now confuse that threading is inherent to any server application, and threads are also easily identifiable in traditional applications like banking, scientific computations etc.

(Refer Slide Time: 26:03)

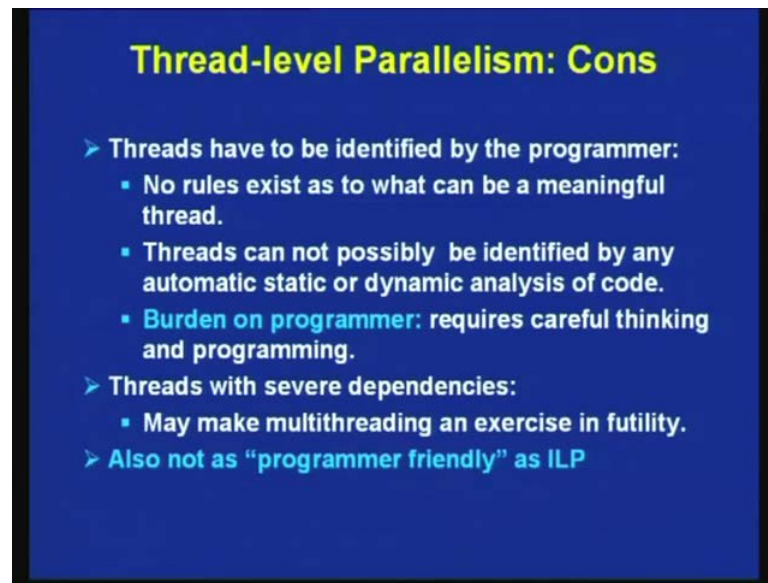


Now, we have discussed about instruction level parallelism, and we have briefly discussed about multi threading, now can one support each other. So, here we can have instruction level parallelism support to exploit (Refer Time: 26:28) thread level parallelism can be done, and you can configure processors accordingly. For example, you can have 4 possible configurations, you can have a super scalar processor with no multi threading support, this is one possibility.

So, here you will be using only instruction level parallelism, but no thread level parallelism, second possibility is a super scalar processor with coarse grained multi threading. So, multi threading can be of two types, fine grained and coarse grained which have discussed about that, and with coarse grained multi threading, third is super scalar with fine grained multi threading, so that can be supported or a super scalar processor with simultaneous multi threading.

So, this fine grained multi threading, coarse grained multi threading and simultaneous multi threading, these techniques these we shall discuss later on and in a modern super scalar processor these can be supported very easily.

(Refer Slide Time: 27:37)



Now, we have, so far we have discussed about the benefits of multi threading, we have seen various advantages, and different applications which inherently support multi threading, and you can gain you can utilize the execution units more efficiently using multi threading. So, the advantages we have discussed, but in this world nothing will be one sided; obviously, there is some disadvantages and here some of them are highlighted. So, threads have to be identified by the programmer, and unfortunately no rules exist as to what can be meaningful thread.

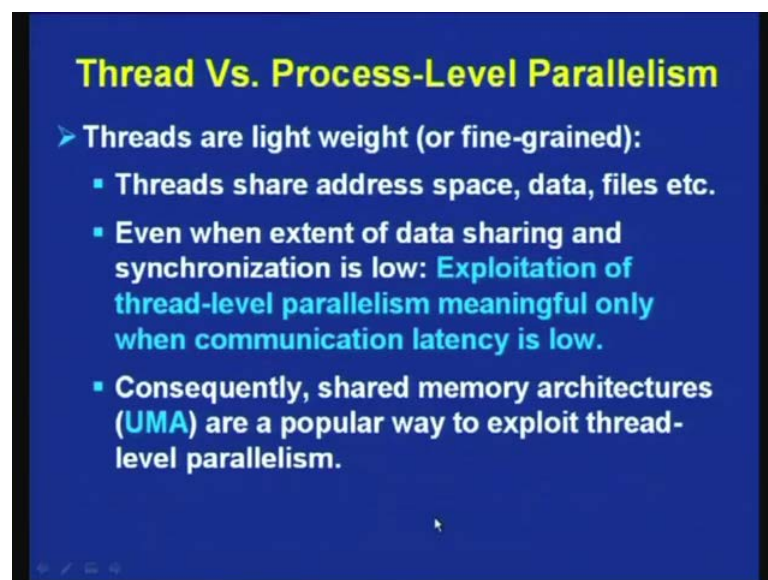
So, a programmer has to use his experience in tuition to create threads, there is no generalize rule say follow this algorithm and create threads, so that type of thing unfortunately does not exist. So, no rules exist as to what can be meaningful thread and threads can possible and cannot possibly be identified by any automatic static or dynamic analysis of code. So, you cannot by analyzing automatic static or dynamic analysis of also code does not really lead to identification of threads, how you can create meaningful threads.

So, what is happening in not sell in putting a burden on the programmer, so it that multi threading can be considered some kind of burden on the programmer, it requires careful thinking and programming. As, I have already told experience in tuitions this plays important role in this type of situation, and more over threads with severe dependencies.

So, you an application program may have severe dependencies, so whenever there are severe dependencies that may make multi threading an exercise in futility.

In other words, so whenever you have got severe dependencies, you may not really achieve I mean foot full multiple threads useful multiple threads, so as a consequence the thread level parallelism is not as programmer friendly as ILP. So, in case of instruction level parallelism, we have seen that programmer or user is not really not much bothered, it is taken care of by hardware in most of the situations. Since, it is taken care of by the programmer by this hardware in most of the situations, it is very programmer friendly programmers are very happy with that, but that is not the situation in thread level parallelism.

