MCA – 503 Distributed Operating System

Unit # 2 and 3

Department of Computer Science

SPU, V V Nagar.

Ms D M Patel

Index

System Models

- Processes run on processors.
- In traditional system there is only one processor so there is no question like how processors should be arranged.
- But here in Distributed System we have multiple CPUs available so here it is a design issue.
- So here multiple CPUs Can be organized in several ways.

- So the different ways into which we can arrange or organize the multiple CPUs are the System Models.
- There are three popular models available namely
    1. The Workstation Model
    2. Processor Pool Model
    3. Hybrid Model

The Workstation Model

- This one is the very simple and straight.
- Here, the Distributed System consists of number of workstations connected by the high speed LAN scattered thought-out the building.
- At any instant one user may be using any one of the workstations or that workstation may be idle.
- So here some of the workstations may be occupied whereas some may be idle. Right!

The Workstation Model Continue……..

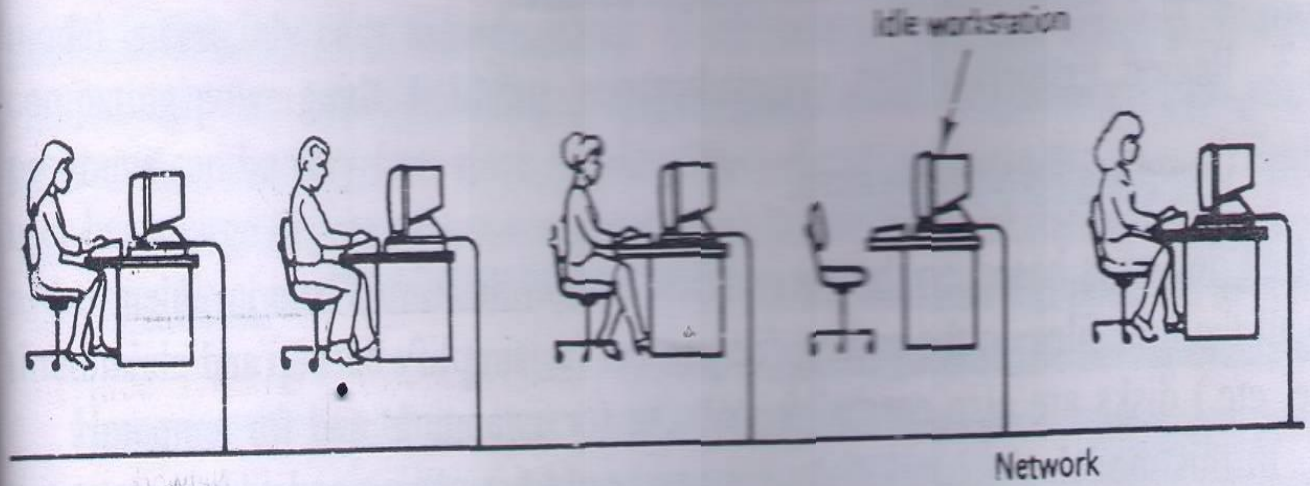


Fig. 4-10. A network of personal workstations, each with a local file system.

The Workstation Model Continue……..

- The workstations which has been shown in the previous slide might be having disks at their disposal or they might not be having.
- So here we can have two types even based on the fact that workstation contains disk or not.
- Distributed System having Diskless workstations(workstations without disks) And Distributed System having Disk full workstations(workstations with disks).

The Workstation Model Continue……..

- So if former is the case that means if the distributed system is diskless or the workstations are not having disks then it is obvious that file system will not be local and it will be provided by the one or more remote servers.
- So here every request for reading or writing the files are sent to the file server, which performs the work and sends reply back.

The Workstation Model Continue……..

- This model of Distributed System(Diskless) is useful for several reasons.

  1. Having the large number of workstations with disks which are small, slow is typically much more expensive than having one or more file servers equipped with huge, fast disks and accessed over the LAN.

  2. Popular because of its ease of maintenance…..

The Workstation Model Continue……..

- For example suppose new release of some program, say some system programs like compiler comes out then it can be installed on small number of file servers so here we do not require to install it all each workstation.

3. Diskless workstations provide symmetry and flexibility because user can use every time different workstation since file system is not local.

The Workstation Model Continue……..

- Now consider later case means Distributed System having local disks or workstations having disk.
- But here even we might be using disks for any one of the following purposes.

1. For storing only paging and temporary files.
2. For storing only paging, temporary files, and system binaries.
3. For storing only paging, temporary files, system binaries, and file caching.

The Workstation Model Continue……..

4. For storing complete file system.

- So as you might have perceived that with increasing number network traffic will decrease.
- That means if we use local disk only for the purpose of paging and temporary files then for rest of work like reading files, writing files requests will have to be sent to file servers.
- Keep in mind that for first three we require both local disk and file servers, but for last option that is forth one that uses disks for complete file system eliminates need for the remote file server.

Disk Usage	Advantages	Disadvantage
(Disk less)	Low cost, easy h\w and s\w maintenance, symmetry and flexibility	Heavy n\w usage, file servers may become bottlenecks.
Paging, Scratch files	Reduces n\w load over diskless case. Obvious thing !	Higher cost due to large # of disks needed.
Paging, scratch files, binaries	Reduces n\w load even more.	Higher cost; additional complexity of updating the binaries
Paging, scratch files, binaries, file caching	Still lower n\w load. Reduces load on file servers as well.	Higher cost; cache consistency problems.
Full local file system	Hardly any n\w load. Eliminates need for file servers.	Loss of transparency.

Disk usage on workstations

- So one thing is clear we can have Distributed System
- 1. D S without disks that is diskless.
- 2. D S with disk but only for paging and temporary or scratch files and for file system we have file server as well.
- 3. D S with disk but only for paging, temporary files and system binaries for file system we have file server as well. .
- 4. D S with disk but only for paging, temporary files, binaries and file caching for file system we have file server as well. .
- 5. D S with disk that provides complete file system and here no need for file servers..

The Workstation Model Continue……..

- But here as we had seen in one of the previous slides that some of the workstations may be busy while some may be idle. Right!
- And as we know that Distributed System should present to its users a single virtual uniprocessor image, that means a user is not aware that where his\her work is being completed that means here if load of the particular machine is more then its some of the processes can be migrated or moved on some idle machine.
- So question is that how to make use of this idle workstations.

The Workstation Model Continue……..

- That means using idle workstation is also a issue here.
- The earliest attempt or earlier way of using idle workstation was the *rsh* program or command that comes with Berkeley UNIX.
- This program is called by rsh machine_name command.
- That means rsh command runs specified command on specified machine.

    1. so here responsibility for identifying idle workstation is totally on the users.

The Workstation Model Continue……..

- So another way or algorithm for utilizing idle workstations got developed.
- This algorithm used to locate idle workstations can be divided into two categories:

  - server driven*
  - client driven

  * server driven approach is also called registry based algorithm because here registry is maintained either centrally at one place or alternatively may be at local machine so here each user will have its on copy of registry.

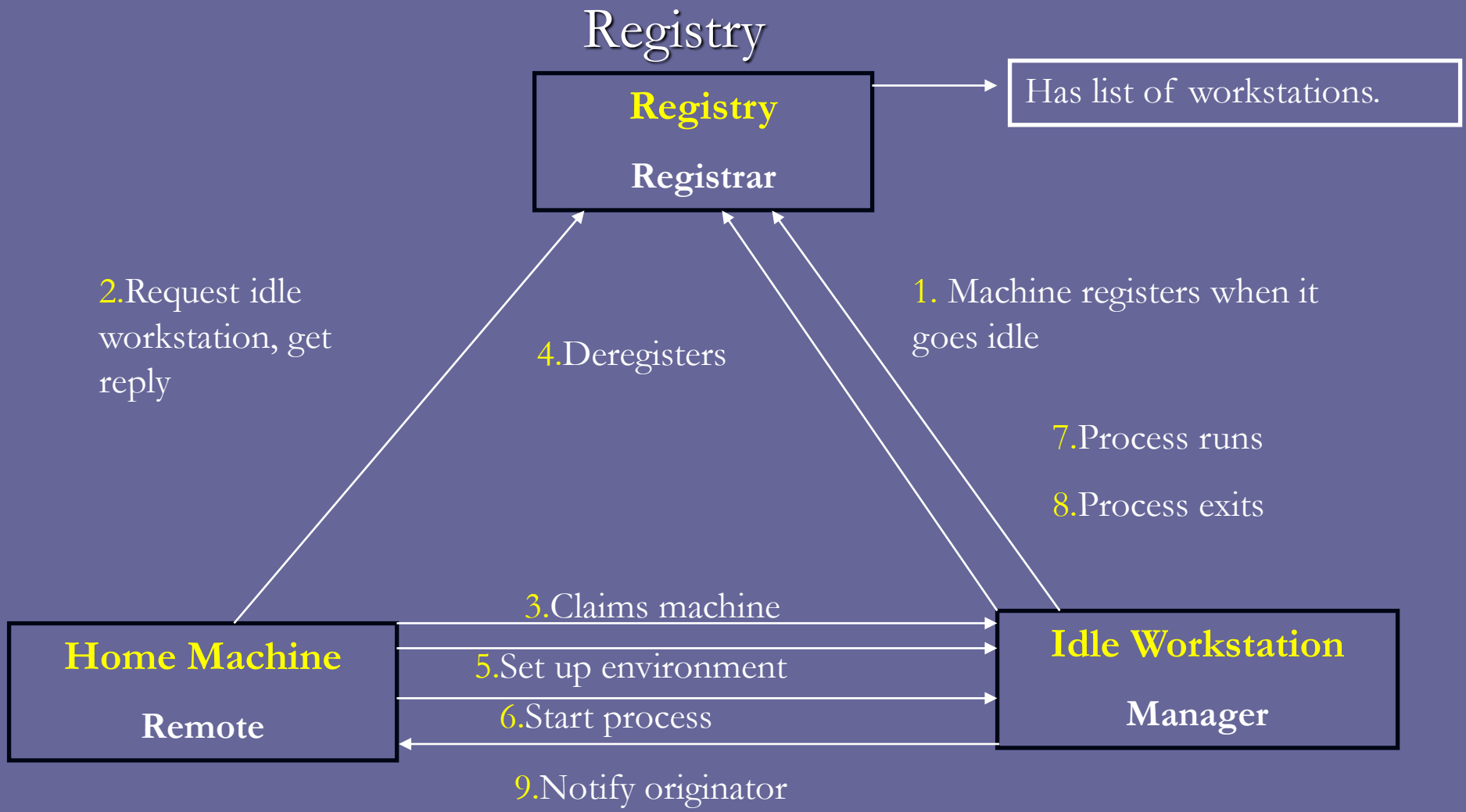
The Workstation Model Continue……..

- In the former approach when the workstation goes idle and thus becomes a potential computer server, it announces this fact and its entry is made in central registry file which might be maintained by any common server.
- It can do this by entering its name, network address, and properties in registry file and the ***remote*** program looks into registry file to find suitable workstation.
- For reliability multiple copies can also be there.
- An alternative way to do the same it may announce this fact that it has become unemployed and broadcast this message on network.

The Workstation Model Continue……..

- So here All other workstations record this fact.
- In effect each machine maintains its own registry file.

The Workstation Model Continue……..



Functioning of Registry based algorithm(server driven & central registry)

The Workstation Model Continue……..

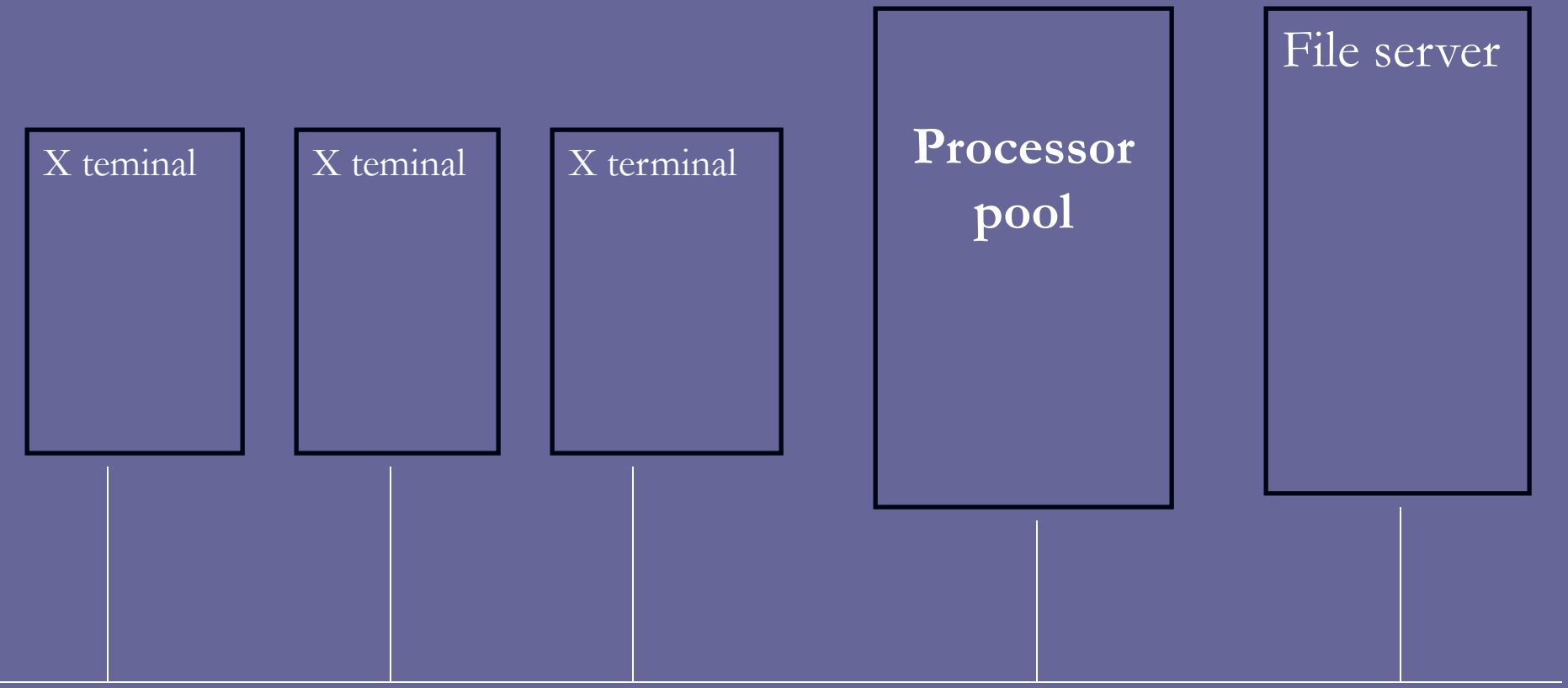
- The other way to locate idle workstations is to use a client driven approach.
- When the particular workstation invokes remote, it broadcasts a request saying what program it wants to execute.
- How much memory will be needed, floating point is needed or note, and so on…..
- So idle workstations will send reply
- When reply comes back ***remote*** picks one of them and proceeds further means runs that process there.

