Distributed Operating System

Unit: 3

Processes and processors in Distributed Systems

Dr. K. Jairam Naik

NIT Raipur

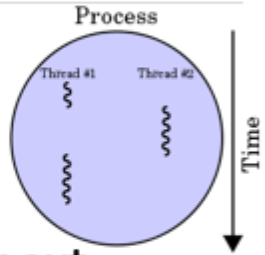
Unit Outline & Weightage %

- **Processes and processors in distributed systems**
  - Threads
  - System model
  - Processor allocation
  - Scheduling in distributed systems
    - Load balancing and sharing approach
    - Fault tolerance
    - Real time distributed systems
    - Process migration and related issues

What is Process?

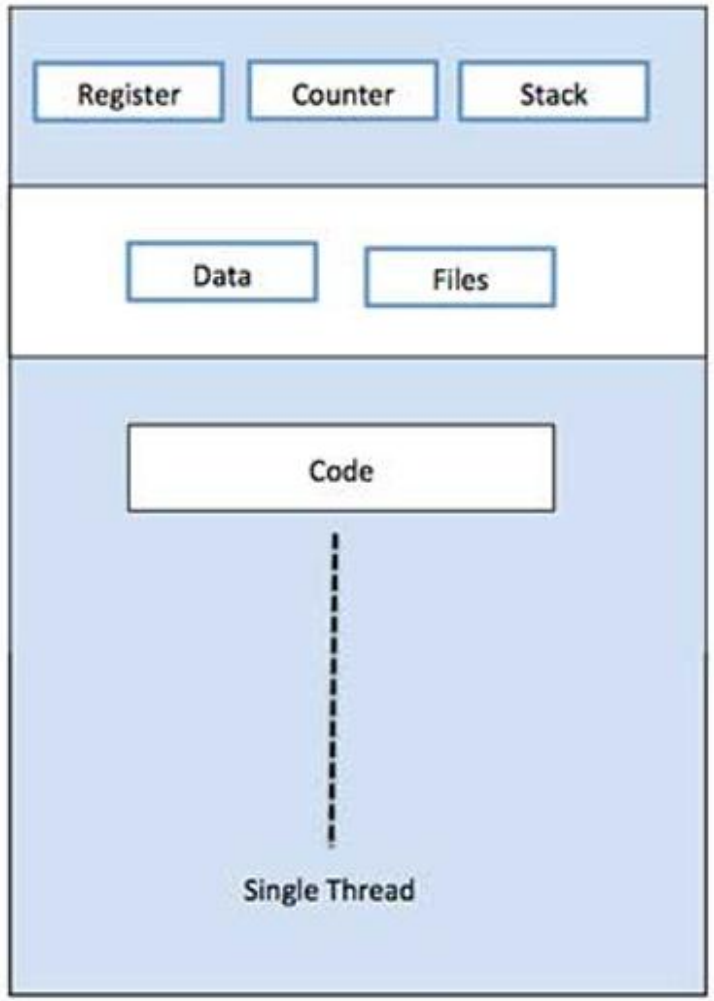
- Process is a **program under execution.**
- Process is an abstraction of a running program.
- Process is an instance of an executing program, including the current values of the program counter, registers & variables.
- Each process has its own virtual CPU.
- Real CPU switches back and forth from process to process called multiprogramming.

What is Threads?

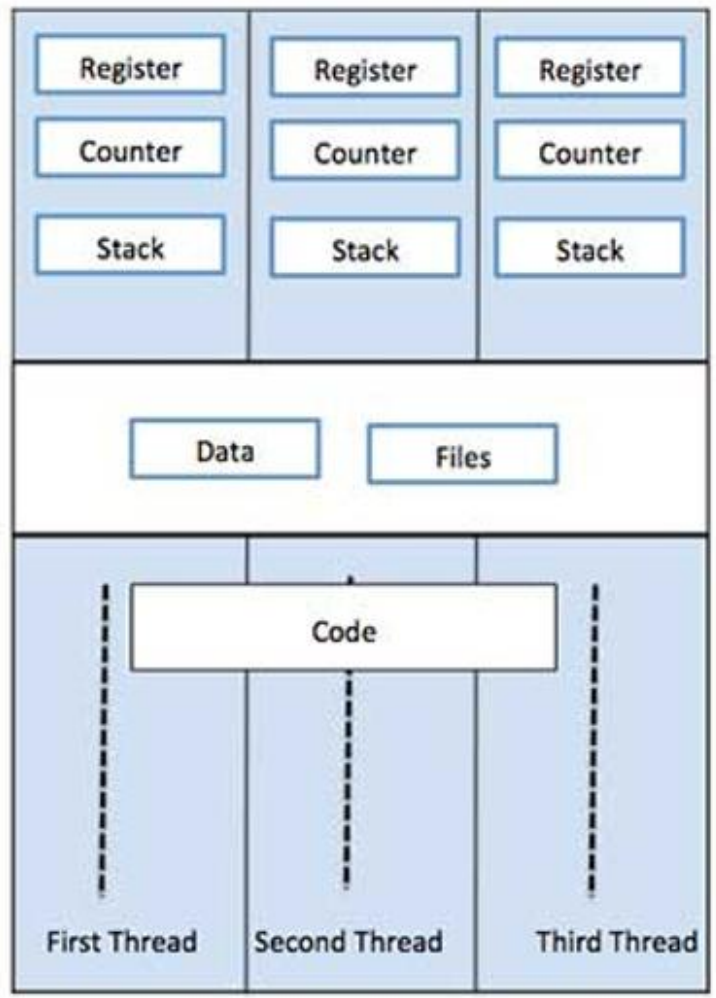


- Thread is a single sequence of execution within a process.
- Thread has it own.
  - **program counter** that keeps track of which instruction to execute next
  - **system registers** which hold its current working variables
  - **stack** which contains the execution history.
- Processes are generally used to execute large, **'heavyweight'** jobs such as working in word, while threads are used to carry out smaller or **'lightweight'** jobs such as auto saving a word document.
- A thread shares few information with its peer threads (having same input) like code segment, data segment and open files.
- Thread is **light weight process** created by a process.

Single Thread VS Multiple Thread



Single Process P with single thread



Single Process P with three threads

Similarities between Process & Thread

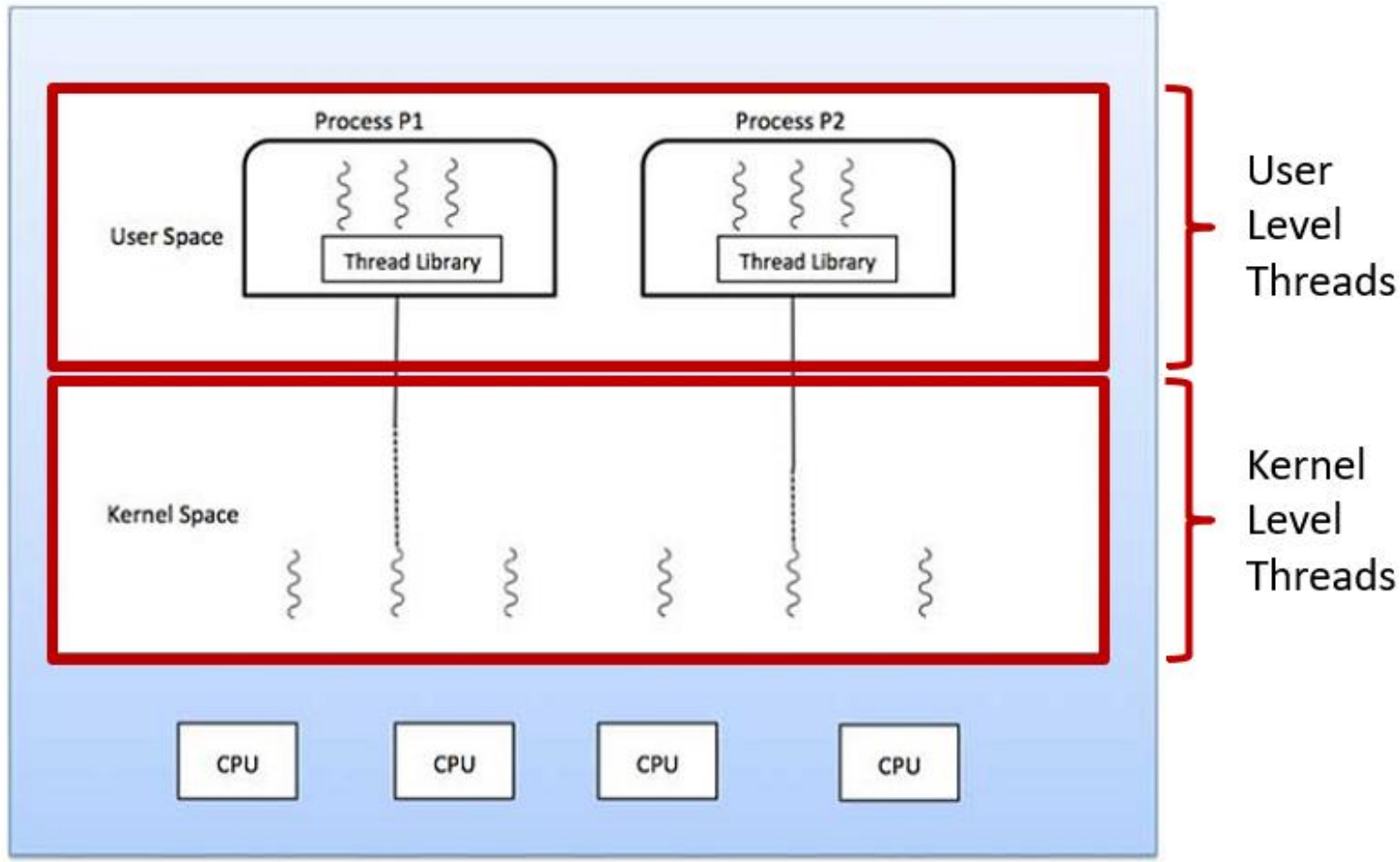
- Like processes threads share CPU and only one thread is running at a time.
- Like processes threads within a process execute sequentially.
- Like processes thread can create children.
- Like a traditional process, a thread can be in any one of several states: running, blocked, ready or terminated.
- Like process threads have Program Counter, Stack, Registers and State.

Dissimilarities between Process & Thread

- Unlike processes threads are not independent of one another.
- Threads within the same process share an address space.
- Unlike processes all threads can access every address in the task.
- Unlike processes threads are design to assist one other. Note that processes might or might not assist one another because processes may be originated from different users.

Types of Threads

1. Kernel Level Thread
2. User Level Thread



Types of Threads (Cont...)

- Click to add text

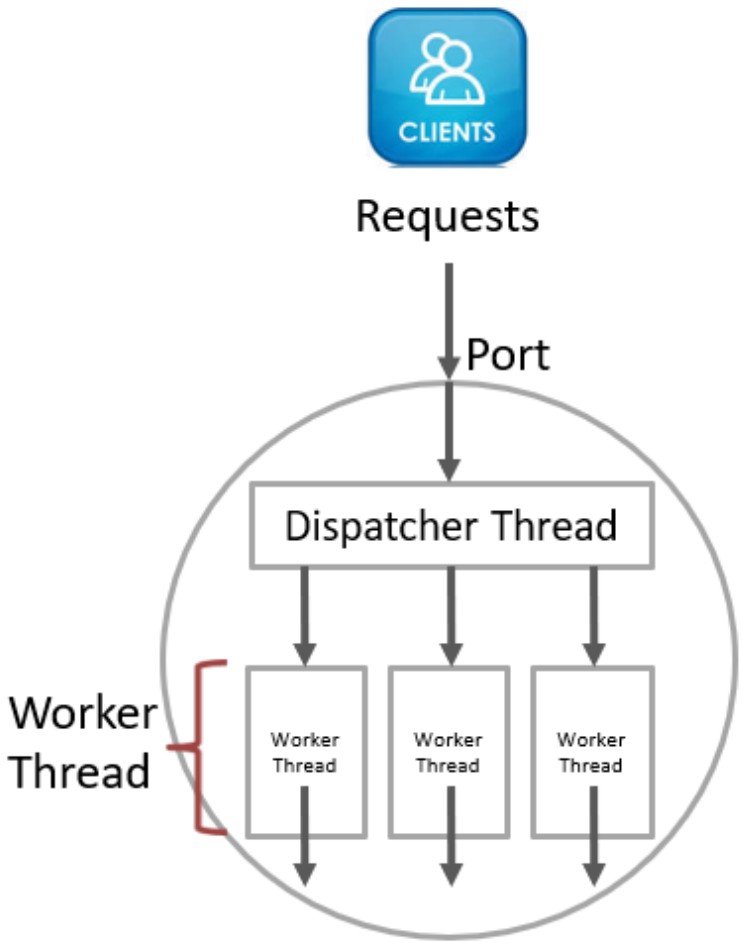
USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread with in same process can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris

Models for Organizing threads

- Depending on the application's needs the threads of a process of the application can be organized in different ways.
- Threads can be organized by three different models.
  - Dispatcher/Worker model
  - Team model
  - Pipeline model

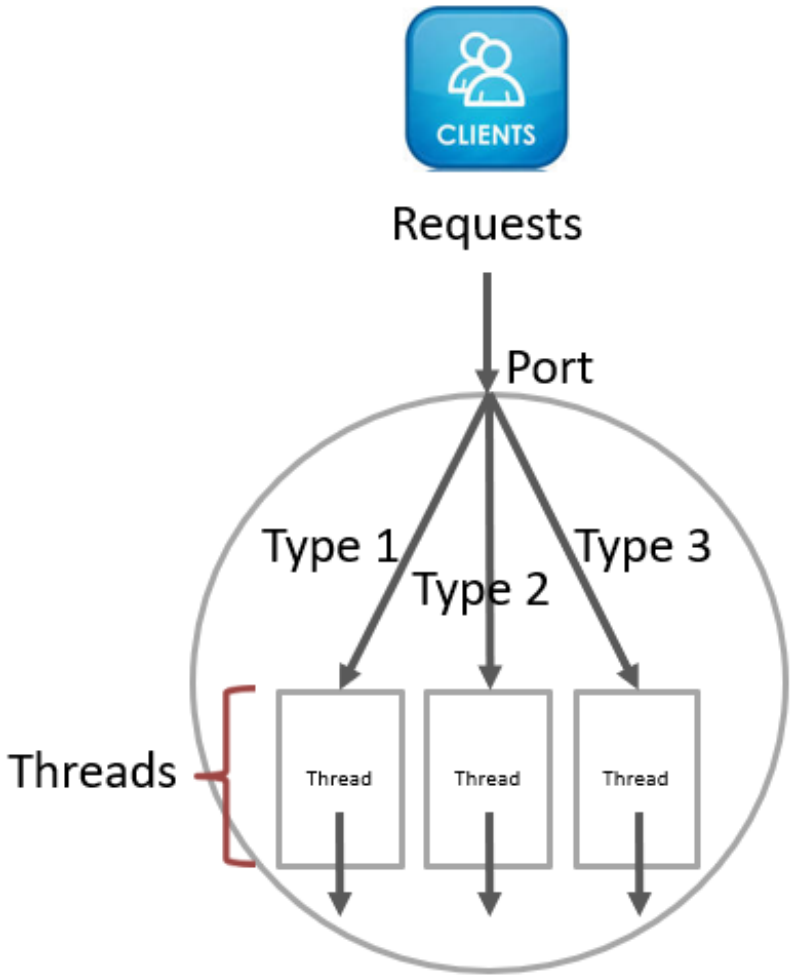
Dispatcher/Worker model

- In this model, the process consists of a single **dispatcher thread** and multiple **worker threads**.
- The dispatcher thread:
  - Accepts requests from clients
  - Examine the request
  - Dispatches the request to one of the free worker threads for further processing of the request.
- Each worker thread works on a different client request.
- Therefore, multiple client requests can be processed in parallel.



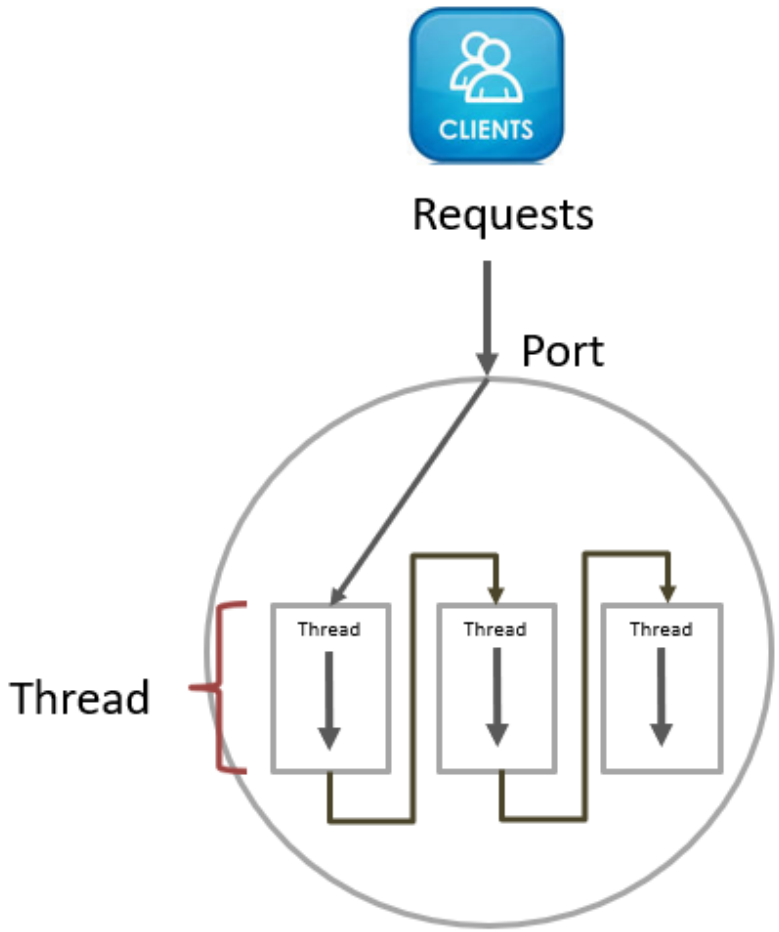
Team model

- In this model, all threads behave as equal.
- Each thread gets and processes clients' requests on its own.
- This model is often used for implementing specialized threads within a process.
- Each thread of the process is specialized in servicing a specific type of requests like copy, save, autocorrect.



