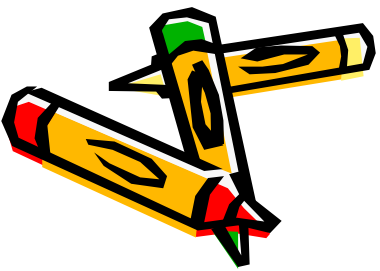
Definition

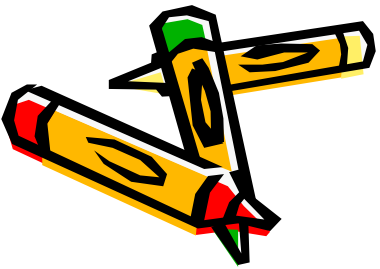
- These Systems
  - Multiple connected CPUs working together
  - A collection of independent computers that appears to its users as a single coherent system

Examples: parallel machines



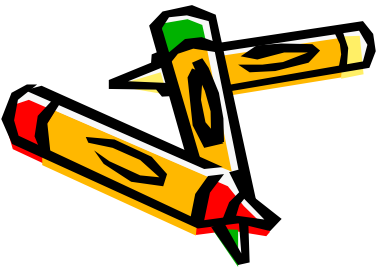
Transparency

- Location transparency: The users cannot tell where resources are located
- Migration transparency: Resources can move at will without changing their names.
- Replication transparency: The users cannot tell how many copies exist.
- Concurrency transparency: Multiple users can share resources automatically.
- Parallelism transparency: Activities can happen in parallel without users knowing

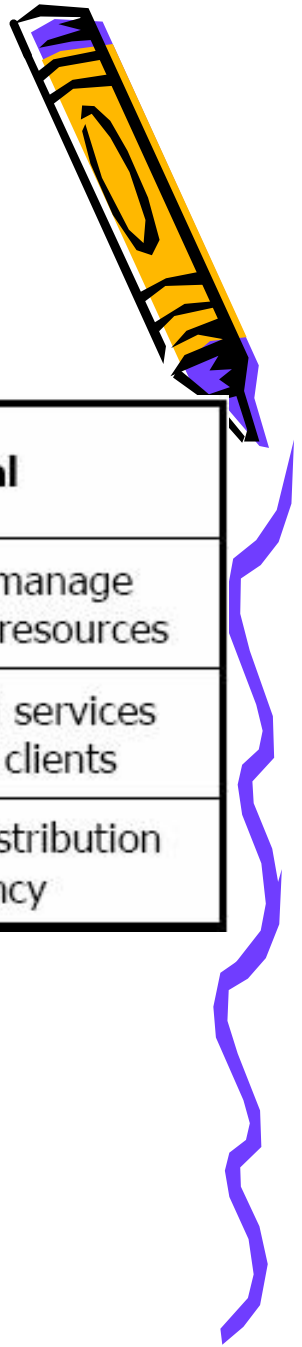


OS Issue

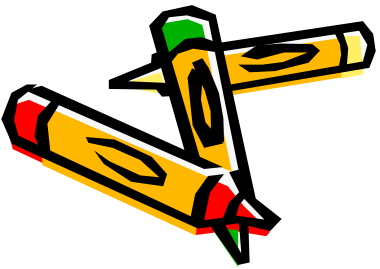
- Manages resources in a parallel or distributed system
  - Seamlessly and transparently to the user
- Looks to the user like a centralized OS
  - But operates on multiple independent CPUs
- Provides transparency
  - Location, migration…
- Presents users with a virtual uniprocessor



