Parallel and Distributed Computing
CS3006 (BDS-6A)
Lecture 13

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

14 March, 2023

Previous Lecture

- Classic Computing
- Distributed Systems:
  - Motivation
  - Need
  - Examples
  - Benefits and disadvantages
  - Cluster, grid and cloud
- Communication Models (transfer time)
- Message Passing
- C/S and P2P paradigms

Distributed Systems: Issues

- *Programming Complexity*
  - How the various programs communicate together?
  - Which data format on the various network nodes?
  - Need to define (application) protocols
  - Operations synchronization may lead to delay and slowing down
- *Scarce Robustness*
  - Higher chance of errors/faults
- *Hard to Optimize*
  - Lack of a global view

Operating System

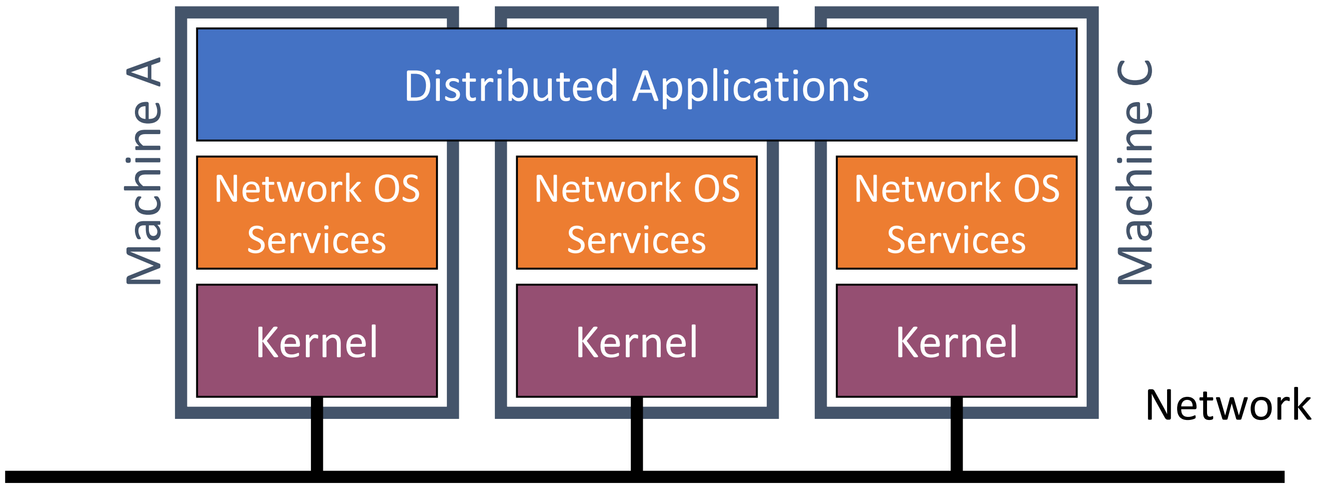
- A program that manages the computer hardware
- Provides a basis for application programs and acts as an intermediary between the user and the hardware
- Offers a reasonable way to solve the problem of creating a usable computing system

Types of Operating Systems:

- Linux and Unix
- Microsoft Windows
- Mac OS X

Network Operating Systems

- Distribution not hidden by OS
- But a number of services offered to support distribution
  - remote login (ssh), remote copying (scp)
  - distributed file systems (samba, nfs)
- Well adapted to heterogeneous DS



Features of Network OS

- Provides basic operating system features; *support for processors*, *protocols*, *automatic hardware detection*, support *multi-processing of applications*
- Security features; *authentication*, *authorization*, *access control*
- Provides files, print, web services, back-up and *replication services*
- Supports Internetworking such as *routing* and *WAN ports*
- User management and support for *login*/*logoff*, *remote access*, *system management*

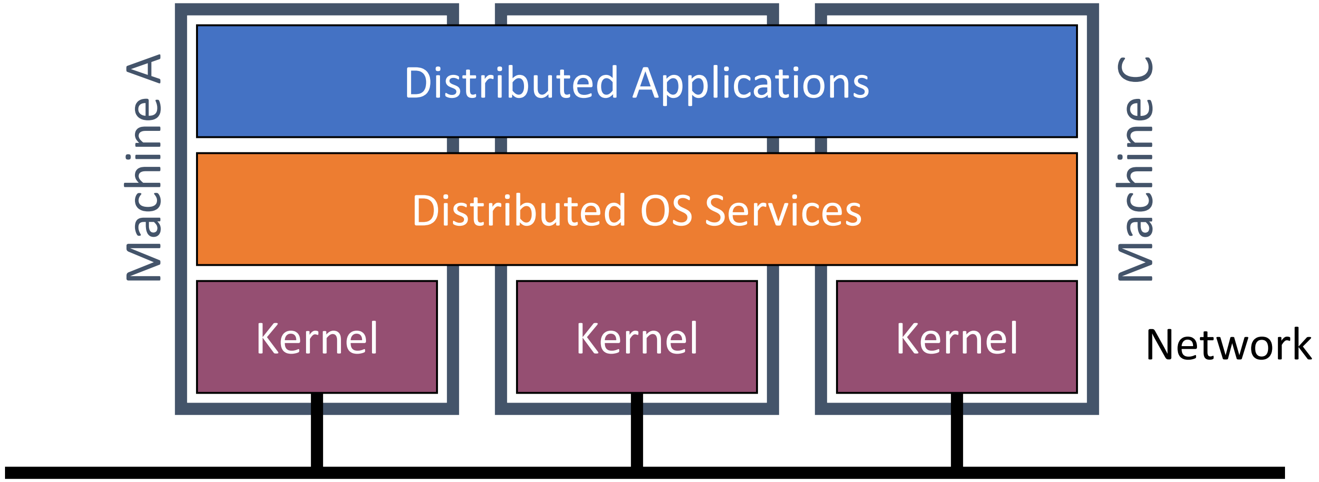
Network and Operating System Security

- OS: *system must protect itself*
- List of possible breaches is almost endless

- Runaway process could constitute an accidental denial-of-service attack
- Travels over private leased lines, shared lines like the internet, wireless connections, or dial-up lines
- Intercepting these data could be harmful as breaking into a computer
- Interruption of communications could constitute a remote denial-of-service attack