- So in client driven approach registry is not maintained at all.

The Processor Pool Model

- Although using idle workstations adds a little computing power to the system, it does not address a more fundamental issue :
- what happens when it is feasible to provide 10 or 100 times as many CPUs as there are active users? So…..
- An alternative approach is to construct a processor pool , rack full of CPUs in the machine room, which can be dynamically allocated to users on demand. This approach is shown in next slide.

A system based on the processor pool model



The Processor Pool Model continue…….

- Instead of giving users personal workstations, in this model they are given high-performance graphics terminals.
- This idea is based on the observation that what many user want is a high-quality graphical interface and good performance.
- The motivation for the processor pool idea comes from taking the diskless workstation idea a step further.
- If the file system can be centralized in a small number of file servers to gain economic gain, it should be possible to do the same for compute servers .

The Processor Pool Model continue…….

- By putting all the CPUs in big rack in the machine room, power supply and other packaging cost can be reduced, giving more amount of power for the given amount of money.
- The model also allows for easy incremental growth.
- If computing load increases by 10 percent then u can buy 10 percent more processing power to the processor pool.
- Actually, In effect we are converting all the computing power into "idle workstations" that can be accessed dynamically.

The Processor Pool Model continue…….

- Users can be assigned as many CPUs they need for short periods. After which they are returned to the pool so that other users can have them.
- There is no concept of ownership here: all processors belong equally to everyone.
- The biggest argument for centralizing the computing power comes from queuing theory.
- Let us call the total input rate λ per second, From all the users combined.

The Processor Pool Model continue…….

- Let us call the rate at which the server can process requests μ per second.
- For stable operation we must have $\mu > \lambda$.
- If server can handle 100 requests/sec, but the user continuously generate 110 requests/sec, the queue will greatly increased… which is obviously understandable.
- It can be proven that the mean time between issuing a request and getting a complete response, T , is related to λ and μ by the formula

The Processor Pool Model continue…….

- $T = 1 / \mu - \lambda$
- Which is purely understandable…..!
- As an example consider a file server that is capable of handling 50 requests/sec but which only gets 40 requests/sec. the mean response time will be 1/10 sec.
- Suppose that we have *n* personal multiprocessors, each with some number of CPUs, and each one forms a separate queuing system with request arrival rate λ and CPU processing rate μ.
- The mean response time will be same as above.

The Processor Pool Model continue…….

- Now consider the situation if we scoop up all the CPUs and place them in a single processor pool.
- Instead of having n small queuing systems running in parallel, we now have one large one, with an input rate $n\lambda$ and a service rate $n\mu$.
- Let us call the mean response time of this combined system $T1$. From the formula above we find
- $T1 = 1 / n\mu - n\lambda$ which is $= T/n$.
- Which is something obvious…..!

The Processor Pool Model continue…….

- This surprising result says that by replacing *n* small resources by one big one that is n times more powerful, we can reduce the average response time *n-fold*.

A Hybrid Model

- Here each user has personal workstation and also processor pool in addition.
- Obviously this solution is expensive than either a pure workstation or processor pool.
- But it combine the advantages of both.
- Interactive work can be done on workstations, giving guaranteed response.
- But here for simplicity utilizing idle workstations concept haven't been involved.

A Hybrid Model Continue…….

- Here all noninteractive processes run on processor pool so heavy computing power can also be utilized.
- This model provides fast interactive response and an efficient use of resources and the simple design.

Processor Allocation

- As we know that distributed system consists of multiple CPUs what ever is the arrangement like as workstation model, processor pool model or as a hybrid model. Right!
- In all cases some algorithm is needed for deciding which process will run on which processor.
- But before we learn actual algorithms let we know underlying assumptions that have been made for all algorithms.

Processor Allocation Continue…….

- All most all algorithms assume that the system is fully interconnected, that is every processor can communicate with every other processor.
- All processor allocation strategies or algorithms may imply either nonmigratory strategy or migratory strategy.
- If nonmigratory is in use then when process is created and at that time decision is taken about where to run the process. And once placed on particular machine then after if load on that machine increases then process can not be migrated or moved some where else. So process will remain there until it terminates.

Processor Allocation Continue…….

- Where as if migratory strategy is in use then even after execution of the process has started some where and suppose load on that machine gets increased then the process or processes can be moved or migrated some where else even.
- So migratory is more flexible than nonmigratory. And it also provides better load balancing. Right!
- Any processor allocation algorithm that assigns some processes to processers they eventually try to optimize some thing. Just like maximizing CPU utilization, response time or any other objective.

Design issues for processor allocation algorithms

1. Deterministic versus Heuristic algorithm
2. Centralized versus Distributed algorithm
3. Optimal versus Suboptimal algorithm
4. Local versus Global algorithm
5. Sender-initiated versus Receiver-initiated algorithm

1. Deterministic versus Heuristic algorithm

- Deterministic algorithms are appropriate when everything about process behavior is known in advance.
- If the computing requirement, file requirement and communication requirement of the process is known in advance then it is possible to make a perfect assignment.

1. Deterministic versus Heuristic algorithm continue….

- At the other extreme are the systems where the load is completely unpredictable.
- Requests for work depend on who is doing what, and can change dramatically from hour to hour, minute to minute.
- Processor allocation in such a system can not be done in a deterministic or mathematical way, but ad hoc techniques called heuristics.

2. Centralized versus Distributed algorithm continue…

- The second design issue is centralized versus distributed.
- Collecting all the information in one place allows a better decision to be made, but is less robust and can put a heavy load on the central machine. Which is something obvious.

3.Optimal versus Suboptimal algorithm

- Finding the optimal solutions are possible in both that is in centralized or decentralized but that are more expensive since it requires more and rigorous information to be obtained.
- But if suboptimal solution suppose acceptable according the application then it will be less expensive.

4.Local versus Global algorithm

- Here we are referring what is process transfer policy that means the decision that is taken for the process, to run it locally or on remote machine is taken based on just local information or the global information is also acquired.
- Local information means just observing the load of the local machine decision is made where as global information means load on other machines are also taken into consideration in the decision making.
- Only information on local machine if decision is taken then sometimes it may not be optimal decision. Why?

4.Local versus Global algorithm Continue……..

- Because suppose local load is more than the some predefined threshold value or load and decision might be taken to transfer the process somewhere else. But it may be that other workstations might be more overloaded than this one. So getting what do we mean…..!

5.Sender initiated versus Receiver initiated

- Some algorithm might be initiated by the machine which is overloaded and wants its processes to be run somewhere else.
- Where as some algorithm might be initiated by the idle machines. Like they broadcast the message like I am free, I can take or share your load to other workstations. So that way.
- So now we can start some example algorithms.

1. A Graph-Theoretic Deterministic Algorithm

- Since this algorithm is deterministic is appropriate for processes whose behavior is predictable. Right!
- If the number of CPUs *K* is smaller than the number of processes, several processes need to be assigned to each CPU.
- Here the idea is to perform the assignment such as to minimize network traffic.

1. A Graph-Theoretic Deterministic Algorithm

- Here system can be represented as a weighted graph, with each node being the process, and each arc representing the flow of messages between two processes.
- Mathematically now problem reduces to finding a way to partition the graph into *K* disjoint sub graphs such that it meets some constraints, here that constraint is to reduce network traffic. Why *K* disjoint sub graphs that u understand. Right!
- And the processes that are coming in single sub graph are placed on the same CPU. So communication becomes easy, so lesser load on n\w. right!

1. A Graph-Theoretic Deterministic Algorithm

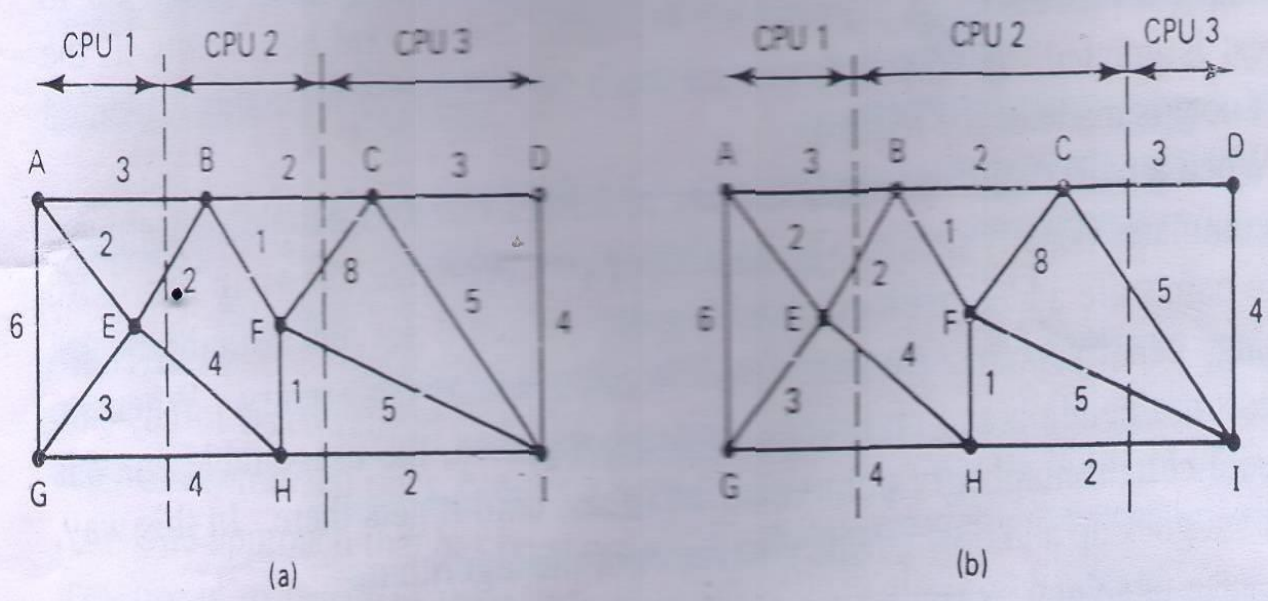


Fig. 4-17. Two ways of allocating nine processes to three processors.

1. A Graph-Theoretic Deterministic Algorithm

- For each solution that meets the constraints, arcs that are entirely within the single subgraph represent intramachine communication and can be ignored while calculating n\w traffic here.
- Arcs that go from one subgraph to another subgraph represent n\w traffic.
- The goal is now to partitioning that minimizes the network traffic.
- In figure we have shown two ways of partitioning the graphs.

1. A Graph-Theoretic Deterministic Algorithm

- Why does this way this algorithm works in practice? Because when we are plotting the weighted graph at that time logically we are clustering or combining the processes at one place that requires communication with each other and later on we are putting them on the same machine. Right!
- So now it is understandable.

2. A Centralized Algorithm OR Up-Down Algorithm

- The previous algorithm which we saw is having limited applicability because it is deterministic by nature so it can be used for the system where the information about the processes are available in advance. Right!
- But every time its not possible……
- So let we learn one heuristic algorithm that does not require information about the nature of the processes and it is a centralized or Up-Down algorithm.

2. A Centralized Algorithm OR Up-Down Algorithm

- Here it is centralized in the sense that coordinator maintains one usage table with one entry per personal workstation initially zero. Obviously!
- When the significant events happen, messages are sent to the coordinator for updating the table.
- Allocation decisions are based on the table.
- And the decisions are needed to be taken when some event occurs like when processor is requested or the processor becomes free.

2. A Centralized Algorithm OR Up-Down Algorithm

- The unusual thing about this algorithm and why it is centralized is that here instead of having the goal to maximize the CPU utilization , goal is to give each user the fair share of computing power.
- This algorithm works as follow.
- Here one thing should be keep in mind that when ever a processor or a workstation running its processes on other workstation or processors it gets per second fixed number of penalty points. And the workstations whose processes are pending to be run on some other machine will get point in minus that means credit. right!
- And this is actually the entry in the central usage table.

2. A Centralized Algorithm OR Up-Down Algorithm

- Now suppose some workstations are making requests for running its processes some where else ok! So this requests will be first sent to central coordinator which maintains the usage table so now the coordinator will make decision for deciding that which processor should be given first chance.
- And coordinator will make this decision based on this fact.
- It will check the usage table entry of each one of these and which ever workstations penalty points are least will be given the chance to run its processes on some other machine. Right!

2. A Centralized Algorithm OR Up-Down Algorithm

- Usage table entry can be positive, zero, or negative.
- A positive entry indicates that a workstation is using other workstation's computing power.
- A zero entry indicates the neutral state.
- And a negative entry indicates that this workstation is having many processes pending and it needs resources.
- So when decision needs to be made at that time the workstation having least score will always beat the workstations having more penalty points.

2. A Centralized Algorithm OR Up-Down Algorithm

- So here attempt is to not to let any single user dominate the system resources.
- And each user should get fair share of computing power.

