Advanced Operating System

Unit – III

Dr.M.Lalli

Dept of CS, Trichy

Unit - III

Distributed File systems – Architecture – Mechanisms – Design Issues –

Distributed Shared Memory – Architecture – Algorithm – Protocols - Design Issues.

Distributed Scheduling – Issues – Components – Algorithms.

DISTRIBUTED FILE SYSTEMS

- A **Distributed File System** (DFS) is simply a classical model of a file system (as discussed before) distributed across multiple machines. The purpose is to promote sharing of dispersed files.
- This is an area of active research interest today.
- The resources on a particular machine are **local** to itself. Resources on other machines are **remote**.
- A file system provides a service for clients. The server interface is the normal set of file operations: create, read, etc. on files.

Definition of a DFS

- DFS: multiple users, multiple sites, and (possibly) distributed storage of files.
- Benefits
  - File sharing
  - Uniform view of system from different clients
  - Centralized administration
- Goals of a distributed file system
  - Network Transparency (access transparency)
  - Availability

Goals

- Network (Access)Transparency
  - Users should be able to access files over a network as easily as if the files were stored locally.
  - Users should not have to know the physical location of a file to access it.
- Transparency can be addressed through naming and file mounting mechanisms

Components of Access Transparency

- Location Transparency: file name doesn't specify physical location
- Location Independence: files can be moved to new physical location, no need to change references to them.
- Location independence $\rightarrow$ location transparency, but the reverse is not necessarily true.

Goals

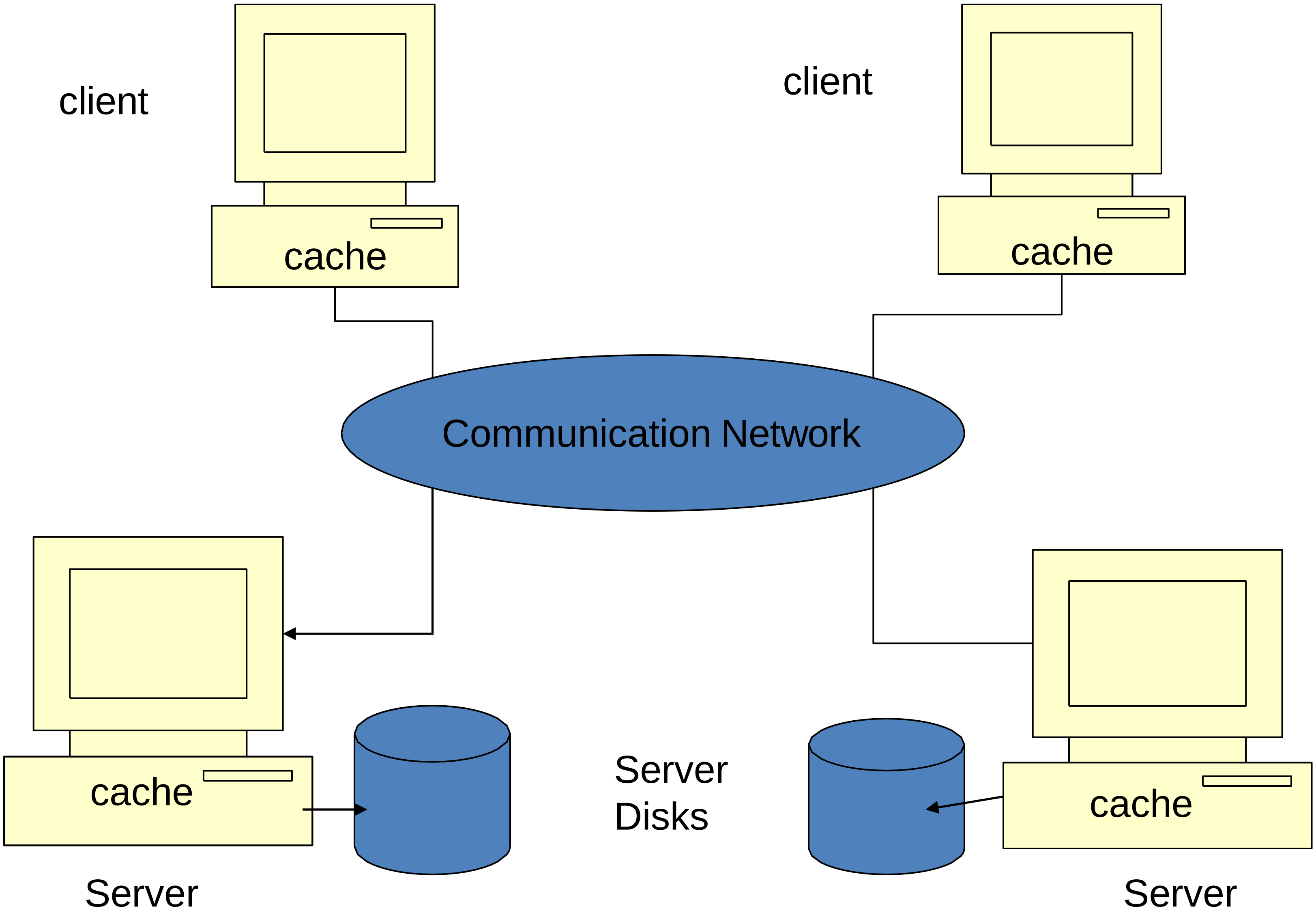
- <u>Availability</u>: files should be easily and quickly accessible.
- The number of users, system failures, or other consequences of distribution shouldn’t compromise the availability.
- Addressed mainly through replication.

Architectures

- Client-Server
  - Traditional; e.g. Sun Microsystem Network File System (NFS)
  - Cluster-Based Client-Server; e.g., Google File System (GFS)
- Symmetric
  - Fully decentralized; based on peer-to-peer technology
  - e.g., Ivy (uses a Chord DHT approach)