(Refer Slide Time: 30:52)



Thread Vs. Process-Level Parallelism

- **Threads are light weight (or fine-grained):**
 - **Threads share address space, data, files etc.**
 - **Even when extent of data sharing and synchronization is low: Exploitation of thread-level parallelism meaningful only when communication latency is low.**
 - **Consequently, shared memory architectures (UMA) are a popular way to exploit thread-level parallelism.**

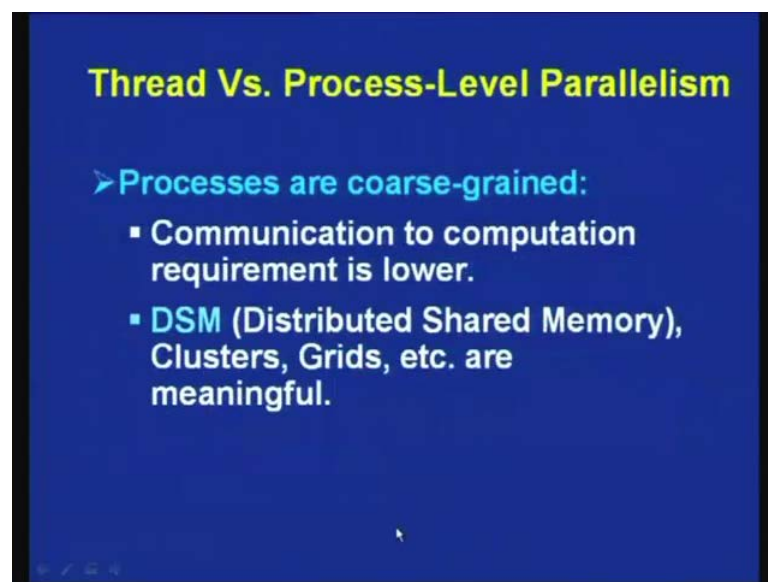
So, I have already discussed about it threads are light weight fine, grained threads are share address space data and files. Even when extent of data sharing and synchronization is low exploitation of thread level parallelism is meaningful only, when communication latency is low, so this is another very important parameter communication. So, different processors and threads have to communicate with each other, so that communication over head you have to consider, you have to take in to consideration.

So, it all depends on how different processors are connected, or they sharing a common bus or they connecting through a switch, or they connected through a internet, so it all depends on how they are inter connected. And so, that communication coarse had to be

taken in to consideration, consequently shared memory architectures are popular way to exploit thread level parallelism, this shared memory architectures are known as uniform memory access.

So, in shared memory architectures you will see, you have a common memory, which is being shared by all the processors, may be through a bus, usually it is through a bus. And are popular weight with exploit thread level parallelism; that means, this thread level parallelism can be more meaningfully a utilized in a situation, we are communication of variety is small, and that happens in a uniform memory access processor system in even more systems.

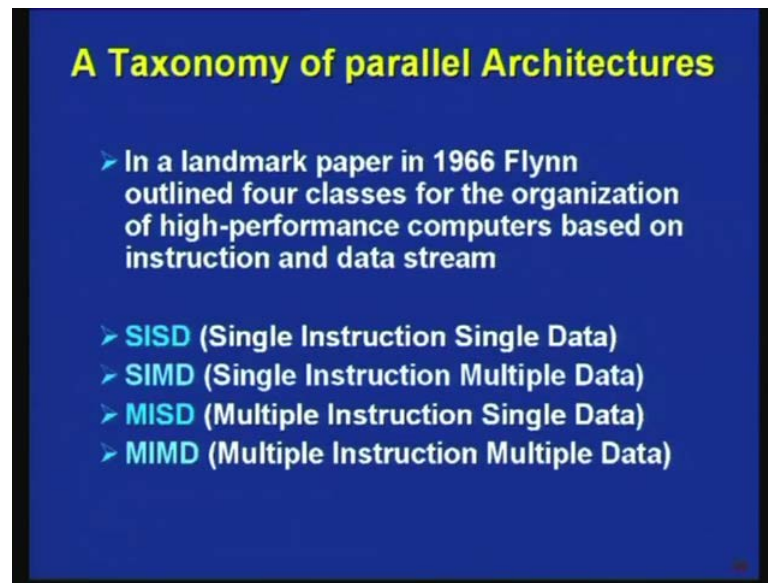
(Refer Slide Time: 32:49)



On the other hand, you can considered processors has coarse grained, coarse grained means communication to computation requirement is I mean whenever you can use these whenever the communication to computation requirement is lower. And you can go for distributed shared memory, DSM in clusters grids etc are meaningful so; that means, the you can use I mean process level parallelism in distributed shared memory architecture.

So, on the other hand in UMA thread level parallelism is more meaningful on the other hand, this coarse grained parallelism I mean you can go for coarse grained parallelism in distributed shared memory architecture.

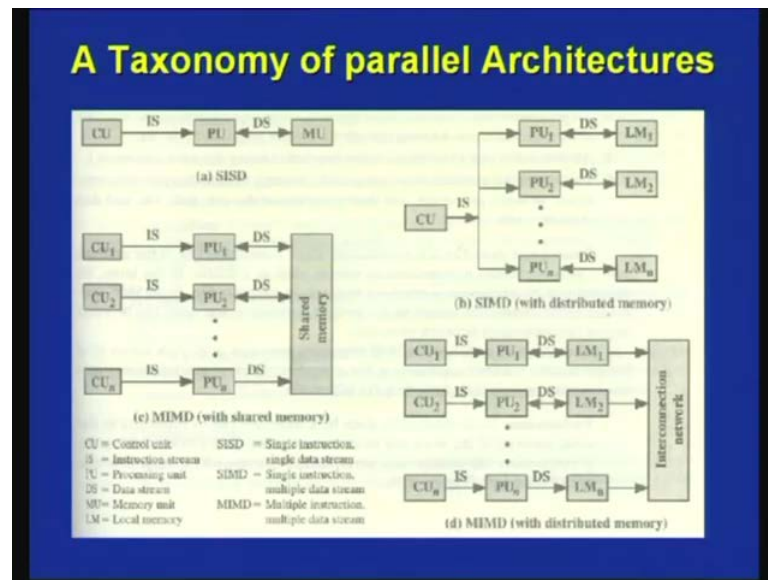
(Refer Slide Time: 33:49)



Now, we shall focus on different types of parallel architecture that are available, this can be considered as taxonomy of parallel architectures. So, back in 1966 try an outlined 4 classes for organization of high performance computer, based on instruction and data streams. So, in a processing you will find that instruction is stream and data streams are there, and the and how the instruction is stream and data streams are flowing, based on that the classification has been done in to 4. types.