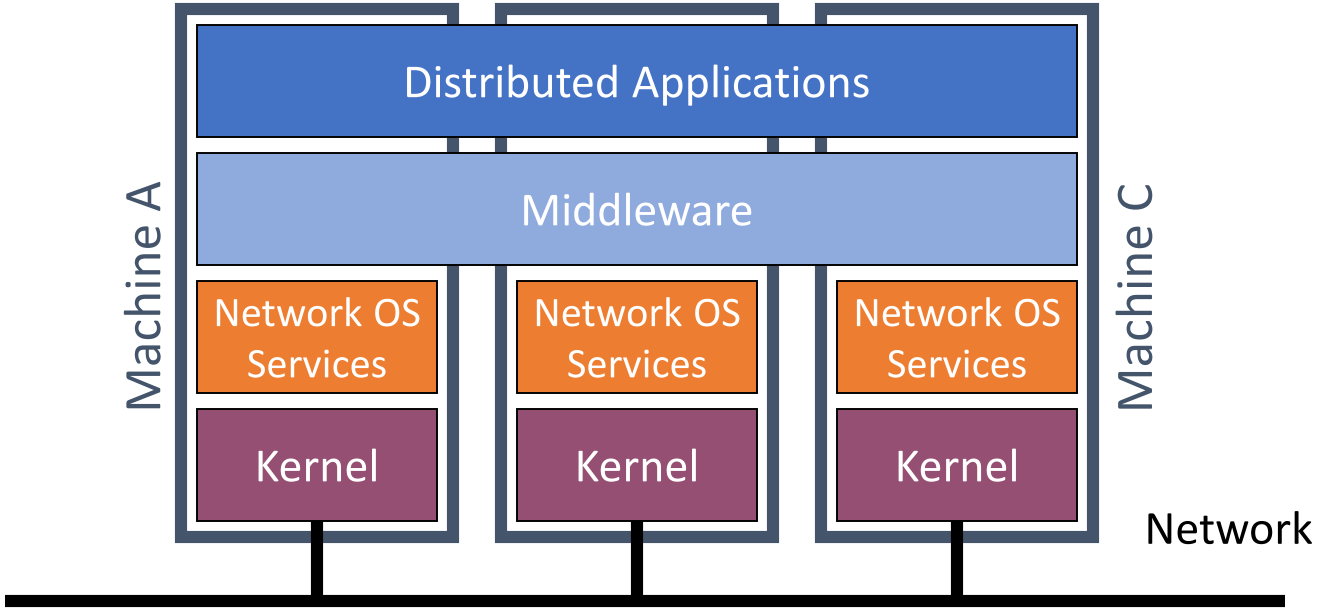
Distributed OSs

- Completely hide the distribution: “single system image”
- Each individual node holds a specific software subset of the global aggregate operating system.
- The user has the illusion to use one single multiprocessor machine (symmetric multi-processing, SMP)
- Essentially used for homogeneous cluster of machines interconnected with high performance networks



Middleware

- Pros and Cons of DOS and NOS
  - DOS are user-friendly but don’t scale well. Reverse for NOS
  - How to get the best of both world?
- Solution: **Middleware!**



System	Description	Main Goal
DOS (Distributed Operating System)	Tightly coupled operating system for multiprocessors and homogenous multicomputers	Hide and manage hardware resources
NOS (Network Operating System)	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 service	Provide distribution transparency