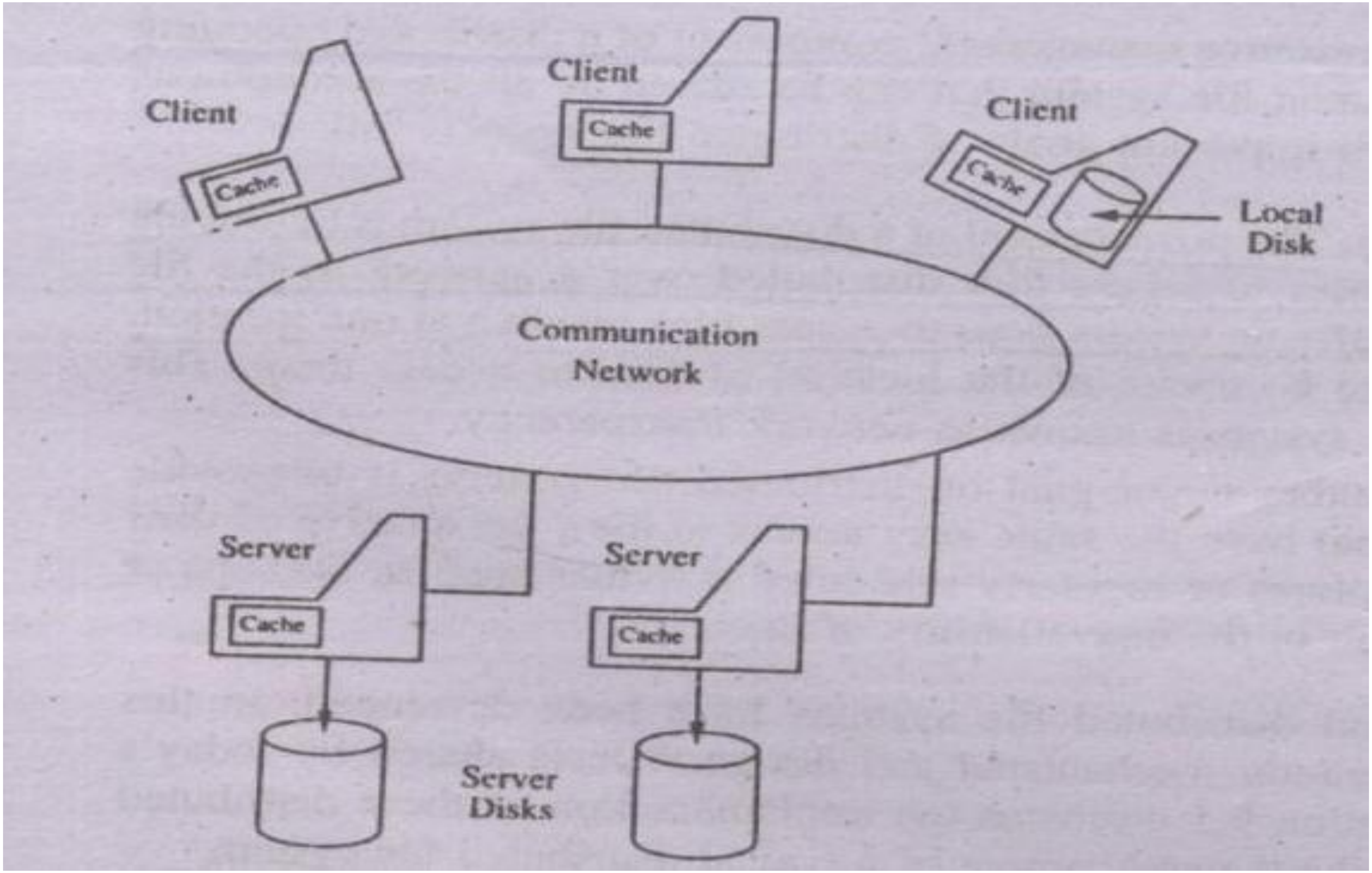
Client-Server Architecture

- One or more machines (file servers) manage the file system.
- Files are stored on disks at the servers
- Requests for file operations are made from clients to the servers.
- Client-server systems centralize storage and management; P2P systems decentralize it.