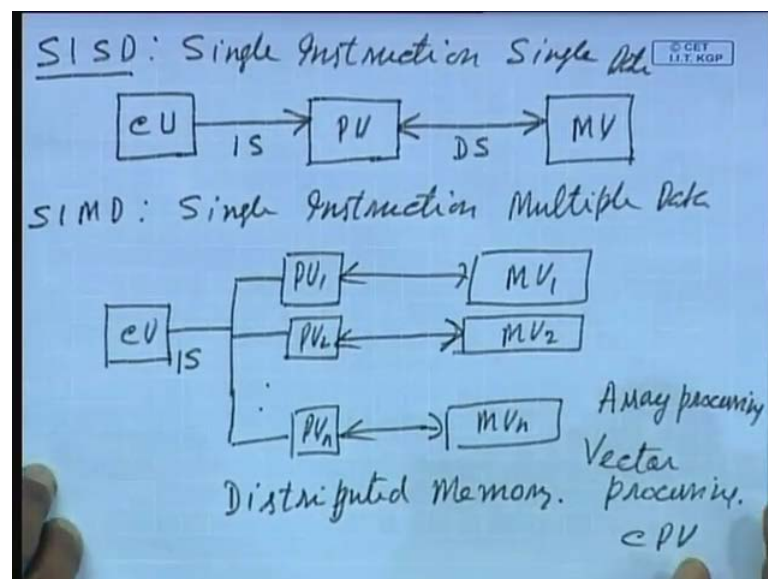
These are known as the one is first one is known as SISD, Single Instruction Single Data, second one is SIMD Single Instruction Multiple Data, third one is MISD Multiple Instruction Single Data, so that means, and fourth one is MIMD Multiple Instruction Multiple Data. So, you have got 4 types 4 classifications of computers, which was proposed by Flynn back in 1966, and the even today that is being used for classifying computers, so I shall briefly discussed about this four types of classifications starting with SISD, MIMD.

(Refer Slide Time: 35:26)



First let us consider SISD.

(Refer Slide Time: 35:30)



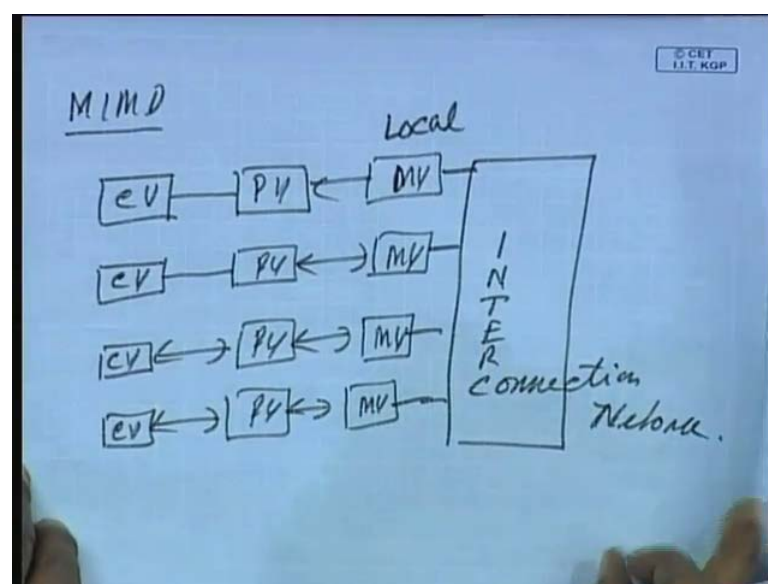
You will require control unit CU, you will require processing unit PU, you will require memory unit, these are the 3 hardware receive resources. You require in your computation control unit, processing unit and memory unit, normally this control unit and processing unit, you know that together is available in the form of a CPU central processing unit.

Now, in case of SISD Single Instruction Single Data, you have a single instruction that is coming from the control unit to the processor; that means, a single instruction is provided to the processing unit. So, you have got a single instruction stream, and then also you have got a single data stream, that is between the processing unit and a memory unit, so here you have got instruction stream and here you have got data stream.

So, this is typically is SISD architecture, where you have got a similar stream of instructions coming from the control unit to the processing unit, and single stream of data flowing between processing unit and memory unit. Then coming to the SIMD Single Instruction Multiple Data, so here you have got the control unit, so control unit is providing a single instruction and you have got multiple processing unit.

So, you have got processing unit 1, processing unit 2 may be processing unit n and there is a separate data stream, multiple data stream between memory unit, memory unit 1, memory unit 2, memory unit n. So, this is your here what is happening, this is not a here as you can see the memory is not shared, so you can say this is distributed memory, so this type of thing processing, you do in case in case of your area processing and vector processing. So, this is the second type of classification Single Instruction And Multiple Data, then coming to the third type of classification that is your MIMD and I mean SIMD, SISD, then you have got the MIMD.