Pipeline model

- This model is useful for applications based on the **producer-consumer model.**
- The output data generated by one part of the application is used as input for another part of the application.
- The threads of a process are organized as a **pipeline.**
- The output data generated by the first thread is used for processing by the second thread, the output of the second thread is used for third thread, and so on.
- The output of the last thread in the pipeline is the final output of the process to which the threads belong.



Designing Issues in Thread Package

- A system that supports thread facility must provide a set of primitives to its users for threads-related operations.
- These primitives of the system are said to form a **thread package.**
- Some of the important issues in designing a thread package are:
  - Thread Creation
  - Thread Termination
  - Thread Synchronization
  - Thread Scheduling
  - Signal Handling

Thread Creation

- Threads can be created either statically or dynamically.
- Static :
  - The number of threads of a process remains fixed for its entire lifetime.
  - Memory space is allocate to each thread.
- Dynamic:
  - The number of threads of a process keeps changing dynamically.
  - Threads are created as and when it is needed during the process life cycle.
  - It exit when task is completed.
  - Here the stack size for the threads is specified as parameter to the system call for thread creation.

Thread Termination

- Threads may terminate or never terminate until life cycle of process.
- Thread termination can be done as follows:
  - Thread destroys itself on task completion by making an EXIT call.
  - Thread is killed from outside using KILL command with thread identifier as parameter.
- In some process, all its threads are created immediately after the process start and then these threads are never killed until the process terminates.

Thread Synchronization

- Since threads belongs to same process share the same address space, thread synchronization is required to ensure that multiple threads don't access the same data simultaneously.
- For example:
  1. If two threads want to increment the same global variable with in the same process.
  2. One thread should exclusive access to shared variable, increment it, and then pass control to the other thread.
  3. It mean that only one thread can execute in critical region at any instance of time.

Thread Scheduling

- Another important issue in designing threads package is to decide an appropriate scheduling algorithm.
- Thread packages provide calls to give the users the flexibility to specify the scheduling policy to be used for their applications.
- Some of the special features for threads scheduling that may be supported by a threads package are as follows:
  1. Priority assignment facility
  2. Flexibility to vary quantum size dynamically
  3. Handoff scheduling
  4. Affinity scheduling

Signal Handling

- Signals provide software-generated interrupts and exceptions.
- Interrupts are externally generated disruptions of a thread or process.
- Exceptions are caused by the occurrence of unusual conditions during a thread's execution.
- The two main issues associated with handling signals in a multithreaded environment are as follows:
    1. A signal must be handled properly no matter which thread of the process receives it.
    2. Signals must be prevented from getting lost when another signal of the same type occurs in some other thread before the first one is handled by the thread in which it occurred.
- An approach for handling the former issue is to create a separate exception handler thread in each process.
- Exception handler thread of a process is responsible for handling all exception conditions occurring in any thread of the process.

Processor Allocation OR Task Assignment

- Each process is divided into multiple tasks.
- These tasks are scheduled to suitable processor to improve performance.
- It requires characteristics of all the processes to be known in advance.
- This approach does not take into consideration the dynamically changing state of the system.
- In this approach, a process is considered to be composed of multiple tasks and the goal is to find an optimal assignment policy for the tasks of an individual process.

Assumptions For Task Assignment Approach

- A process has already been split into pieces called tasks.
- The amount of computation required by each task and the speed of each processor are known.
- The cost of processing each task on every node of the system is known.
- The Inter process Communication (IPC) costs between every pair of tasks is known.
- Other constraints, such as resource requirements of the tasks and the available resources at each node, precedence relationships among the tasks, and so on, are also known.

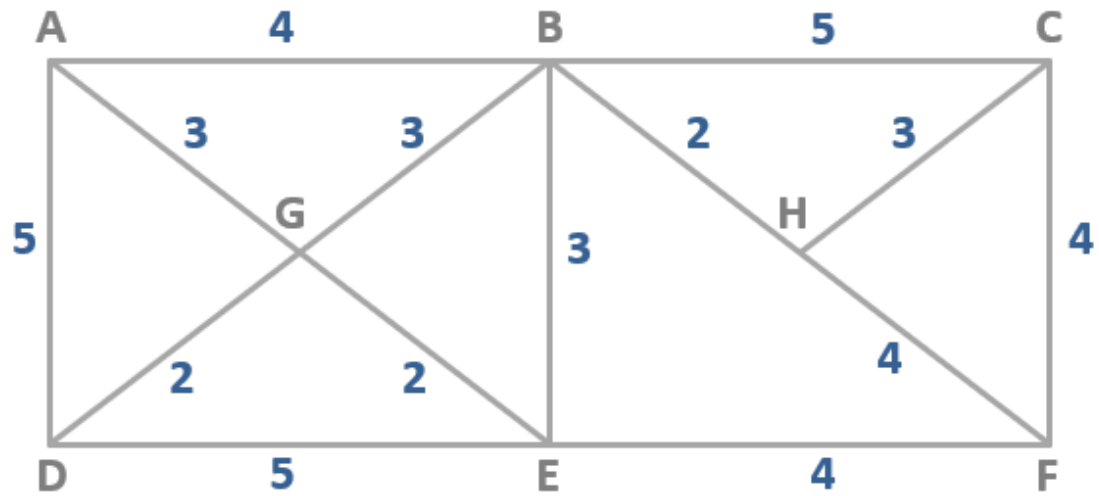
Task Assignment Approach algorithms

1. Graph Theoretic Deterministic Algorithm
2. Centralized Heuristic Algorithm
3. Hierarchical Algorithm

Graph Theoretic Deterministic Algorithm

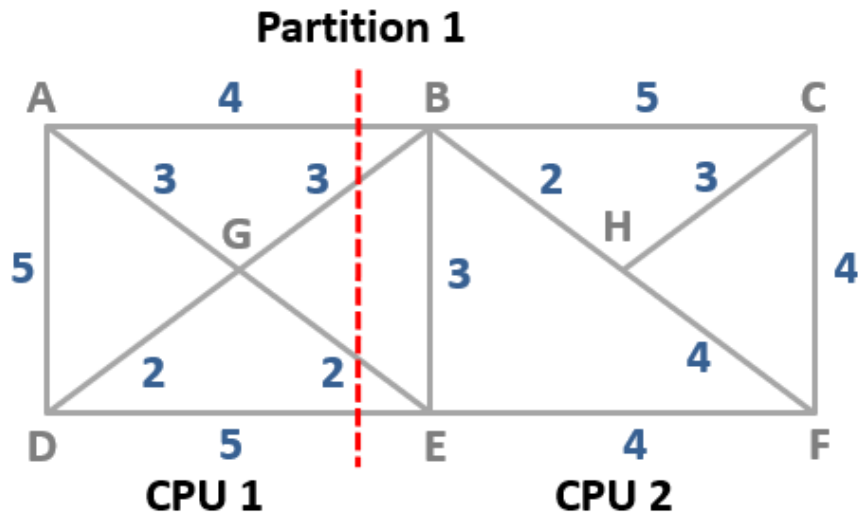
- A system with m CPUs and n processes has any of the following three cases:
  - **m=n:** Each process is allocated to one CPU.
  - **M>n:** Some CPUs may remain idle (free) or work on earlier allocated processes.
  - **M<n:** There is a need to schedule processes on CPUs, and several processes may be assigned to each CPU.
- The main objective of performing CPU assignment is to:
  - Minimize IPC cost.
  - Obtain quick turnaround time.
  - Achieve high degree of parallelism for efficient utilization.
  - Minimize network traffic.

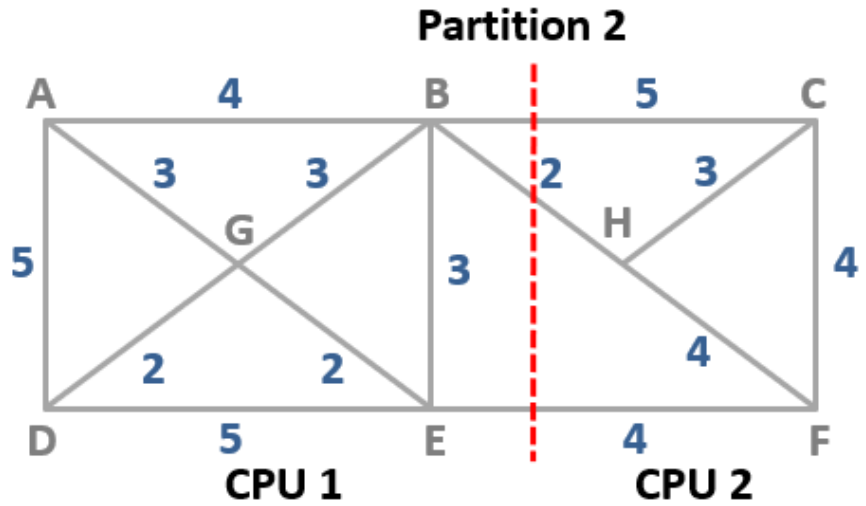
Graph Theoretic Deterministic Algorithm- Example



- Processes are represented as nodes A, B, C, D, E, F, G and H.
- Arcs between sub-graphs represent network traffic and their weights represent IPC costs.
- Total network traffic is the sum of the arcs intersected by the dotted cut lines.

Graph Theoretic Deterministic Algorithm- Example



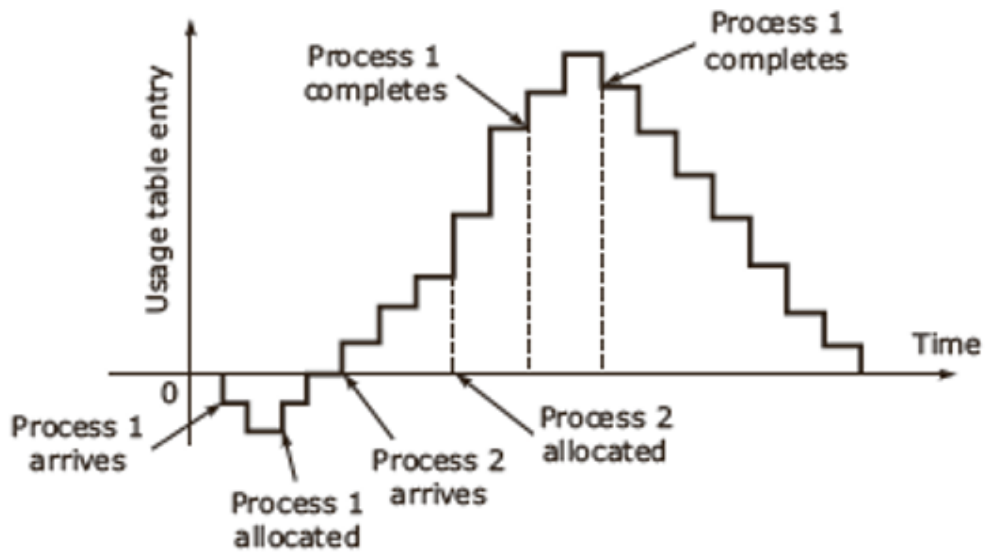


- Partition 1
  - CPU 1 runs A,D,G
  - CPU 2 runs B,E,F,H and C
  - Network traffic= 4+3+2+5=14

- Partition 2
  - CPU 1 runs processes A,D,E,G,B
  - CPU2 runs processes H,C and F
  - Network traffic= 5+2+4=11

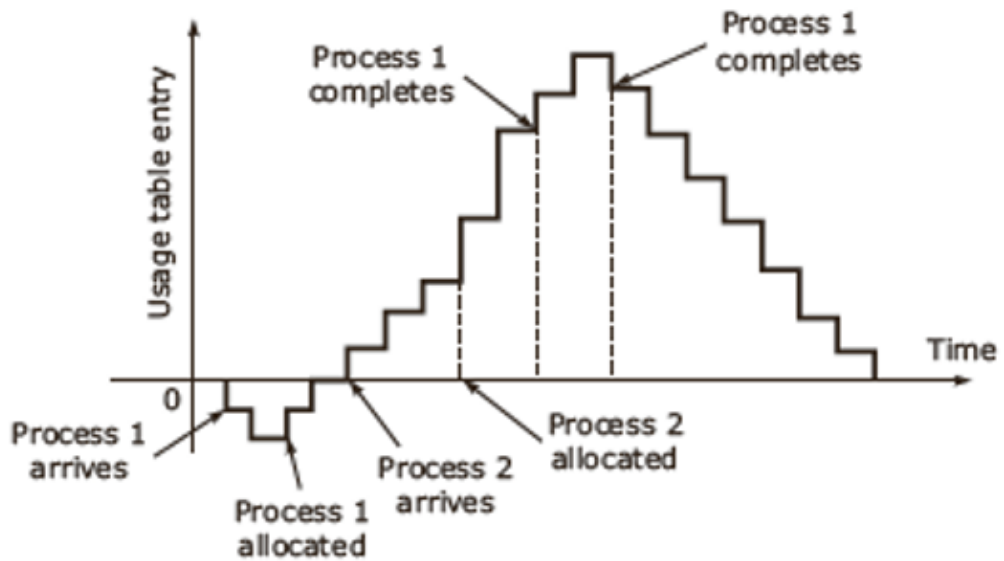
- Thus partition 2 communication generates less network traffic as compared to partition 1.

Centralized Heuristic Algorithm



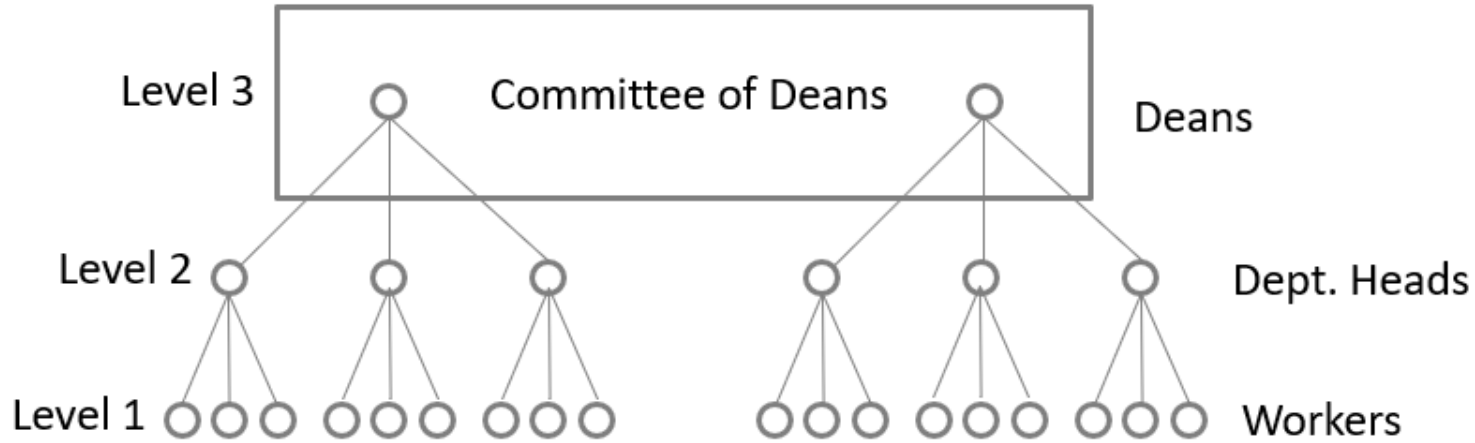
- It is also called Top down algorithm.
- It doesn’t require advance information.
- Coordinator maintains the usage table with one entry for every user (processor) and this is initially zero.

Centralized Heuristic Algorithm



- Usage table entries can either be zero, positive, or negative.
  - Zero value indicates a neutral state.
  - Positive value implies that the machine is using system resources.
  - Negative value means that the machine needs resources.

Hierarchical Algorithm



- Process hierarchy is modelled like an organization hierarchy.
- For each group of workers, one manager machine (department head) is assigned the task of keeping track of who is busy and who is idle.
- If the system is large, there will be number of department heads, so some machines will function as “deans”.
- Each processor maintains communication with one superior and few subordinates.
- When a dean or department head stops functioning (crashes), promote one of the direct subordinates of the faulty manager to fill in for the boss.
- The choice of which can be made by the subordinates themselves.

Scheduling in distributed systems

- A resource manager schedules the processes in a distributed system to make use of the system resources.
- Scheduling is to optimize **resource usage, response time, network congestion**. It can be broadly classified into three types:

1. Task assignment approach
   - In which each process is viewed as a collection of related tasks.
   - These tasks are scheduled to suitable nodes so as to improve performance.
2. Load-balancing approach
   - In which all the processes submitted by the users are distributed among the nodes of the system to equalize the workload among the nodes.
3. Load-sharing approach
   - Which simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed.