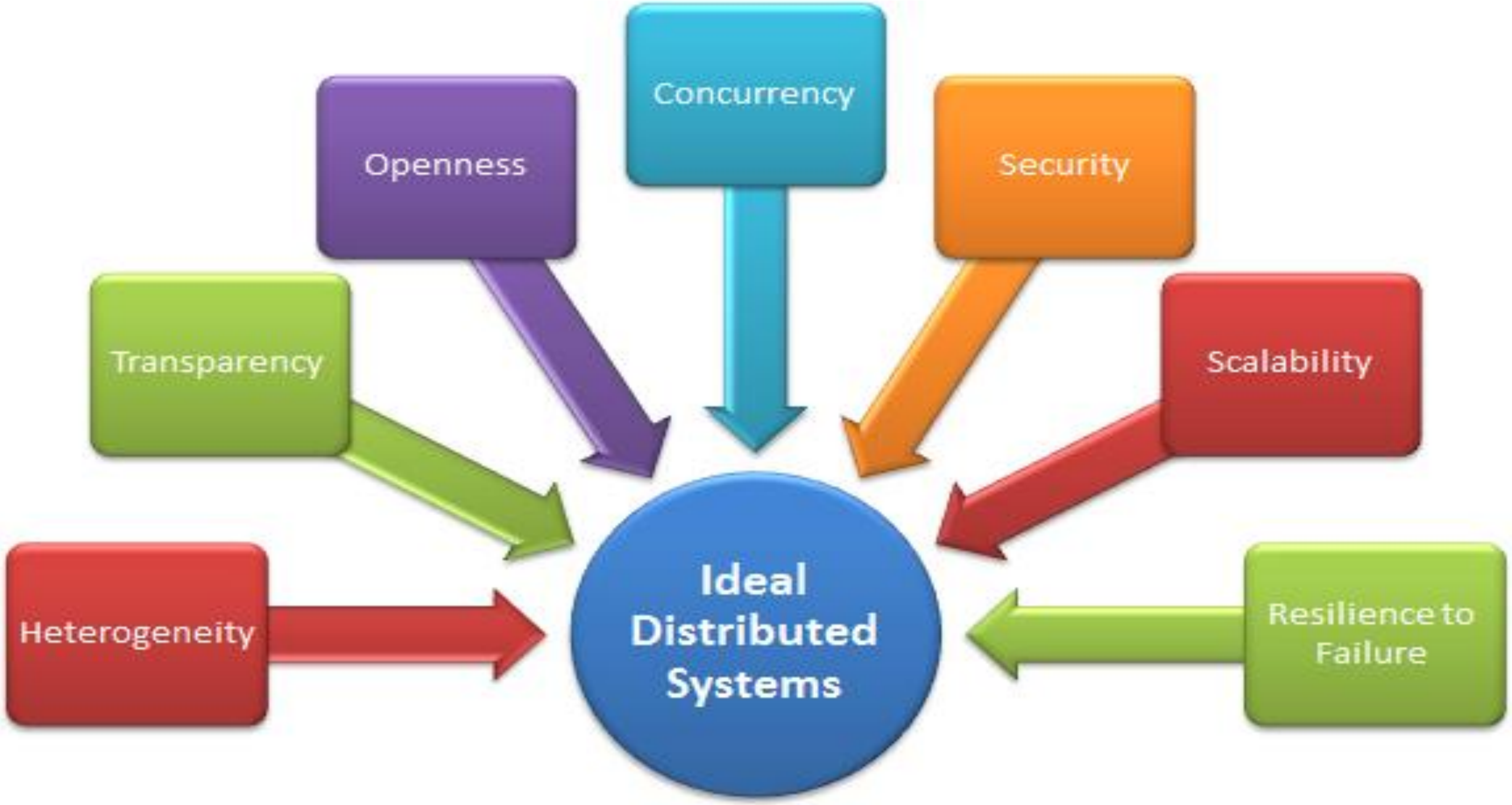
Goals of a Middleware Platform

- Resource sharing
  - The ability to access and share resources in a distributed environment
  - The bread and butter of distributed systems
- Transparency
  - The ability to view a distributed system as if it were a single computer
  - Varying dimensions of transparency incl. location, access, migration, etc.
  - The degree of transparency is a key decision in any systems architecture

Goals of a Middleware Platform

- Openness
  - The offering of services according to standard rules (syntax and semantics)
  - Openness provides support for the key properties of portability and interoperability
  - Again, the degree of openness is a key factor in systems design
- Extensibility
  - The ability to be able to introduce new or modified functionality

Challenges of distributed Systems



Heterogeneity

- Distributed systems usually combine *multiple underlying systems* to solve a problem. These systems can differ in the following ways
  - networks
  - computer hardware
  - operating systems
  - programming languages
  - implementations by different developers

- The aim for *middleware* is to provide an abstraction level such that all these heterogeneities *are hidden from the user*

Transparency

- Actual System:
  - a collection of *independent systems*

- User's viewpoint:
  - a *single unified system*

- Transparency is the *concealment of the distributed system from the user*

Challenges for transparency

- To hide the location where a resource is located
- To hide the migration of resource
- To hide the replication of resource
- To hide the differences in data representation
- To hide the failure and recovery of resource
- Hide that a resource may be shared by several competitive users

Openness

- An *open distributed system* is a system that offers services according to standard rules that describe the *syntax* and *semantics* of those services
- Example
  - Implementation of the services in the network
  - Interface Definition
- Different Implementations of the same Interface
  - ***Interoperability*** - the extent by which *two implementations of systems from different manufacturers* can work together by relying on each other's services as specified by a *common standard*.
  - ***Portability*** – An application developed for a distributed system *can be executed* (without modification) on a *different system that implements the same interface*

Concurrency

- Any object that represents a *shared resource* in a distributed system must be responsible for ensuring that it *operates correctly* in a concurrent environment

- For an object to be *safe* in a *concurrent environment*, its operations must be *synchronized* in such a way that its data remains *consistent*.

Security

- Resources are shared however they must be kept *secure* – multiple users with their own data
- Components of Security
  - *confidentiality* (protection against disclosure to unauthorized individuals)
  - *integrity* (protection against alteration or corruption)
  - *availability* (protection against interference with the means to access the resources)

Security (Cont..)

- the challenge is to
  - *send sensitive information* in a message over a network in a *secure manner*
  - knowing for sure the *identity of the user* or other agent on whose behalf a message was sent
- *Major Security threats*
  - *Denial of service attacks*
  - *Security of mobile code (Executing Code)*