Types

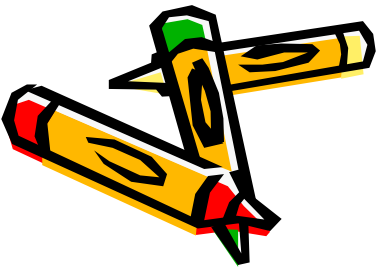


System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

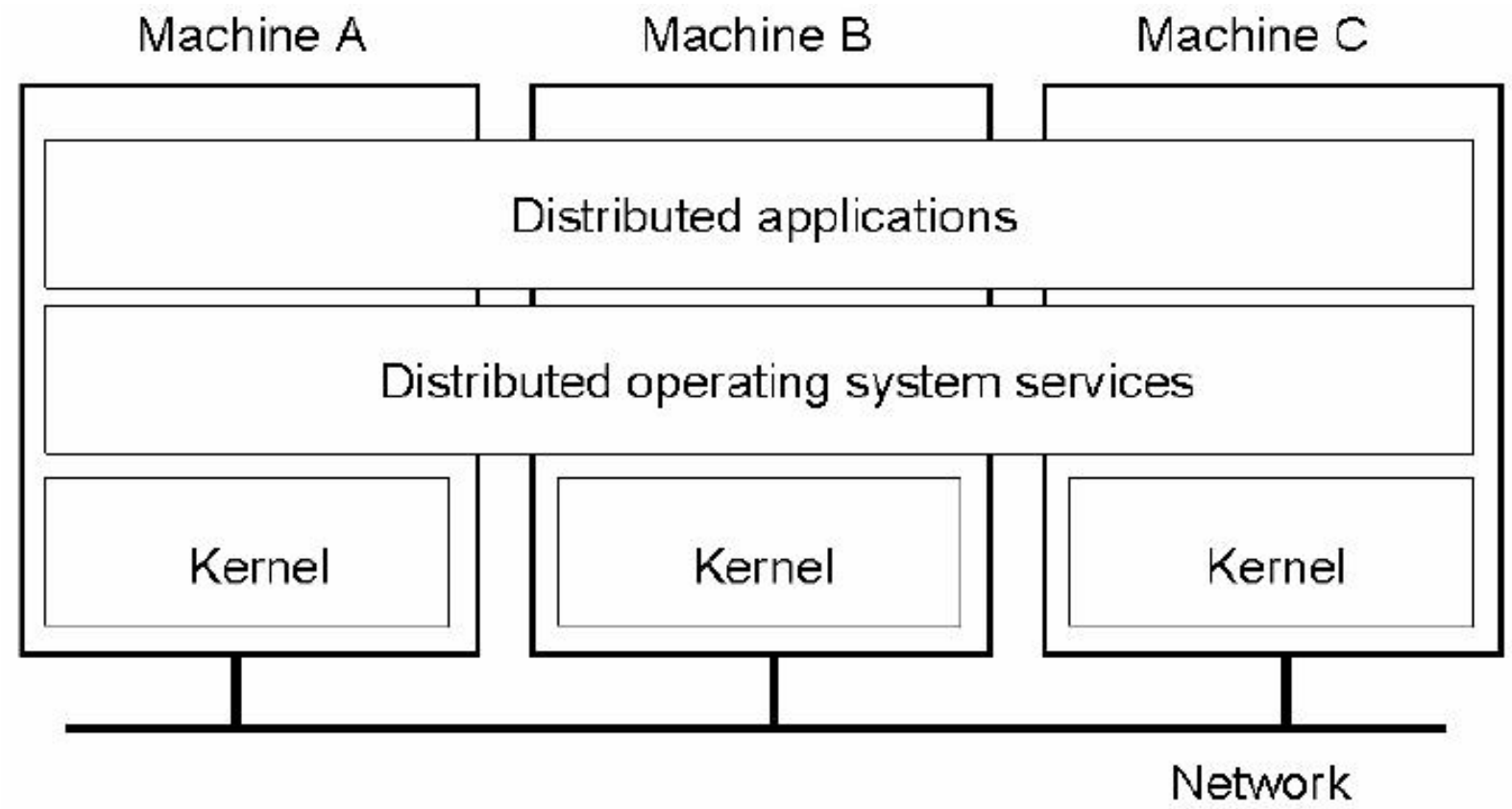