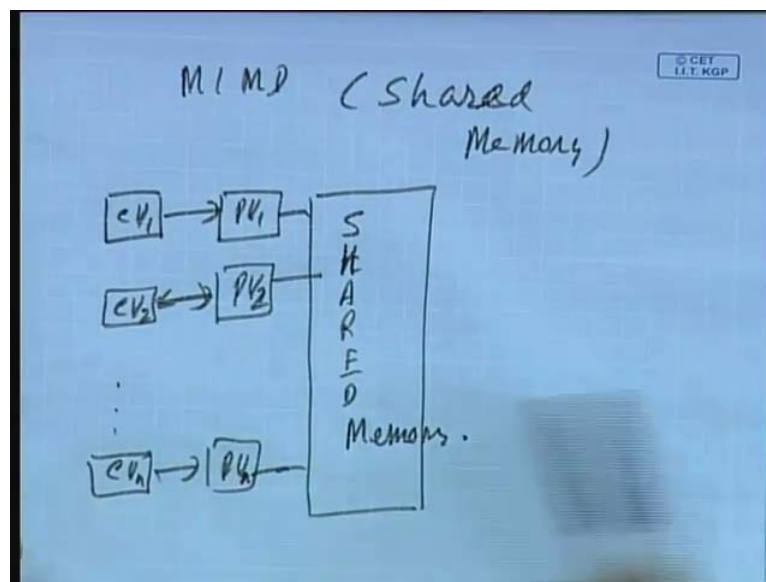
(Refer Slide Time: 39:18)



In MIMD, you have got and multiple control unit is connected to multiple processing unit and each of them is connected to separate memory unit, so this is where you do with in this case we can say that they can be connected through a inter connection network. So, through the inter connection network are connected, then you have got, then you can these are each of the processors are have much are having their local memory, this is your local memory is connected to the to the processors.

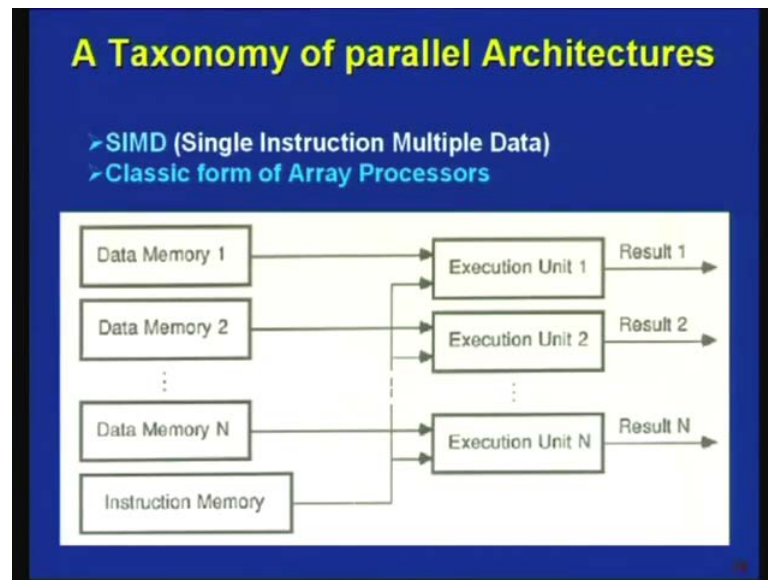
And they are connected through the inter connection network, so this is a kind of MIMD architecture connected through inter connection network, you can have MIMD with shared memory as well.

(Refer Slide Time: 41:12)



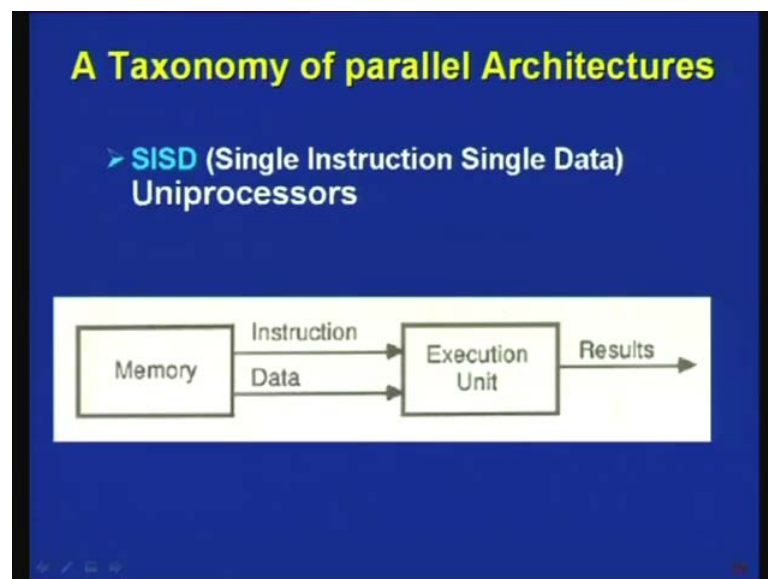
So, MIMD with shared memory is also possible, where you can have multiple control unit, control unit 1, control unit 2, control unit n, these are connected to multiple processing unit, processing unit 2, processing unit n, and these are connected to a shared memory, so this is how the MIMD shared memory is occurring.