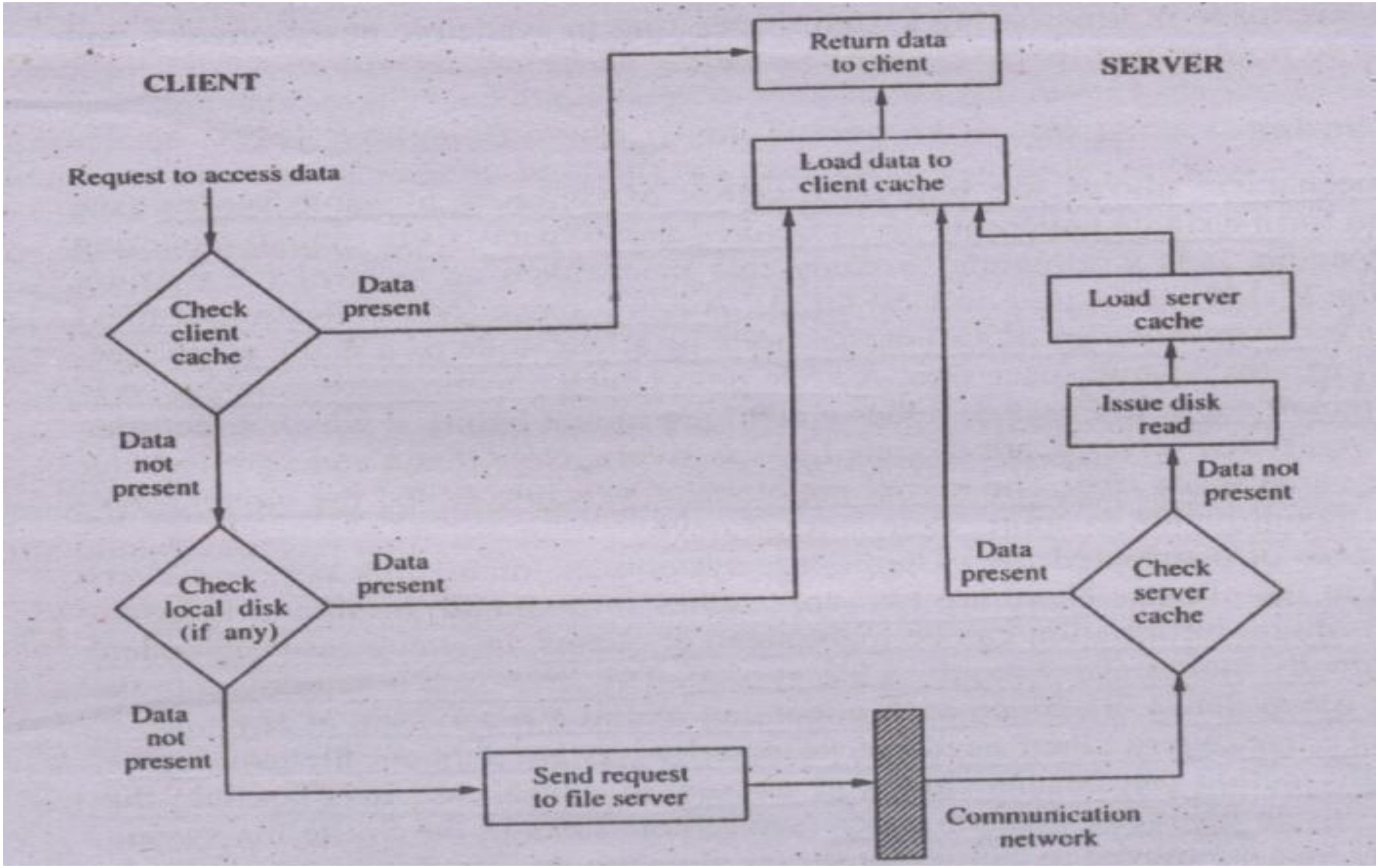


Architecture of a distributed file system: client-server model

Architecture of DFS



Data Access Actions in DFS



Mechanisms for Building DFS

- Mounting
  - Allows the binding together of different filename spaces to form a single hierarchically structured name space
  - Kernel maintains a structure called the mount table which maps mount points to appropriate storage devices.
- Caching
  - To reduce delays in the accessing of data by exploiting the temporal locality of reference exhibited by program
- Hints
  - An alternative to cached data to overcome inconsistency problem when multiple clients access shared data
- Bulk Data Transfer
  - To overcome the high cost of executing communication protocols, i.e. assembly/disassembly of packets, copying of buffers between layers
- Encryption
  - To enforce security in distributed systems with a scenario that two entities wishing to communicate establish a key for conversation

Design Goals

- Naming and Name Resolution
- Caches on Disk or Main Memory
- Writing Policy
- Cache Consistency
- Availability
- Scalability
- Semantics

Naming and Name Resolution

- Name in file systems is associated with an object (e.g. a file or a directory)
- Name resolution refers to the process of mapping a name to an object, or in case of replication, to multiple objects.
- Name space is a collection of names which may or may not share an identical resolution mechanism
- Three approaches to name files in DE
  - Concatenation
  - Mounting (Sun NFS)
  - Directory structured (Sprite and Apollo)
- The Concepts of Contexts
  - A context identifies the name space in which to resolve a given name
  - Examples: x-Kernel Logical File System, Tilde Naming Scheme
- Name Server
  - Resolves the names in distributed systems. Drawbacks involved such as single point of failure, performance bottleneck. Alternate is to have several name servers, e.g. Domain Name Servers

Caches on Disk or Main Memory

- Cache in Main Memory
  - Diskless workstations can also take advantage of caching
  - Accessing a cache is much faster than access a cache on local disk
  - The server-cache is in the main memory, and hence a single cache design for both
  - Disadvantages
    - It competes with the virtual memory system for physical memory space
    - A more complex cache manager and memory management system
    - Large files cannot be cached completely in memory
- Cache in Local Disk
  - Large files can be cached without affecting performance
  - Virtual memory management is simple
  - Example: Coda File System

Writing Policy

- Decision to when the modified cache block at a client should be transferred to the server
- Write-through policy
  - All writes requested by the applications at clients are also carried out at the server immediately.
- Delayed writing policy
  - Modifications due to a write are reflected at the server after some delay.
- Write on close policy
  - The updating of the files at the server is not done until the file is closed

Cache Consistency

- Two approaches to guarantee that the data returned to the client is valid.
  - Server-initiated approach
    - Server inform cache managers whenever the data in the client caches become stale
    - Cache managers at clients can then retrieve the new data or invalidate the blocks containing the old data
  - Client-initiated approach
    - The responsibility of the cache managers at the clients to validate data with the server before returning it
- Both are expensive since communication cost is high
  - Concurrent-write sharing approach
    - A file is open at multiple clients and at least one has it open for writing.
    - When this occurs for a file, the file server informs all the clients to purge their cached data items belonging to that file.
  - Sequential-write sharing issues causing cache inconsistency
    - Client opens a file, it may have outdated blocks in its cache
    - Client opens a file, the current data block may still be in another client's cache waiting to be flushed. (e.g. happens in Delayed writing policy)

Availability

- Immunity to the failure of server of the communication network
- Replication is used for enhancing the availability of files at different servers
- It is expensive because
  - Extra storage space required
  - The overhead incurred in maintaining all the replicas up to date
- Issues involve
  - How to keep the replicas of a file consistent
  - How to detect inconsistencies among replicas of a file and recover from these inconsistencies
- Causes of Inconsistency
  - A replica is not updated due to failure of server
  - All the file servers are not reachable from all the clients due to network partition
  - The replicas of a file in different partitions are updated differently

Availability (contd.)

- Unit of Replication
  - The most basic unit is a file
  - A group of files of a single user or the files that are in a server (the group file is referred to as volume, e.g. Coda)
  - Combination of two techniques, as in Locus
- Replica Management
  - The maintenance of replicas and in making use of them to provide increased availability
    - Concerns with the consistency among replicas
      - A weighted voting scheme (e.g. Roe File System)
      - Designated agents scheme (e.g. Locus)
      - Backups servers scheme (e.g. Harp File System)

Scalability

- The suitability of the design of a system to cater to the demands of a growing system
- As the system grow larger, both the size of the server state and the load due to invalidations increase
- The structure of the server process also plays a major role in deciding how many clients a server can support
  - If the server is designed with a single process, then many clients have to wait for a long time whenever a disk I/O is initiated
  - These waits can be avoided if a separate process is assigned to each client
  - A significant overhead due to the frequent context switches to handle requests from different clients can slow down the server
  - An alternate is to use Lightweight processes (threads)