TASK ASSIGNMENT APPROACH

Typical **assumptions** found in task assignment work are as follows:

- A process has already been split into pieces called tasks. This split occurs along natural boundaries, so that each task will have integrity in itself and data transfers among the tasks will be minimized.
- The amount of computation required by each task and the speed of each processor are known.
- The cost of processing each task on every node of the system is known. This cost is usually derived based on the information about the speed of each processor and the amount of computation required by each task.
- The interprocess communication (IPC) costs between every pair of tasks is known. The IPC cost is considered zero (negligible) for tasks assigned to the same node. They are usually estimated by an analysis of the static program of a process. For example, during the execution of the process, if two tasks communicate n times and if the average time for each intertask communication is t, the intertask communication cost for the two tasks is $n \times t$.
- Other constraints, such as resource requirements of the tasks and the available resources at each node, precedence relationships among the tasks, and so on, are also known.
- Reassignment of the tasks is generally not possible.

- With these assumptions, the task assignment algorithms seek to assign the tasks of a process to the nodes of the distributed system in such a manner so as **to achieve goals such as the following:**

- Minimization of IPE costs
- Quick turnaround time for the complete process
- A high degree of parallelism
- Efficient utilization of system resources in general

Intertask communication costs						
	t_1	t_2	t_3	t_4	t_5	t_6
t_1	0	6	4	0	0	12
t_2	6	0	8	12	3	0
t_3	4	8	0	0	11	0
t_4	0	12	0	0	5	0
t_5	0	3	11	5	0	0
t_6	12	0	0	0	0	0

(*a*)

Execution costs		
Tasks	Nodes	
	n_1	n_2
t_1	5	10
t_2	2	∞
t_3	4	4
t_4	6	3
t_5	5	2
t_6	∞	4

(*b*)

Serial assignment	
Task	Node
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_2
t_5	n_2
t_6	n_2

(*c*)

Optimal assignment	
Task	Node
t_1	n_1
t_2	n_1
t_3	n_1
t_4	n_1
t_5	n_1
t_6	n_2

(*d*)

Serial assignment execution cost (x) = $x_{11} + x_{21} + x_{31} + x_{42} + x_{52} + x_{62}$
$= 5 + 2 + 4 + 3 + 2 + 4 = 20$

Serial assignment communication cost (c) = $c_{14} + c_{15} + c_{16} + c_{24} + c_{25} + c_{26} + c_{34} + c_{35} + c_{36}$
$= 0 + 0 + 12 + 12 + 3 + 0 + 0 + 11 + 0 = 38$

Serial assignment total cost = $x + c = 20 + 38 = 58$

Optimal assignment execution cost (x) = $x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{62}$
$= 5 + 2 + 4 + 6 + 5 + 4 = 26$

Optimal assignment communication cost (c) = $c_{16} + c_{26} + c_{36} + c_{46} + c_{56}$
$= 12 + 0 + 0 + 0 + 0 = 12$

Optimal assignment total cost = $x + c = 26 + 12 = 38$

Fig. 7.1 A task assignment problem example. (*a*) intertask communication costs; (*b*) execution costs of the tasks on the two nodes; (*c*) serial assignment; (*d*) optimal assignment.

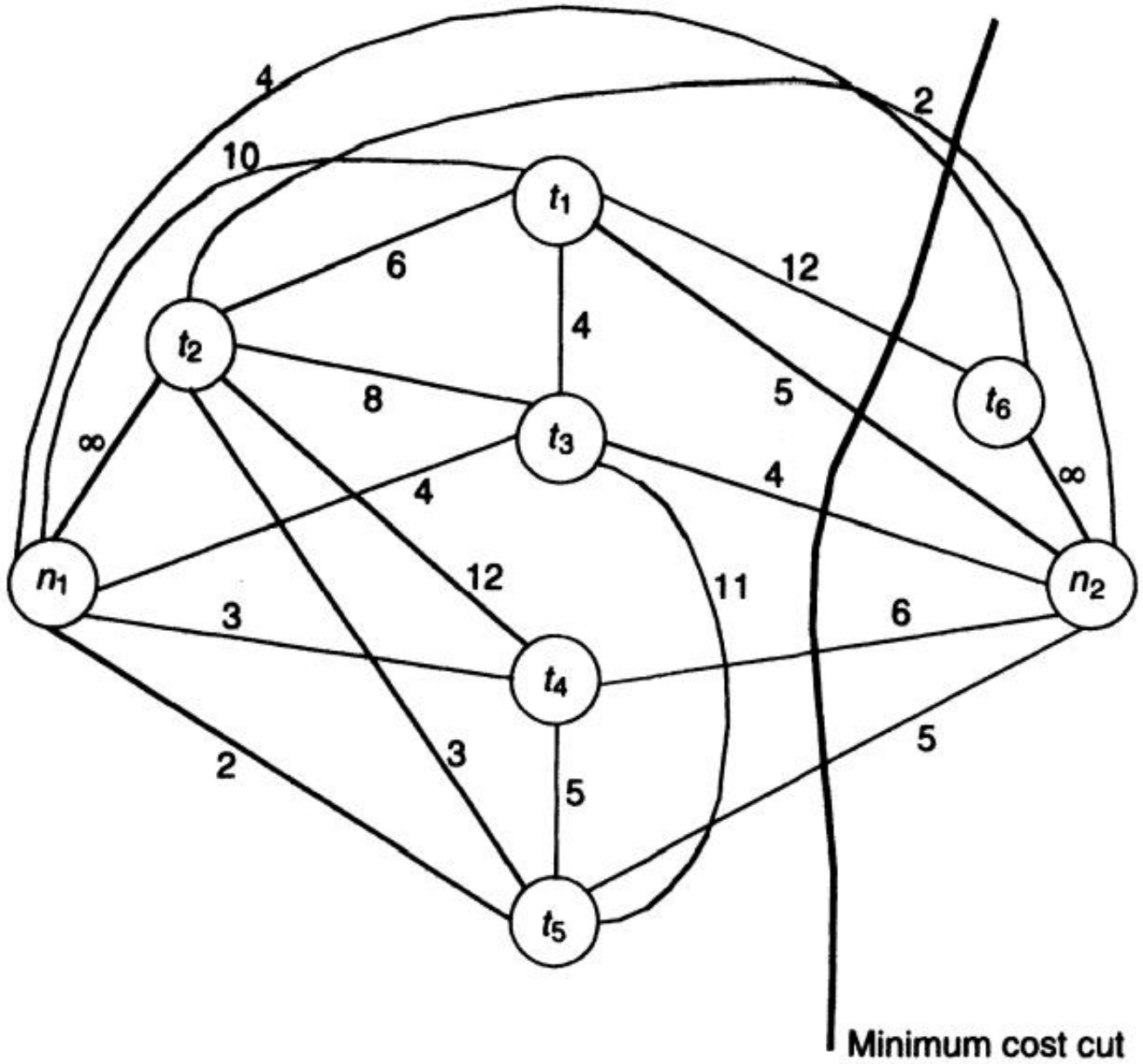


Fig. 7.2 Assignment graph for the assignment problem of Figure 7.1 with minimum cost cut.

Load Balancing Approach

- The **distribution of loads** to the processing elements is simply called the load balancing.
- The goal of the load balancing algorithms is to **maintain the load** to each processing element such that all the processing elements become neither overloaded nor idle.
- Each processing element ideally has **equal load at any moment** of time during execution to obtain the maximum performance (minimum execution time) of the system.

Classification of Load Balancing

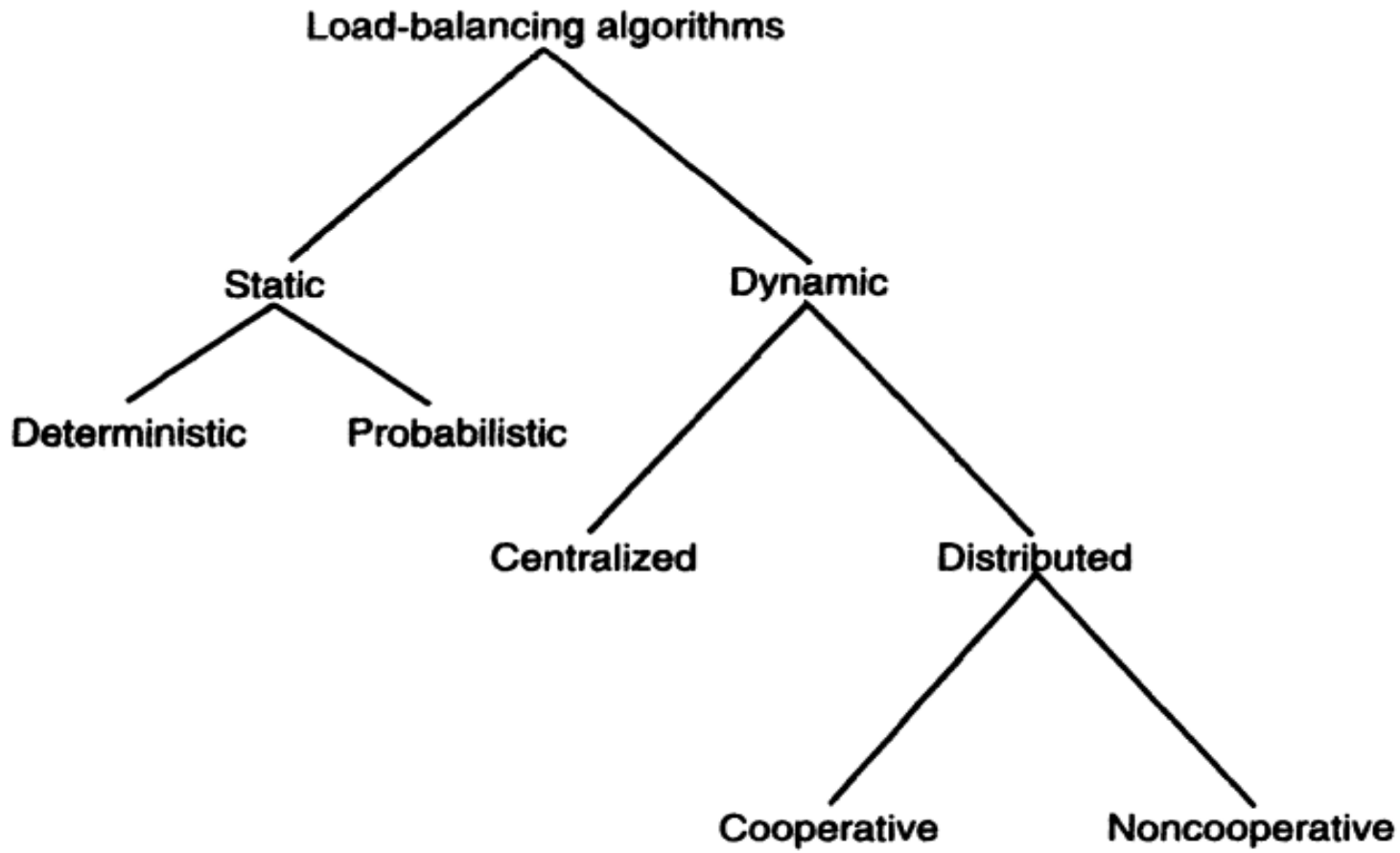


Fig. 7.3 A taxonomy of load-balancing algorithms.

Static Load Balancing

- In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes.
- Once the processes are assigned, no change or reassignment is possible at the run time.
- Number of jobs in each node is fixed in static load balancing algorithm.
- Static algorithms do not collect any information about the nodes.
- The static load balancing algorithms can be divided into two sub classes:
  1. Optimal static load balancing (Deterministic)
  2. Sub optimal static load balancing (Probabilistic)

Static Load Balancing (cntd.)

1. Optimal Static Load Balancing Algorithm
   - If all the information and resources related to a system are known optimal static load balancing can be done.
2. Sub optimal static load balancing Algorithm
   - Sub-optimal load balancing algorithm will be mandatory for some applications when optimal solution is not found.

Deterministic vs Probabilistic

- Deterministic algorithms are suitable when the process behavior is known in advance.
- If all the details like list of processes, computing requirements, file requirements and communication requirements are known prior to execution, then it is possible to make a perfect assignment.
- In the case load is unpredictable or variable from minute to minute or hour to hour, a Probabilistic/heuristic processor allocation is preferred.

Dynamic Load Balancing

- In dynamic load balancing algorithm assignment of jobs is done at the runtime.
- In DLB jobs are reassigned at the runtime depending upon the situation.
- The load will be transferred from heavily loaded nodes to the lightly loaded nodes.
- No decision is taken until the process gets executed.
- This strategy collects the information about the system state and about the job information.
- As more information is collected, the algorithm can make better decision.

Design Issues of load-balancing Algorithms

Designing a good load-balancing algorithm is a difficult task because of the following issues:

- Load estimation policy, which determines how to estimate the workload of a particular node of the system
- Process transfer policy, which determines whether to execute a process locally or remotely
- State information exchange policy, which determines how to exchange the system load information among the nodes
- Location policy, which determines to which node a process selected for transfer should be sent
- Priority assignment policy, which determines the priority of execution of local and remote processes at a particular node
- Migration limiting policy, which determines the total number of times a process can migrate from one node to another

Load Estimation Policies

A node's workload can be estimated based on some measurable parameters.

These parameters could include time dependent and node-dependent factors such as the following:

- Total number of processes on the node at the time of load estimation
- Resource demands of these processes
- Instruction mixes of these processes
- Architecture and speed of the node's processor

- Several load-balancing algorithms *use the total number of processes present on the node* as a measure of the node's workload.
- However, this is an *unsuitable measure for such an estimate since the true load could vary widely* depending on the remaining service times for those processes.
- Another measure used for estimating a node's workload is the sum of *the remaining service times of all the processes on that node*.
- However, *how to estimate the remaining service time of the processes*.
- Bryant and Finkel [1981] have proposed the *use of one of the following methods for this purpose.*

    1. *Memoryless method*
    2. *Pastrepeats*
    3. *Distribution method*

Process Transfer Policies

- The strategy of load-balancing algorithms is based on the idea of transferring some processes from the heavily loaded nodes to the lightly loaded nodes for processing.
- To facilitate this, it is necessary to devise a policy to decide whether a node is lightly or heavily loaded.
- Most of the load-balancing algorithms use the threshold policy to make this decision.
- The threshold value of a node may be determined by any of the following methods:
  - **Static policy:-** Threshold value does not vary with the dynamic changes in workload at local or remote nodes.
  - **Dynamic policy:-**
    - In this method, the threshold value of a node (n_i) is calculated as a product of the average workload of all the nodes and a predefined constant (c_i).
    - For each node n_i, the value of c_i, depends on the processing capability of node n_i, relative to the processing capability of an other nodes.

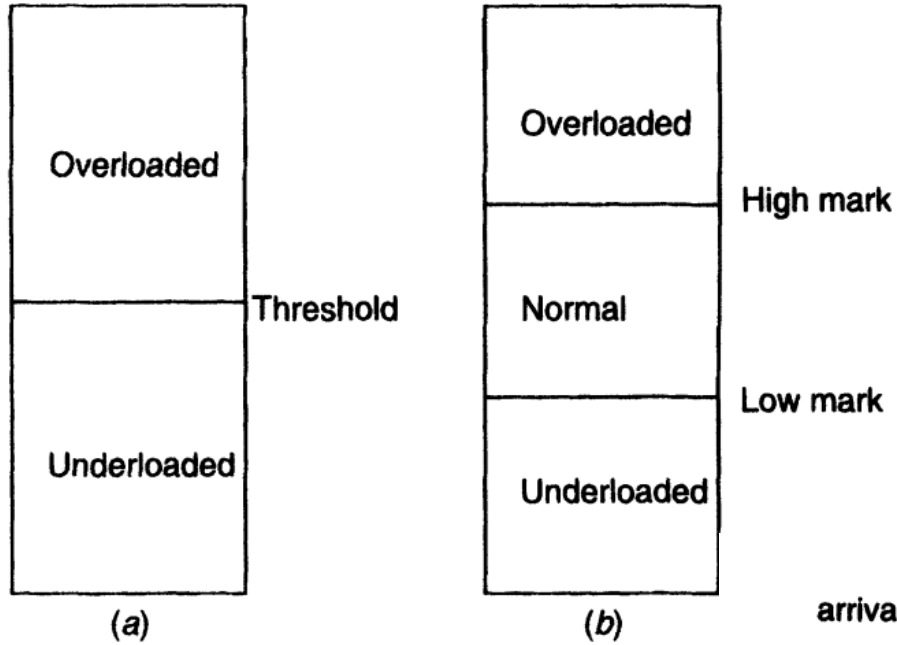


Fig. 7.4 The load regions of (*a*) single-threshold policy and (*b*) double-threshold policy.

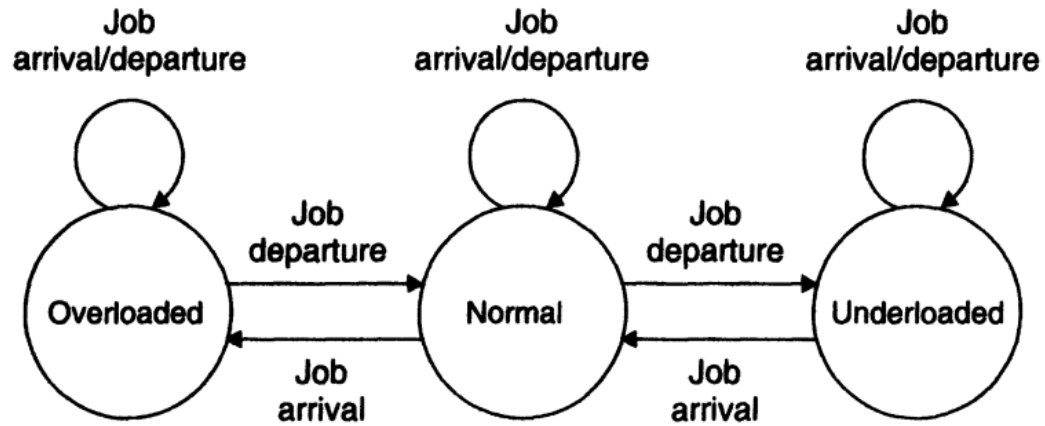


Fig. 7.5 State transition diagram of the load of a node in case of double-threshold policy.