(Refer Slide Time: 42:11)



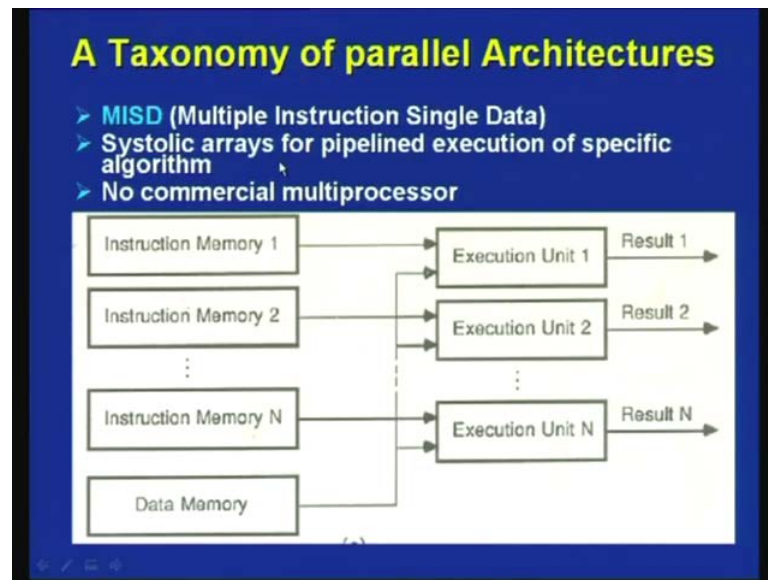
And so, I have already discussed about these things.

(Refer Slide Time: 42:20)



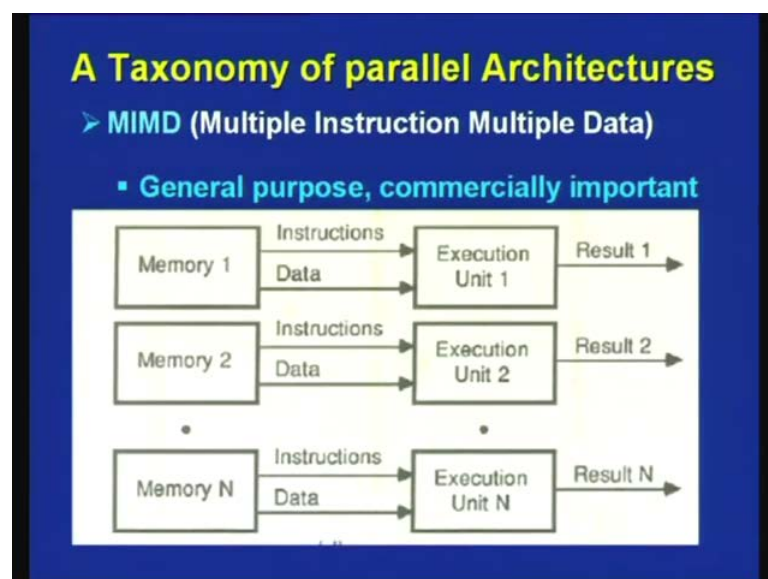
First one is that this is the typical uni processors system, SISD, then you have got SIMD classic form of array processors and vector processors.

(Refer Slide Time: 42:35)



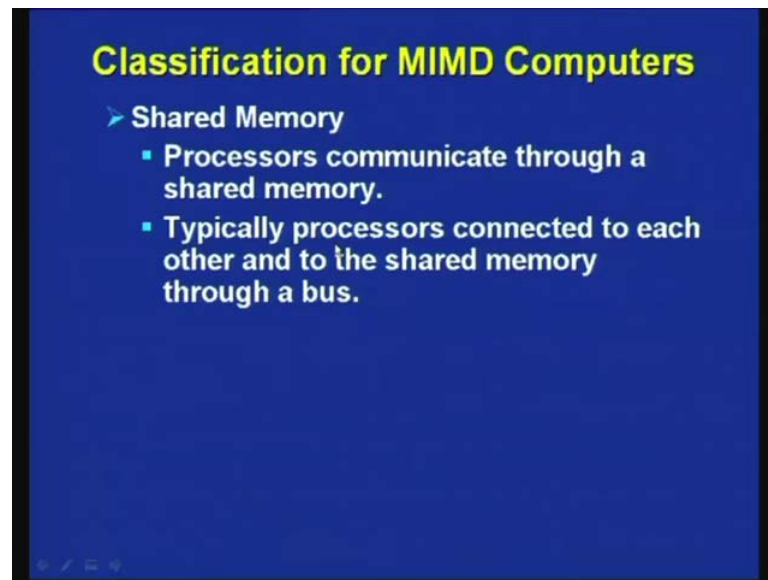
Then this is your MIMD, MISD, where multiple instruction single data is being process, so this one as you can see here you have got different instructions coming from different processors and single data is being processed. And, as it happens in a systolic array and results are generated by multiple execution units, so this is the typical MISD architecture and finally, of course this one was proposed systolic architecture was proposed in 70's, but there was no commercial implementation of it.

(Refer Slide Time: 43:20)



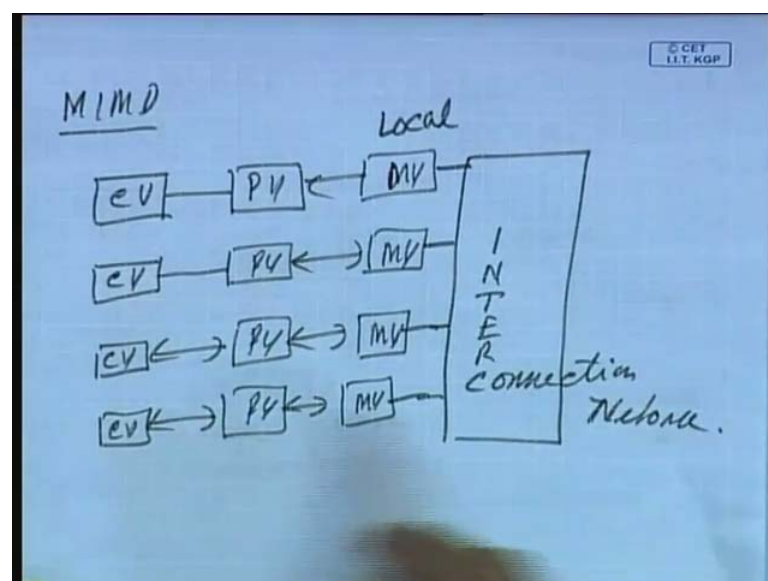
And finally, the MIMD, which is general purpose and commercially important.