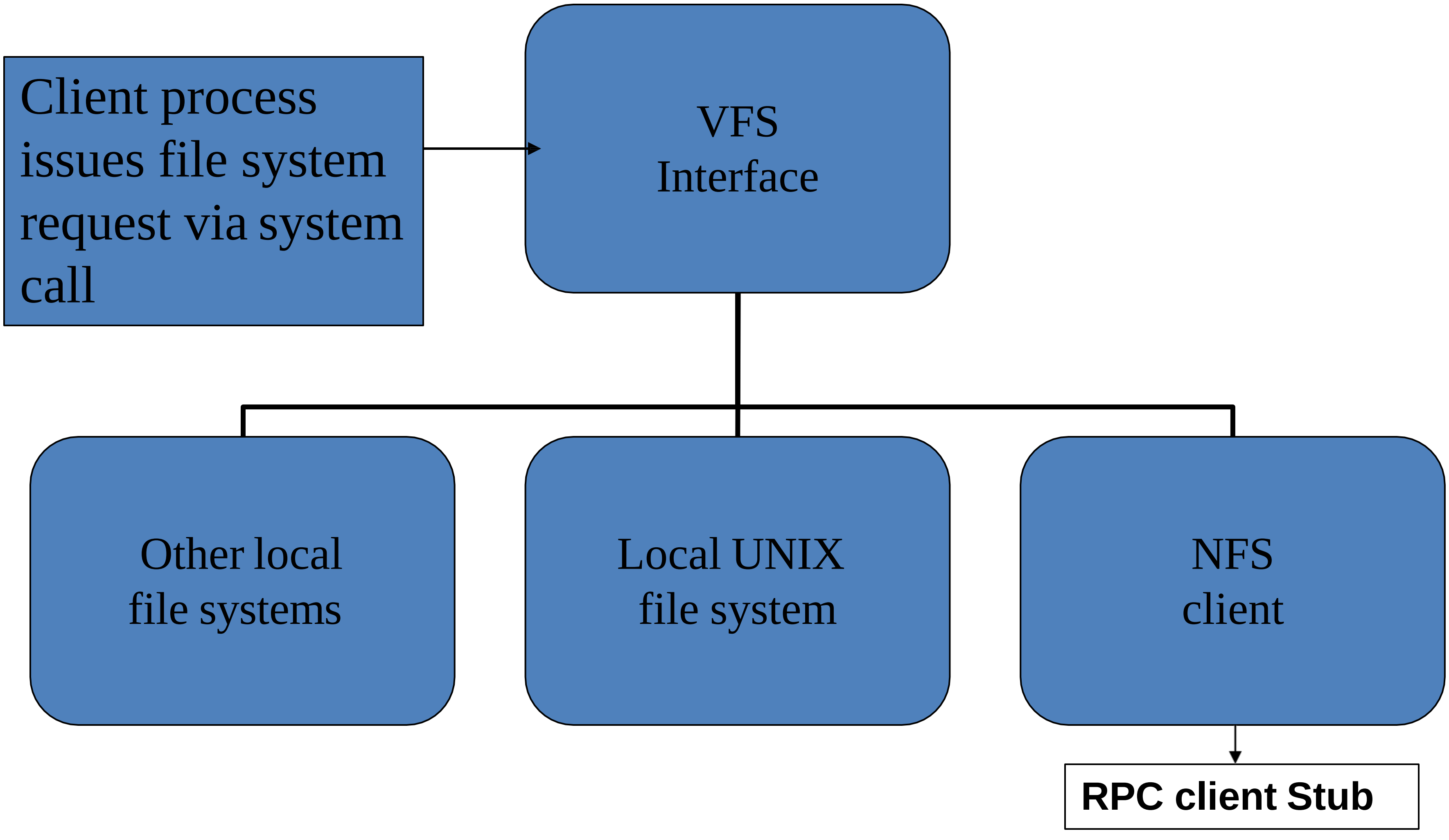
Semantics

- The semantics of a file system characterizes the effects of accesses on files
- Guaranteeing the semantics in distributed file systems, which employ caching, is difficult and expensive
  - In server-initiated cache the invalidation may not occur immediately after updates and before reads occur at clients.
  - This is due to communication delays
- To guarantee the above semantics all the reads and writes from various clients will have to go through the server
- Or sharing will have to be disallowed either by the server, or by the use of locks by applications

NFS- System Architecture

- Virtual File System (VFS) acts as an interface between the operating system's system call layer and all file systems on a node.
- The user interface to NFS is the same as the interface to local file systems. The calls go to the VFS layer, which passes them either to a local file system or to the NFS client
- VFS is used today on virtually all operating systems as the interface to different local and distributed file systems.

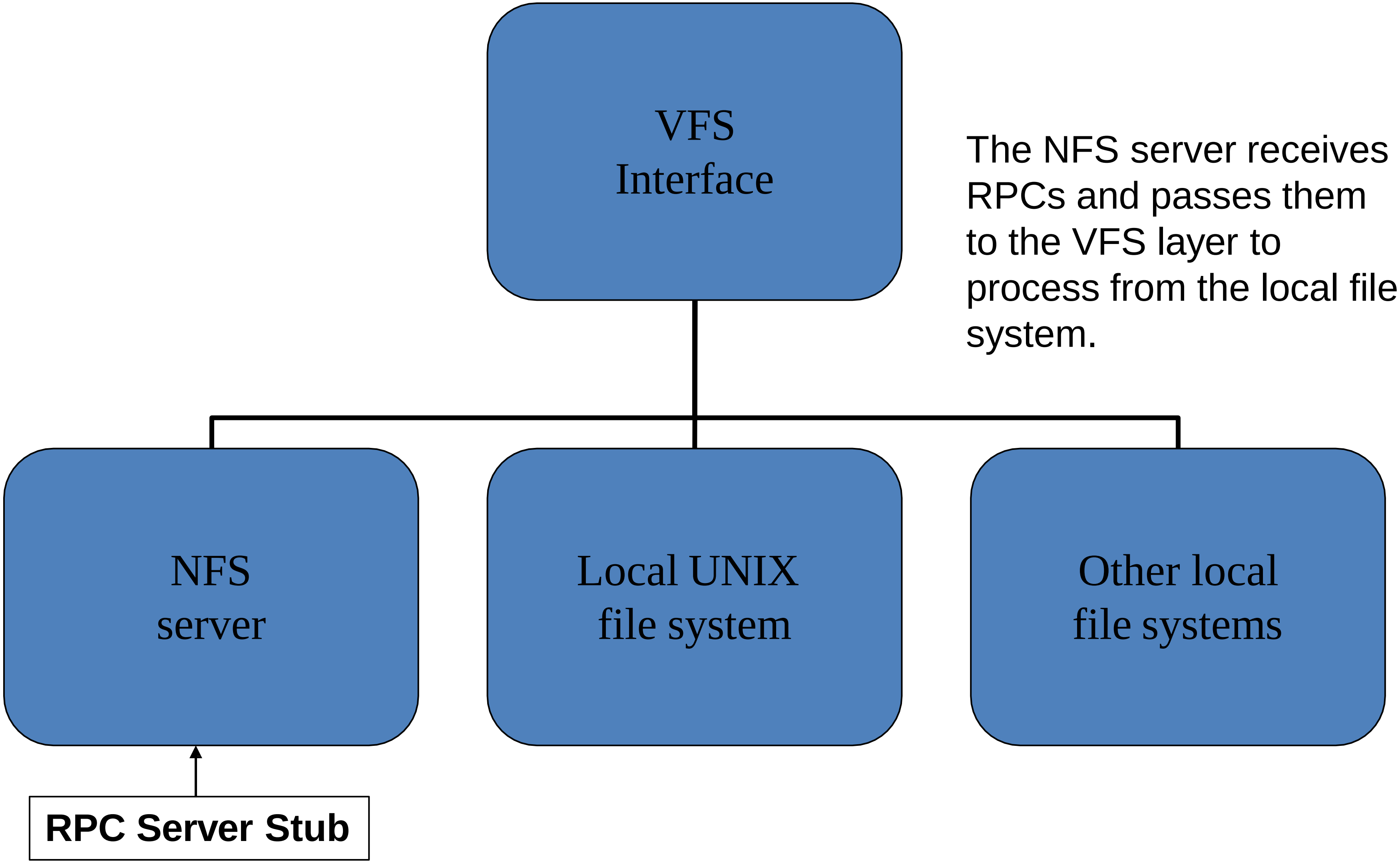
Client-Side Interface to NFS



NFS Client/Server Communication

- The NFS client communicates with the server using RPCs
  - File system operations are implemented as remote procedure calls
- At the server: an RPC server stub receives the request, "un-marshalls" the parameters & passes them to the NFS server, which creates a request to the server's VFS layer.
- The VFS layer performs the operation on the local file system and the results are passed back to the client.