- A node should only transfer one or more of its processes to another node if such a transfer greatly improves the performance of the rest of its local processes.
- A node should accept remote processes only if its load is such that the added workload of processing these incoming processes does not significantly affect the service to the local ones.

Location Policies

- Once a decision has been made through the transfer policy to transfer a process from a node, the next step is to select the destination node for that process's execution.
- This selection is made by the location policy of a scheduling algorithm.
- Few of them are:
  - **Threshold:-** A destination node is selected at .random and a check is made to determine whether the transfer of the process to that node would place it in a state that prohibits the node to accept remote processes.
  - **Shortest:-** The process is transferred to the node having the minimum load value, unless that node's load is such that it prohibits the node to accept remote processes.
  - **Bidding:-** Each node in the network is responsible for two roles with respect to the bidding process: manager and contractor.
  - **Pairing:-** The method of accomplishing load balancing by the pairing policy is to reduce the variance of loads only between pairs of nodes of the system.

State Information Exchange Policies

- The proposed load-balancing algorithms use one of the following policies for this purpose:
- **Periodic Broadcast.** In this method each node broadcasts its state information after the elapse of every t units of time.
- **Broadcast When State Changes.** This method avoids the problem of fruitless message exchanges of the periodic broadcast method by ensuring that a node broadcasts its state information only when the state of the node changes.
- **On-Demand Exchange.** A node needs to know about the state of other nodes only when it is either underloaded or overloaded. The method of on-demand exchange of state information is based on this observation.
- **Exchange by Polling.** When a node needs the cooperation of some other node for load balancing, it can search for a suitable partner by randomly polling the other nodes one by one.

Priority Assignment Policies

1. **Selfish.** Local processes are given higher priority than remote processes.
2. **Altruistic.** Remote processes are given higher priority than local processes.
3. **Intermediate.** The priority of processes depends on the number of local processes and the number of remote processes at the concerned node.

Migration-Limiting Policies:

To decide about the total number of times a process should be allowed to migrate.
One of the following two policies may be used for this purpose:

Uncontrolled:

- A remote process arriving at a node is treated just as a process originating at the node.
- A process may be migrated any number of times.
- This policy has the unfortunate property of causing instability.

Controlled:

- The upper limit of the value of migration count is fixed to 1, and hence a process cannot be migrated more than once under this policy.
- However, some system designers feel that multiple process migrations, especially for long processes, may be very useful for adapting to the dynamically changing states of the nodes.
- Thus this group of designers sets the upper limit of the value of migration count to some value k > 1. The value of k may be decided either statically or dynamically.

LOAD-SHARING APPROACH

- ✓ Because the overhead involved in gathering state information to achieve this objective is normally very large, especially in distributed systems having a large number of nodes.
- ✓ Moreover, load balancing in the strictest sense is not achievable because the number of processes in a node is always fluctuating and the temporal unbalance among the nodes exists at every moment.
- ✓ Therefore this rectification is often called dynamic load sharing instead of dynamic load balancing.

Issues in Designing load-Sharing Algorithm

- ❖ The design of a load-sharing algorithm also requires that proper decisions be made regarding load estimation policy, process transfer policy, state information exchange policy, location policy, priority assignment policy, and migration limiting policy.
- ❖ Unlike load balancing algorithms, load-sharing algorithms do not attempt to balance the average workload on all the nodes of the system. Rather, they only attempt to ensure that no node is idle when a node is heavily loaded

Other policies for load sharing are described below:

- Load Estimation Policies
- Process Transfer Policies
- Location Policies
  - Sender-initiated policy, in which the sender node of the process decides where to send the process
  - Receiver-initiated policy, in which the receiver node of the process decides from where to get the process
- State Information Exchange Policies
  - Broadcast When State Changes.
  - Poll When State Changes.

DISTRIBUTED FILE SYSTEMS

Dr. K. Jairam Naik
NIT Raipur

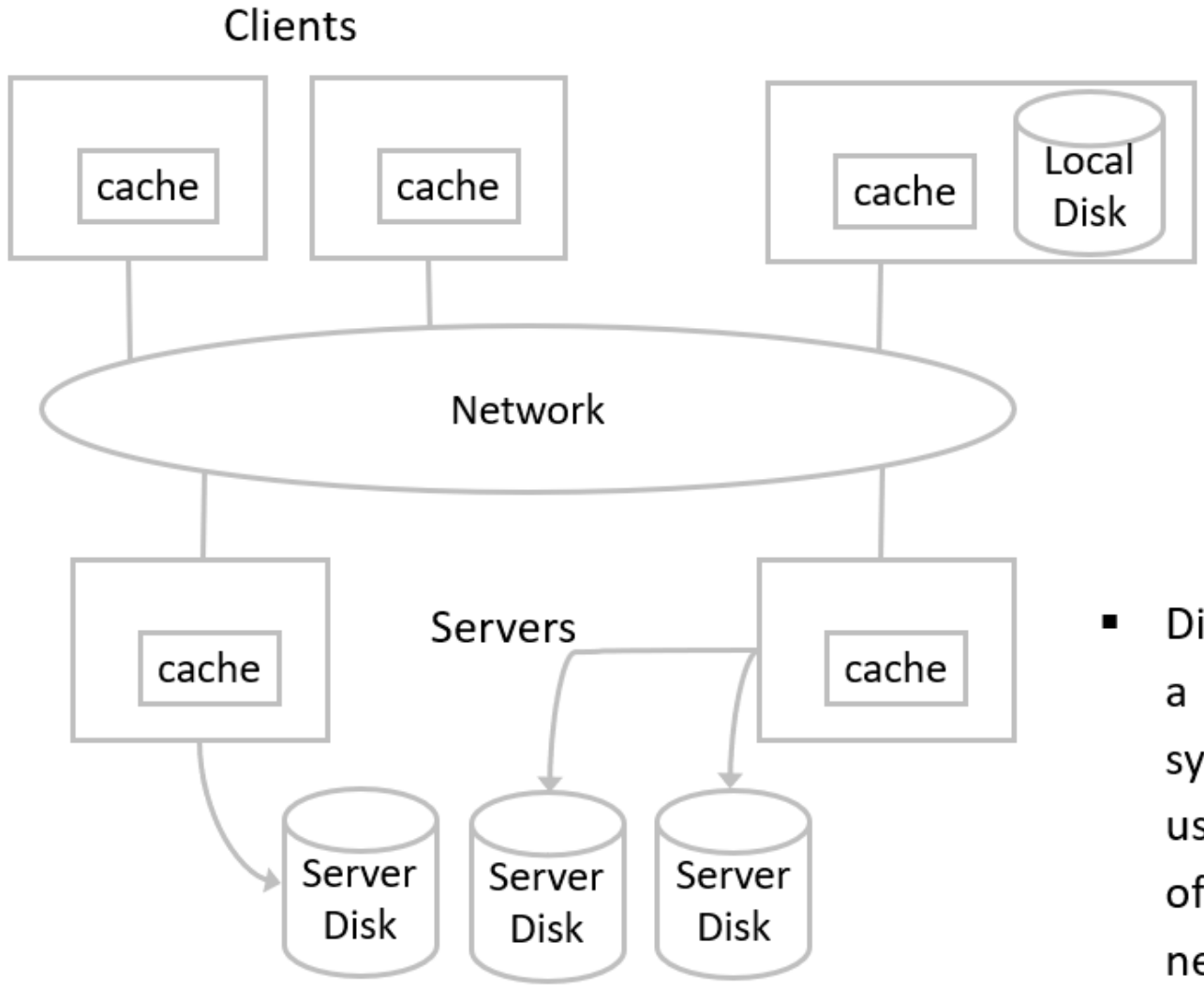
Topics . . .

- **Distributed File Systems**
  - Introduction
  - Features & goal of distributed file system
  - File models
  - File accessing models
  - File sharing semantics
  - File caching scheme
  - File replication
  - Fault tolerance

What is Distributed file system?

- A distributed file system (DFS) is a file system with data stored on a server.
- The data is accessed and processed as if it was stored on the local client machine.
- The DFS makes it convenient to share information and files among users on a network in a controlled and authorized way.
- Server allows the client users to share files and store data just like they are storing the information locally.
- Servers have full control over the data and give access control to the clients.

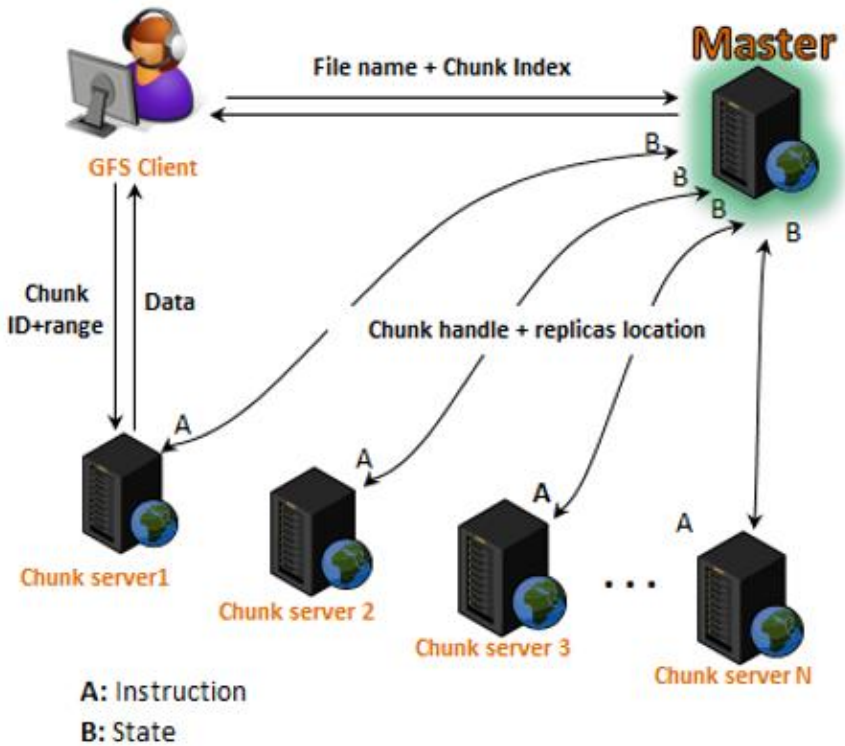
Distributed file system



- Distributed file system is a part of distributed system that provides a user with a unified view of the files on the network.

Distributed file system- Example

- Google File System (GFS or GoogleFS) is a distributed file system developed by Google to provide efficient, reliable access to data using large clusters of hardware.



Benefits of distributed file system

- Distributed file system supports the following:
  1. **Remote information sharing:** It allows a file to be transparently accessed by processes of any node of the system irrespective of the file's location.
  2. **User mobility:** It implies that a user should have the flexibility to work on different nodes at different times.
  3. **Availability:** Files should be available for use even in the event of temporary failure of one or more nodes of the system.
  4. **Diskless workstations:** A distributed file system with transparent remote-file accessing capability, allows the use of diskless workstations in a system.

Services of distributed file system

- Distributed file system provides following types of services:
  1. **Storage service/Disk service:** It deals with the allocation and management of space on a secondary storage device that is used for storage of files in the file system.
  2. **True file service:** It is concerned with the operations on individual files, such as operations for accessing and modifying the data in files and for creating and deleting files.
  3. **Name/Directory service:** It provides a mapping between text names for files and references to files, that is, file IDs.

Desirable features of a good distributed file system

1. Transparency

I. Structure transparency

- Clients should not know the number or locations of file servers and the storage devices.

II. Access transparency

- Both local and remote files should be accessible in the same way.
- The file system should automatically locate an accessed files and arrange for the transport of data to the client's site.

III. Naming transparency

- The name of a file should give no hint as to where the file is located.
- The name of the file must not be changed when moving from one node to another.

IV. Replication transparency

- If a file is replicated on multiple nodes, both the existence of multiple copies and their locations should be hidden from the clients.

Desirable features of a good distributed file system(cntd.)

2. **User mobility**
   - The user should not be forced to work on a specific node but should have the flexibility to work on different nodes at different times.
   - This can be achieved by automatically bringing the users environment to the node where the user logs in.
3. **Performance**
   - Performance is measured as the average amount of time needed to satisfy client requests.
   - This time includes CPU time + time for accessing secondary storage + network access time.
   - It is desirable that the performance of a distributed file system should be comparable to that of a centralized file system.
4. **Simplicity and ease of use**
   - User interface to the file system must be simple and the number of commands should be as small as possible.

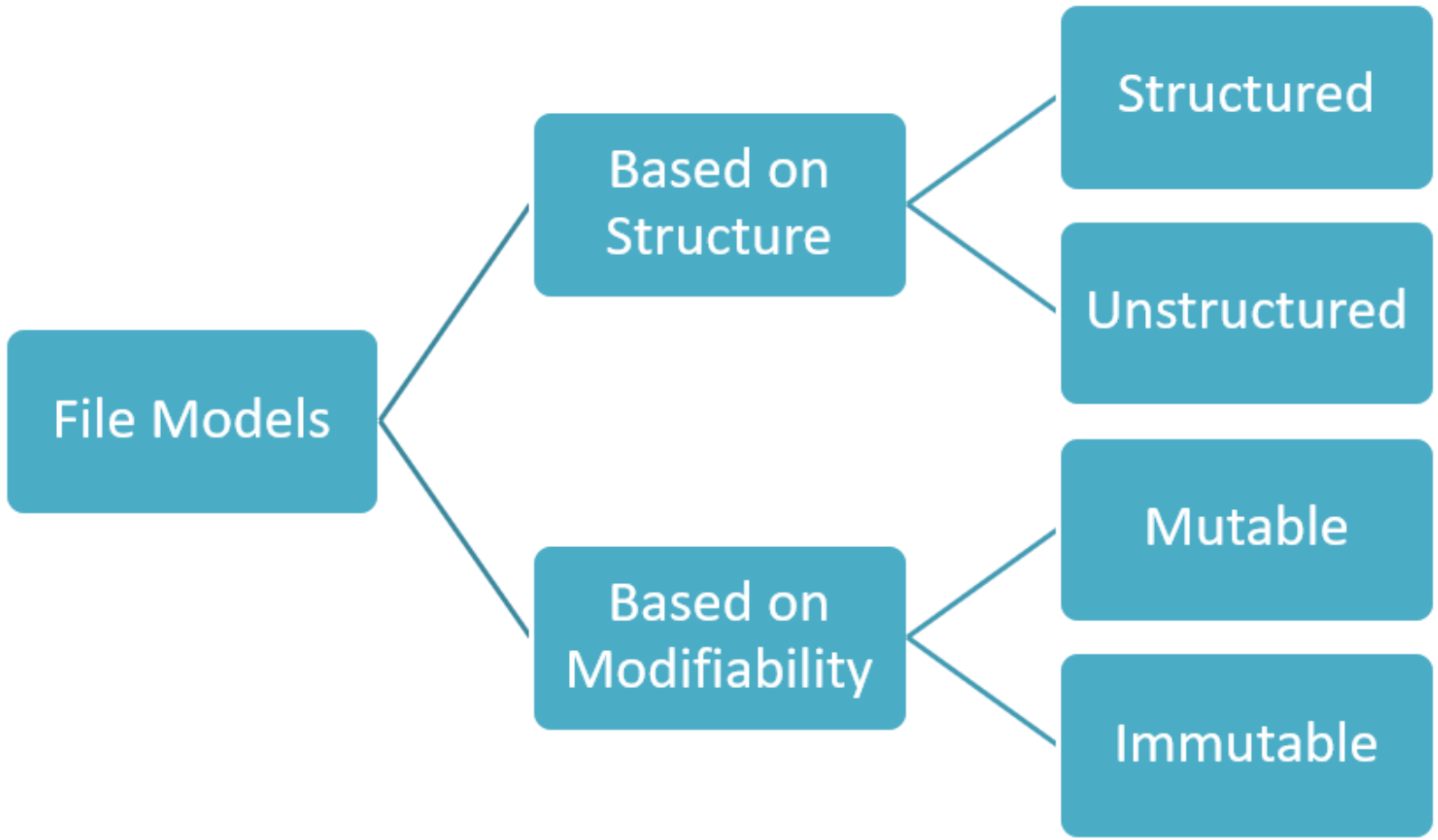
Desirable features of a good distributed file system(cntd.)

5. **Scalability**
   - A good distributed file system should be designed to easily cope with the growth of nodes and users in the system.
   - Such growth should not cause serious disruption of service or significant loss of performance to users.
6. **High availability**
   - A distributed file system should continue to function even when partial failures occur due to the failure of one or more components.
   - It should have multiple and independent file servers controlling multiple and independent storage devices.
7. **High reliability**
   - The probability of loss of stored data should be minimized.
   - System should automatically generate backup copies of critical files that can be used in the event of loss of the original ones..

Desirable features of a good distributed file system(cntd.)

8. **Data integrity**
   - Concurrent access requests from multiple users must be properly synchronized by the use of some concurrency control mechanism.
   - Atomic transactions are a high-level concurrency control mechanism provided to users by a file system for data integrity.
9. **Security**
   - Users should be confident about the privacy of their data.
   - Necessary security mechanisms must be implemented against unauthorized access of files.
10. **Heterogeneity**
   - There should be easy access to shared data on diverse platforms (e.g. Unix workstation, Wintel platform etc.).