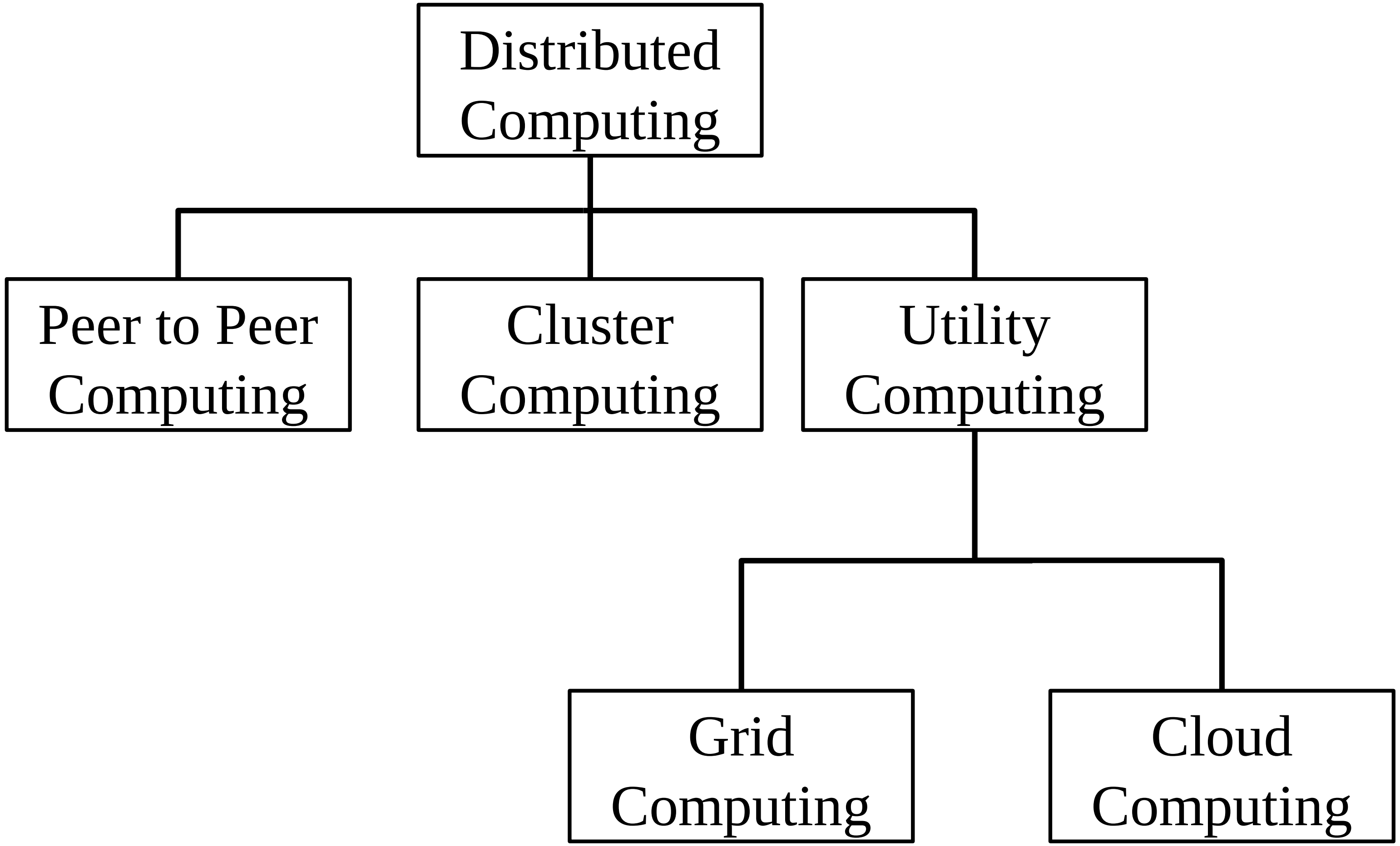
Scalability

- A system is described as *scalable if it will remain* effective when there is a *significant increase in the number of resources and the number of users*
  - *Controlling the cost of physical resources*
  - *Controlling the performance loss*
  - *Preventing software resources running out*
- Dimensions
  - *Scalable with respect to size*
  - *Scalable with respect to geography*
  - *Scalable with respect to administration*
- The challenge is that the system and application software *should not need to change* when the scale of the system increases