Server-Side Interface to NFS



The NFS server receives RPCs and passes them to the VFS layer to process from the local file system.

NFS as a Stateless Server

- NFS servers historically did not retain any information about past requests.
- Consequence: crashes weren't too painful
  - If server crashed, it had no tables to rebuild – just reboot and go
- Disadvantage: client has to maintain all state information; messages are longer than they would be otherwise.
- NFSv4 is state*ful*

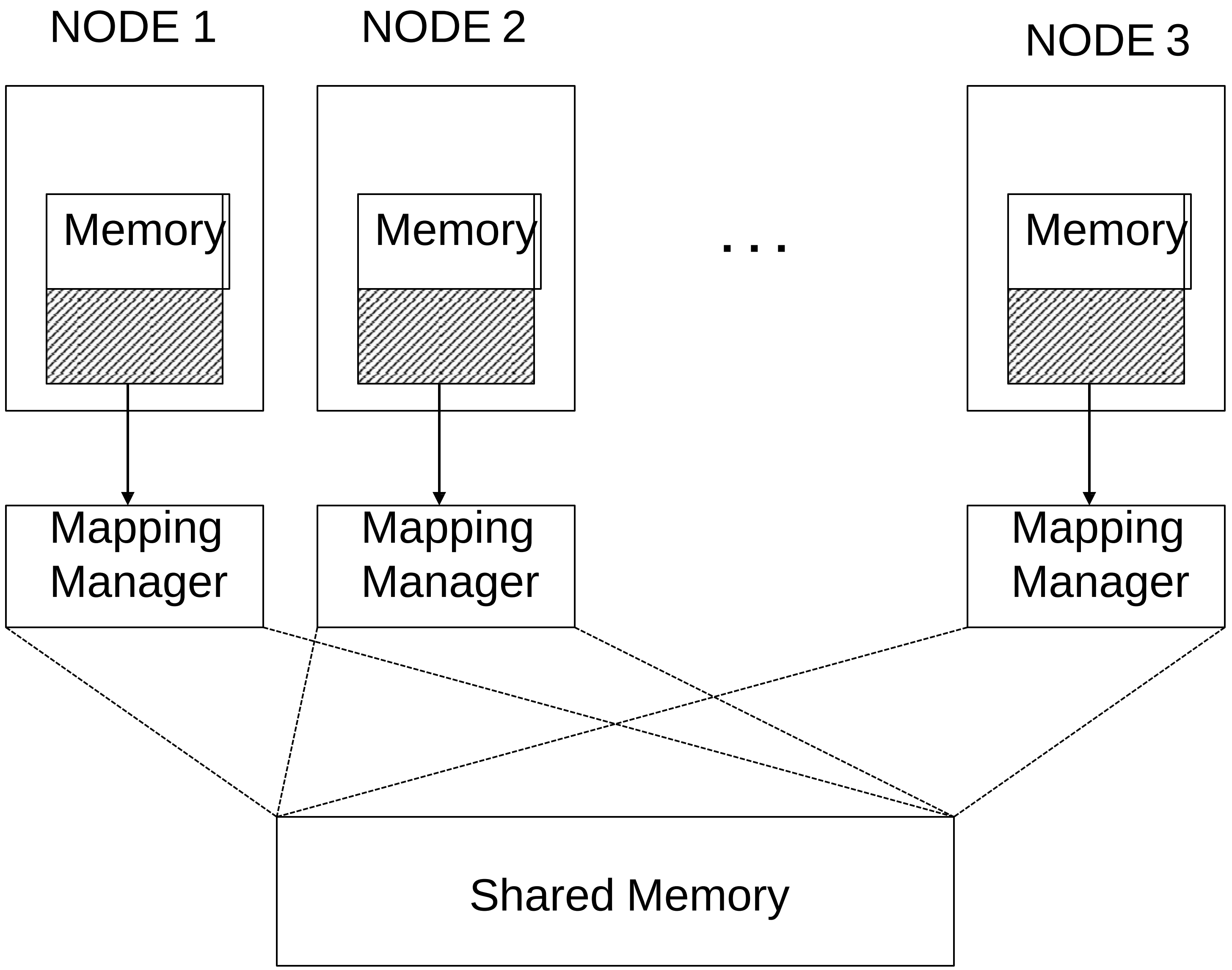
Advantages/Disadvantages

- Stateless Servers
  - Fault tolerant
  - No open/close RPC required
  - No need for server to waste time or space maintaining tables of state information
  - Quick recovery from server crashes
- Stateful Servers
  - Messages to server are shorter (no need to transmit state information)
  - Supports file locking
  - Supports idempotency (don't repeat actions if they have been done)

Distributed shared memory (DSM)

- What
  - The distributed shared memory (DSM) implements the shared memory model in distributed systems, which have no physical shared memory
  - The shared memory model provides a virtual address space shared between all nodes
  - The overcome the high cost of communication in distributed systems, DSM systems move data to the location of access
- How:
  - Data moves between main memory and secondary memory (within a node) and between main memories of different nodes
  - Each data object is owned by a node
    - Initial owner is the node that created object
    - Ownership can change as object moves from node to node
  - When a process accesses data in the shared address space, the mapping manager maps shared memory address to physical memory (local or remote)

Distributed shared memory (Cont.)



Advantages of distributed shared memory (DSM)

- Data sharing is implicit, hiding data movement (as opposed to ‘Send’/‘Receive’ in message passing model)
- Passing data structures containing pointers is easier (in message passing model data moves between different address spaces)
- Moving entire object to user takes advantage of locality difference
- Less expensive to build than tightly coupled multiprocessor system: off-the-shelf hardware, no expensive interface to shared physical memory
- Very large total physical memory for all nodes: Large programs can run more efficiently
- No serial access to common bus for shared physical memory like in multiprocessor systems
- Programs written for shared memory multiprocessors can be run on DSM systems with minimum changes

Algorithms for implementing DSM

- Issues
  - How to keep track of the location of remote data
  - How to minimize communication overhead when accessing remote data
  - How to access concurrently remote data at several nodes
- 1. The Central Server Algorithm
  - Central server maintains all shared data
    - Read request: returns data item
    - Write request: updates data and returns acknowledgement message
  - Implementation
    - A timeout is used to resend a request if acknowledgment fails
    - Associated sequence numbers can be used to detect duplicate write requests
    - If an application's request to access shared data fails repeatedly, a failure condition is sent to the application
  - Issues: performance and reliability
  - Possible solutions
    - Partition shared data between several servers
    - Use a mapping function to distribute/locate data

Algorithms for implementing DSM (cont.)