2. A Centralized Algorithm OR Up-Down Algorithm

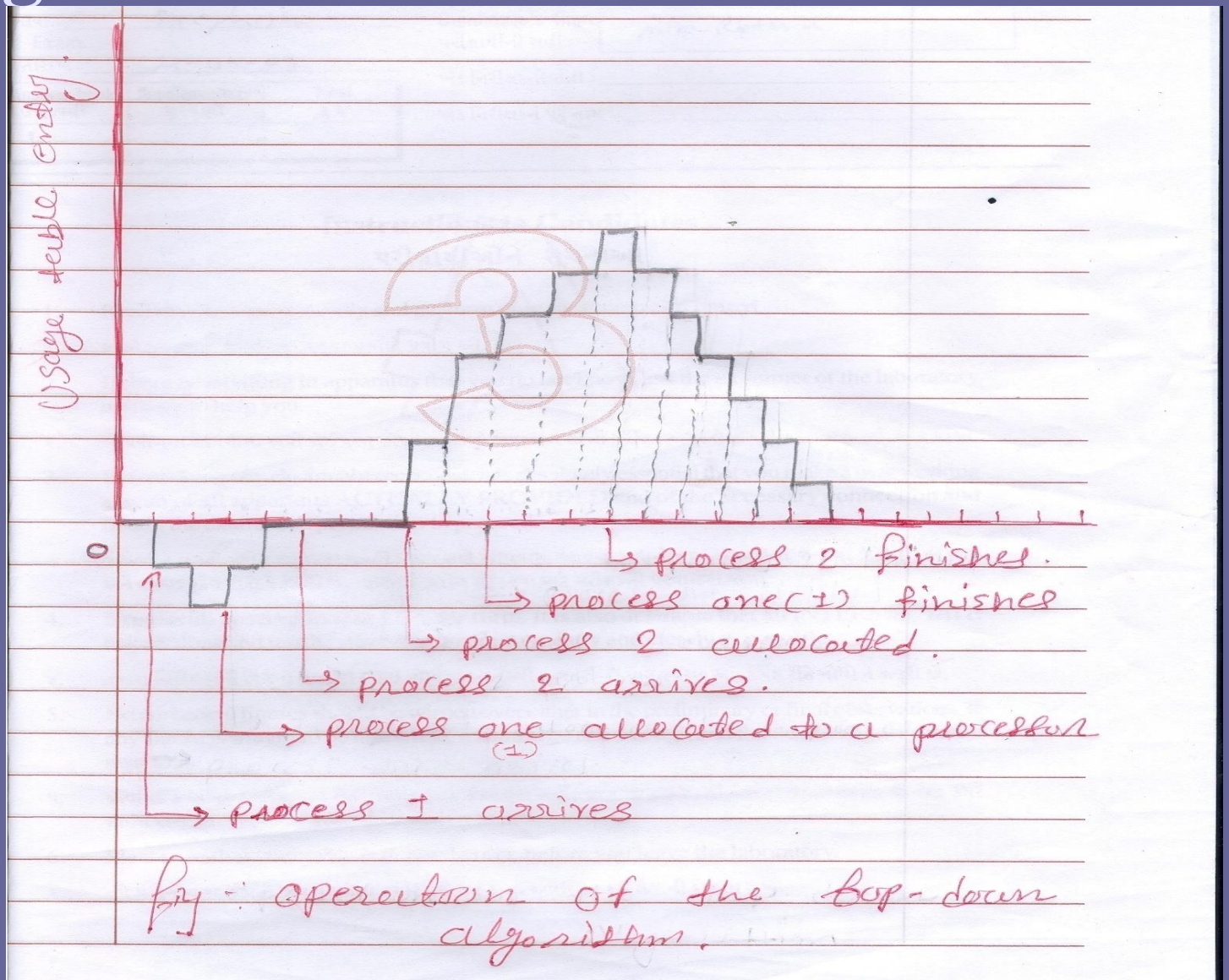


Fig: operation of the top-down algorithm.

3.A Hierarchical Algorithm

- Centralized algorithm such as Up-Down do not scale well for the large system because it results in heavy load on that central component which is managing this algorithm. Right!
- One approach that has been proposed for keeping tabs on a collection of processors is to organize them in a logical hierarchy independent on the logical structure.
- For each group of K workers, one manager machine (the Department head) is assigned the task of keeping track of who is busy and who is idle.

3.A Hierarchical Algorithm

- If the system is large then there will be more departmental heads, so some machines will function as a dean. And each governing some number of this departmental heads. And this way it can be extended.
- An obvious question is that what happens when a department head stops functioning? One solution is to promote one of the direct subordinates of the faulty manager to fill in for the boss.
- To avoid having the single authority at top we can truncate the tree so that now supreme authority is not single. Means the failure of one does not bring entire system down. Getting……

3.A Hierarchical Algorithm

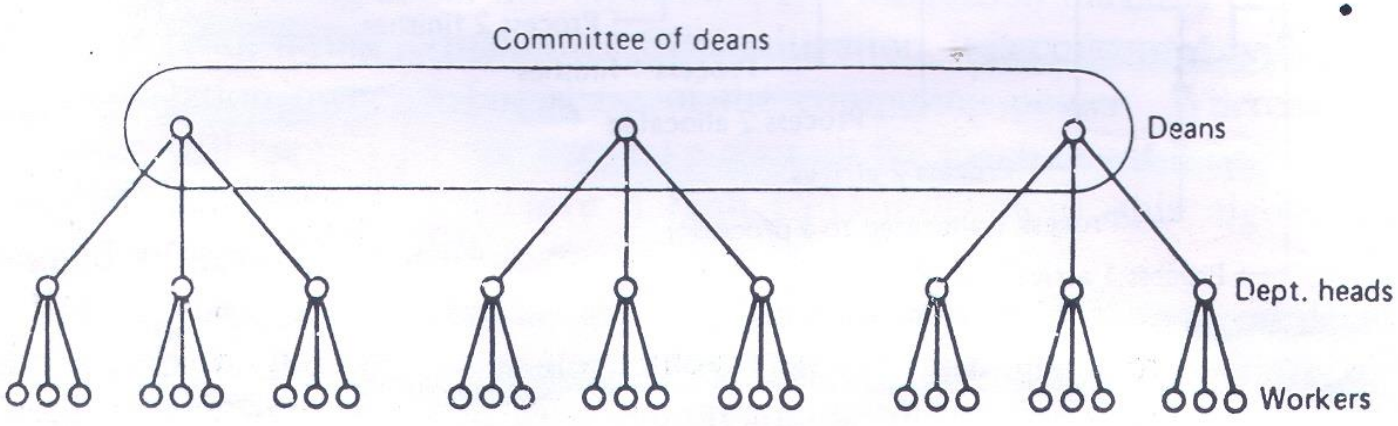


Fig. 4-19. A processor hierarchy can be modeled as an organizational hierarchy.

3.A Hierarchical Algorithm

- So here when one member malfunctions or crashes the remaining members promote someone one level down as replacement. Right!
- So we can say that this system is self repairing and can survive occasional crashes of both workers and managers and manages without any long-term effects.

4.A sender initiated Distributed Algorithm

- So far the algorithm that we have seen, are all centralized or semi-centralized. But in practice distributed algorithm also exits.
- Here the machine on which the process has been created and wants to run or send it somewhere else , asks the probing question or inquiry question to some randomly selected machine. If that machine's load is below some threshold value then it accepts that process otherwise denies.
- And if it denies then some other machine is again randomly chosen and inquiry is made.

4.A sender initiated Distributed Algorithm

- And this algorithm is repeated for N number of times.
- If the suitable environment is not found in N probes then decision is made to run the process where it was originated.

5.A receiver initiated Distributed Algorithm

- The previous algorithm is initiated by Overloaded sender where as complementary algorithm can be the one where underloaded receiver initiates the process.
- So here when on particular machine process finishes so that underloaded machine randomly selects one machine and asks for work.
- If it does not have work to give then it selects second machine and so on……
- It continues this process for N probes and then stops.

5.A receiver initiated Distributed Algorithm

- Again when second process finishes then it again initiates this algorithm or process.

6. Bidding Algorithm

- Another class of algorithms tries to turn the computer system a miniature economy.
- With buyers and sellers of services and prices set by supply and demand.
- The key players in the economy are the processes, which must buy CPU time to get their work done, and processors which auction their cycles off to the highest bidder.

6. Bidding Algorithm

- Each processor advertises its approximate price by putting it in a publicly readable file.
- This price is not guaranteed, but gives an indication of what the service is worth.
- Different processors may have different prices, depending on their speed, memory size, presence of floating point hardware, and other features.
- An indication of the service provided, such as expected response time can also be published.

6. Bidding Algorithm

- When the process wants to start up a child process, it goes around and checks who is currently offering the service that it needs.
- It then determines the set of processors whose services it can afford.
- From this set, it computes the best candidate, where best mean cheapest, fastest, or best price/performance, depending on the application.

6. Bidding Algorithm

- It then generates the bid and sends the bid to its first choice.
- The bid may be higher or lower than the advertised price.
- Processors collects all the bids sent to them, and make a choice, presumably by picking the highest one.
- The winners and losers are informed, and the winning process is executed. The published price of the server is then updated to reflect the new going rate.

Distributed File System

- A key component of any distributed system is the file system.
- As an single processor systems, in distributed systems the job the file system is to store programs and data and make them available as needed.
- Many aspects of the distributed file systems are similar to conventional file system, which we are familiar to.
- First of all it is necessary to keep in mind that it is important to distinguish between the concept of the file service and the file server.

Distributed File System

- The file service is the specification of what the file system offers to its clients.
- It describes the primitive available, what parameters they take, and what action they perform.
- To the clients file service defines precisely what service they can count on, but says nothing about how it is implemented.
- In effect, the file service specifies the file system's interface to the clients.

Distributed File System

- A file server in contrast is the process that runs on some machine and helps implementing file service.
- A system may have one file server or several.
- In particular, users should not know how many file servers there are and what the location of these file servers.
- All they need to know is that when they call the particular procedure specified in the file service, the required work is performed somehow and the required result are returned.

Distributed File System

- Ideally it should look like normal single processor file system.
- Since a file server is normally just a user process running on some machine, a system may contain multiple file servers and each offering even different file service.
- For example a distributed system might have two file servers that offers UNIX file service and MS-DOS file service. So this way there will be multiple windows on the single terminal.

Distributed File System Design

- A distributed file system has two components.
    1. The true file service.
    2. The directory service.
- The former is concern with operations on the individual files, such as reading writing and appending.
- Whereas latter is concerned with creating and managing directories, adding and deleting files from the directories, and so on.

1.File service interface

- For any file service, whether it is a single processor or distributed system, the most fundamental issue is : What is a file?
- In many systems such as UNIX and MS-DOS, a file is an uninterrupted sequence of bytes.
- On mainframes, however many types of files exists each with different properties. Here file can be structured as a sequence of records.

1.File service interface

- Files can have attributes, which are pieces of information about the file, but which are not the part of the file itself.
- Typical attributes are the owner, size, creation date, and access permissions.
- The file service usually provides primitives to read and write some of the attributes.
- For example it may be possible to change the access permission but not the size.

1.File service interface

- Another aspect of file model is whether files can be modified after they have been created. Both are possible.
- Once the file has been created and if it can not be changed then such a file is called immutable.
- Having files immutable it makes much easier to support file caching and replication. Which is something obvious! Because it eliminates the problems associated with having to updates all copies of a file whenever it changes.

1.File service interface

- Another aspect is protection. Which uses same techniques as in single-processor systems. That is capabilities and access control list.
- With capabilities each user has a kind of ticket, called a capability, for each object to which it has access.
- The capabilities specifies the kind of access the user possesses whether reading or writing or both.
- All access control list schemes associate with each file a list of users who may access the file and how. Just like in unix operating system.

1.File service interface

- File services can be split into two types. Depending on whether they support upload/download model or a remote access model .
- In the next slide figure (a) shows Upload/download model and (b) shows Remote access model.
- In the Upload/download model file service provides only two major operations.
- And that is read file and write file.
- In reading operation entire file is transferred from server to client machines.

1.File service interface

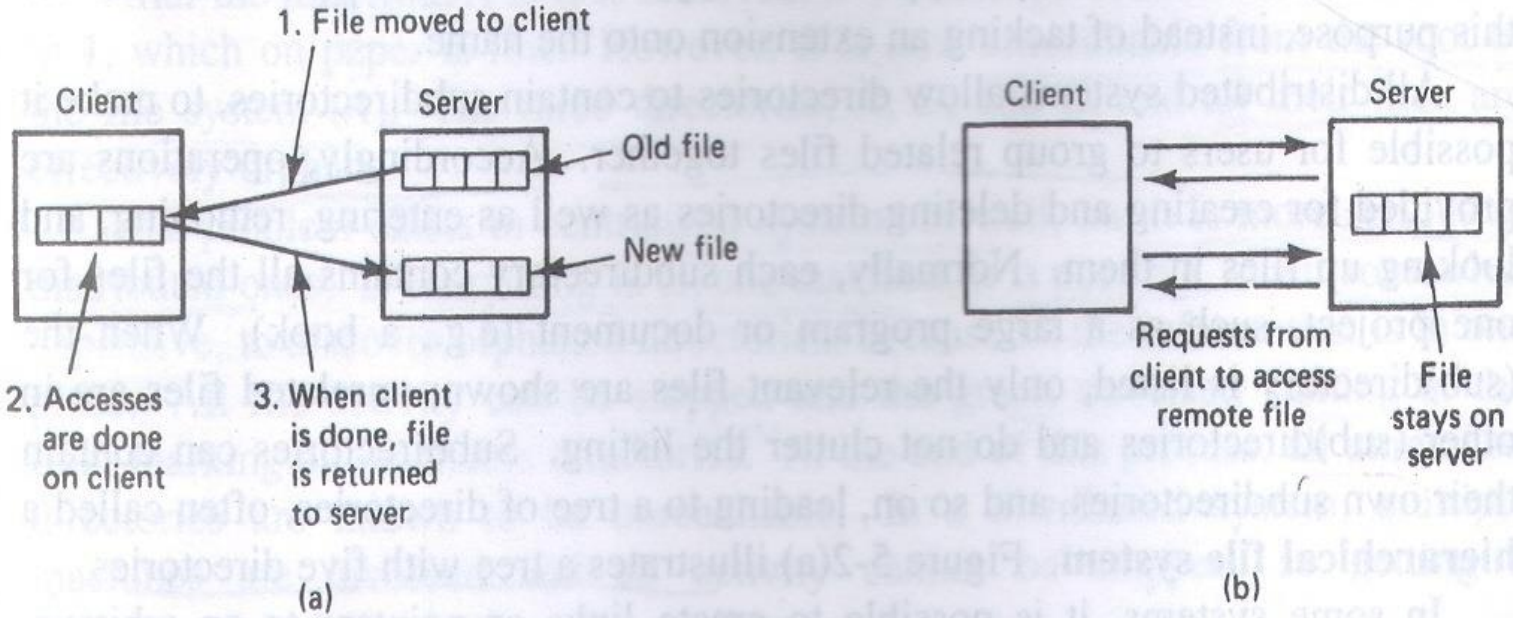


Fig. 5-1. (a) The upload/download model. (b) The remote access model.