File models

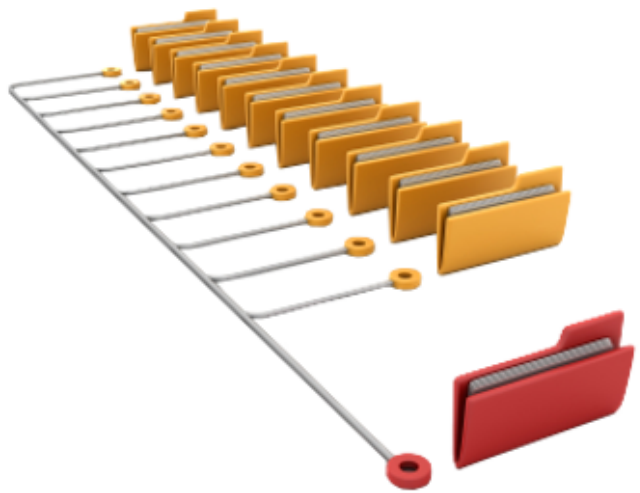

File Models
Based on Structure
Structured
Unstructured
Based on Modifiability
Mutable
Immutable

Unstructured files

- In this model, there is no substructure known to the file server.
- Contents of each file of the file system appears to the file server as an uninterpreted sequence of bytes.
- Interpretation of the meaning and structure of the data stored in the files are entirely up to the application programs.
- UNIX, MS-DOS and other modern operating systems use this file model.
- This is mainly because sharing of a file by different applications is easier compared to the structured file model.

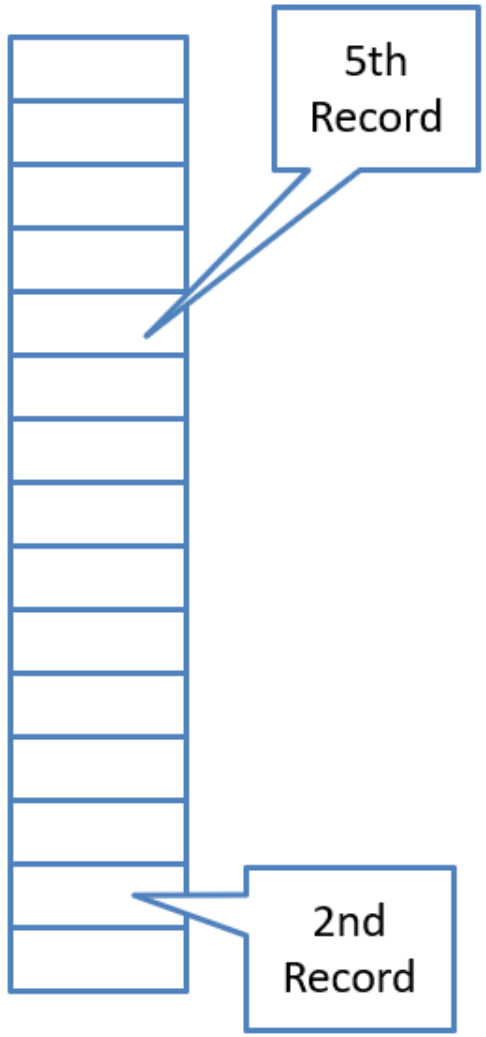
Structured files

- In structured files (rarely used now) a file appears to the file server as an ordered sequence of records.
- Records of different files of the same file system can be of different sizes.
- NetBSD operating system use this file model.
- Two types of structured files are:
  1. Files with non indexed records
  2. Files with indexed records



Structured files(cntd.)

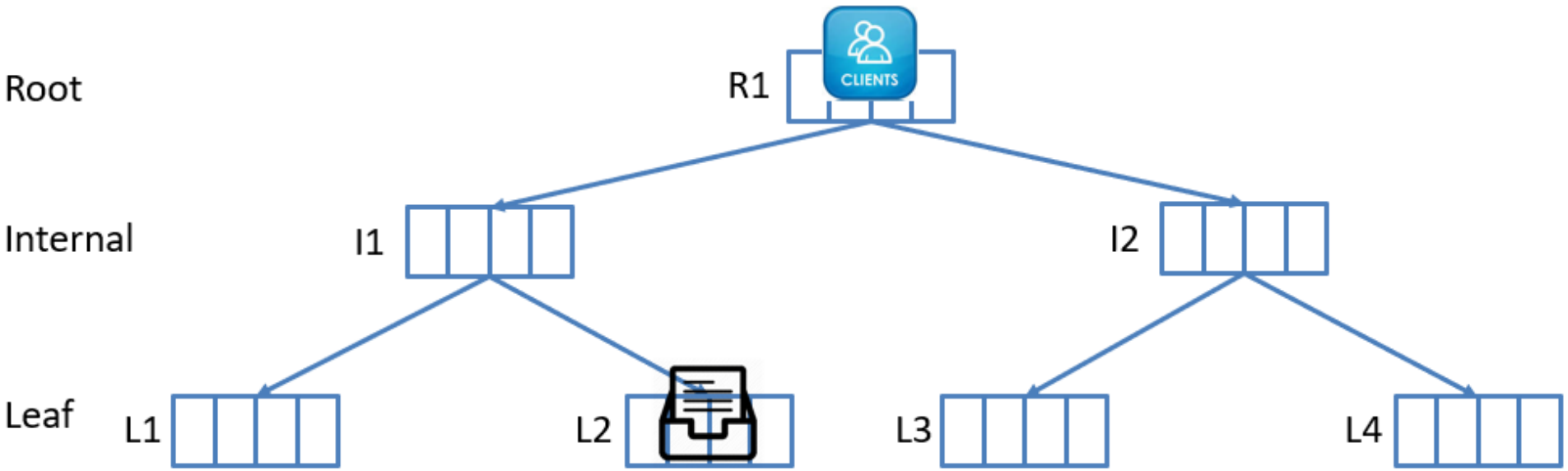
1. **Files with non indexed records**
   - File record is accessed by specifying its position within the file.
   - For example, the fifth record from the beginning of the file or the second record from the end of the file.



Structured files(cntd.)

2. Files with indexed records

- Records have one or more key fields and can be addressed by specifying the values of the key fields.
- File is maintained as B-tree or other suitable data structure or hash table to locate records quickly.

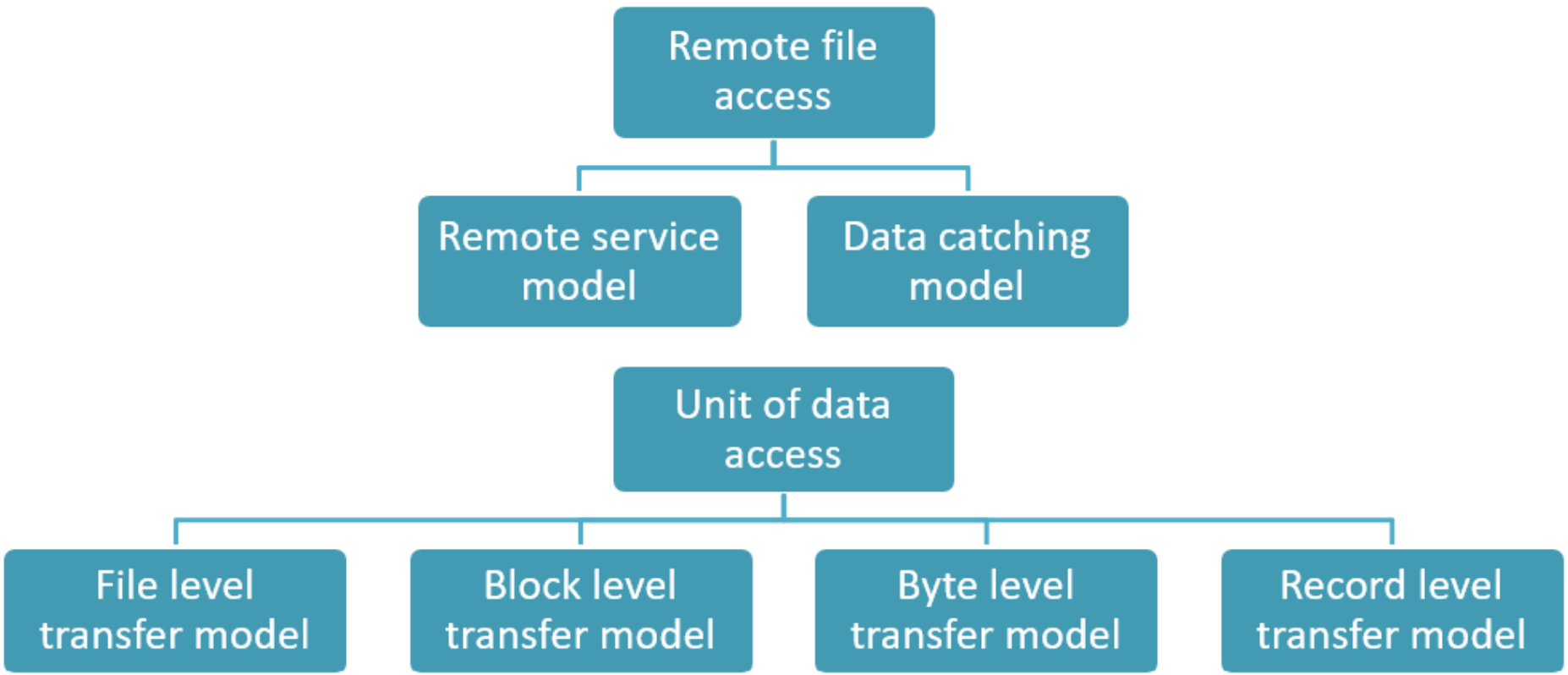


Mutable and Immutable files

MUTABLE FILES	IMMUTABLE FILES
Update performed on a file overwrites on its old contents to produce the new contents.	Rather than updating the same file, a new version of the file is created each time a change is made to the file contents.
A file can be modified by each update operation.	A file cannot be modified once it has been created except to be deleted.
File overwrites its old contents to produce the new contents.	File versioning approach is used to implement file updates.
Most existing operating systems use the mutable file model.	It is rarely used now a days.

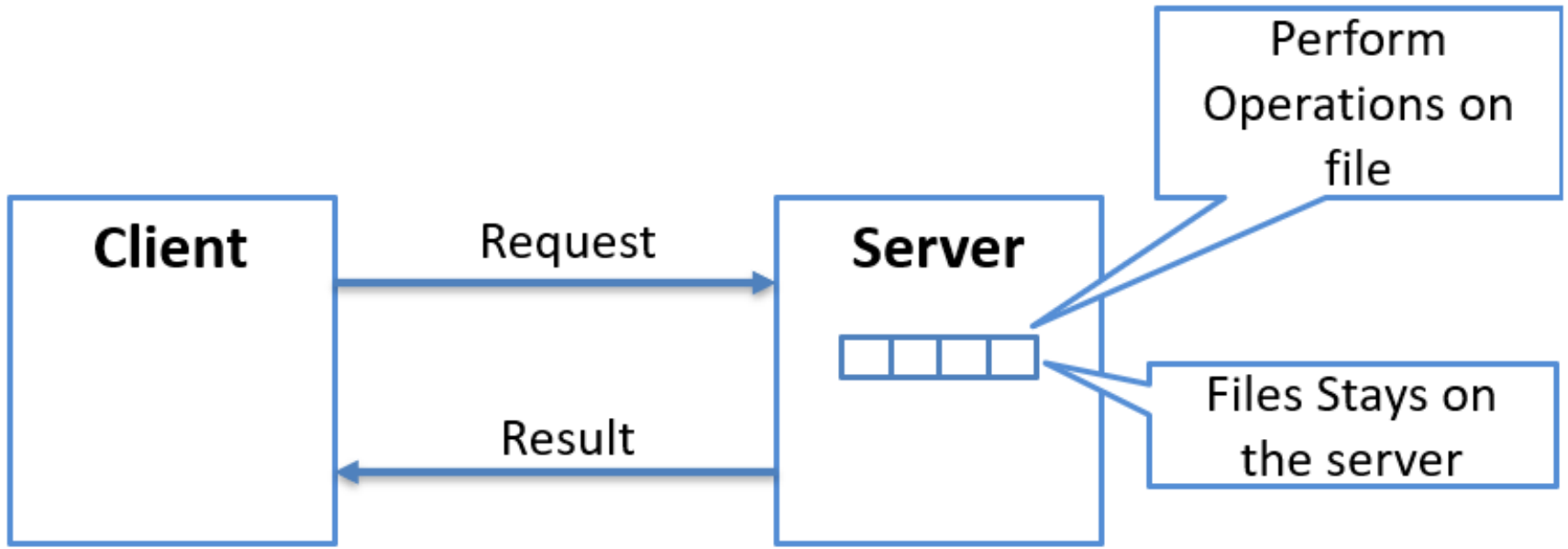
File accessing models

- The file accessing model of a distributed file system mainly depends on two factors:
  1. The method used for accessing remote files
  2. The unit of data access



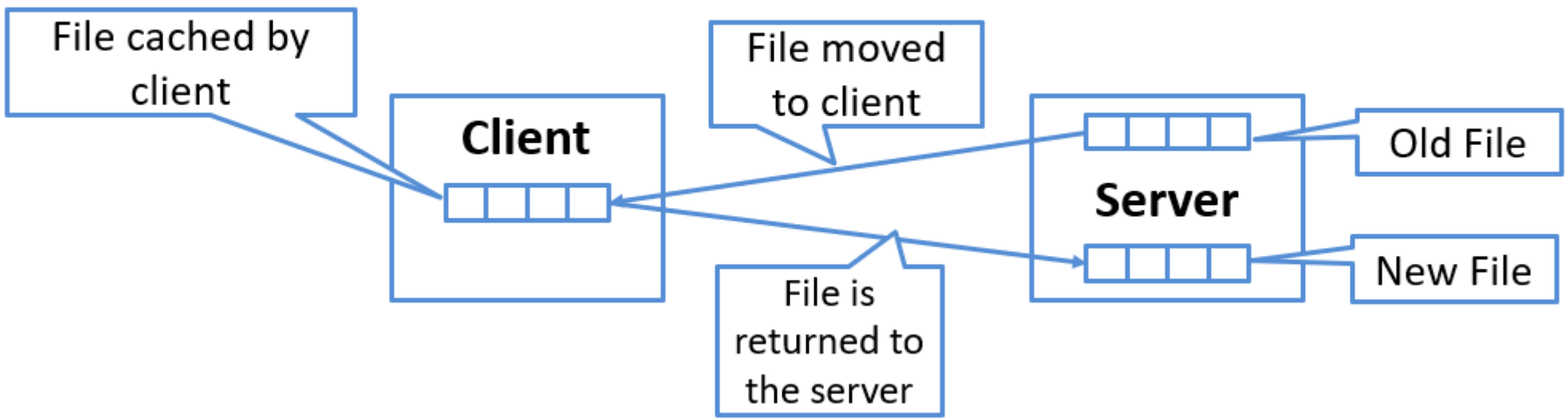
Remote service model

- Client's request for file access is delivered to server, the server machine performs on it and result is forwarded back to the client.
- Requests and replies transferred across network as messages.



Data catching model

- It attempts to reduce the amount of network traffic by taking advantage of the locality feature found in file accesses.
- Data is copied from the server's node to the client's node and is cached there.
- The client's request is processed on the client's node itself by using the cached data.



Unit of data transfer

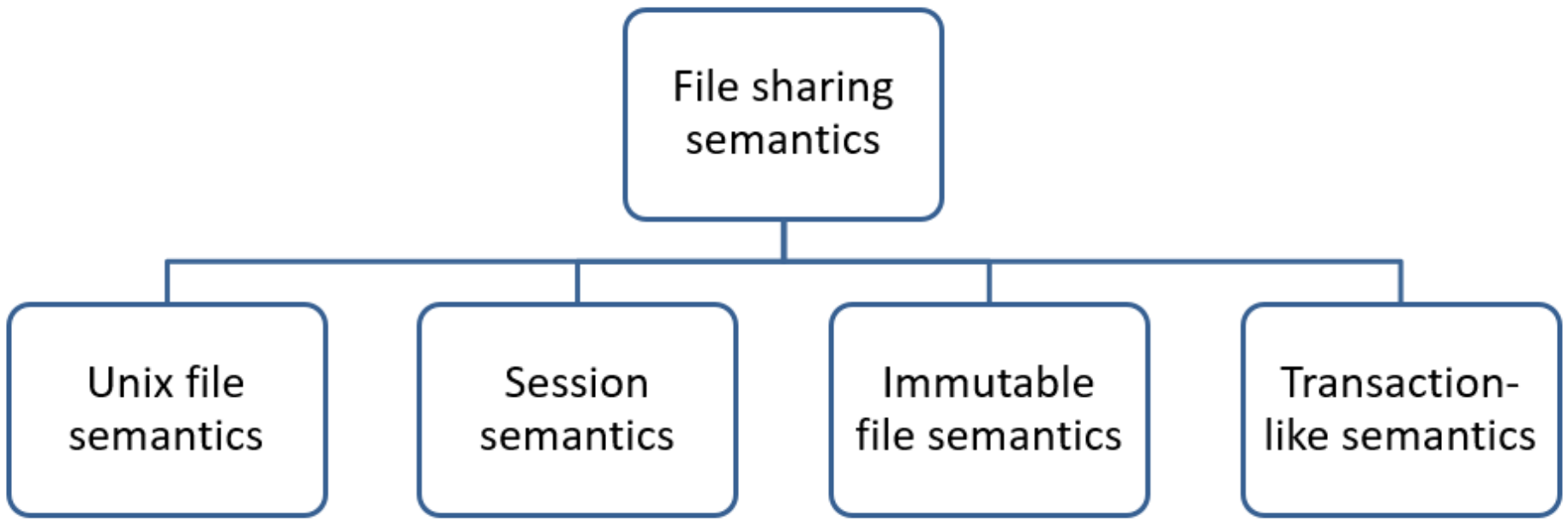
- Unit of data transfer refers to the fraction (or its multiples) of a file data that is transferred to and from clients as a result of a single read or write operation.
- The four commonly used data transfer models based on this factor are as follows:
  1. File-level transfer model
  2. Block-level transfer model
  3. Byte-level transfer model
  4. Record-level transfer model

Comparison of unit of data transfer models

Type of model	Unit of transfer	Advantages	Disadvantages
File-level (Amoeba, AFS)	File	• Fewer accesses to file server and reduces network traffic. • Good for small-sized files. • Supports heterogeneous environments.	• Not suitable for diskless work-stations. • Not suitable for large-sized files. • Network bandwidth and storage space are wasted if only a small part of the file is needed.
Block-level (LOCUS, Sprite)	Block	• Storage space is saved. • Suitable for diskless workstations.	• For large files, there is a need to make multiple requests for accessing the same file. • Increases network traffic.
Byte-level (Cambridge file server)	Byte	• Flexibility for any range of data storage and retrieval.	• Cache management is difficult.
Record-level (Research Storage)	Record	• Easier to protect data. • Ideal for database environment.	• Increases network traffic in case large number of records have to be accessed.