Multiprocessor Operating Systems

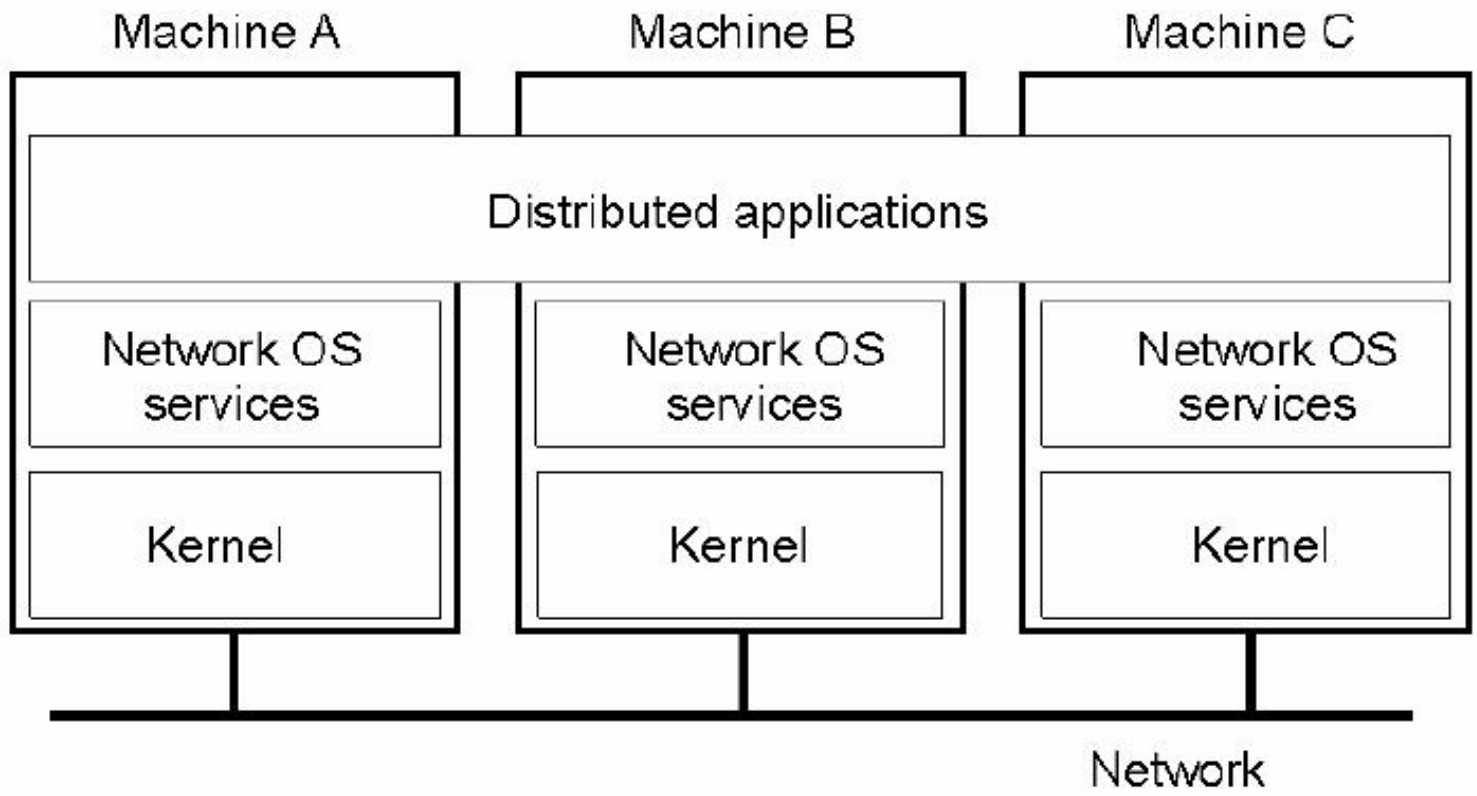
- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache