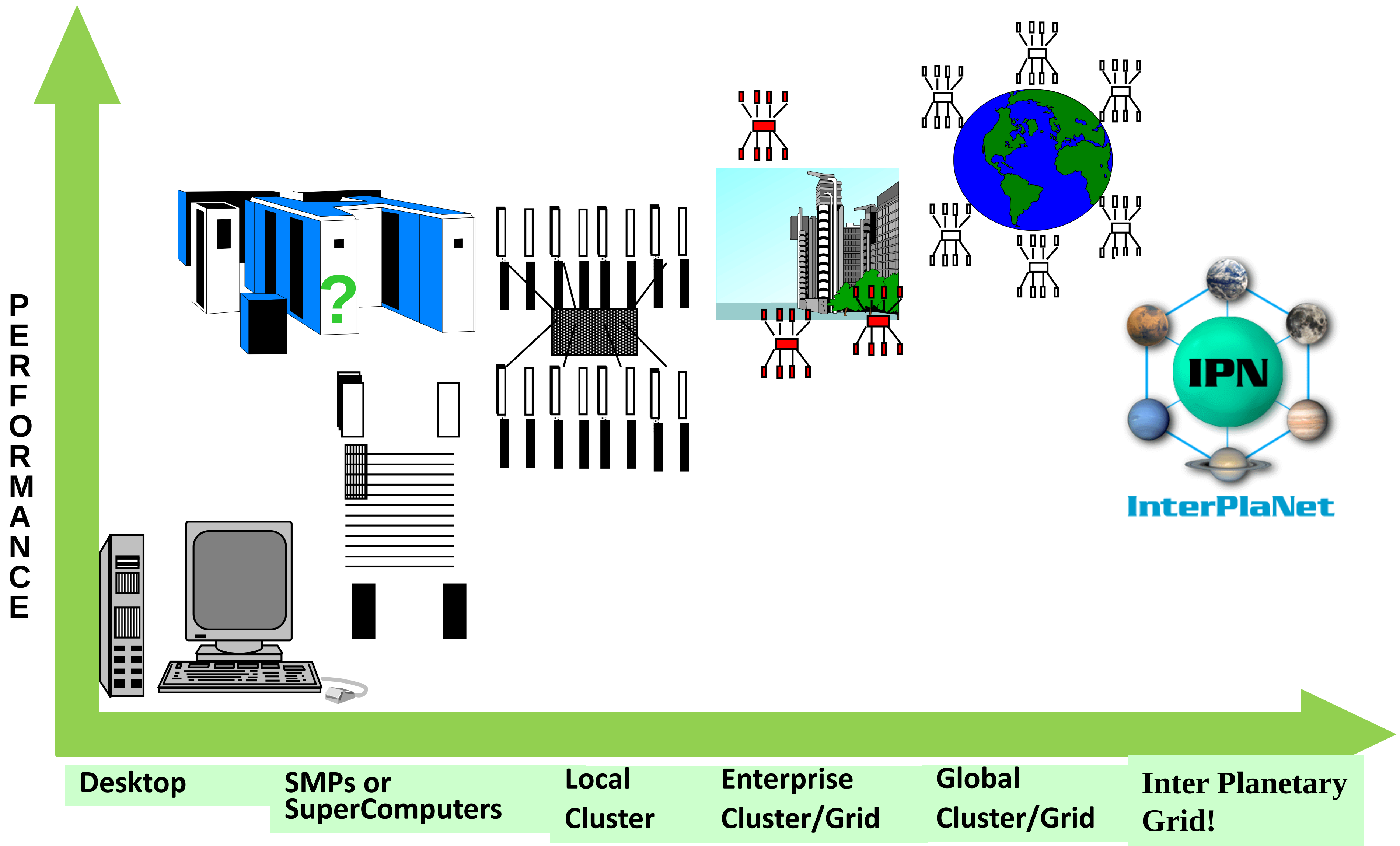
Failure Handling

- Failures in a distributed system are partial – that is, some components fail while others continue to function
  - *Masking* Failures - Resending the dropped messages
  - *Tolerating* Failures – Reloading Pages
  - *Recovering* Failures – Roll back – Removing Inconsistency in data
- The design of effective techniques for keeping *replicas* of rapidly changing data up-to-date without excessive loss of performance is a challenge

Distributed Computing

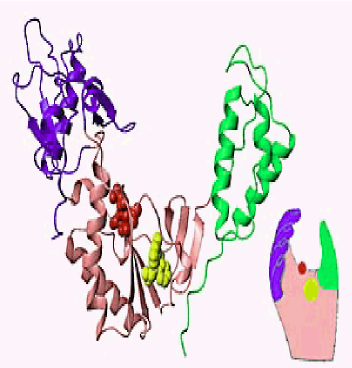


Scalable High Performance Computing

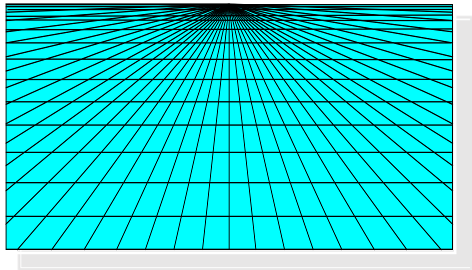


Why we need it?

- **Solving grand challenge applications using *modeling*, *simulation* and *analysis***

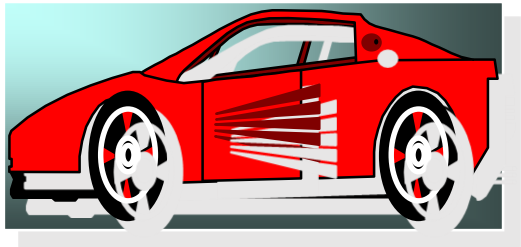


Life Sciences



Aerospace

Internet &
Ecommerce



CAD/CAM

Digital Biology

Military Applications

Clusters

- A cluster is a type of a *distributed processing system*, which consists of a collection of interconnected *stand-alone computers* cooperatively working together as a *single*, integrated computing resource.

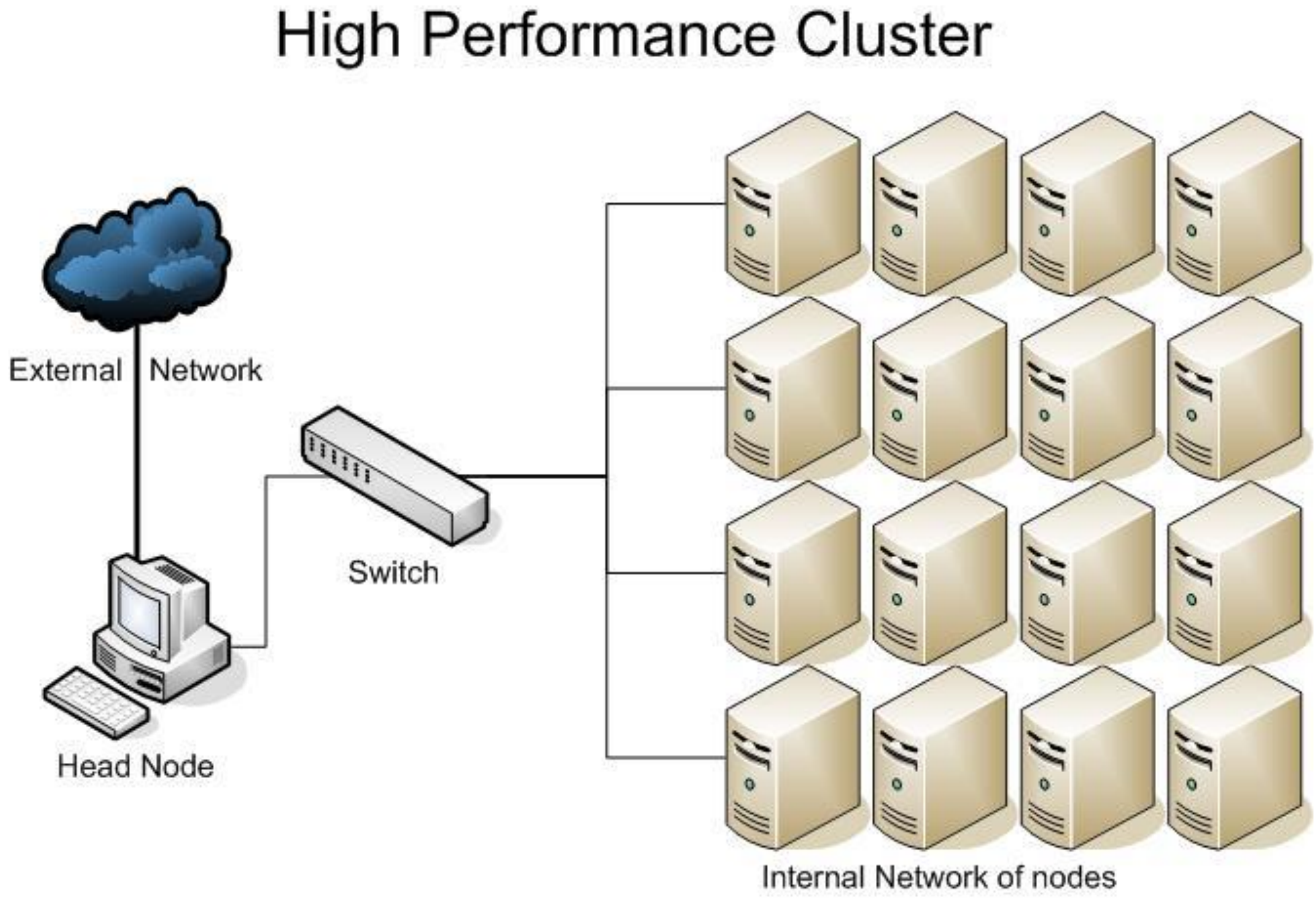
- "stand-alone" (whole computer) that can be used on its own (full hardware and OS).