(Refer Slide Time: 43:28)



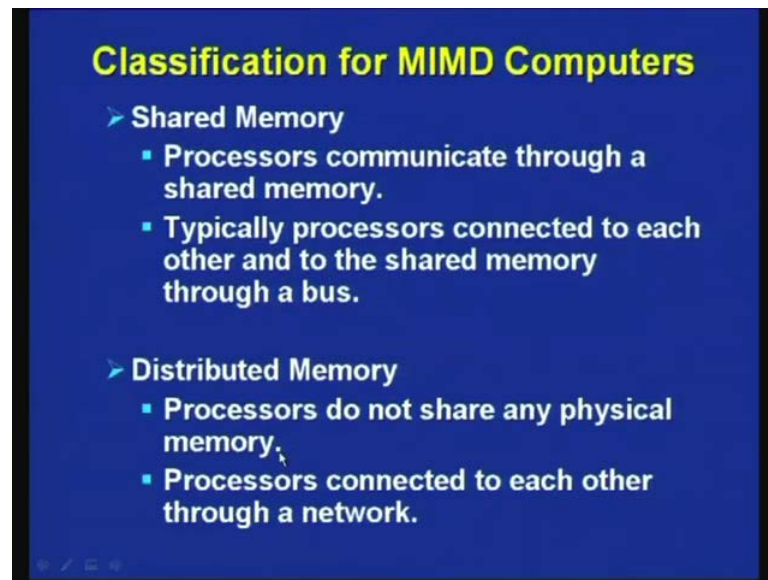
So, now a days this MIMD processors are becoming impingingly popular and widely used, so multiple instruction, multiple data processors are becoming more and more popular. So, we shall divert some time on classifications of MIMD computers, MIMD computers can have shared memory, so the processors communicate through a shared memory, and I have already discussed about this, this is your MIMD with shared memory.

(Refer Slide Time: 43:58)



So, through a shared memory the communication take place.

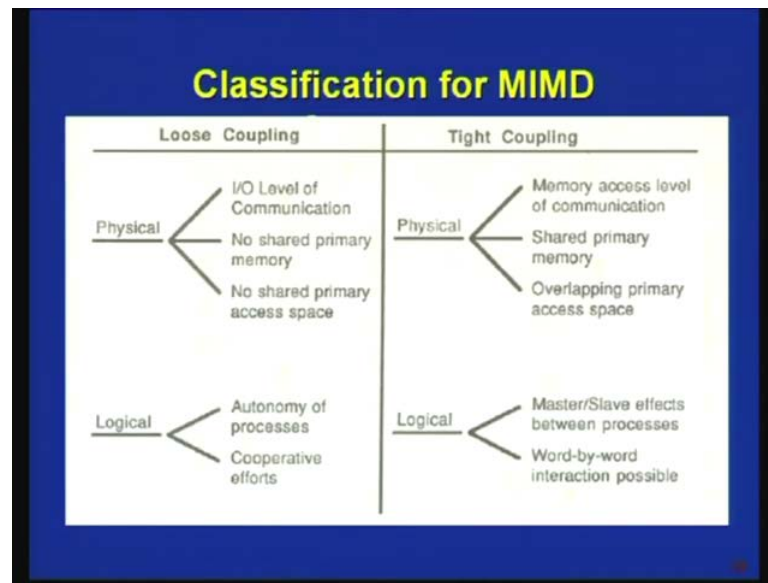
(Refer Slide Time: 44:04)



And typically can a processors are connected to each other to the shared memory through a bus, so usually a bus is use to communicate; that means, each of the processing elements are connected to a bus. And to the bus you can say, it will be like this say processing element, all the processing elements will be connected to a bus, and to that bus they will be a shared memory. So, this is the typical situation of shared memory, on the other hand and distributed memory processors do not share any physical memory.

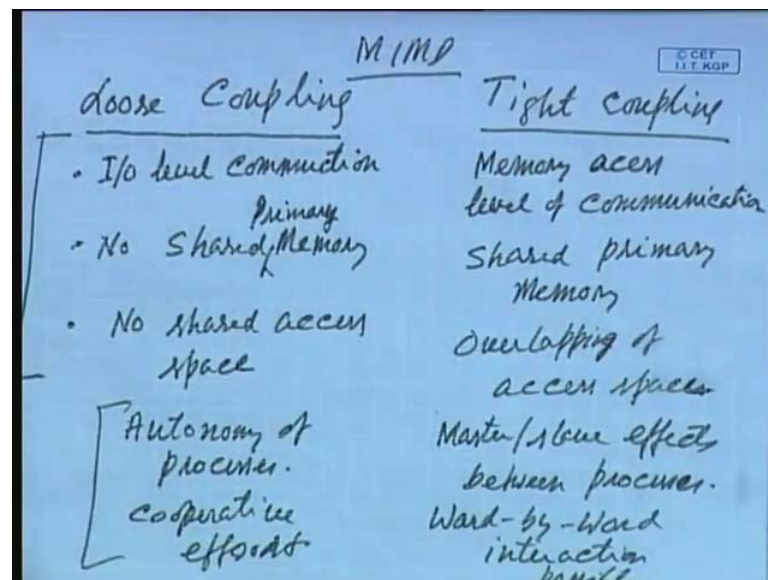
So, they do not share any physical memory, only local memory is been shared as I have already shown here this is the distributed memory here, they have some local memory, but they do not share any physical memory processors are connected to each other through a network. So, it is use were an inter connection network through which each of these processors are connected, they are having they are local memory units, and they do the processing using they are local memory units and through the inter connection network they exchange information they communicate with each other.