Multicomputer Operating Systems

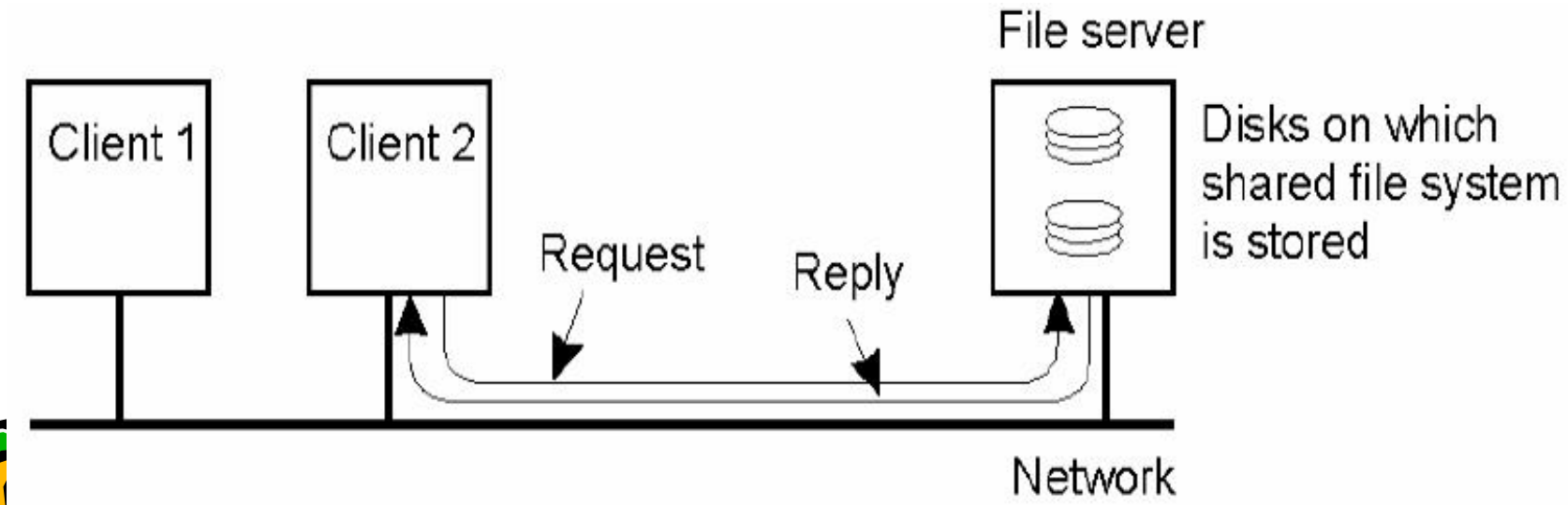


Network Operating System

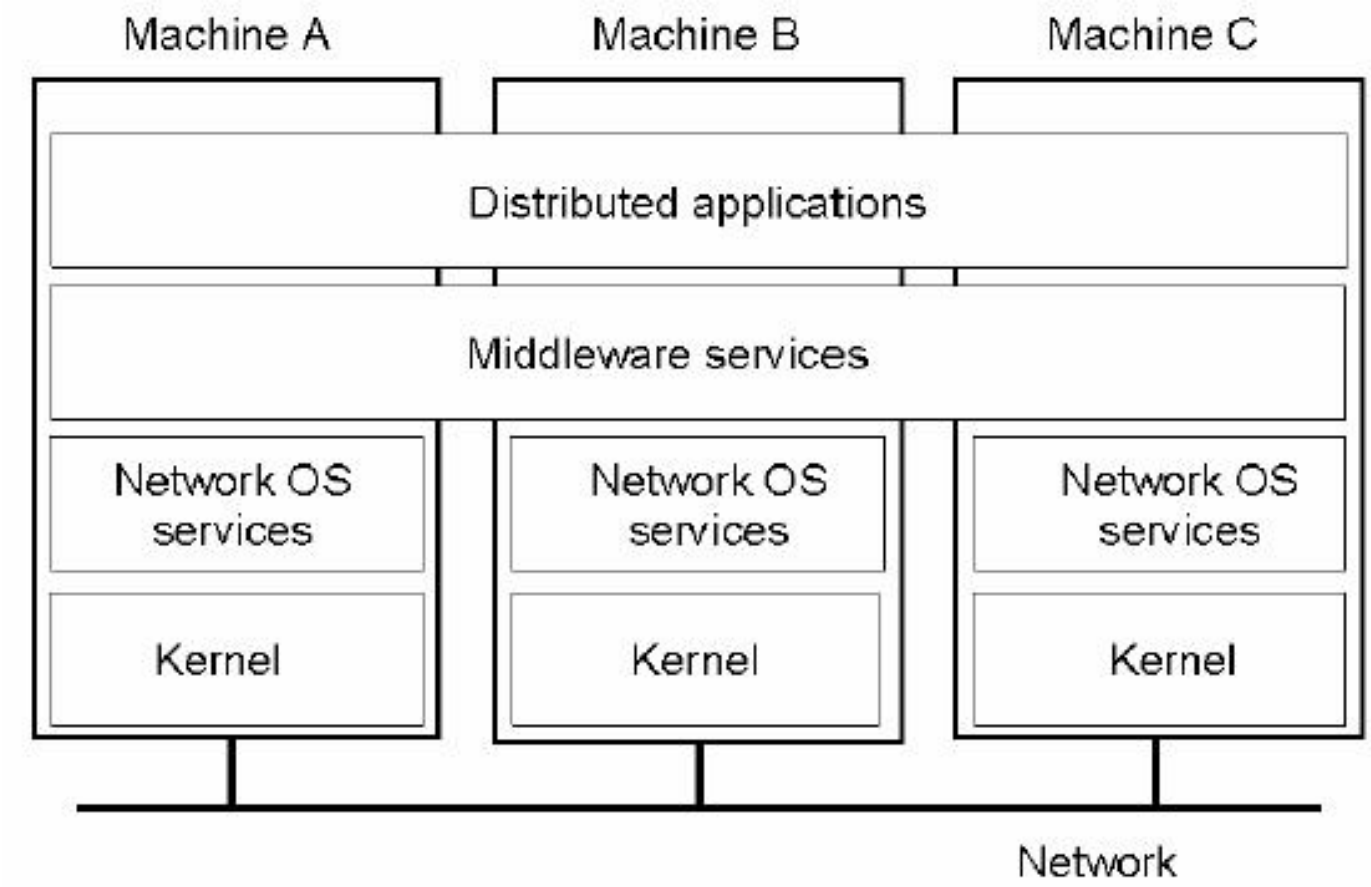


Network Operating System

- Employs a client-server model
  - Config I: All stand-alone workstation each with its own CPU, memory, hard disk, and OS, and connect by a LAN