- Nodes can be used individually or combined

Cluster Computing

- Clusters are usually deployed to improve *speed and/or reliability* over that provided by a *single* computer.
- Much more *cost effective* than single computer of comparable speed or reliability.
- In cluster computing each node within a cluster is an independent system, with its own
  - *operating system*
  - *private memory*
  - in some cases, its *own file system*
- Processors on one machine *cannot directly access the memory* on the other machines
- Programs employ *message passing* to get data and execution code from one machine to another

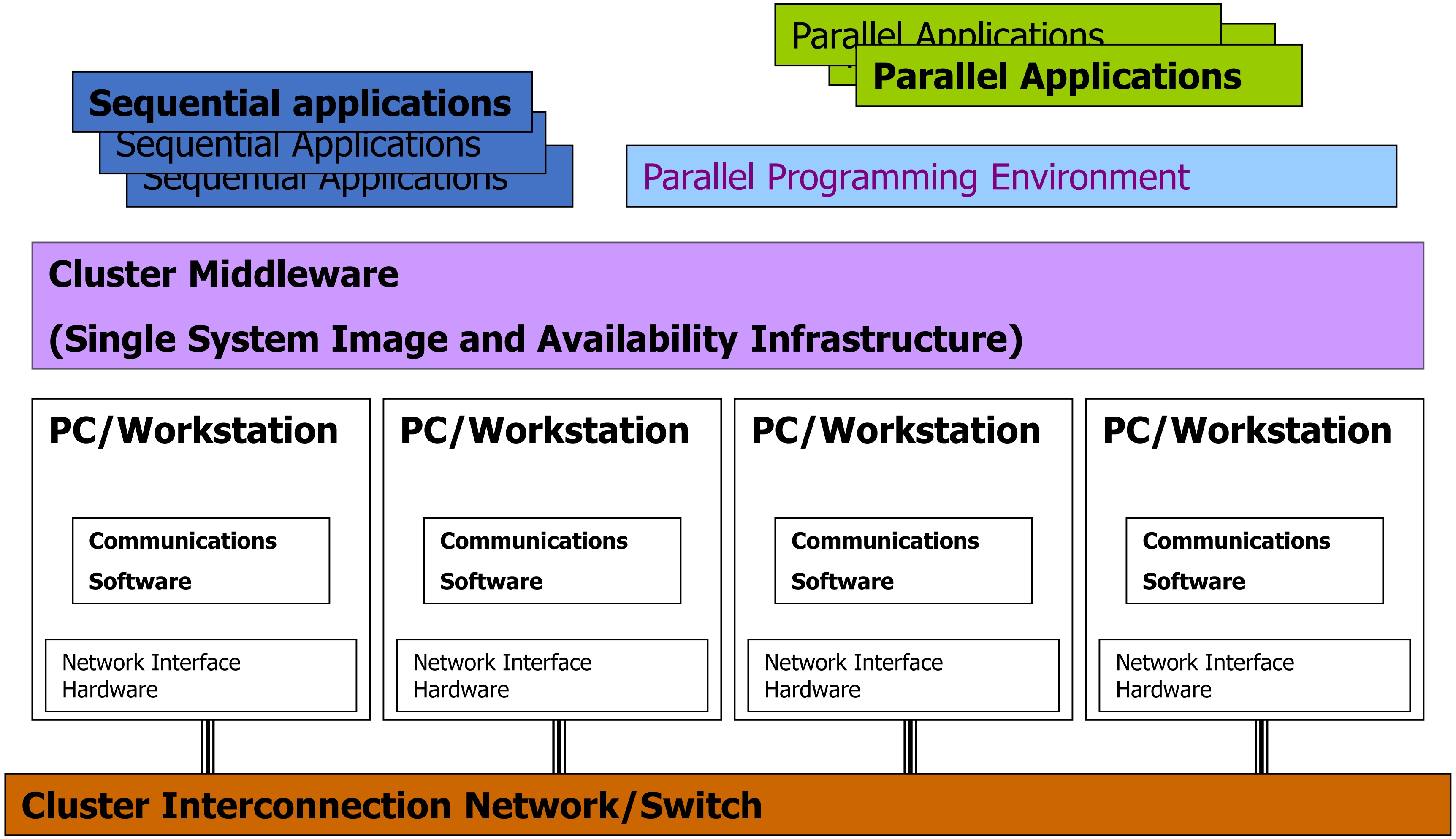
A cluster is a group of computers connected by a local area network (LAN)



Functioning of a cluster

- A cluster typically has one *head* or *master node* (the main node that handles management tasks) and several *compute nodes*.
- *Management and programming software* is essential to break down *tasks* into *smaller parts* and then *assign* those parts to individual systems.
- The *assigned* systems process their *portions* before they *return* their *results* to cluster management software, which manages and controls the cluster.
- To achieve more with less from a technological standpoint, you can harness the efforts of *numerous smaller and less-expensive processors* to meet or exceed the capabilities of a *larger and more-expensive processor*