The advantage of the upload/download model is its conceptual simplicity. Application programs fetch the files they need, then use them locally. Any

1.File service interface

- And in writing operation entire file is transferred from client machine to server machine.
- So basic operation is to move entire file to either direction.
- Advantage of this upload/download model is simplicity.
- However disadvantage is that enough storage is needed on local side.

1.File service interface

- Whereas in remote access model file service provides large number of operations for opening and closing files, reading and writing parts of files, moving around within the files, examining and changing file attributes and so on. Unlike upload/download model which requires physical transfer of files.
- Advantage of using remote access model is that local storage is not wasted and there is no need for much space local side.

2. The Directory service

- The other part of the file service is the directory service.
- Which provides operations for creating and deleting directories, naming and renaming files, and moving them from one directory to another.
- The nature of the directory service does not depend on the fact that upload/download model is in use or remote access model.
- The directory service defines some alphabet and syntax for composing file , directory names.
- Some system divide file names into two parts.

2. The Directory service

- Usually separated by period. Such as pro.c for C program or man.txt for text file. The second part is called extension.
- All distributed file systems allow directories to contain subdirectories to make it possible for user to group related data together. And subdirectory even may contain other subdirectories and so on. Leading to a tree of directories called hierarchical file system.
- In the next slide figure(a) shows the same.

2. The Directory service

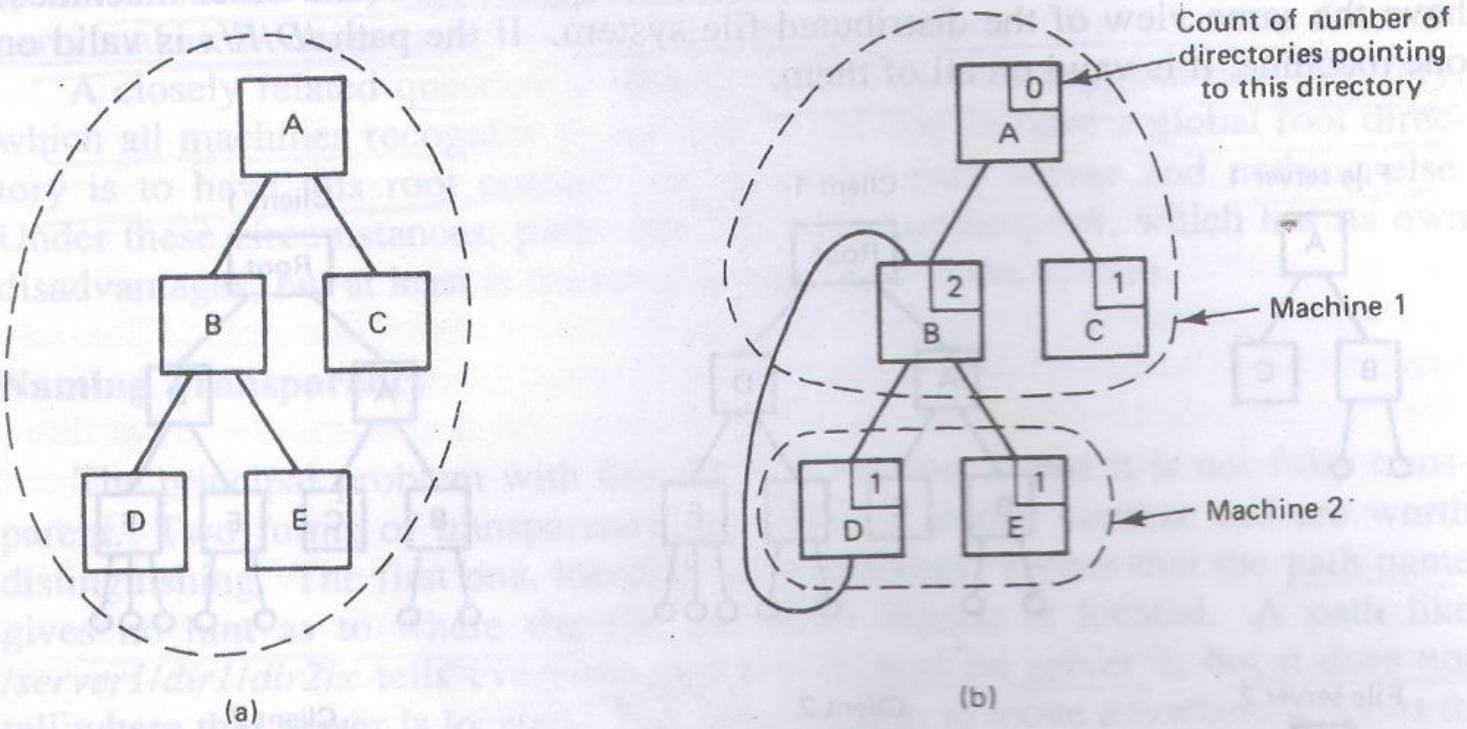


Fig. 5-2. (a) A directory tree contained on one machine. (b) A directory graph on two machines.

2. The Directory service

- In some systems, it is possible to create links or pointers to an arbitrary directory.
- And these can be put in any directory, making it possible to create not only trees but directory graph also.
- The deference between tree structure and graph structure is important.
- One of the difficulty can be seen from the aspect of nature difficulty.

2. The Directory service

- Which has been shown in previous slide in figure(b) (Graph structure).
- In this figure directory D has a link to directory B. The problem occurs when link from A to B is removed.
- In tree structure it is like if directory is empty then link can be removed means can be deleted so there is no problem.
- But in graph a link can be removed if at list one other link is present.

2. The Directory service

- After the link from A to B is removed, B's reference count reduces to 1 from 2, which on paper is still fine, but what will happen is directories B, D, and E, and all their files are orphans.
- This problem(orphons) will also be there on centralized system as well if graph structure has been employed, but it is more severe on distributed system because on centralized one all orphan directories are at least on single machine so locating is still easy, but on distributed they might be on different - different machine so it is more severe so difficulty is understandable.

2. The Directory service

- Another key design issue in designing distributed file system is that clients have the same view(figure (b)) of file system or different-different views (figure (c))are allowed.
- Views have been shown in next slide.
- If same view is there then valid path on one machine is also valid on another machine as well.
- But if different-different views are allowed then valid path on one machine is not valid on another.

2. The Directory service

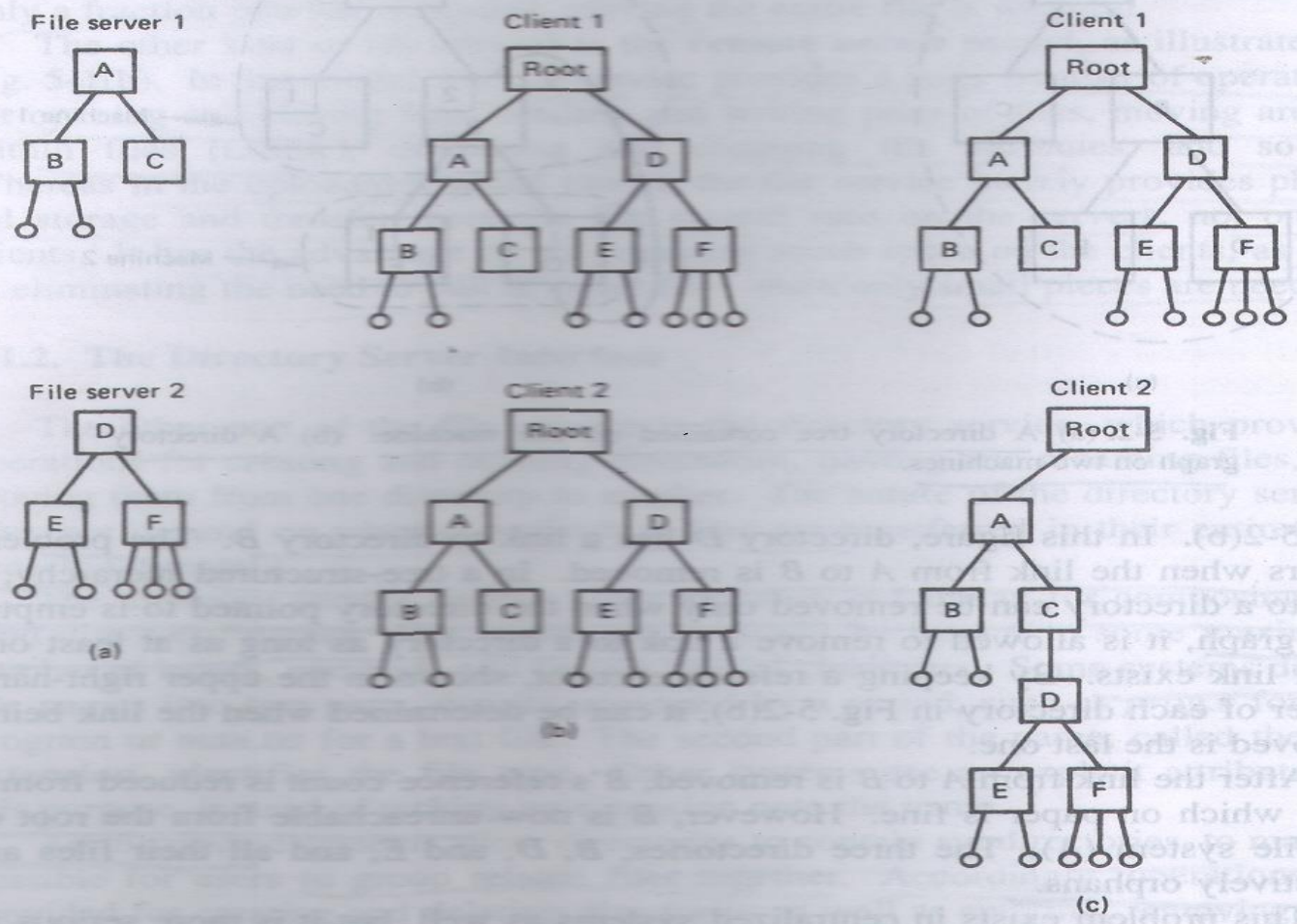


Fig. 5-3. (a) Two file servers. The squares are directories and the circles are files. (b) A system in which all clients have the same view of the file system. (c) A system in which different clients may have different views of the file system.

Synchronization In Distributed System

- Synchronization in distributed system is more complicated than centralized one because synchronization has to do with time that means event A occur before event B or vice versa.
- So if two processes are interacting on the same machine then there is no problem because both processes are following the same clock time but that is not the same case with distributed systems.
- Because there are many machines present which do not share same memory and one process might be running on one machine and another on other machine.

Synchronization In Distributed System

- So here it is bit more difficult to achieve synchronization. Because all processes follow their local clocks. Which is some thing obvious!
- To understand this imagine the following situation.
- Suppose that on centralized system or on some PC editor for C program and compiler runs so here we do not have synchronization problem because say, output.o will have higher time than output.c since they both follow the same clock. And if output.c has higher time than output.o then compiler knows that output.c must have been modified so it needs to be recompiled.

Synchronization In Distributed System

- But here in distributed system is not the same case.
- Consider the following figure.

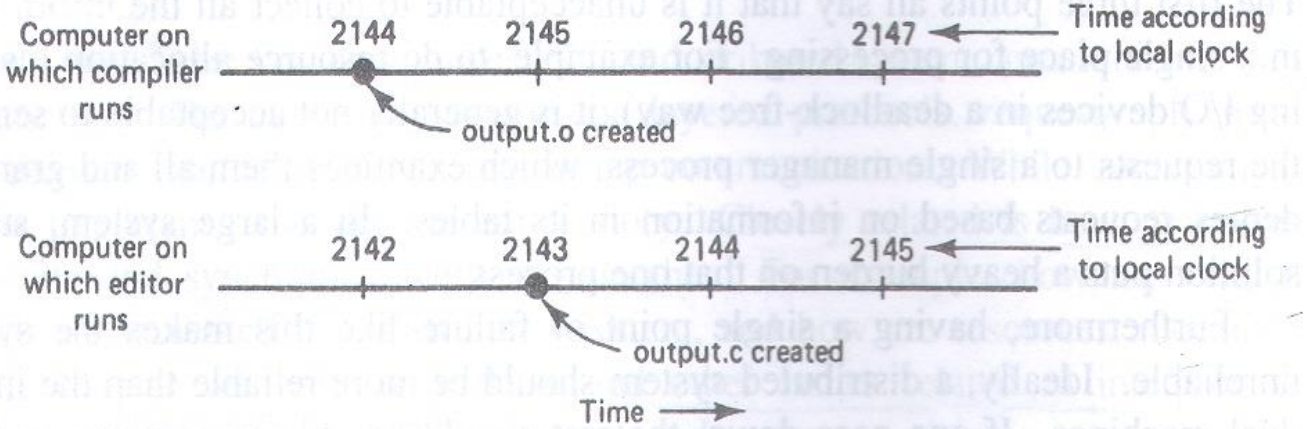


Fig. 3-1. When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Synchronization In Distributed System

- U can see that on one machine editor runs and on other compiler runs.
- U can also see that output.c has been created or has been modified after the output.o had been created. But since output.c has been assigned time lesser than output.o because clock on that machine runs bit slower than the machine on which compiler runs. So compiler will not recompile the modified source file and programmer will go crazy finding what is wrong with code.