  - Config II: System like the above but with some dedicated servers -- file servers, application servers, ftp server, mail server, name server, PPP server

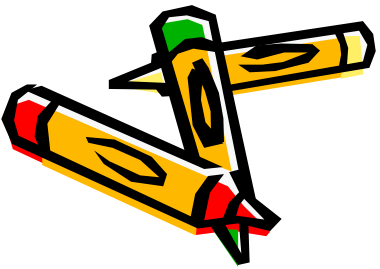


Middleware-based Systems

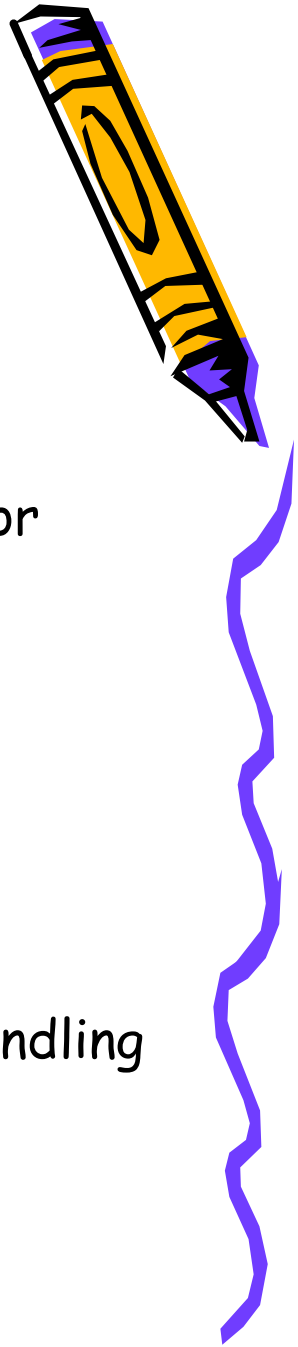


Comparison

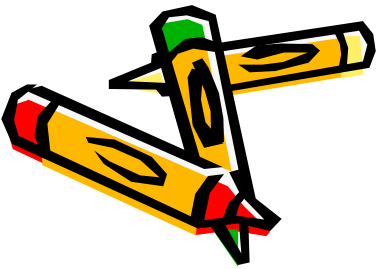
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open