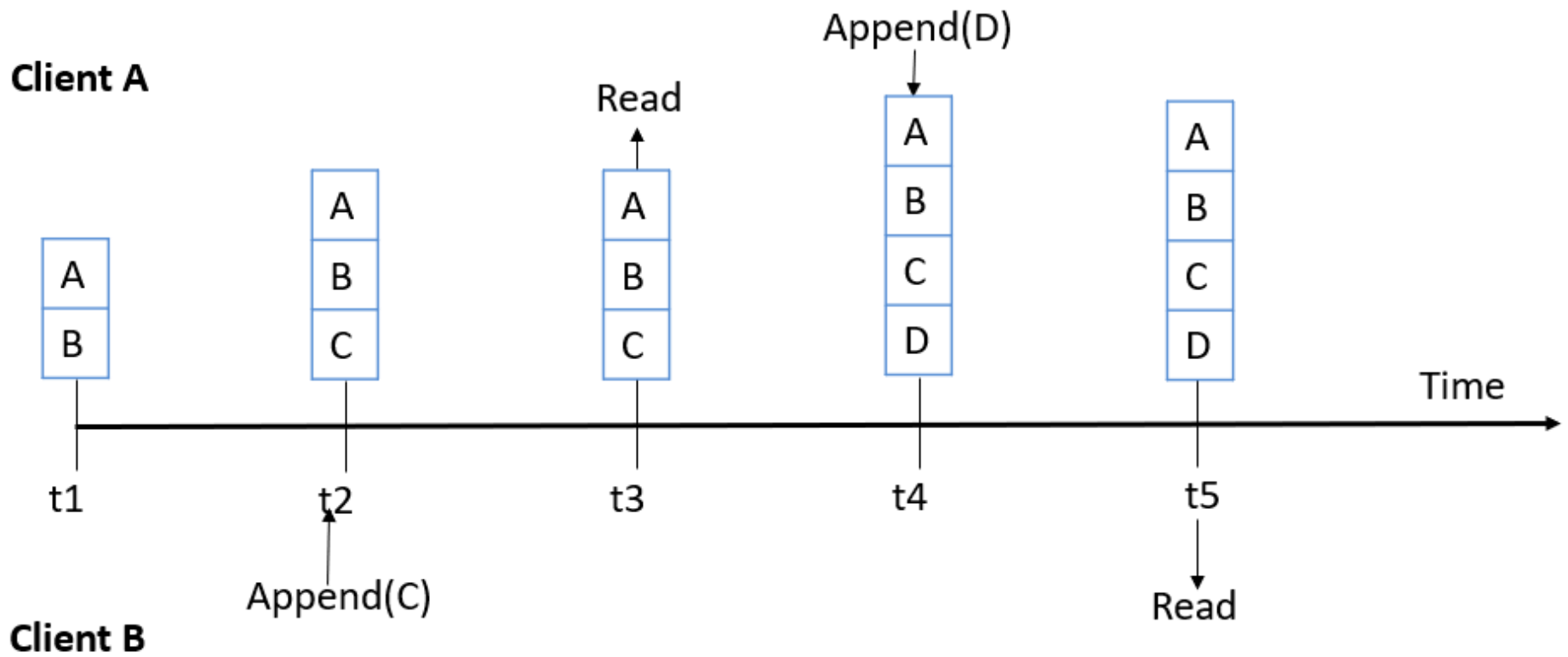
File sharing semantics

- A shared file may be simultaneously accessed by multiple users.
- In such a situation, an important design issue for any file system is to clearly define when modifications of file data made by a user are observable by other users.



File sharing semantics - Unix file semantics

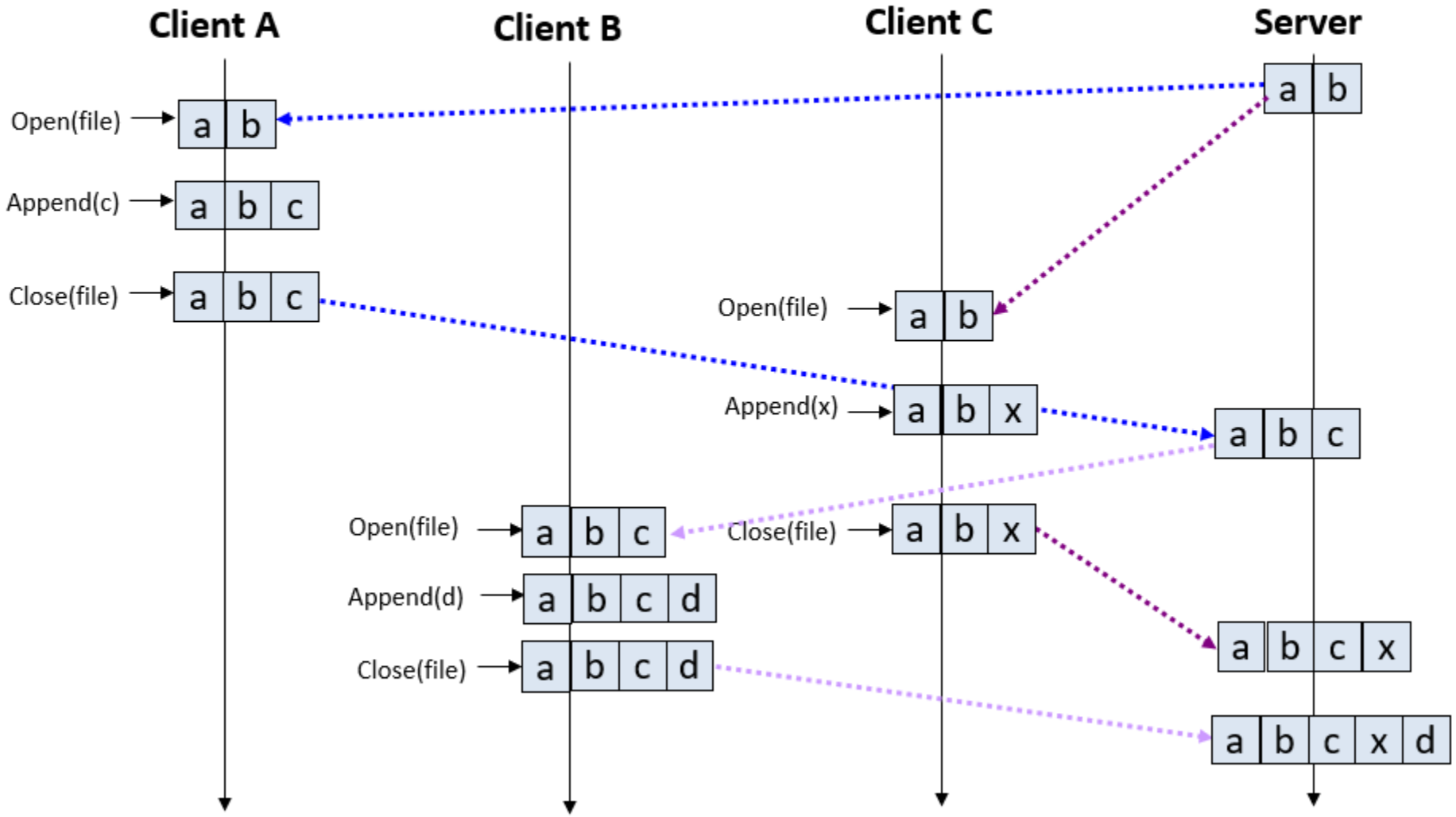
- Every read operation on a file sees the effects of all previous write operations performed on that file.



File sharing semantics- Session semantics

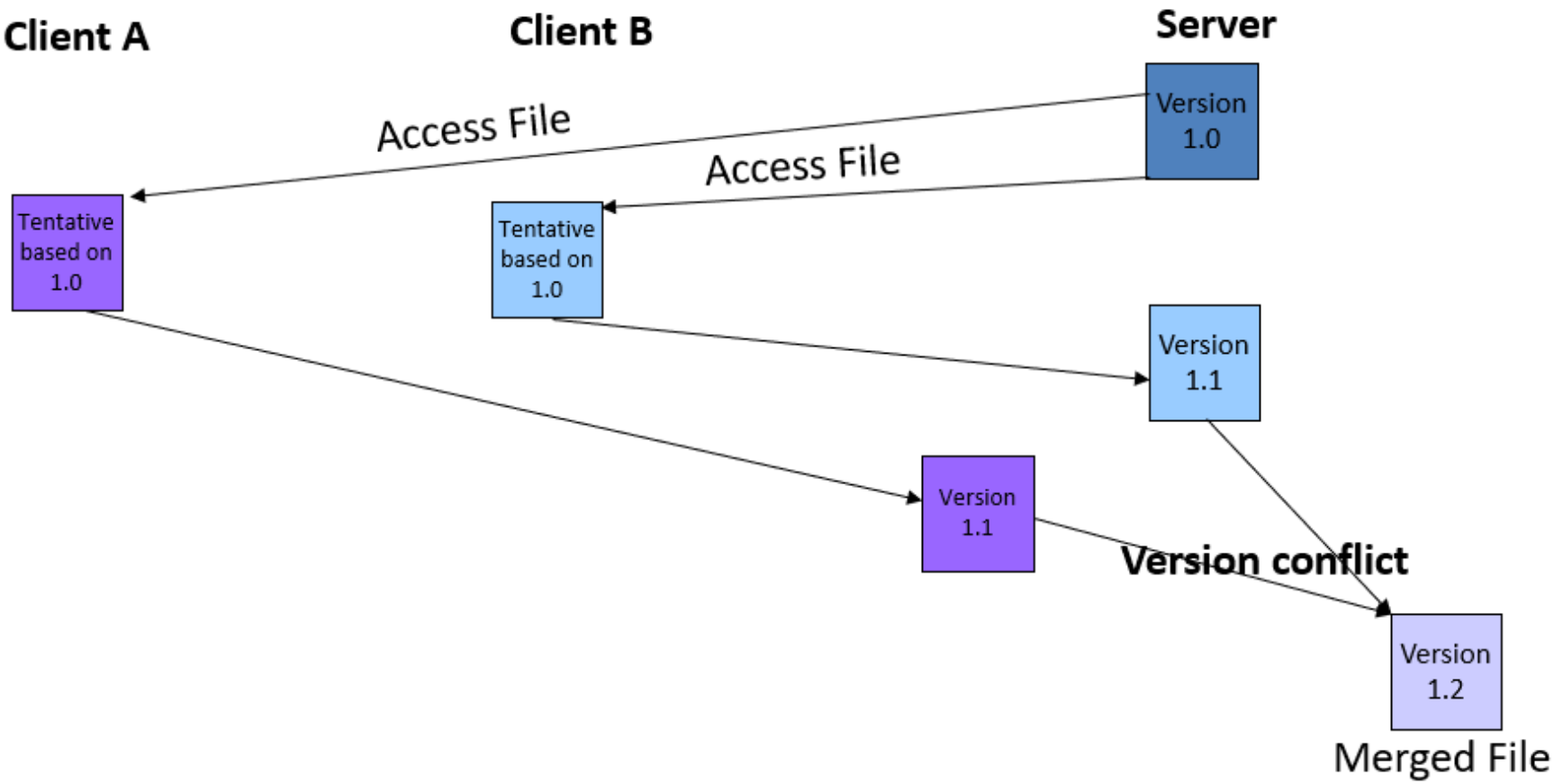
- In session semantics, all changes made to a file during a session are initially made visible only to the client process that opened the session.
- It is invisible to other remote processes who have the same file open simultaneously.
- Once the session is closed, the changes made to the file are made visible to remote processes only in later starting sessions.

File sharing semantics- Session Semantics



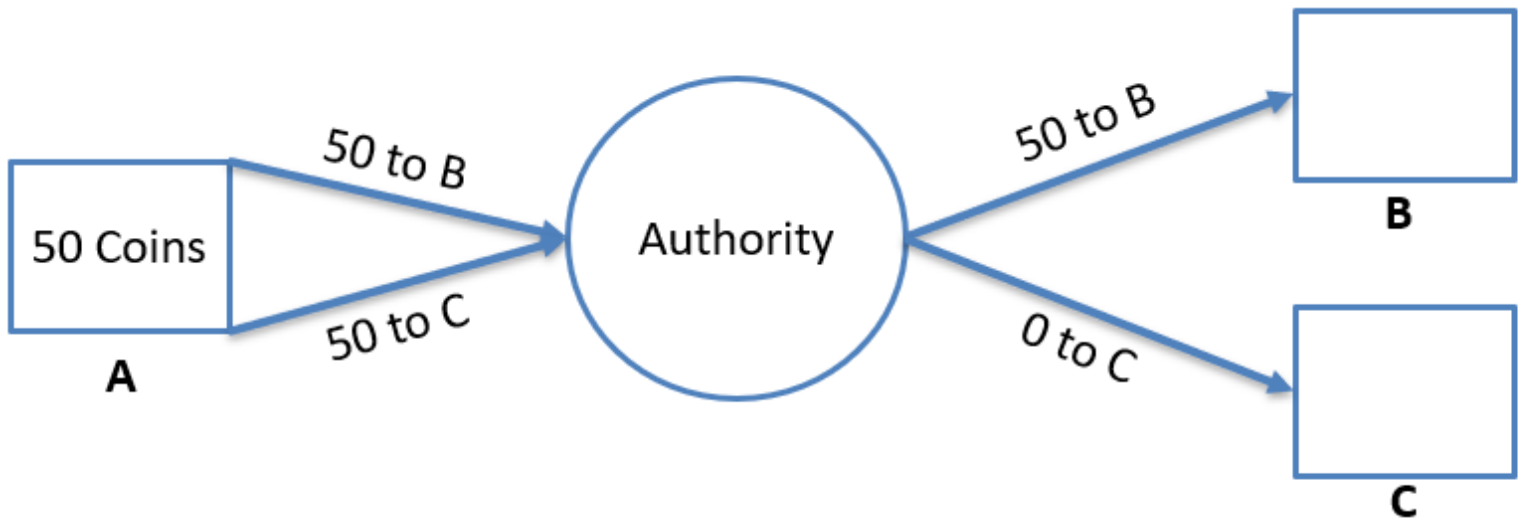
File sharing semantics- Immutable shared-files Semantics

- Change to the file are handled by creating a new updated version of the file.
- Therefore, the semantics allows files to be shared only in the read-only mode.



File sharing semantics- Transaction like Semantics

- A transaction is a set of operations enclosed in-between a pair of **begin_transaction** and **end_transaction** like operations.
- All the transactions will be carried out in order, without any interference from other concurrent transactions.
- Partial modifications will not be visible to other concurrently executing transactions until the transaction ends.
- The example of this is a banking system.



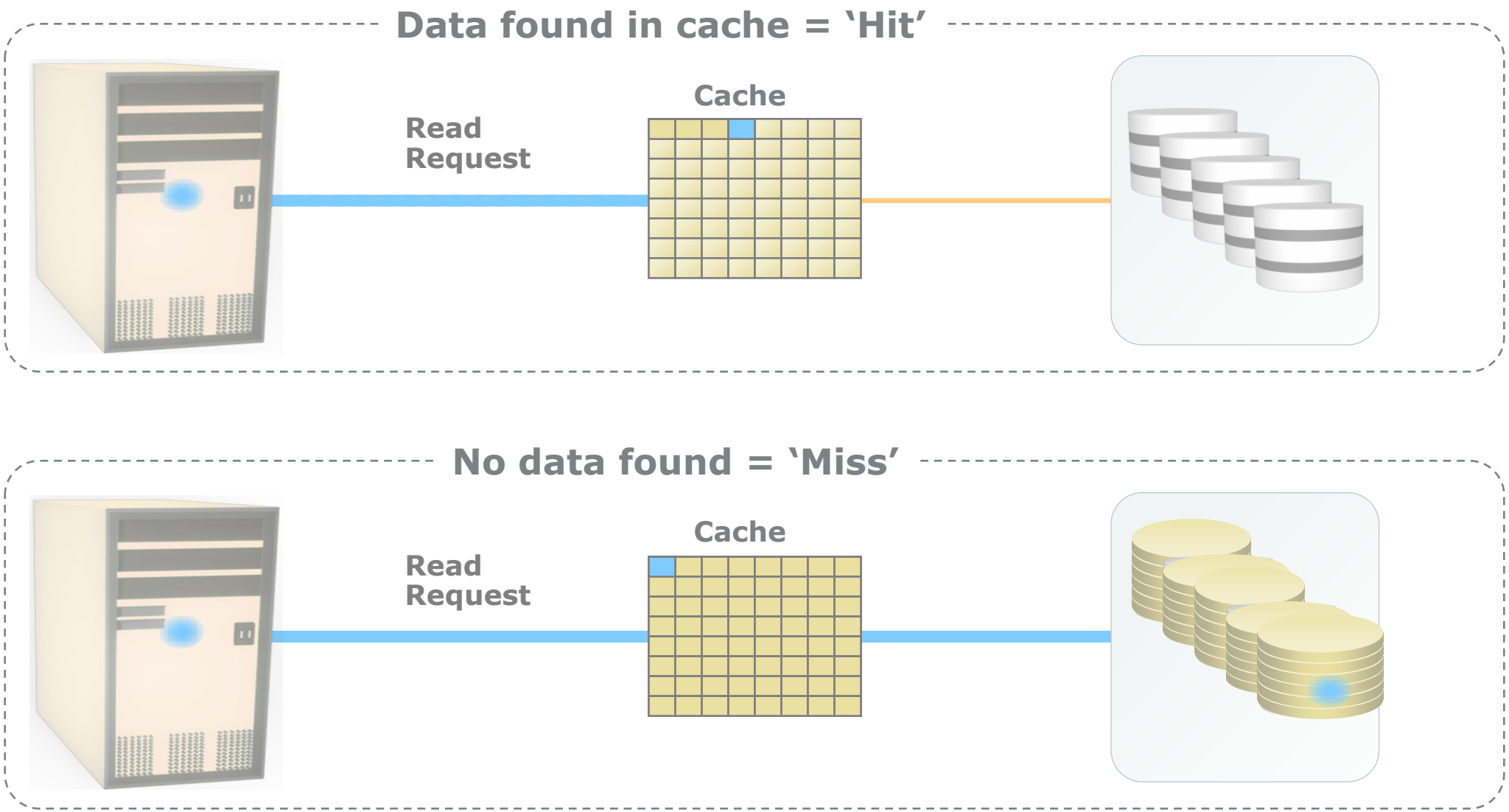
Brief of File sharing semantics

METHOD	FEATURES
Unix semantics	Operations on files are instantaneously visible to all processes.
Session semantics	Changes are not visible to other processes till the file is closed.
Immutable Files	Updates are not allowed. Simplifies sharing and replication.
Transaction-like semantics	Changes have the all-or-nothing property.