Synchronization In Distributed System

- So here we should opt for different technique.
- We will shortly discuss it but before that some basic terms we should see.
- Clock skew: The difference in time values, crystals in different computers run at slightly different rates. Each computer has its own clock.
- Logical Clock: For certain class of algorithms, it is the internal consistency of clock that matters, not whether they are particularly close to the real time. For many purposes it is sufficient that all machines agree on same time. This is called logical clock.

Synchronization In Distributed System

- Physical Clock: When the additional constraint is present that the clock must not only be the same but also must not deviate from the real time, by more than a certain amount than clocks are called physical clocks.
- To Synchronize logical clocks, Lamport defined a relation called "happens before".
- Here latent(Underlying) idea is that rather than time here we concern with whether event A occur before event B or not so that processes can be synchronized or cooperated in a true sense.

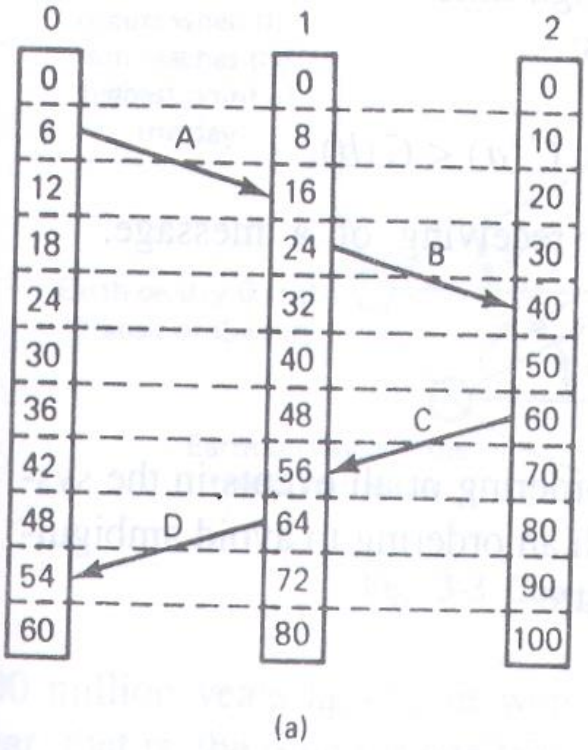
Synchronization In Distributed System

- So lamport's algorithm says that as far as u follow this algorithm u will always find that clocks are logically synchronized so it will always state true that whether event A occur first or event B occur first.
- Here expression a $\longrightarrow$ b is read as event a happens before event b.
- 1. If a and b are the events in the same process, and event a occurs before event b then a$\longrightarrow$b is true.
- 2. If a is the event of a message being sent by one process and b is the event of the message being received by another process then a$\longrightarrow$b is also true.

Synchronization In Distributed System

- Because message can not be received before it is sent, or even at the same time it is sent, since it takes a finite amount of time to arrive.
- Consider figure in the next slide.

Synchronization In Distributed System



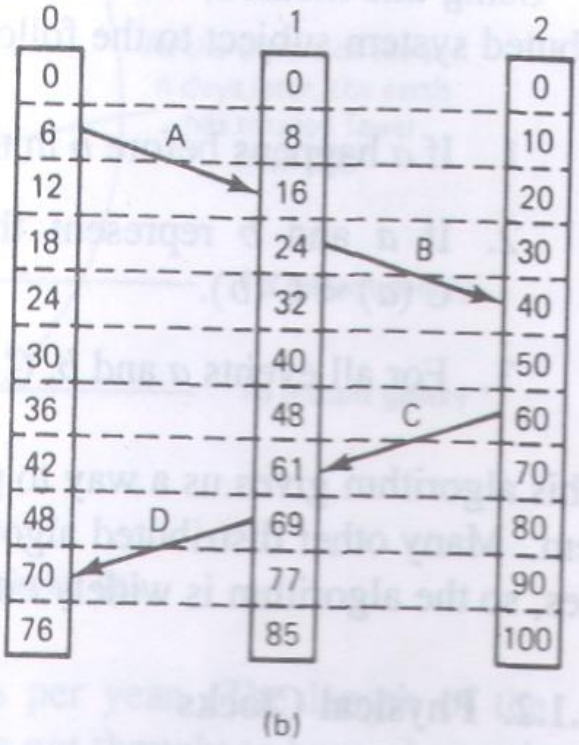


Fig. 3-2. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

Synchronization In Distributed System

- Consider the three processes depicted in the figure(a).
- The processes run on different machines, each with its on clock, running at its own speed.
- U should also see that when on machine 0 time tick is 6 at that time on machine 1 time tick is 8 and on machine 2 it is 10. u can also consider 0,1 and 2 as processes here.
- At time 6 process 0 sends message A to process 1. so here at tick 16 in process 1 it arrives means it takes 10 ticks to arrive. So no problem here sending process has lesser time than receiving process.

Synchronization In Distributed System

- And u can also see that message B from 1 to 2 takes 16 ticks so again its valid.
- But just see figure(a) again and just see the fun that comes around, message C from 2 to1 leaves at 60 and arrives at 56. similarly message D from 1 to 0 leaves at 64 and arrives at 54. these values are clearly impossible.
- Lamport's solution follows directly from the happens before relation.
- Since C left at 60 it must arrive at 61 or later. Therefore each message carries the sending time, according to the senders clock.

Synchronization In Distributed System

- When a message arrives and the receiver's clock shows a value prior to the time the message was sent, the receiver fast forwards it's clock to be one more than the sending time. Here in figure(b) we can see that C now arrives at 61. similarly D arrives at 70.

Unit # 3 Parallel Processing

Parallel Processing: Basic concepts

- **Parallel processing** is information processing that emphasizes the concurrent manipulation of data elements belonging to one or more processes ***solving a single problem.***
- A **parallel computer** is a multiple processor computer capable of parallel processing.
- A **supercomputer** is a general purpose computer capable of solving individual problems at extremely high speeds.
- All contemporary supercomputers are parallel computers

Continue…….

- The **throughput** of the device is the number of results it produces **per unit time**.
- There are many ways we can improve the **throughput of the device.**
- **Speed** can be increased or **concurrency** may be implemented means – **Number of operations that are being performed at any one time**.
- **Pipelining and Data parallelism** are two ways to **increase concurrency** of the computation.

Continue…….

- A **pipelined computation** is divided into a number of steps, called **segments** or **stages.**
- Each segment works at full speed on particular part of the computation.
- The output of the one segment is the input of the next segment.
- **Data parallelism** is the use of multiple functional units to apply the **same operation simultaneously to elements of the dataset.**

Continue…….

- A k-fold increase in the number of functional units leads to a k fold increase in the throughput of the system.
- **Speedup** is the ratio between the time needed for the most efficient sequential algorithm to perform a computation and the time needed to perform the same computation on the machine pipelining or parallelism.
- **Contrasting pipelining and data parallelism.**

Contrasting pipelining and data parallelism.

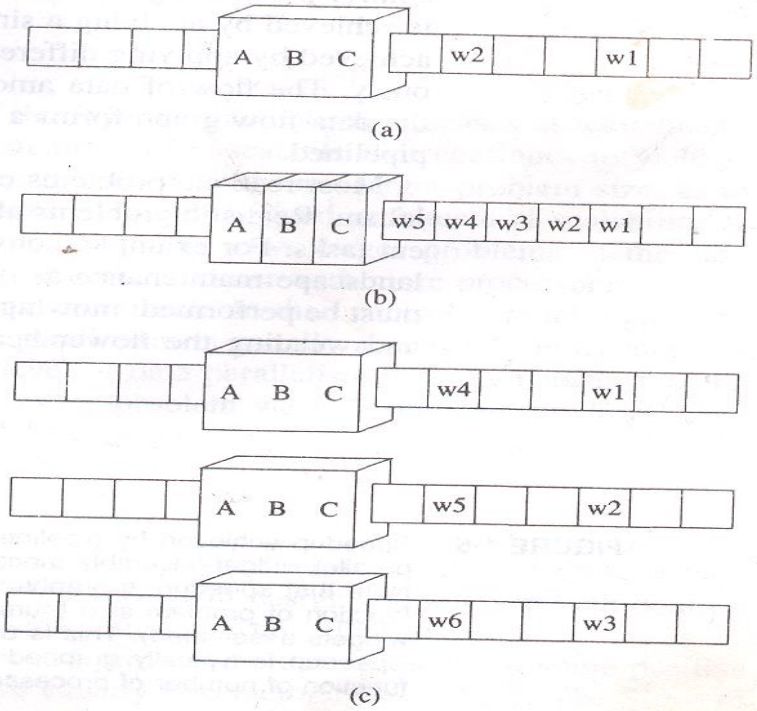


FIGURE 1-4 Three methods to assemble widgets. (a) A sequential widget-assembly machine produces one widget every three units of time. (b) A three-segment pipelined widget-assembly machine produces the first widget in three units of time and successive widgets every time unit thereafter. (c) A three-way data-parallel widget-assembly machine produces three widgets every three units of time.

Processor organization for Parallel Computation

- This section defines important processor organizations – methods of connecting processors in a parallel computer.
- A processor organization can be represented by a graph in which the nodes (vertices) represent processors and the edges represent communication lines between pairs of processors.
- We evaluate these processor organizations according to criteria that help us understand their effectiveness in implementing efficient parallel algorithms on real hardware.

This criteria are:

- Diameter: The diameter of a network is the largest distance between two nodes. Low diameter is better, because the diameter puts a lower bound on the complexity of parallel algorithms requiring communication between arbitrary pairs of nodes.
- Bisection width: Bisection width of a network is the minimum number of edges that must be removed in order to divide network into two halves. High bisection width is better, because in algorithms requiring large amounts of data movement, the size of the data set divided by the bisection width puts a lower bound on the complexity of the parallel algorithm.

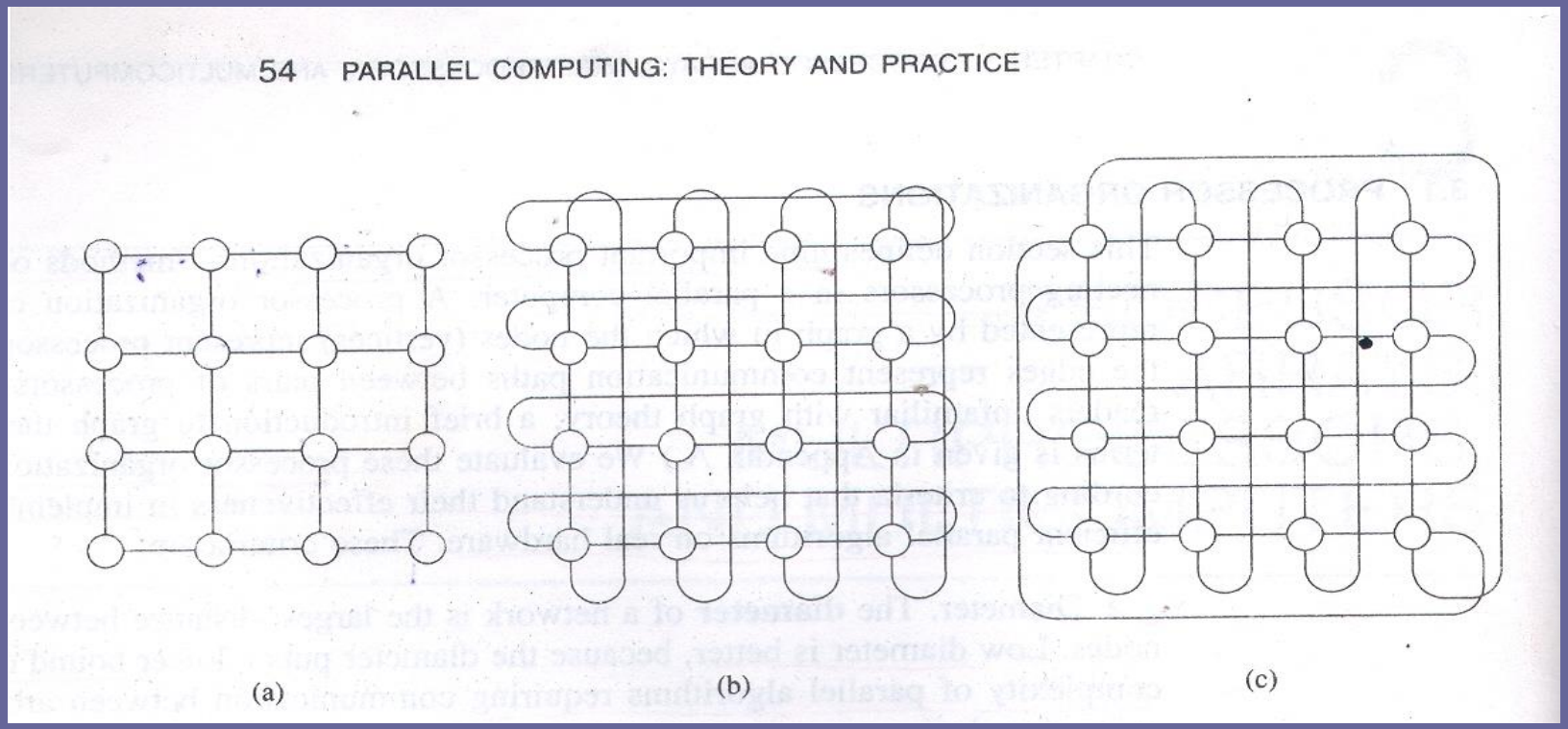
This criteria are: Continue……

- Number of edges per node. It is best if the number of edges per node is a constant independent of the network size, because then the processor organization scales more easily to systems with large numbers of nodes.
- Maximum edge length. For scalability reasons it is best if the nodes and edges of the network can be laid out in three-dimensional space so that the maximum edge length is a constant independent of network size.