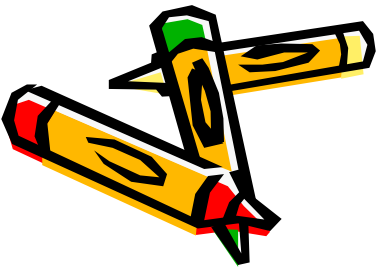
Design Issues



- Transparency
- Flexibility
  - Monolithic kernel vs. Microkernel
  - Microkernel -- keep it small, use user-level servers for system services
    - An interprocess communication mechanism
    - Some memory management
    - A small amount of low-level process management and scheduling
    - Low-level input / output
  - Monolithic kernel
    - provide the file system and directory system
    - full process management, and much system call handling

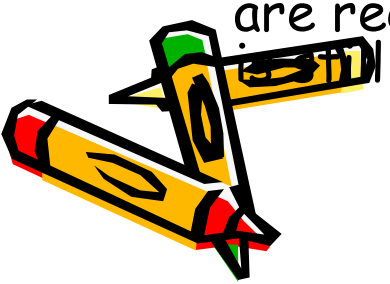


- Reliability
  - Fault tolerance
- Performance
  - Fine-grained parallelism
  - Coarse-grained parallelism
- Scalability
  - Potential bottle-necks in very large distributed systems
    - Centralized components: A single mail server for all users
    - Centralized algorithms: Doing routing based on complete information
  - Use decentralized algorithms:
    - No machine has complete information about the system state
    - Machines make decisions based only on local information
    - Failure of one machine does not ruin the algorithm
    - There is no implicit assumption that a global clock exists



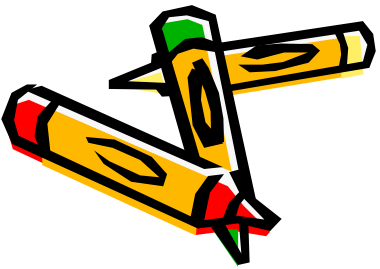
Communication

- Multicasting: It is possible to create a special network address (for example, indicated by setting one of the high-order bits to 1), to which multiple machines can listen. When a packet is sent to one of these addresses, it is automatically delivered to all machines listening to the address. Implementation is easy, just assign each group a different multicast address
- Broadcasting: Packet contains a certain address (e.g. 0) are delivered to all machines. Broadcasting can also be used to implement group, but it is less efficient. Each machine receives each broadcast, so its software must check to see if the packet is intended for it. If not, the packet is discarded. The above two methods still takes only one packet to reach all the members of the group.
- Unicasting: Group communication is implemented by having the sender transmit separate packets to each of the members of the group. For a group of n members, n packets are required. If most groups are small, this implementation is still workable.

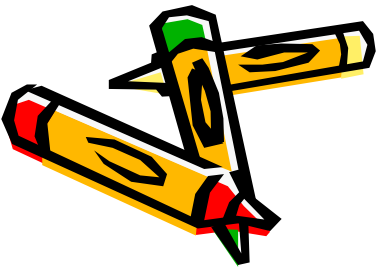


Scheduling

- Each processor does its own local scheduling (assuming that it has multiple processes running on it), without regard to what the other processors are doing.
- However, when a group of related, heavily interacting processes are all running on different processors, independent scheduling is not always the most efficient way.



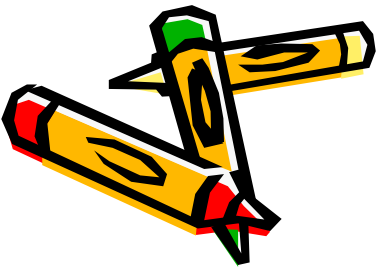
- Although it is difficult to determine dynamically the inter-process communication patterns, in many cases, a group of related processes will be started off together.
- We can further assume that a sufficiently large number of processors is available to handle the largest group, and that each processor is multi-programmed with N process slots



Parallel Sorting

Sorting an array which consists of 1000000 random numbers

- Linear Quick Sort: 0.390000 secs
- Parallel Quick Sort 2 procs:0.250000 secs
- Parallel Quick Sort 3 procs: 0.210000 secs
- Parallel Quick Sort 4 procs: 0.190000 secs



References

- http://lass.cs.umass.edu/~shenoy
- http://www.comp.hkbu.edu.hk/~jng
- http://www.sci.hkbu.edu.hk/hpccc/