Cluster Computer Architecture



Motivations for clustering

- Improving the *time to solution*, especially for *large* and *complex simulations*, *models*, or *forecasts*

- Squeezing a better return out of existing investments in computing technology, especially by *increasing overall utilization of computing resources*

- Establishing more *control over computing workloads* and pushing *important jobs to the head of the line* for speedy, reliable processing

- Improving overall computing *reliability*, *serviceability*, and *availability*, without necessarily forking over huge amounts of cash for custom-built fault-tolerant systems

Clusters

- A cluster is *tightly coupled*, whereas a grid or a cloud is *loosely coupled*
- All nodes work cooperatively together as a *single integrated computing resource* so conceptually it is smashing up many machines to make a *powerful* machine

- Types of clusters
  - High Availability clusters
  - Load balancing clusters
  - High Performance Computing clusters

High Availability Clusters

- These clusters are designed to *provide uninterrupted availability* of data or services (typically web services) to the *end-user community*.

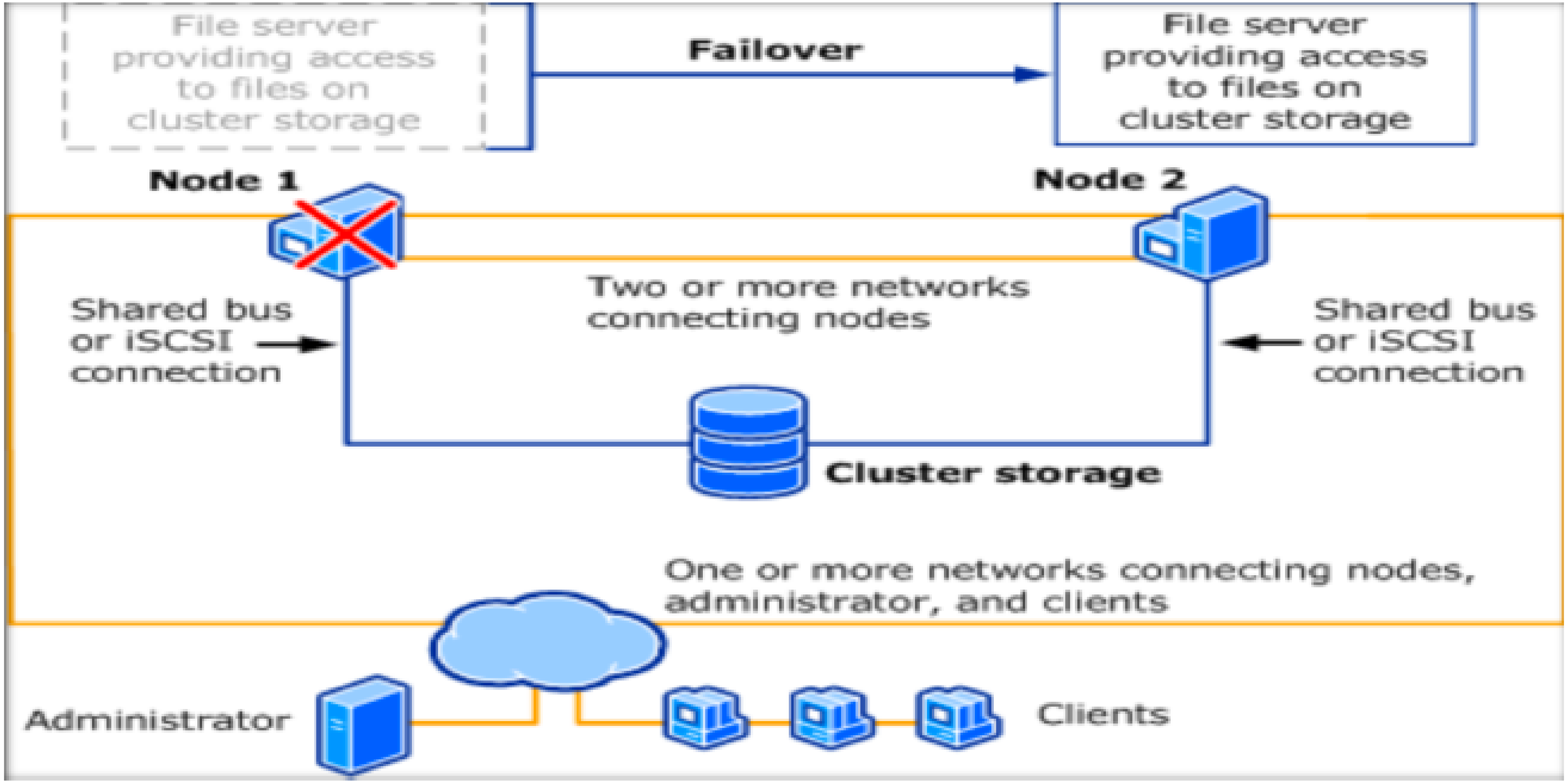
- If a *node fails*, the service can be *restored* without affecting the availability of the services provided by the cluster. While the application will still be available, there will be a *performance drop* due to the missing node.

- The purpose of these clusters is to ensure that a single instance of an application is only ever running on one cluster member at a time but if and when that cluster member is no longer available, the *application will failover to another cluster member*.

High Availability Clusters

- Also known as Failover Clusters
- Support server applications that can be *reliably utilized with a minimum of down-time*
- *Redundant computers in groups* that provide continued service even in case of *component failure*
- It is done by *detecting hardware/software faults*, and immediately *restarting the application on another system* without requiring administrative intervention
- High availability clusters implementations are best for *mission-critical applications or databases*, *mail*, *file* and *print*, *web* or *application servers*.