1. Mesh Networks

- In a mesh network, the nodes are arranged into a q-dimensional lattice.
- Communication is allowed only between neighboring nodes: hence interior nodes communicate with 2q other processors.
- In the next slide figure (a) shows two-dimensional mesh(2 – D). Some variants of the mesh model have been shown in figure (b) and (c). Wrap-around connection in the same row or column figure (b). And wrap-around connection in the adjacent row or column figure (c).

1. Mesh Networks Continue…..



Figure

1. Mesh Networks Continue…..

- Let's evaluate the mesh network according to our four criteria. Assume there is no wrap-around connection.
- The diameter of a q-dimensional mesh with k^q nodes is q(k-1).
- The bisection width of a q-dimensional mesh with k^q nodes is k^{q-1} .
- The maximum number of edges per node is 2q.
- The maximum edge length is constant , independent of number of nodes, for two and three dimensional meshes.

2.Binary Tree Network

- In a binary tree network the 2^{k-1} nodes are arranged into a complete binary tree of depth k-1.
- A node has at most three links.
- Every interior node can communicate with its two children and every node other than the root can communicate with its parent.
- Next slide shows Binary tree network of size 15 and depth 3.

2.Binary Tree Network continue….

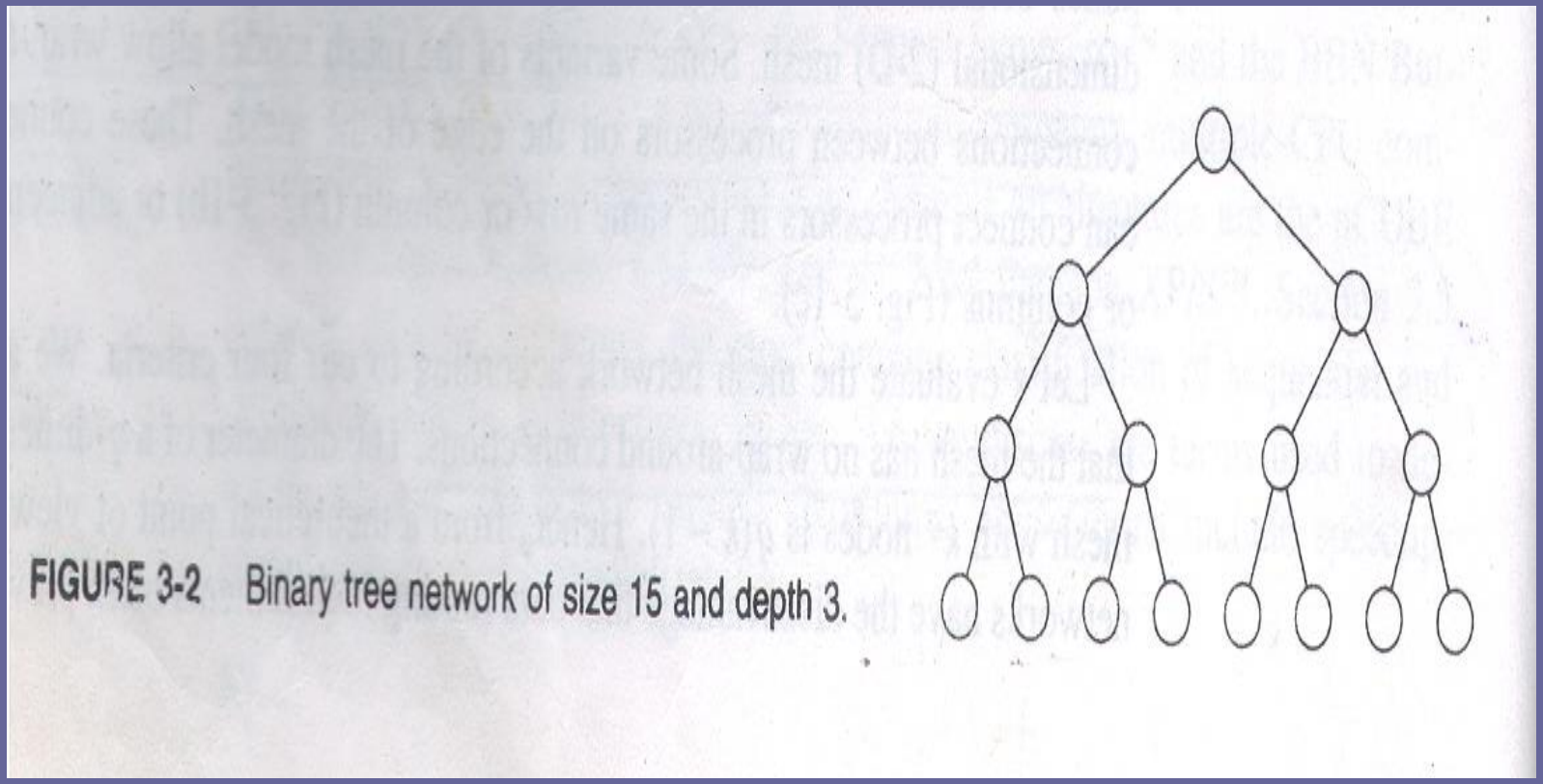


FIGURE 3-2 Binary tree network of size 15 and depth 3.

Figure

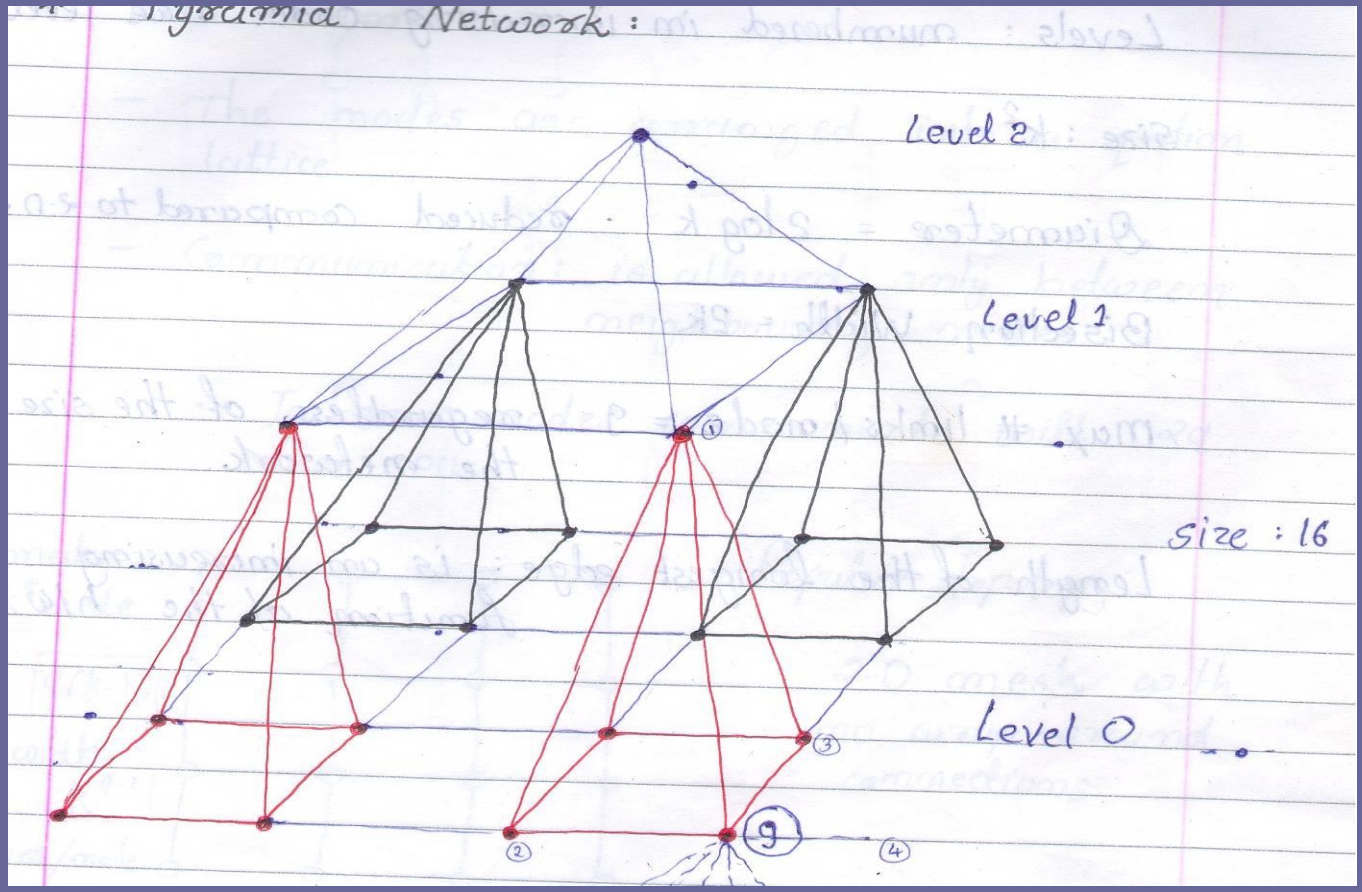
2.Binary Tree Network continue….

- The binary tree has low diameter, 2(k-1), but has poor bisection width of one.
- It is impossible to arrange the nodes of a binary tree in three-dimensional space such that as the number of nodes increases. The length of the longest edge is always less than a specified constant.

3.Pyramid Network

- The pyramid network can be seen as an attempt to combine the advantages of mesh networks with those of tree networks.
- A pyramid network of size k^2 is a complete 4-ary rooted tree of height $\log_2 k$ augmented with additional interprocessor links so that the processors in every tree level form a 2-D mesh network.
- The pyramid of size k^2 has at its base a 2-D mesh network containing k^2 processors.

3.Pyramid Network Continue…..



Figure

3.Pyramid Network Continue…

- The total number of processors of size k^2 is $(4/3)\ k^2 - (1/3)$.
- The levels of the pyramid are numbered in ascending order such that the base has level number 0.
- Every interior node or processor is connected to nine other processors: One parent, Four neighbors, and Four children.
- Previous slide shows the pyramid of size 16.
- The diameter of a pyramid of size k^2 is $2\log k$. which is better than the 2-D mesh.

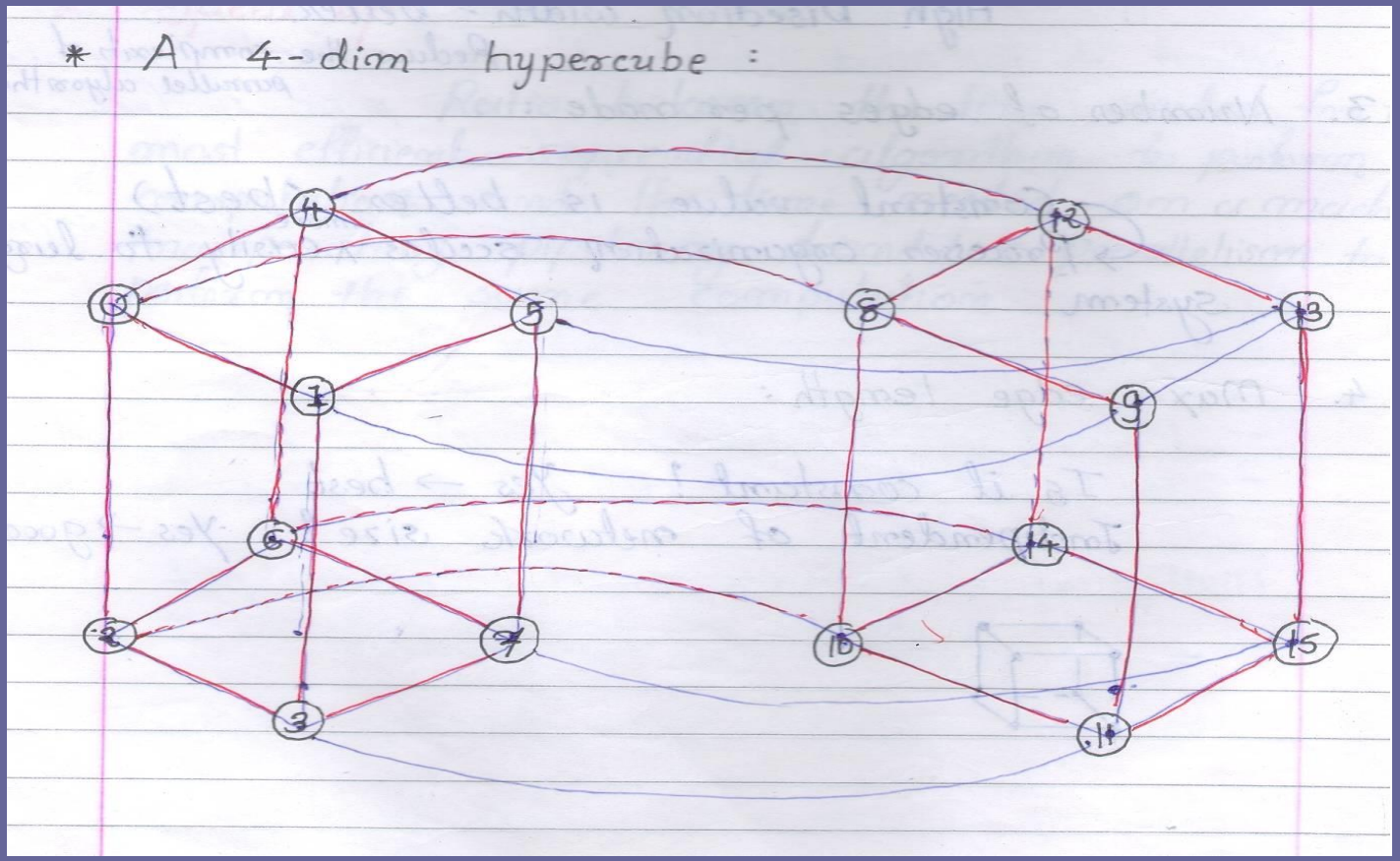
3.Pyramid Network Continue…

- The bisection width of a pyramid of size k^2 is 2k.
- Maximum number of links per node is no longer than nine, regardless of the size of the network.
- Unlike the 2-D mesh length of the longest edge in the pyramid network is an increasing function of the network size.