(Refer Slide Time: 45:33)



So, this is again some classification based on classifications of MIMD, and actually there are two terms which are used in this context.

(Refer Slide Time: 45:52)



one is known as loosely coupling, another is known as tight coupling, loose coupling and tight coupling in the context of MIMD. So, physically in case of loose coupling here IO level communication is done, and here it is we have seen it is shared memory. So, it will be memory level communication, memory access level of communication and in loosely couple systems, there is no shared primary memory. So, primary memory let me remind

you what do you mean by primary memory, so in a computer system you have different types of memory, primary memory, secondary memory.

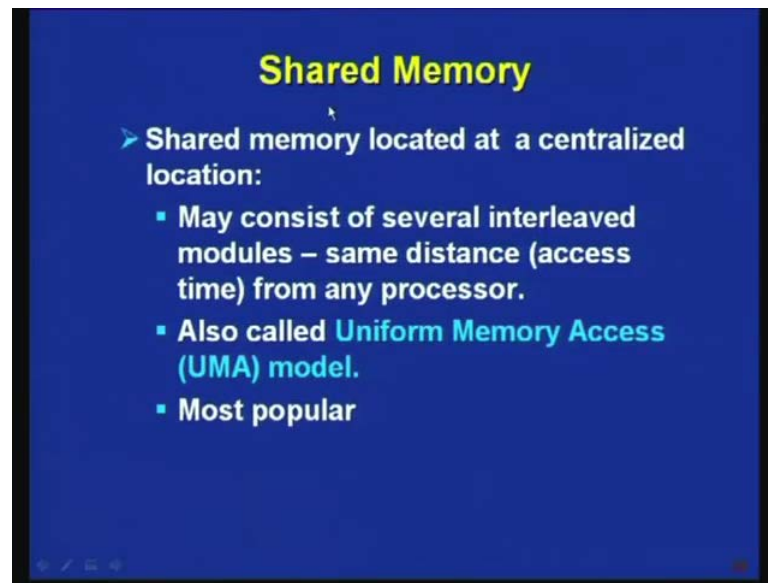
Secondary memory is a destorage, primary memory is from where, instructions are fetched and executed, so when the processors are executing a program the instructions are fetch from the primary memory that is why special designation primary memory is given. And in case of tightly coupled system, it has got shared primary memory, and no shared access space overlapping of access space.

What do you really mean by that this, so no shared access space and overlapping of access space, we have seen that whenever we discussed about partial memory. They are we have seen, we can have pages and each of the pages can be made, I mean there is some flag bits, which can be said, and which can allow sharing of a page by other users or a particular page may not be allowed by shared by other users.

So, whenever a page is allowed to be shared by other users, that is a essentially leads to overlapping of access space; that means, the same page can be access by multiple programs or multiple processors, but in this case there is no shared access space. So, in that case the memory pages are only a I mean you cannot be access by other users, so there is no shared access space, here overlapping access space. And this is from the view point of physical inter connection can logically, in case of loosely couple system, there is autonomy of processors.

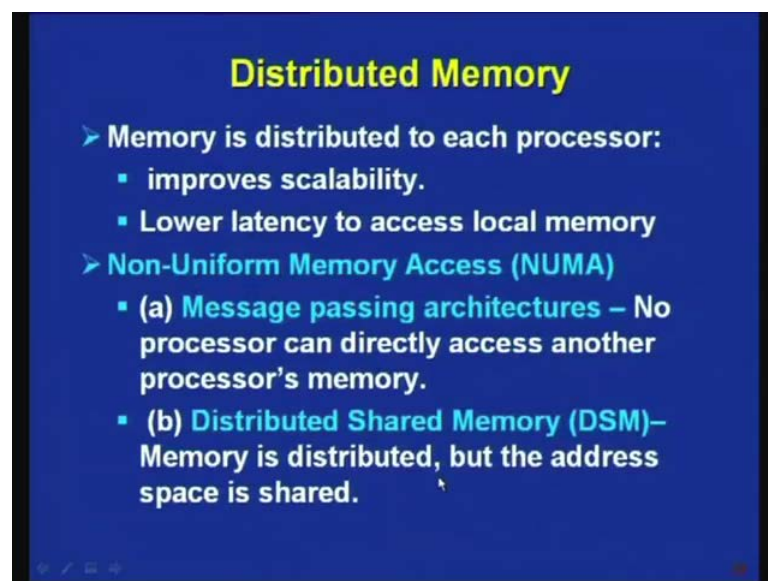
And here some kind of masters here, master slave effects between processors, so this is different distinguish in terms of logical access, and here you can say whenever it is loosely coupled you can say it is a kind of co operative effort. And in this particular case it is possible to have word by word interaction, so the distinguish between loosely couple systems and tightly couple systems, in the context of MIMD has been discussed in detail.

(Refer Slide Time: 50:39)