- 2. The Migration Algorithm
  - Operation
    - Ship (migrate) entire data object (page, block) containing data item to requesting location
    - Allow only one node to access a shared data at a time
  - Advantages
    - Takes advantage of the locality of reference
    - DSM can be integrated with VM at each node
      - Make DSM page multiple of VM page size
      - A locally held shared memory can be mapped into the VM page address space
      - If page not local, fault-handler migrates page and removes it from address space at remote node
  - To locate a remote data object:
    - Use a location server
    - Maintain hints at each node
    - Broadcast query
  - Issues
    - Only one node can access a data object at a time
    - Thrashing can occur: to minimize it, set minimum time data object resides at a node

Algorithms for implementing DSM (cont.)

- 3. The Read-Replication Algorithm
  - Replicates data objects to multiple nodes
  - DSM keeps track of location of data objects
  - Multiple nodes can have read access or one node write access (multiple readers-one writer protocol)
  - After a write, all copies are invalidated or updated
  - DSM has to keep track of locations of all copies of data objects. Examples of implementations:
    - IVY: owner node of data object knows all nodes that have copies
    - PLUS: distributed linked-list tracks all nodes that have copies
  - Advantage
    - The read-replication can lead to substantial performance improvements if the ratio of reads to writes is large

Algorithms for implementing DSM (cont.)

- 4. The Full–Replication Algorithm
  - Extension of read-replication algorithm: multiple nodes can read and multiple nodes can write (multiple-readers, multiple-writers protocol)
  - Issue: consistency of data for multiple writers
  - Solution: use of gap-free sequencer
    - All writes sent to sequencer
    - Sequencer assigns sequence number and sends write request to all sites that have copies
    - Each node performs writes according to sequence numbers
    - A gap in sequence numbers indicates a missing write request: node asks for retransmission of missing write requests

Memory coherence

- DSM are based on
  - Replicated shared data objects
  - Concurrent access of data objects at many nodes
- Coherent memory: when value returned by read operation is the expected value (e.g., value of most recent write)
- Mechanism that control/synchronizes accesses is needed to maintain memory coherence
- Sequential consistency: A system is sequentially consistent if
  - The result of any execution of operations of all processors is the same as if they were executed in sequential order, and
  - The operations of each processor appear in this sequence in the order specified by its program
- General consistency:
  - All copies of a memory location (replicas) eventually contain same data when all writes issued by every processor have completed

Memory coherence (Cont.)

- Processor consistency:
  - Operations issued by a processor are performed in the order they are issued
  - Operations issued by several processors may not be performed in the same order (e.g. simultaneous reads of same location by different processors may yields different results)
- Weak consistency:
  - Memory is consistent only (immediately) after a synchronization operation
  - A regular data access can be performed only after all previous synchronization accesses have completed
- Release consistency:
  - Further relaxation of weak consistency
  - Synchronization operations must be consistent which each other only within a processor
  - Synchronization operations: Acquire (i.e. lock), Release (i.e. unlock)
  - Sequence: Acquire
    - Regular access
    - Release

Coherence Protocols

- Issues
  - How do we ensure that all replicas have the same information
  - How do we ensure that nodes do not access stale data
- 1. Write-invalidate protocol
  - A write to shared data invalidates all copies except one before write executes
  - Invalidated copies are no longer accessible
  - Advantage: good performance for
    - Many updates between reads
    - Per node locality of reference
  - Disadvantage
    - Invalidations sent to all nodes that have copies
    - Inefficient if many nodes access same object
  - Examples: most DSM systems: IVY, Clouds, Dash, Memnet, Mermaid, and Mirage
- 2. Write-update protocol
  - A write to shared data causes all copies to be updated (new value sent, instead of validation)
  - More difficult to implement

Design issues

- Granularity: size of shared memory unit
  - If DSM page size is a multiple of the local virtual memory (VM) management page size (supported by hardware), then DSM can be integrated with VM, i.e. use the VM page handling
  - Advantages vs. disadvantages of using a large page size:
    - (+) Exploit locality of reference
    - (+) Less overhead in page transport
    - (-) More contention for page by many processes
  - Advantages vs. disadvantages of using a small page size
    - (+) Less contention
    - (+) Less false sharing (page contains two items, not shared but needed by two processes)
    - (-) More page traffic
  - Examples
    - PLUS: page size 4 Kbytes, unit of memory access is 32-bit word
    - Clouds, Munin: object is unit of shared data structure

Design issues (cont.)

- Page replacement
  - Replacement algorithm (e.g. LRU) must take into account page access modes: shared, private, read-only, writable
  - Example: LRU with access modes
    - Private (local) pages to be replaced before shared ones
    - Private pages swapped to disk
    - Shared pages sent over network to owner
    - Read-only pages may be discarded (owners have a copy)

Distributed Scheduling

- Good resource allocation schemes are needed to fully utilize the computing capacity of the DS
- Distributed scheduler is a resource management component of a DOS
- It focuses on judiciously and transparently redistributing the load of the system among the computers
- Target is to maximize the overall performance of the system
- More suitable for DS based on LANs

Motivation

- A locally distributed system consists of a collection of autonomous computers connected by a local area communication network
- Users submit tasks at their host computers for processing
- Load distributed is required in such environment because of random arrival of tasks and their random CPU service time
- There is a possibility that several computers are heavily loaded and others are idle of lightly loaded
- If the load is heavier on some systems or if some processors execute tasks at a slower rate than others, this situation will occur often

Distributed Systems Modeling

- Consider a system of N identical and independent servers
- Identical means that all servers have the same task arrival and service rates
- Let ρ be the utilization of each server, than P=1- ρ, is the probability that a server is idle
- If the ρ=0.6, it means that P=0.4,
- If the systems have different load than load can be transferred from highly loaded systems to lightly load systems to increase the performance

Issues in Load Distribution

- Load
  - Resource queue lengths and particularly the CPU queue length are good indicators of load
  - Measuring the CPU queue length is fairly simple and carries little overhead
  - CPU queue length does not always tell the correct situation as the jobs may differ in types
  - Another load measuring criterion is the processor utilization
  - Requires a background process that monitors CPU utilization continuously and imposes more overhead
  - Used in most of the load balancing algorithms

Classification of LDA

- Basic function is to transfer load from heavily loaded systems to idle or lightly loaded systems
- These algorithms can be classified as :
  - Static
    - decisions are hard-wired in the algorithm using a prior knowledge of the system
  - Dynamic
    - use system state information to make load distributing decisions
  - Adaptive
    - special case of dynamic algorithms in that they adapt their activities by dynamically changing the parameters of the algorithm to suit the changing system state

Basic Terminologies

- Load Balancing vs. Load sharing
  - Load sharing algorithms strive to reduce the possibility for a system to go to a state in which it lies idle while at the same time tasks contend service at another, by transferring tasks to lightly loaded nodes
  - Load balancing algorithms try to equalize loads at al computers
  - Because a load balancing algorithm transfers tasks at higher rate than a load sharing algorithm, the higher overhead incurred by the load balancing algorithm may outweigh this potential performance improvement

Basic Terminologies (contd.)