4.A 4 Dimensional Hypercube

- Consists of 2k nodes.
- Forming a k-dimensional hypercube.
- The nodes are labeled 0,1,2…….,2^{k-1} .
- Next slide shows the figure.

4.A 4 Dimensional Hypercube



Figure

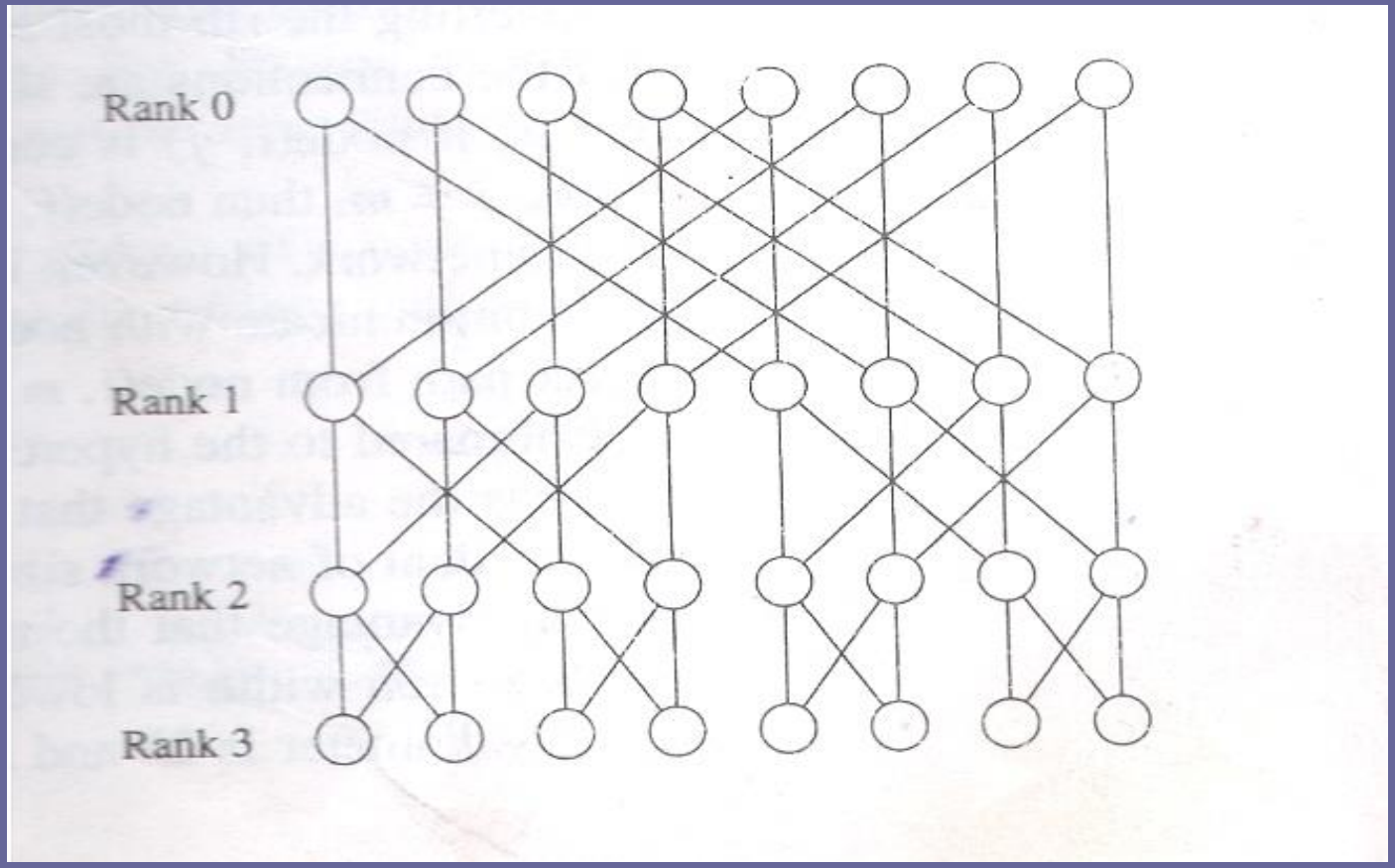
4.A 4 Dimensional Hypercube

- Diameter : k
- Bisection width : 2^{k-1}
- # of edges / node : k
- Length of the longest edged increases as the size of the network.
- In particular in 4 – dim hypercube:
  - Consists of 2^4 node = 16 means k=4 here.
  - Diameter : 4
  - Bisection width $2^{4-1} = 2^3 = 8$
  - # edges/node = 4 its constant.
  - Max edge-length : not constant

5.Butterfly Network

- Consists of $(k+1)2^k$ nodes
- (k+1) rows or ranks
- Each containing $n = 2^k$ nodes
- Diameter : 2k
- Bisection width : 2^k
- Length of the longest network edges : Increases as the # of nodes increases
- Constant # edges : yes
- Constant edge's length : no

5.Butterfly Network



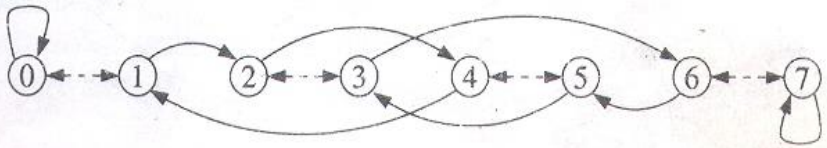
Figure

6.Shuffle-Exchange Network

- A shuffle-exchange network consists of $n = 2^k$ nodes, numbered 0,1…..,n-1, and two kinds of connections, called shuffle and exchange.
- Exchange connections link pairs of nodes whose numbers differ in their least significant bit.
- The perfect shuffle connection links node i with node 2i modulo (n -1), with the exception that node n – 1 is connected to itself.
- Consider the figure in the next slide which shows drawing for an eight-node shuffle-exchange network.

6.Shuffle-Exchange Network

FIGURE 3-8 Shuffle-exchange network with eight nodes. Solid arrows denote shuffle connections. Dashed arrows denote exchange connections.



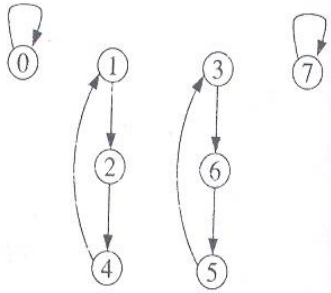


FIGURE 3-9 Necklaces of the shuffle-exchange network with eight nodes.

6.Shuffle-Exchange Network

- Diameter : 2k -1
- Bisection width >= 2^{k-1} / k
- Constant # edges : yes
- Constant edge length : No
- Let a_{k-1} a_{k-2}……a_2 a_1 a0 be the address of a node before performing the shuffle operation (for understanding purpose consider addresses in binary notation).
- Then new address after shuffle operation

a_{k-2} , a_{k-3}……a_1,a_0,a_{k-1} [left cycle rotation]

6.Shuffle-Exchange Network

- U can also see that how necklace is formed here.
- The nodes through which a data items beginning at address i travels in response to a sequence of shuffles are called the necklace of i .
- No necklace is larger than k and a necklace shorter than k is called short necklace.
- Every node in a shuffle-exchange network has two outgoing and two incoming links.

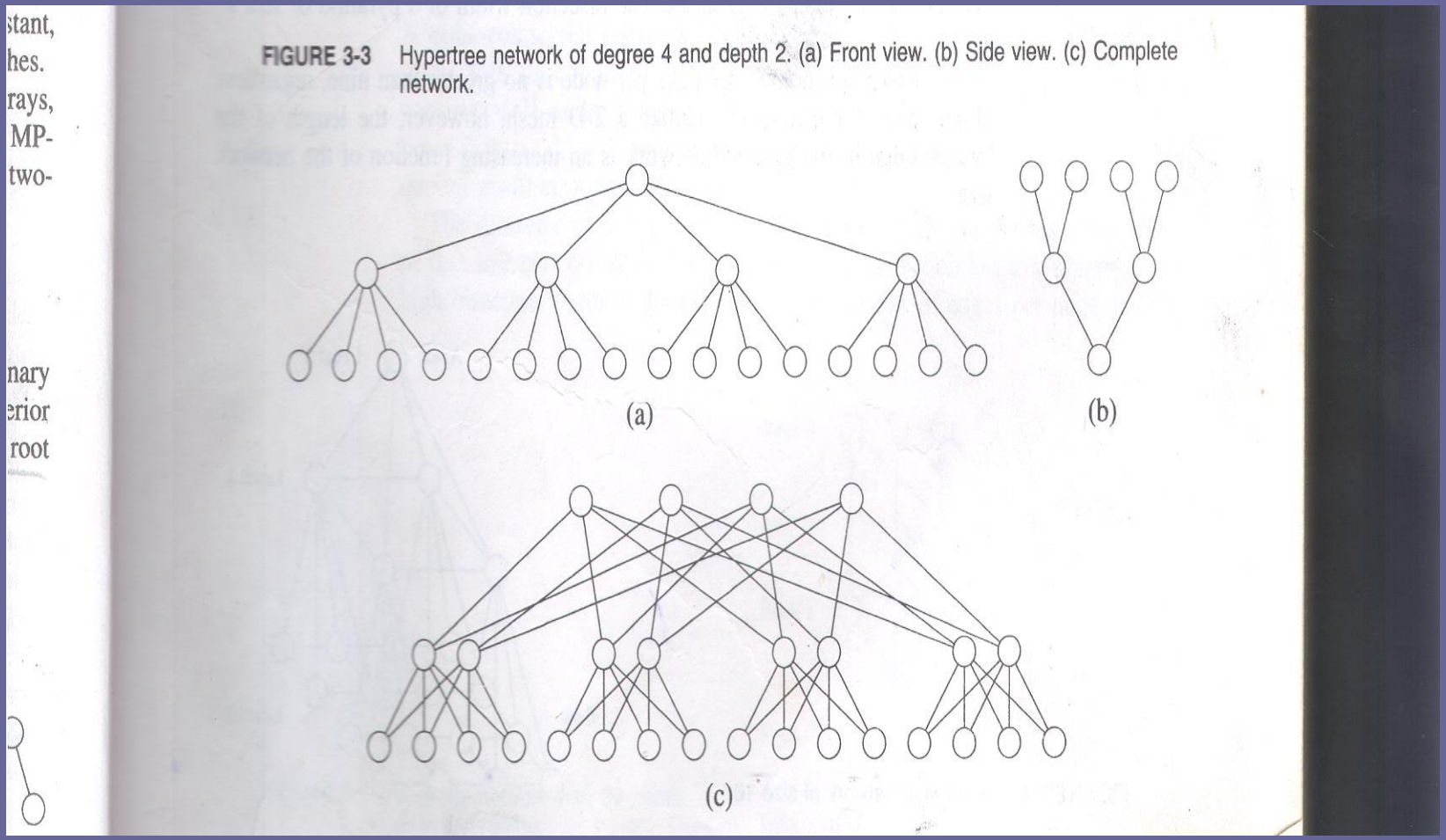
7.Hypertree Networks

- A hypertree represents one approach to building a network with the low diameter of a binary tree but an improved bisection width.
- The easiest way to think of a hypertree network of degree k and depth d is to consider the network from the two aspect.
- From the front a hypertree network of degree k and depth d looks like a complete k-ary tree of height d. (figure 3.3(a))
- From side the same hypertree network looks like an upside down binary tree of height d.(figure 3.3(b)).

7.Hypertree Networks

stant,
hes.
rays,
MP-
two-

nary
erior
root



figure

7.Hypertree Networks

- Joining the front and side views yields the complete network of degree 4 height 2 (figure-3.3(c)).
- A 4-ary hypertree with depth d has 4^d leaves and $2^d(2^{d+1} – 1)$ nodes in all.
- Diameter is 2d and its bisection width is 2^{d+1}.
- The number of edges per node is never more than six and the maximum edge length is an increasing function of the problem size.

Odd-even transposition Sort

- This one is one parallel sorting algorithm.
- It has been designed for the processor array model in which processing elements are organized into a one-dimensional mesh.
- Assume that,

  A = $(a_0,a_1,\ldots\ldots a_{n-1})$ is the set of n-elements to be sorted.
- The algorithm performs n/2 iterations.
- Each iteration has 2 phases.

Odd-even transposition Sort

- In the first phase called odd-even exchange the value of a in every odd-numbered processor (except for n-1) is compared with the value of a stored in the successor processor.
- The values are exchanged if necessary, so that the lower-numbered processor contains the smaller value.
- In the second phase, called even-odd exchange the value of a in every even-numbered processor is compared with the value of a in the successor processor.
- The values are exchanged if necessory.

Algorithm

ODD-EVEN TRANSPOSITION SORT (ONE-DIMENSIONAL MESH PROCESSOR ARRAY):

```
Parameter   n
Global      i
Local       a   {Element to be sorted}
            t   {Element taken from adjacent processor}
begin
  for i ← 1 to n/2 do
    for all P_j, where 0 ≤ j ≤ n − 1 do
      if j < n − 1 and odd(j) then
                                          {Odd-even exchange}
        t ⇐ successor(a)                  {Get value from successor}
        successor(a) ⇐ max(a, t)          {Give away larger value}
        a ← min(a,t)                      {Keep smaller value}
      endif
      if even(j) then
                                          {Even-odd exchange}
        t ⇐ successor(a)                  {Get value from successor}
        successor(a) ⇐ max(a,t)           {Give away larger value}
        a ← min(a,t)                      {Keep smaller value}
      endif
    endfor
  endfor
end
```

FIGURE 10-2 Odd-even transposition sort algorithm for the one-dimensional mesh processor array model.

Tracing

FIGURE 10-3 Odd-even transposition sort of eight values on the one-dimensional mesh processor array model.