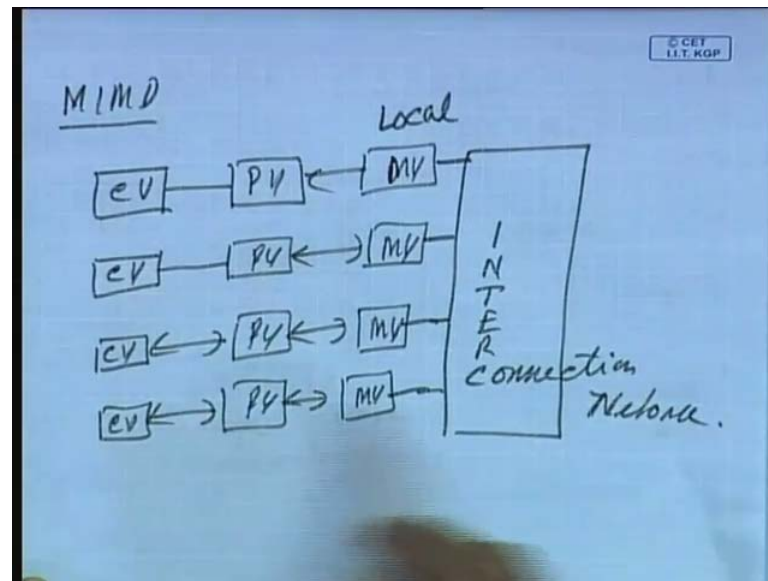
And very briefly I shall talk about this shared memory and distributed memory shared memory located at centralize location, may consists of several interleaved modules. Than the; that means, the same distance from any processors, and that is the reason why it is called uniform memory access model UMA model; that means, each processor will take same time to access it. So, in case of UMA, so you have got multiple processors, so of a since they are sharing through a bus kind of thing, each of them will take the same amount of time, and that is why the term uniform memory access model.

(Refer Slide Time: 51:21)



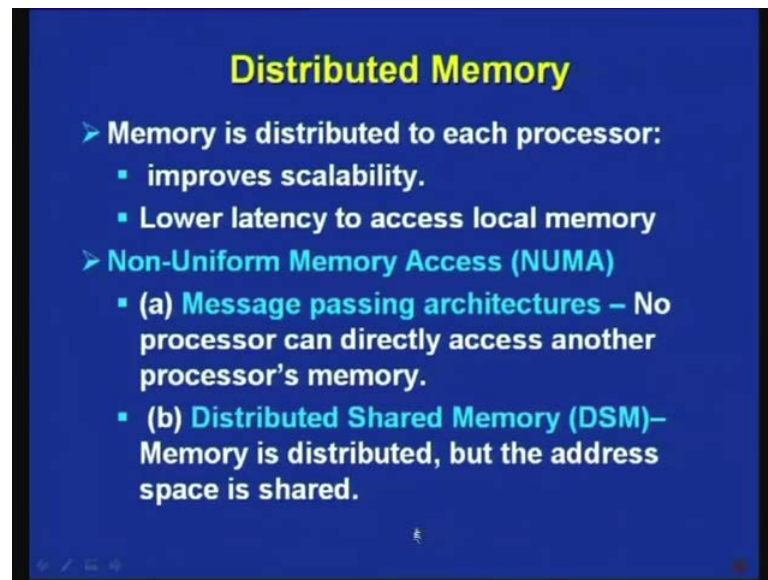
And this is very popular, on the other hand whenever memory is distributed to each processor, although it improves scalability. It has got lower latency to access local memory, on the other hand it has got higher latency for access of local memory so; that means, whenever we discussed, this local this particular situation whenever this processor is accessing.

(Refer Slide Time: 51:49)



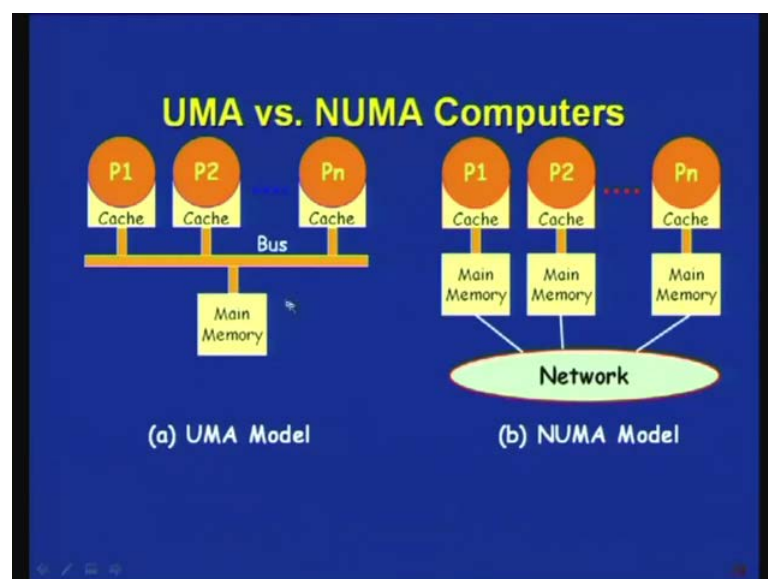
This memory the latency is very small on the other hand if this processor is allowed to access this memory; obviously, the access time will be longer. So, you can see the access time is not uniform in such a situation, whenever the connection is through a inter connection network.

(Refer Slide Time: 52:09)



And that has lead to the model known as non uniform memory access, so whenever it is non uniform memory access, the we use message passing architecture, no processor can directly access other processors memory. So, the communication is done by using message passing or distributed shared memory, memory is distributed, but the address space is shared that I have already mention how it can done with the help of that virtual memory concept.

(Refer Slide Time: 52:39)



And this is the typical diagram of the two different models, this is the uniform memory access model, so here there is a bus, and this is the main memory or primary memory through, which sharing is occurring. Although, each of the processor is having their own cache memory, but the memory that is being shared is the main memory or the this is the prime memory.

So, all are connected through a bus and as a consequence, the access time is uniform for each of the processors, on the other hand whenever you have got this non uniform memory access model. Each processor with they are built in cache is connected to a main memory and that that is being connected to the network or it may be inter connection network. And whenever, sharing is done this processor will take much less time to access this memory, compare to this processor trying to access trough the network to this memory.

And not only I mean in this case the time I mean access time will be different, for different processor for accessing different memory devices. So, with this we have to come to the end of today's lectures, we have discuss details of I mean differences between process, what is process, what is thread and the relationship between process, and thread. And also we have discussed about the different processors models, like SISD SIMD and MIMD with this we conclude today's lecture.

Thank you.