- Preemptive vs. Non-preemptive transfer
  - Preemptive task transfers involve the transfer of a task that is partially executed
  - Non-preemptive task transfers involve the transfer of the tasks that have not begun execution and hence do not require the transfer of the task's state
  - Preemptive transfer is an expensive operation as the collection of a task's state can be difficult
  - What does a task's state consist of?
  - Non-preemptive task transfers are also referred to as task placements

Components of a Load Balancing Algorithm

- Transfer Policy
  - determines whether a node is in a suitable state to participate in a task transfer
  - requires information on the local nodes' state to make decisions
- Selection Policy
  - determines which task should be transferred
- Location Policy
  - determines to which node a task selected for transfer should be sent
  - requires information on the states of remote nodes to make decisions
- Information policy
  - responsible for triggering the collection of system state information
  - Three types are: Demand-Driven, Periodic, State-Change-Driven

Stability

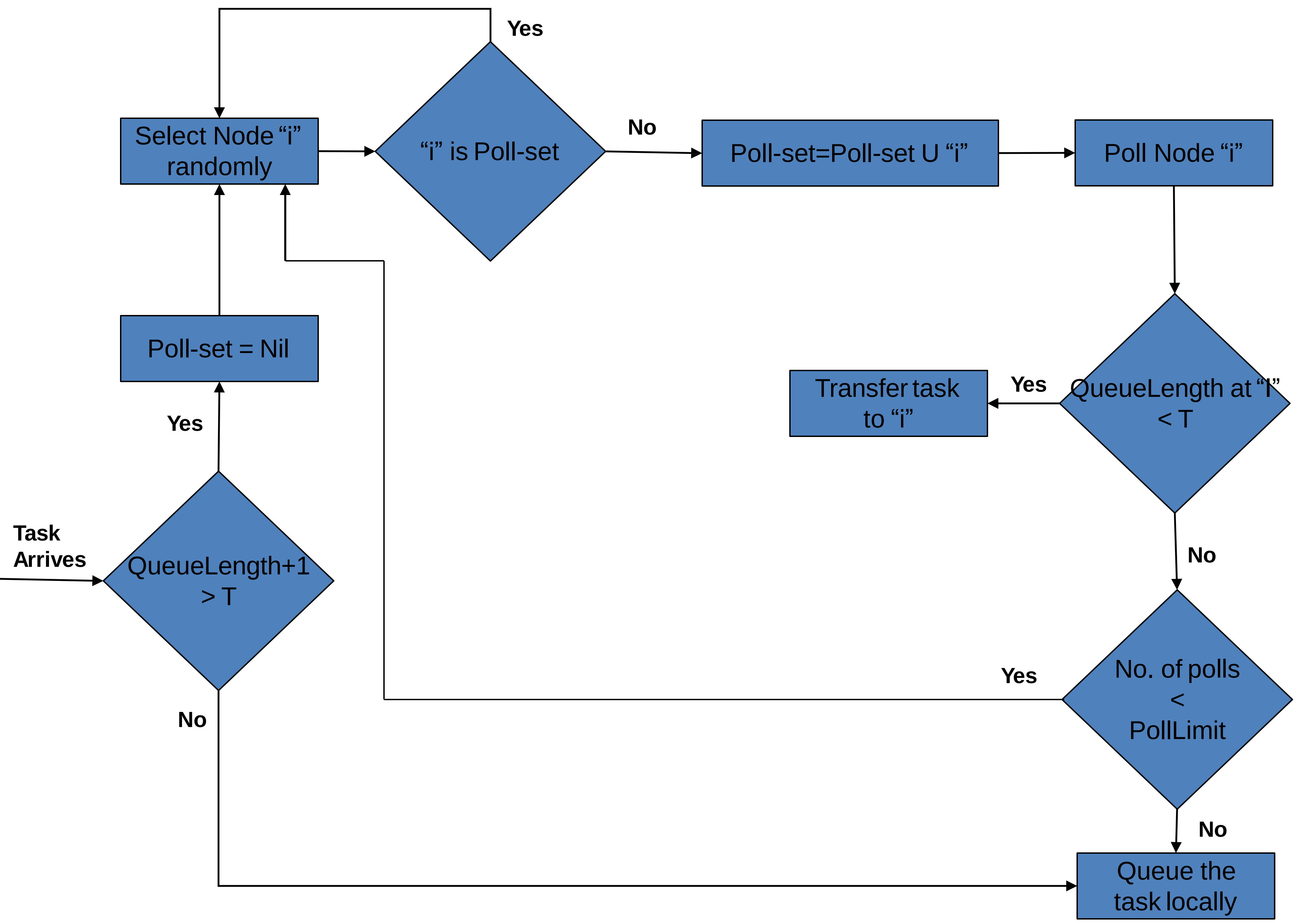
- The two views of stability are,
  - The Queuing-Theoretic Perspective
    - A system is termed as unstable if the CPU queues grow without bound when the long term arrival rate of work to a system is greater than the rate at which the system can perform work.
  - The Algorithmic Perspective
    - If an algorithm can perform fruitless actions indefinitely with finite probability, the algorithm is said to be unstable.