High Availability clusters



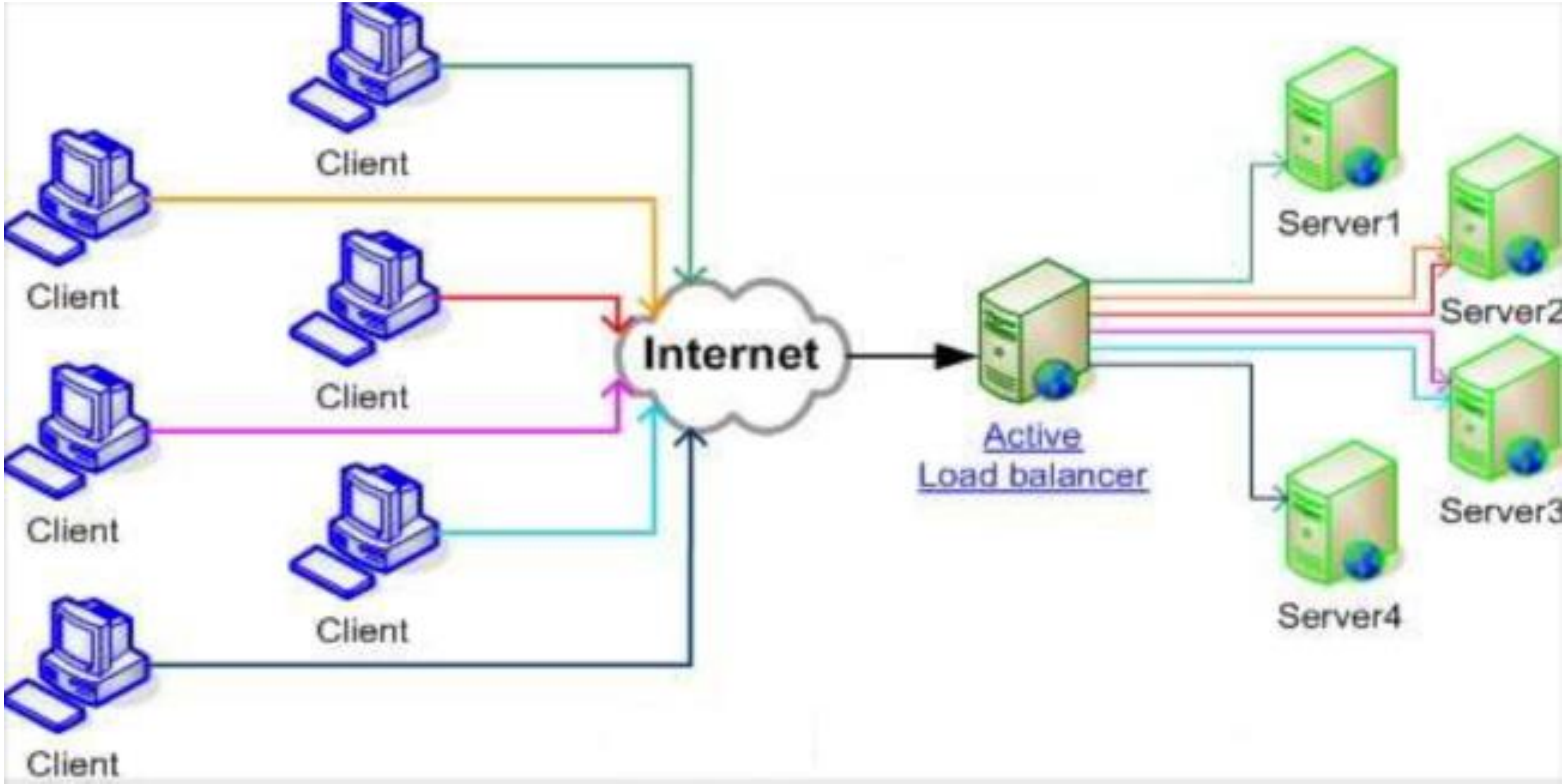
Load Balancing Clusters

- Load balancing is required in systems that *handle large volumes of client requests*
- Support *multiuser* and *multitasking* environments
- Overall distribution of the workload of a cluster is *hard to predict* at any particular moment
- Static approach to I/O planning is almost useless
- The two major categories of load balancing are:
- **Software-based load balancing**
    - consists of special software that is installed on the servers which dispatches requests from clients to servers
- **Hardware-based load balancing**
    - consists of a specialized switch or router with software to provide load balancing functionality

Load Balancing Cluster

- This type of *cluster distributes incoming requests for resources* or content among multiple nodes running the same programs or having the same content.
- Both the *high availability* and *load-balancing cluster* technologies can be combined to increase the reliability, availability, and scalability of application and data resources that are widely deployed for web, mail, news, or FTP services.
- Every node in the cluster is able to handle requests for the *same content or application*.
- This type of distribution is typically seen in a *web-hosting environment*.

Load Balancing Cluster



High Performance Computing Clusters

- *High performance clusters* are used where
  - *time to solution* is important
  - *problem* is *huge* and can not be executed on one single computer
- The cluster manages the resources needed for the job and assigns the job to a *work queue*
- An ideal solution for problems in which users need to *run many similar jobs* with *different parameters or data sets*
- *Computing is local to a cluster node*, node doesn't communicate with other nodes
- May need *high speed file system access*

High Performance Cluster

- Donald Becker of NASA assembled such a cluster in 1994
- It is also known as the *Beowulf cluster*.
- Essentially, any *group of Linux machines* dedicated to a *single purpose* with a *centralized node* (for coordination) can be called a *Beowulf cluster*.
- These types of clusters *increase performance* and *scalability* for applications, particularly *computationally or data intensive tasks*.
- Applications of such systems include *data mining*, *simulations*, *parallel processing*, *weather modeling*, etc.

References

1. Slides of Dr. Haroon Mahmood
2. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City, CA: Benjamin/Cummings.
3. Quinn, M. J. Parallel Programming in C with MPI and OpenMP, (2003).

Helpful Links:

1. https://en.wikipedia.org/wiki/Java_remote_method_invocation