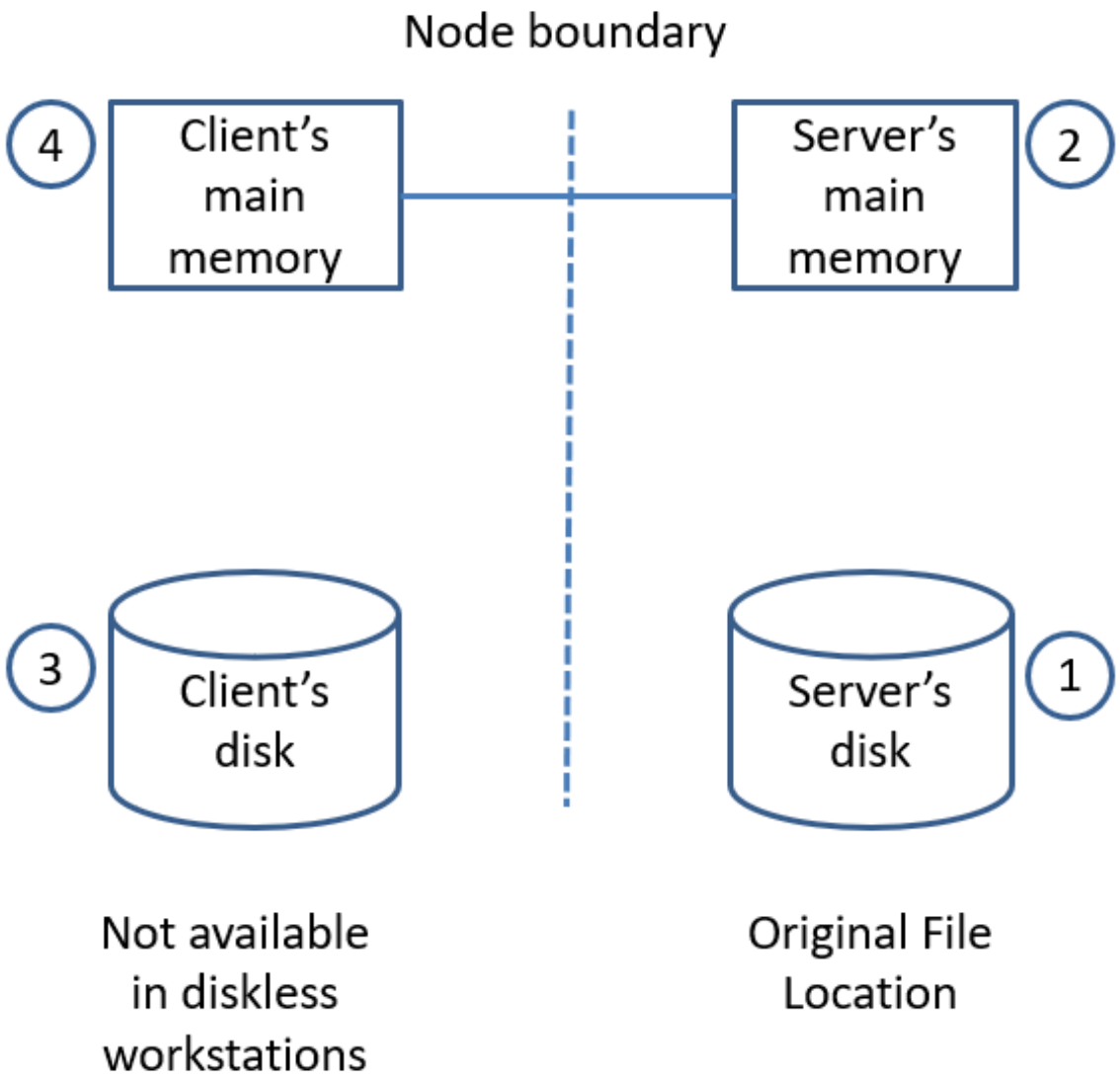
File caching schemes

- The idea in file caching is to retain recently accessed file data in main memory, to avoid repeated accesses.
- File caching reduces disk transfers substantially, resulting in better overall performance of the file system.
- A file-caching scheme for a distributed file system should address the following key decisions:
  1. Cache location
  2. Modification propagation
  3. Cache validation

Read Operation with Cache: ‘Hits’ and ‘Misses’



Possible cache location in File caching scheme



1. No caching
2. Cache located in server's main memory
3. Cache located in client's disk
4. Cache located in client's main memory

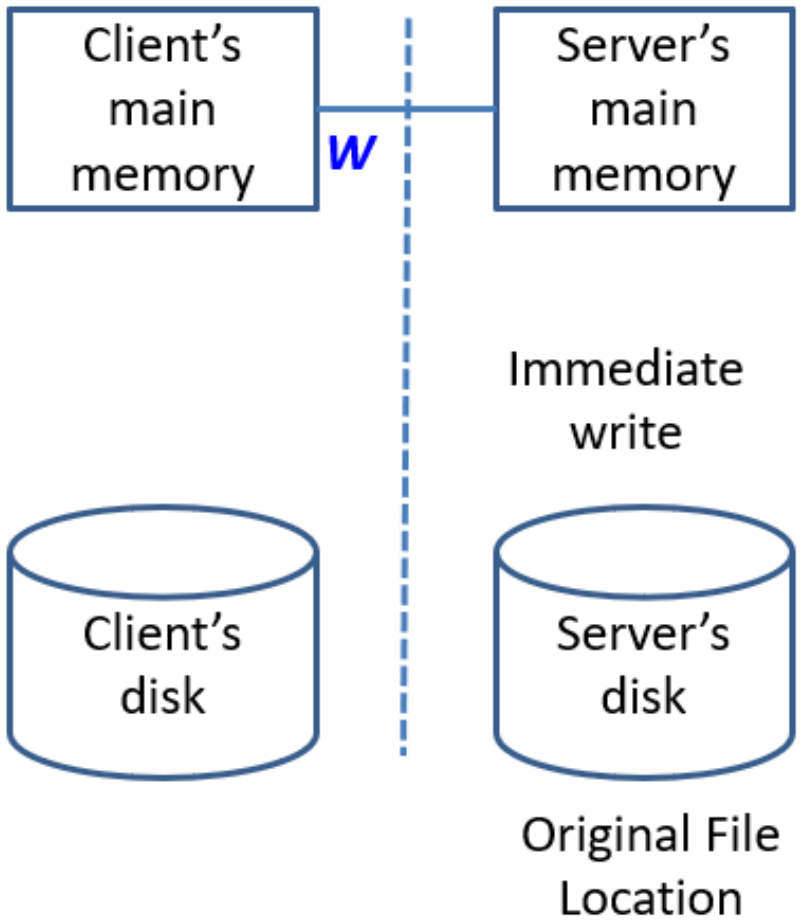
Cache location

CACHE LOCATION	ACCESS COST (CACHE HIT)	MERITS
Server's main memory	One network access	• Easy implementation • Transparent to clients • Consistency maintained easily between cached data and original file • Supports UNIX like file-sharing semantics
Client's disk	One disk access	• Crash reliability • Large storage space availability • Supports operations on files even when disconnected • Scalable
Client's main memory	Nil	• High performance as compared to other methods • Supports diskless workstations • Scalable and reliable

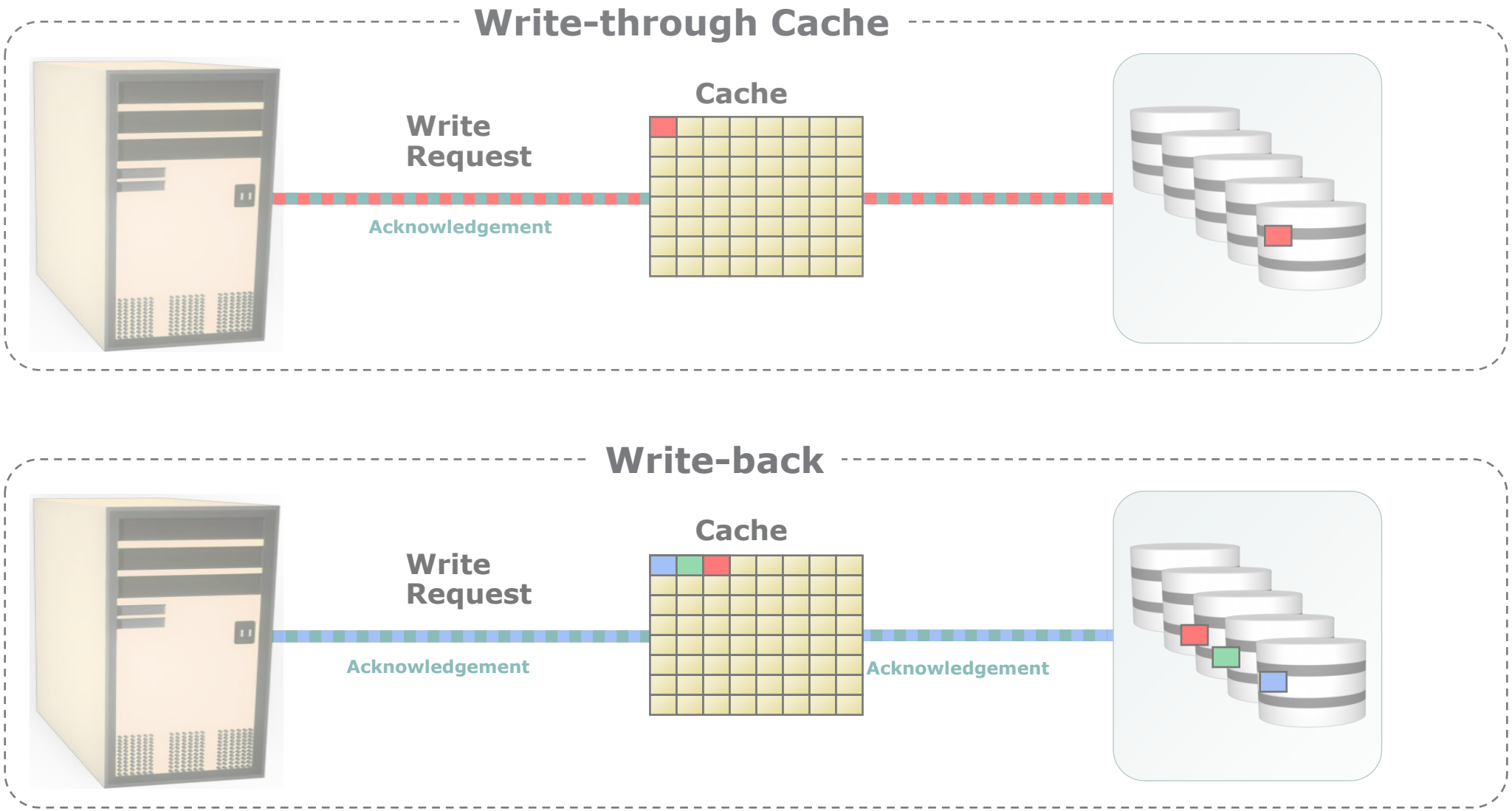
Modification propagation

1. Write-through scheme

- In this scheme, when a cache entry is modified, the new value is immediately sent to the server for updating the master copy of the file.
- **Pros:**
  - High degree of reliability and suitability for UNIX-like semantics
  - Risk of updated data getting lost is very low
- **Cons:**
  - Poor write performance
  - Suitable when read operations are more.



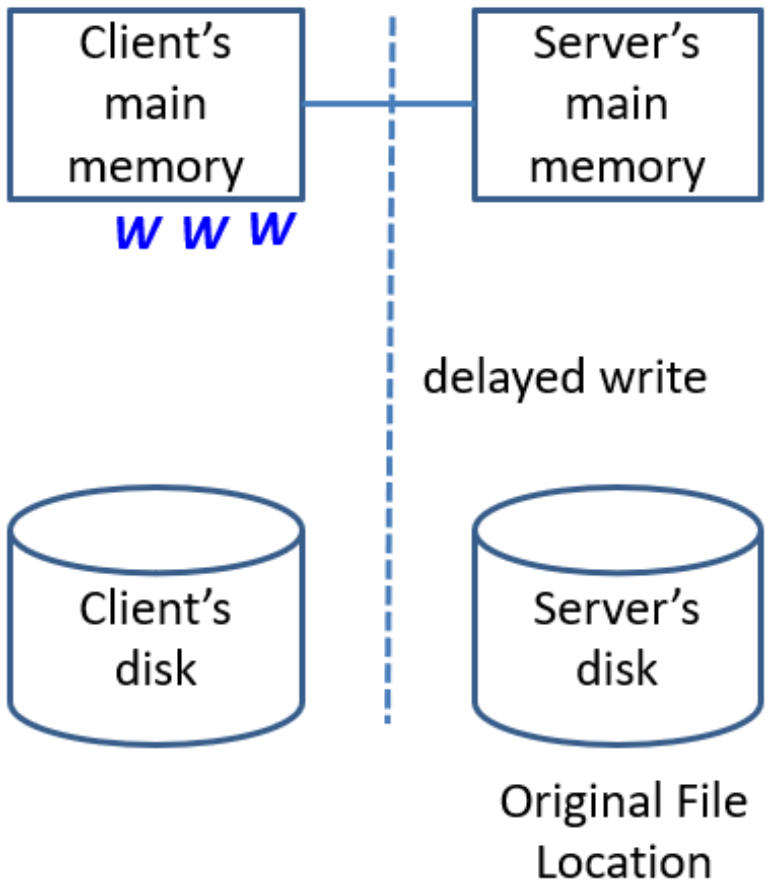
Write Operation with Cache



Modification Propagation

2. Delayed Write

- In this scheme, when a cache entry is modified, client just makes a note that the cache entry has been updated.
- After some time, all updated cache entries corresponding to a file are gathered together and sent to the server at a time.
- **Pros:**
  - Write accesses complete quickly
  - Some writes may be omitted by the following writes.
  - Gathering all writes reduces network overhead.
- **Cons:**
  - Delaying of write propagation results in fuzzier file-sharing semantics.



Cache Validation Schemes

- It becomes necessary to verify if the data cached at a client node is consistent with the master copy.
- When the caches of all nodes contain exactly the same copies of the file data then caches is **consistent**.
- If not, the cached data must be invalidated and the updated version of the data must be fetched again from the server.
- There are two approaches:
  - Client Initiated Approach
  - Server Initiated Approach

Client-Initiated Approach

- Client contacts the server and checks whether its locally cached data is consistent with the master copy or not.
- One of the following approaches may be used:
  - Checking before every access (Unix-like semantics but too slow)
  - Checking periodically (better performance but fuzzy file-sharing semantics)
  - Checking on file open (simple, suitable for session-semantics)
- Problem: High network traffic

Server-Initiated Approach

- Keeping track of clients having a copy
- Keeping a record of which client has which file open and in what mode.
- When a new client makes a request to open an already open file, It denying a new request, queuing it or disabling caching.
- Notifying all clients of any update on the original file.
- Problem:
  - Violating client-server model
  - Stateful servers
  - A check-on-open, client-initiated cache validation approach must still be used along with the server-initiated approach.

File Replication

- File replication is the primary mechanism for improving file availability.
- A replicated file is a file that has multiple copies, with each copy located on a separate file server.
- The main problem in Replication is consistency, i.e. when one copy changes, how do other copies reflect that change?
- Often there is a tradeoff: consistency versus availability and performance.