Load Distributing Algorithms

- Sender-Initiated Algorithms
- Receiver-Initiated Algorithms
- Symmetrically Initiated Algorithms
- Adaptive Algorithms

Sender-Initiated Algorithms

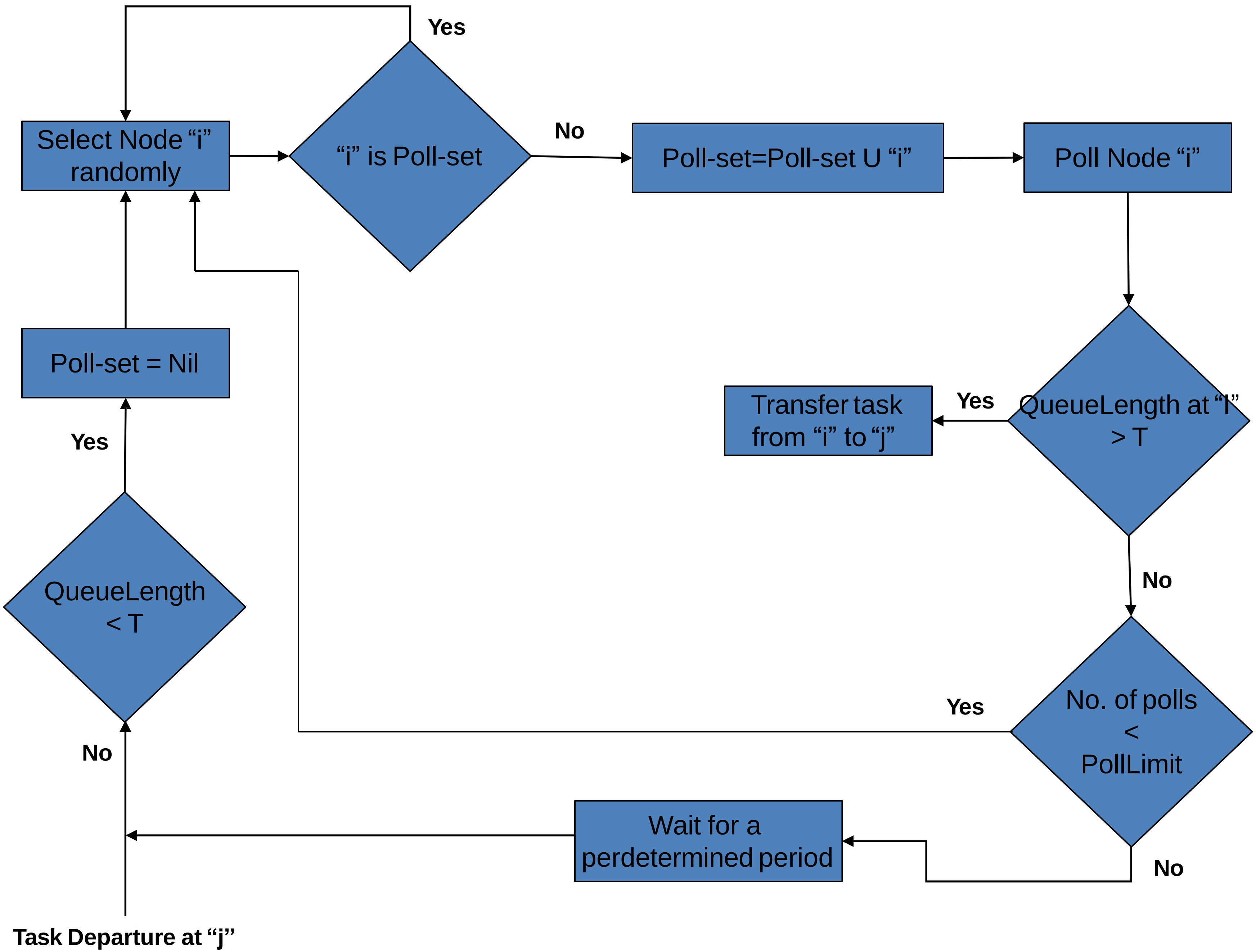
- Activity is initiated by an overloaded node (sender)
- A task is sent to an underloaded node (receiver)
  - Transfer Policy
    - A node is identified as a sender if a new task originating at the node makes the queue length exceed a threshold T.
  - Selection Policy
    - Only new arrived tasks are considered for transfer
  - Location Policy
    - Random: dynamic location policy, no prior information exchange
    - Threshold: polling a node (selected at random) to find a receiver
    - Shortest: a group of nodes are polled to determine their queue
  - Information Policy
    - A demand-driven type
  - Stability
    - Location policies adopted cause system instability at high loads



Yes
Select Node “i” randomly
“i” is Poll-set
No
Poll-set=Poll-set U “i”
Poll Node “i”
Poll-set = Nil
Transfer task to “i”
Yes
QueueLength at “I” < T
Yes
Task Arrives
QueueLength+1 > T
No
No. of polls < PollLimit
Yes
No
No
Queue the task locally

Receiver-Initiated Algorithms

- Initiated from an underloaded node (receiver) to obtain a task from an overloaded node (sender)
  - Transfer Policy
    - Triggered when a task departs
  - Selection Policy
    - Same as the previous
  - Location Policy
    - A node selected at random is polled to determine if transferring a task from it would place its queue length below the threshold level, if not, the polled node transfers a task.
  - Information Policy
    - A demand-driven type
  - Stability
    - Do not cause system instability in high system load, however, in low load it spare CPU cycles
    - Most transfers are preemptive and therefore expensive


Yes
Select Node "i" randomly
"i" is Poll-set
No
Poll-set=Poll-set U "i"
Poll Node "i"
Poll-set = Nil
Transfer task from "i" to "j"
Yes
QueueLength at "I" > T
Yes
QueueLength < T
No
No. of polls < PollLimit
Yes
No
Wait for a perdetermined period
No
Task Departure at "j"

Symmetrically Initiated Algorithms

- Both senders and receivers search for receiver and senders, respectively, for task transfer.
- The Above-Average Algorithm
  - Transfer Policy
    - Thresholds are equidistant from the node's estimate of the average load across all node.
  - Location Policy
    - Sender-initiated component: Timeout messages TooHigh, TooLow, Accept, AwaitingTask, ChangeAverage
    - Receiver-initiated component: Timeout messages TooLow, LooHigh, Accept, AwaitingTask, ChangeAverage
  - Selection Policy
    - Similar to both the earlier algorithms
  - Information Policy
    - A demand-driven type but the acceptable range can be increased/decreased by each node individually.

Adaptive Algorithms

- A Stable Symmetrically Initiated Algorithm
  - Utilizes the information gathered during polling to classify the nodes in the system as either Sender, Receiver or OK.
  - The knowledge concerning the state of nodes is maintained by a data structure at each node, comprised of a senders list, a receivers list, and an OK list.
  - Initially, each node assumes that every other node is a receiver.
  - Transfer Policy
    - Triggers when a new task originates or when a task departs.
    - Makes use of two threshold values, i.e. Lower (LT) and Upper (UT)
  - Location Policy
    - Sender-initiated component: Polls the node at the head of receiver's list
    - Receiver-initiated component: Polling in three order
      - Head-Tail (senders list), Tail-Head (OK list), Tail-Head (receivers list)
  - Selection Policy: Newly arrived task (SI), other approached (RI)
  - Information Policy: A demand-driven type

Thank U