Indices:	0	1	2	3	4	5	6	7
Initial values:	G	H	F	D	E	C	B	A
After odd-even exchange:	G	F	<H	D	<E	B	<C	A
After even-odd exchange:	F	<G	D	<H	B	<E	A	<C
After odd-even exchange:	F	D	<G	B	<H	A	<E	C
After even-odd exchange:	D	<F	B	<G	A	<H	C	<E
After odd-even exchange:	D	B	<F	A	<G	C	<H	E
After even-odd exchange:	B	<D	A	<F	C	<G	E	<H
After odd-even exchange:	B	A	<D	C	<F	E	<G	H
After even-odd exchange:	A	<B	C	<D	E	<F	G	<H

Bitonic Merge

- It's a parallel sorting algorithm.
- Invented by Batcher in 1968.
- Fundamental operation : compare-exchange.
- Two numbers are routed into a comparator, where they are exchanged if necessary, so that they are in proper order.
- Definition: A bitonic sequence is a sequence of values $a_0,a_1,\ldots\ldots,a_{n-1}$ with the property that

  (1) there exists an index i, where 0 < I <= n-1 such that a_0 through a_i is monotonically increasing and a_i through a_{n-1} is monotonically decreasing.

Bitonic Merge

Or (2) there exists a cycle shift of indices so that the first condition is true.

- Graph of a bitonic sequence contains at most one "peak" and one "valley".

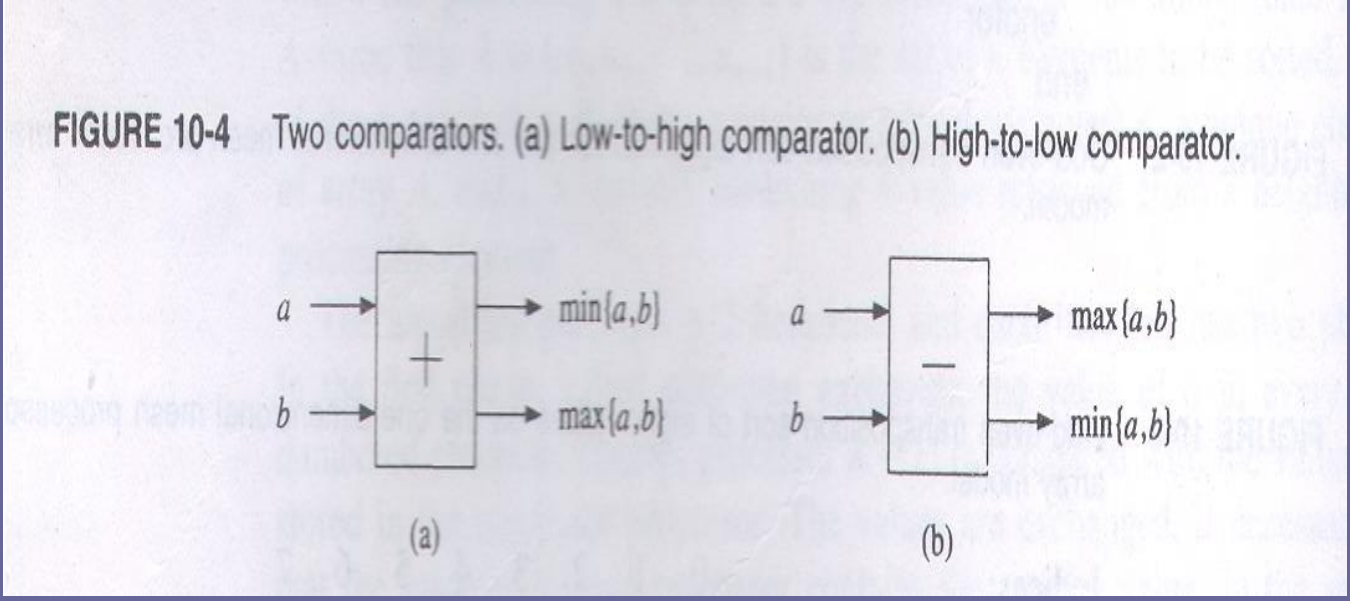
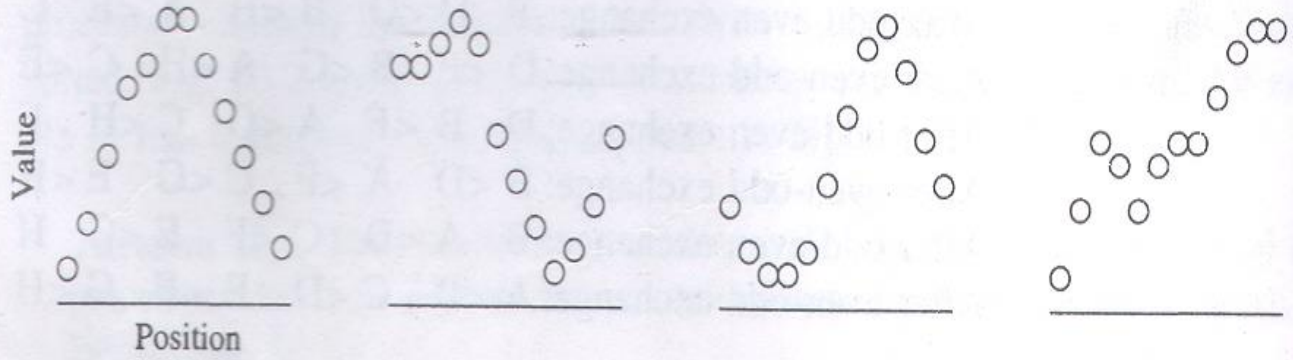


FIGURE 10-4 Two comparators. (a) Low-to-high comparator. (b) High-to-low comparator.

Bitonic Merge

FIGURE 10-5 The first three sequences are bitonic sequences; the last sequence is not.



Bitonic Merge

- How to produce 2 bitonic sequence from one bitonic sequence.
- Lemma: A single compare exchange step can shift a single bitonic sequence into two bitonic sequence.
- If n is even then n/2 comparisons are sufficient to transform a bitonic sequence of n-values

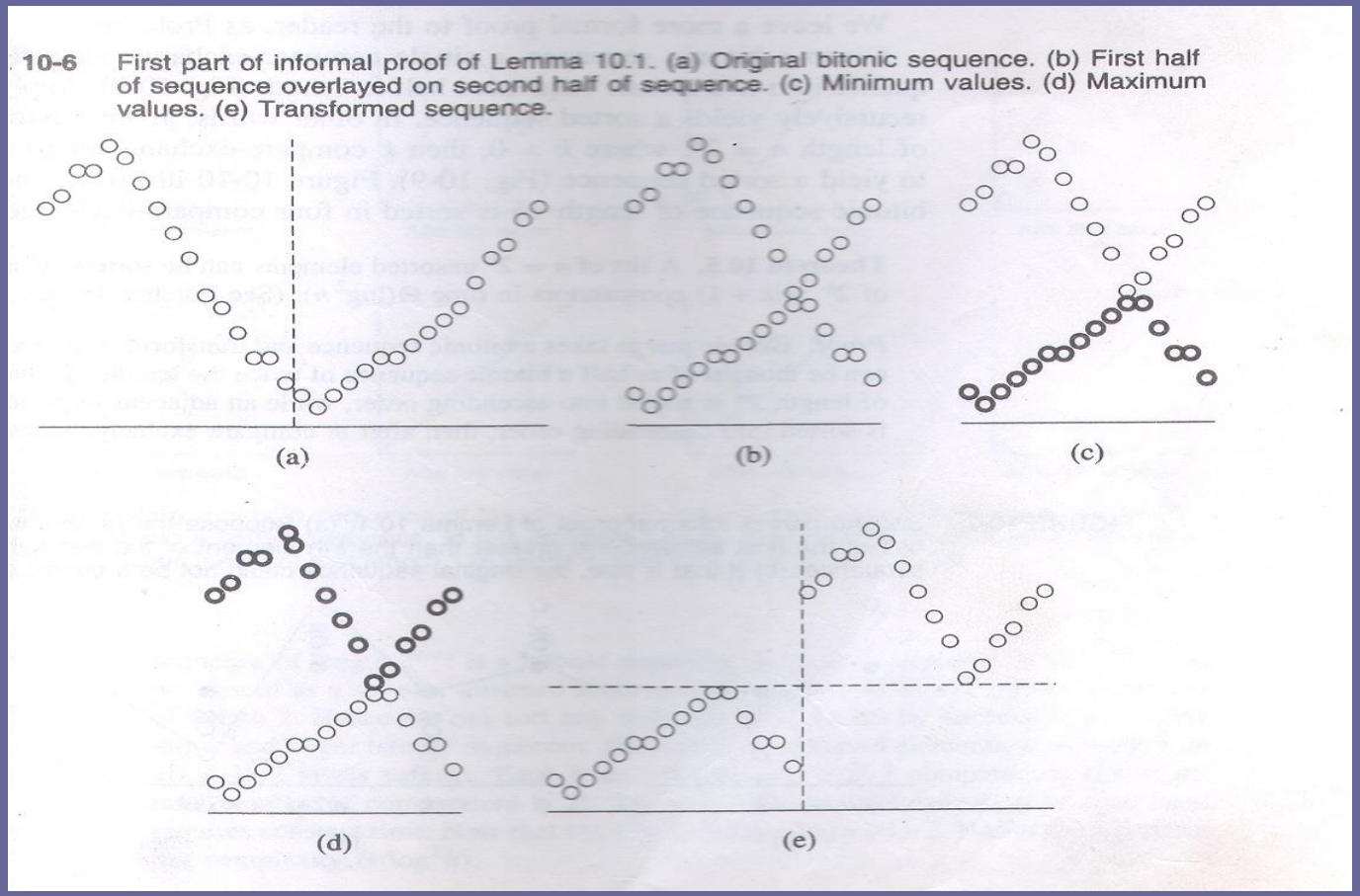
  $a_0,a_1,a_2,\ldots..,a_{n-2},a_{n-1}$ into two bitonic sequences of n/2 values, $\min(a_0,a_{n/2}),\min(a_1,a_{n/2+1})\ldots.\min(a_{n/2-1}, a_{n-1})$ and

$\max(a_0,a_{n/2}),\max(a_1,a_{n/2+1})\ldots.\max(a_{n/2-1}, a_{n-1})$ such that no value in the first sequence is greater than any value in the second sequence.

Bitonic Merge

- Actually first of all bitonic sequence of n elements is divided into two half then we use n/2 comparators and then we compare first element of the first half to first element of second half and second element of first half with second element of second half and so on….
- If we perform only single step of this compare-exchange on n elements then it will split single bitonic sequence into two bitonic sequence. if we perform this step k times then the sequence will be sorted. ($n=2^k$)

Bitonic Merge



10-6 First part of informal proof of Lemma 10.1. (a) Original bitonic sequence. (b) First half of sequence overlayed on second half of sequence. (c) Minimum values. (d) Maximum values. (e) Transformed sequence.

Bitonic Merge

- Given a bitonic sequence a single compare-exchange step divides the sequence into two bitonic sequences half its length.
- Applying this step recursively yields a sorted sequence.
- In other words given a bitonic sequence of length $n=2^k$, where $k > 0$ then k compare exchange steps are sufficient to yield a sorted sequence.
- Next slide shows a bitonic sequence of length 16 is sorted in four compare-exchnge steps.

Bitonic Merge

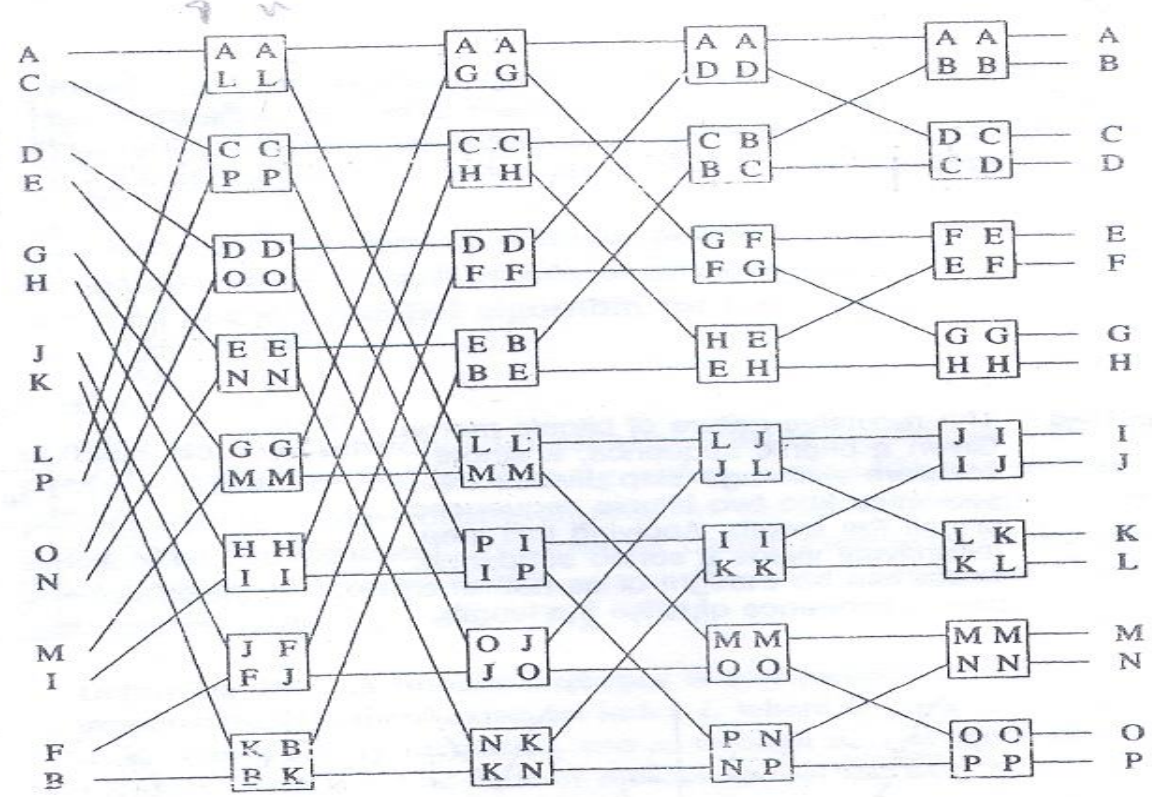


FIGURE 10-10 Sorting a bitonic sequence of length 16 by using bitonic merge.