Replication Transparency

- Multiple copies of a replicated files should be appeared as a single logical file to its users.
- Two important issues related to replication transparency are as follows:
  1. Naming of Replicas
     - A single identifier to all replicas of an object
  2. Replication control
     - Determining number and locations of replicas

Multicopy Update Problem

- As soon as a file system allows multiple copies of the same (logical) file to exist on different servers, it is faced with the problem of keeping them mutually consistent.
- Some of the commonly used approaches to handle this issues are described below:
  1. Read-Only Replication
  2. Read-Any-Write-All Protocol
  3. Available-Copies Protocol
  4. Primary-Copy Protocol
  5. Quorum-Based Protocols

Approaches to handle Multicopy Update Problem

1. Read-Only Replication
   - Allows the replication of only immutable files.
2. Read-Any-Write-All Protocol
   - Read any copy of file and write to all copies of file.
3. Available-Copies Protocol
   - When a server recover after failure, it bring itself up to date by copying from other server before accepting any user request.
4. Primary-Copy Protocol
   - Read operation can be performed from any copy but write operations directly performed only on primary copy.
   - Each server having a secondary copy updates its copy by..
     - Receiving notification of changes from the server having the primary copy.
     - Requesting the updated copy from it.

Approaches to handle Multicopy Update Problem(cntd.)

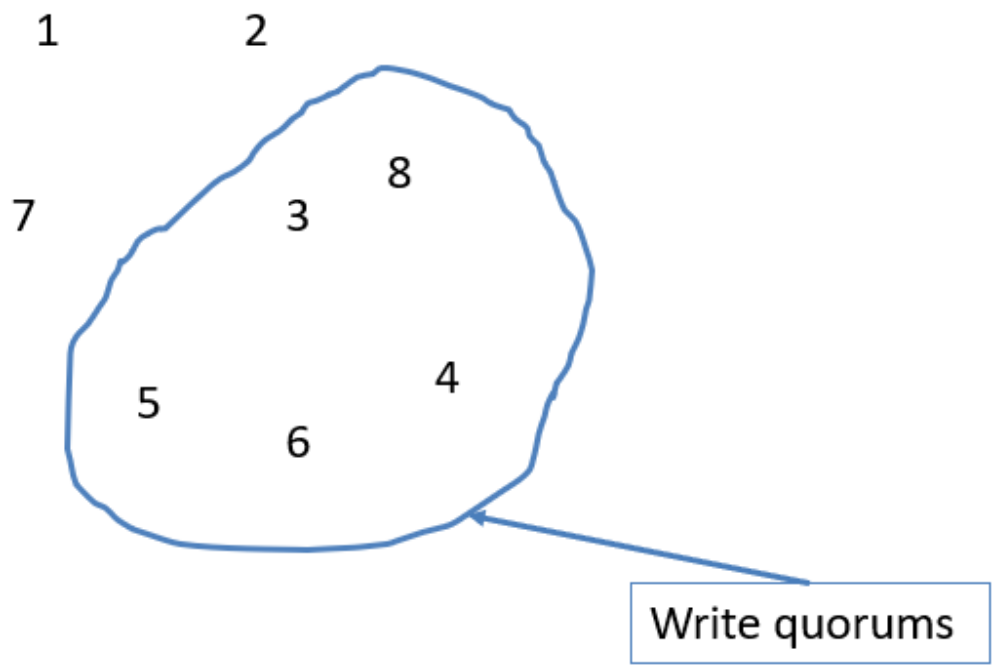
5. Quorum-Based Protocols
   - Suppose that there are a total of n copies of a replicated file F.
   - To read the file, a minimum r copies of F have to be consulted.
   - This set of r copies is called a read quorum.
   - To perform a write operation on the file, a minimum w copies of F have to be written.
   - This set of w copies is called a write quorum.
   - The restriction on the choice of the values of r and w is (r+w>n).
   - That is, there is at least one common copy of the file between every pair of read and write operations resulting in at least one up-to-date copy in any read/write quorum.

Approaches to handle Multicopy Update Problem(cntd.)

- The version number of a copy is updated every time the copy is modified.
- A copy with the largest version number in a quorum is current.
- A read is executed as follows:
  1. Retrieve a read quorum (any r copies) of F.
  2. Of the r copies retrieved, select the copy with the largest version number.
  3. Perform the read operation on the selected copy.
- A write is executed as follows:
  1. Retrieve a write quorum (any w copies) of F.
  2. Of the w copies retrieved, get the version number of the copy with the largest version number.
  3. Increment the version number.
  4. Write the new value and the new version number to all the w copies of the write quorum.

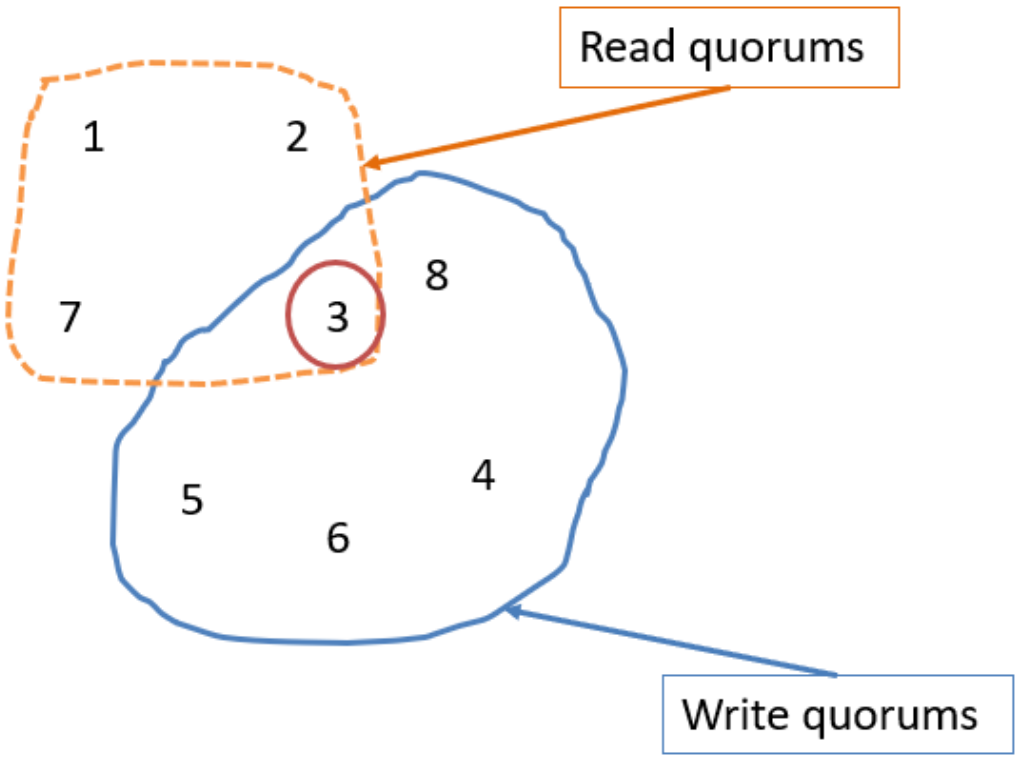
Approaches to handle Multicopy Update Problem(cntd.)

- Examples of quorum consensus algorithm:(n=8, r=4, w=5)
  - There are total of eight copies of the replicated file (n = 8).
  - The values of read and write quorums are 4 and 5, respectively (r =4, w= 5).
  - The condition r +w > n is satisfied.
  - Now suppose a write operation is performed on the write quorum comprised of copies 3, 4, 5, 6, and 8.



Approaches to handle Multicopy Update Problem(cntd.)

- Examples of quorum consensus algorithm:(n=8, r=4, w=5)
  - All these copies get the new version and the new version number.
  - Now any subsequent read operation will require a read quorum of four copies because r=4.
  - any read quorum will have to contain at least one copy of the previous write quorum.



Difference between Replication and Caching

- Click to add text

REPLICATION	CACHING
Replica is associated with a server.	Cached copy is normally associated with a client.
Existence of a replica depends on availability and performance requirement.	Existence of a cached copy is dependent on the locality in file access patterns.
More persistent.	Less persistent.
Widely known, secure, available, complete and accurate.	Less known, secure, available, complete, and accurate.
A replica does not change upon cached.	A cached copy is subject to change upon a replica.

Advantages of Replication

1. Increased availability
   - The system remains operational and available to the users despite failures.
2. Increased reliability
   - Replication allows the existence of multiple copies of their files.
3. Improved response time
   - It enables data to be accessed either locally or from a node to which access time is lower than the primary copy access time.
4. Reduced network traffic
   - If a file's replica is available with a file server on a client's node, the client's access requests can be serviced locally, resulting in reduced network traffic.

Advantages of Replication(cntd.)

5. Improved system throughput
   - Replication also enables several client's requests for access to the same file to be serviced in parallel by different servers, resulting in improved system throughput.
6. Better scalability
   - By replicating the file on multiple servers, the same requests can now be serviced more efficiently by multiple servers due to workload distribution.
7. Autonomous operation
   - In a distributed system that provides file replication as a service to their clients.
   - All files required by a client for operating during a limited time period may be replicated on the file server residing at the client's node.
   - This will facilitate temporary autonomous operation of client machines.

Fault Tolerance in DFS

- The primary file properties that directly influence the ability of a distributed file system to tolerate faults are as follow:
- Availability
  - Availability of a file refers to the fraction of time for which the file is available for use during communication link failure.
- Robustness
  - Robustness of a file refers to its power to survive crashes of the storage device and decays of the storage medium on which it is stored.
- Recoverability
  - Recoverability of file refers to its ability to be rolled back to an earlier state when an operation on the file fails or is aborted by the client.

Stable Storage

- In context of crash resistance capability, storage may be broadly classified into three types:

1. Volatile Storage (RAM)
   - It cannot withstand power failure or machine crash.
   - Data is lost in case of power failure or machine crash.
2. Nonvolatile storage (Disk)
   - It can withstand CPU failures but cannot withstand transient I/O faults and decay of the storage media.
3. Stable Storage
   - It can withstand transient I/O fault and decay of storage media.

Effect of Service Paradigm on Fault Tolerance

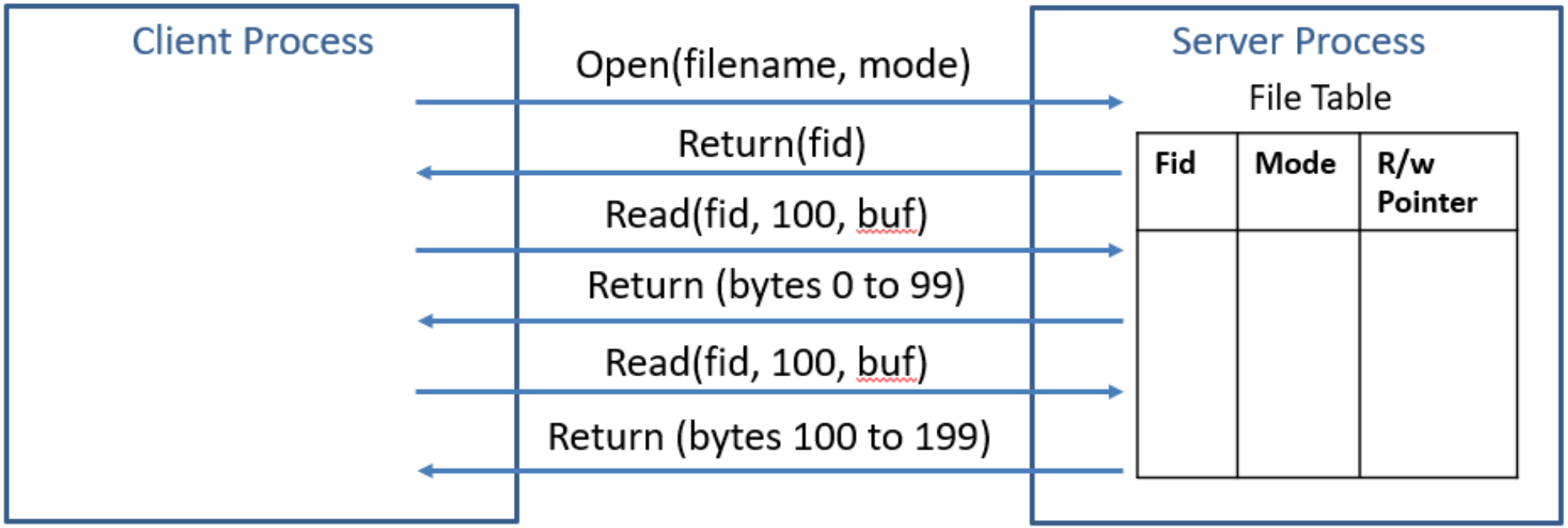
- A server may be implemented by using any one of the following two service paradigms **stateful** and **stateless**.
- The two paradigms are distinguished by one aspect of the client server relationship i.e.
  - Whether or not the history of the serviced requests between a client and a server affects the execution of the next service request.
- The stateful approach depends on the history of the serviced requests, but the stateless approach does not depend on it.

Effect of Service Paradigm on Fault Tolerance

1. Stateful File Servers

- A stateful server maintains client's state information from one remote procedure call to the next.
- These clients state information is subsequently used at the time of executing the second call.
- To illustrate how a stateful file server works, let us consider a file server for byte-stream files that allows the following operations on files:
  - Open(filename,mode)
  - Read(fid,n,buffer)
  - Write(fid,n,buffer)
  - Seek(fid,position)
  - Close(fid)

Operations on Files in Stateful File Server

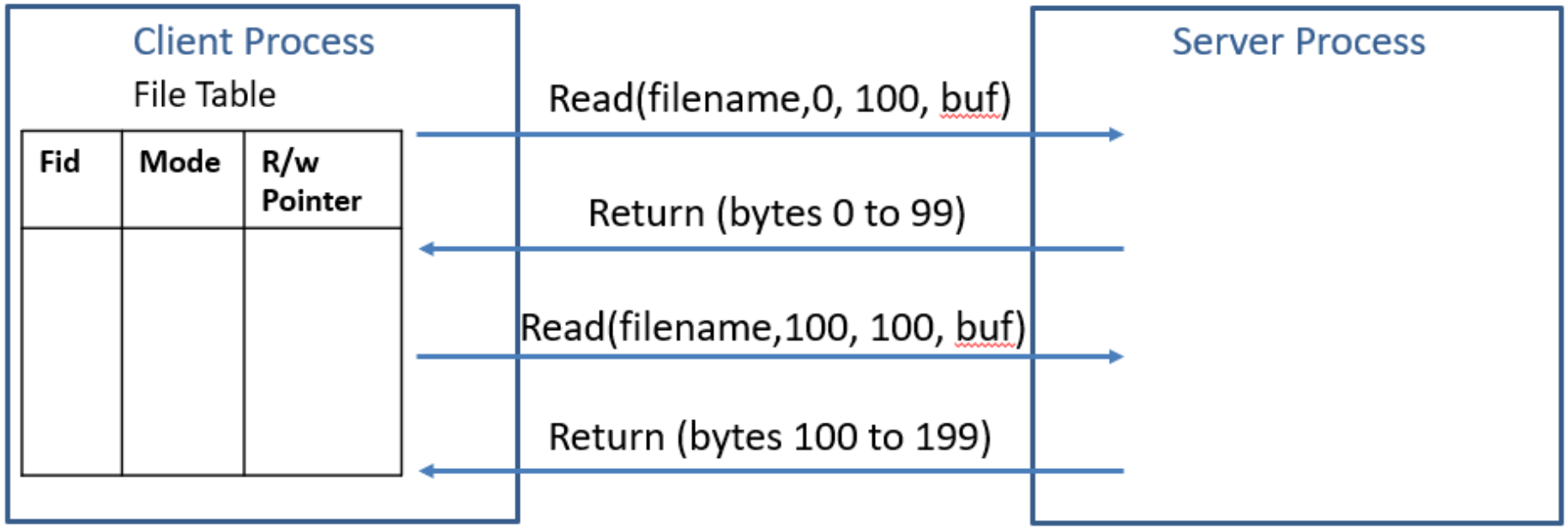


- After opening a file, if a client makes two subsequent Read (fid, 100, buf) requests, for the first request the first 100 bytes (bytes 0 to 99) will be read and for the second request the next 100 bytes (bytes 100 to 199) will be read.

Stateless File Server

- A stateless file server does not maintain any client state information.
- Therefore every request from a client must be accompanied with all the necessary parameters to successfully carry out the desired operation.
- Each request identifies the file and the position in the file for the read/write access.
- Operations on files in Stateless File server:
  - Read(filename,position,n,buffer): On execution, the server returns n bytes of data of the file identified by filename.
  - Write(filename,position,n,buffer): On execution, it takes n bytes of data from the specified buffer and writes it into the file identified by filename.

Operations on Files in Stateless File Server



- This file server does not keep track of any file state information resulting from a previous operation.
- Therefore, if a client wishes to have similar effect as previous figure, the following two read operations must be carried out:
  - Read(filename, 0, 100, buf)
  - Read(filename, 100, 100, buf)

Difference between Stateful & Stateless

PARAMETERS	STATEFUL	STATELESS
State	A Stateful server remember client data (state) from one request to the next.	A Stateless server does not remember state information.
Programming	Stateful server is harder to code.	Stateless server is straightforward to code.
Efficiency	More because clients do not have to provide full file information every time they perform an operation.	Less because information needs to be provided.
Crash recovery	Difficult due to loss of information.	Can easily recover from failure because there is no state that must be restored.
Information transfer	The client can send less data with each request.	The client must specify complete file names in each request.
Operations	Open, Read, Write, Seek, Close	Read, Write

Trends in Distributed File System

- Some of the changes that can be expected in the future and some of the implications these changes may have for file systems are:
  1. **New Hardware:** Within few years memory may become so cheap that the file system may permanently reside in memory, and no disks will be needed.
  2. **Scalability:** Algorithms that work well for systems with 100 machines should also work same with larger systems.
  3. **Wide Area Networking:** Bringing fiber optics into everywhere will increase communication speed.
  4. **Mobile Users:** Large caching will be used for mobile users.
  5. **Multimedia:** To handle applications such as video-on-demand, completely different file systems will be needed.

~END