DFS
Presentation to IBM

4/11/91

Locus

DFS SESSION

- DFS Overview
- Name Space
- VFS+
- Volume Location Data Base
- Fileset Replication
- Fileset Storage (Episode)
- Client Cache Server
- DFS File Server
- Token Management
- Access Control Lists
- NFS Interoperability
- Administration

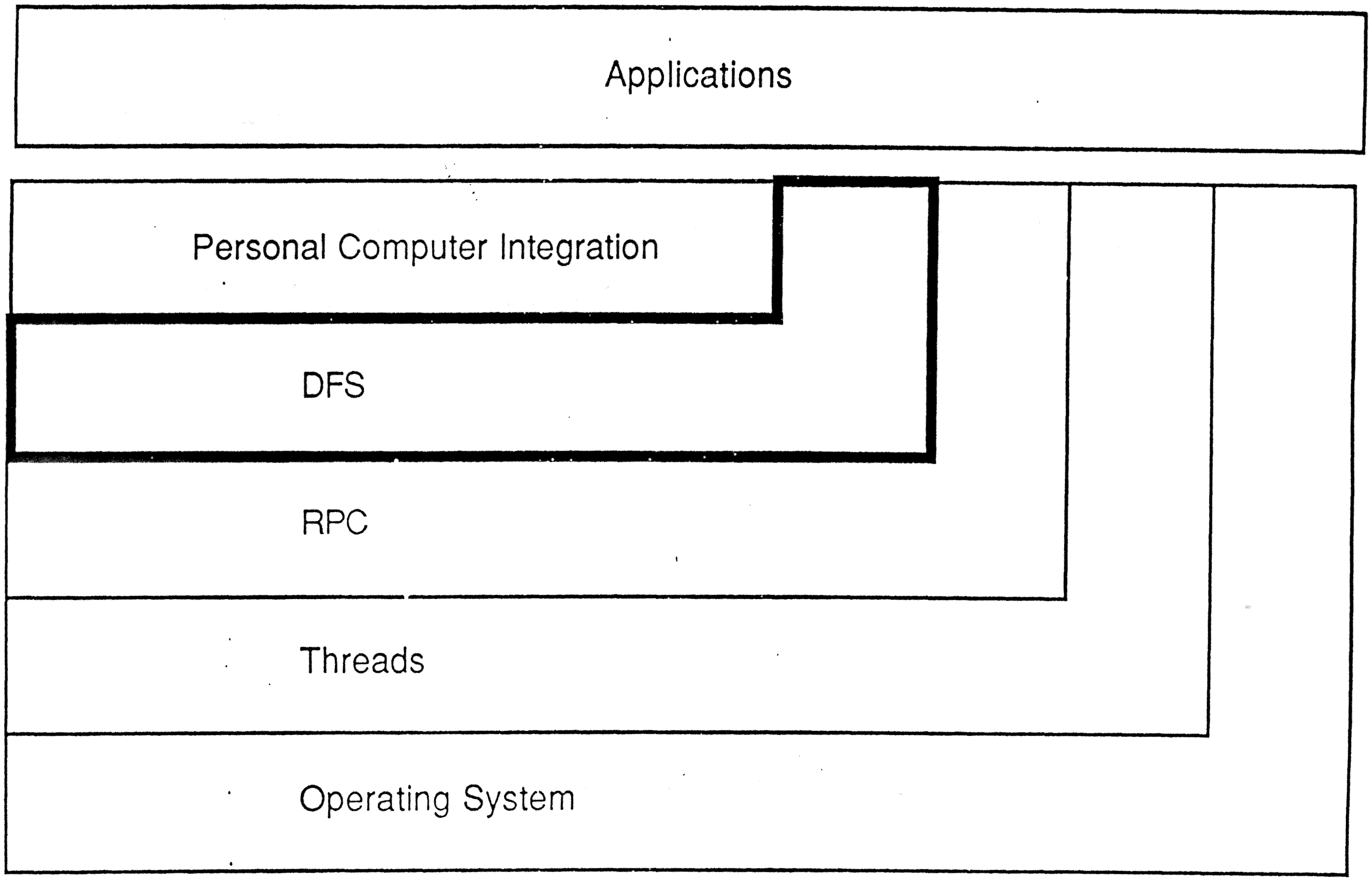
DFS OVERVIEW

- Concepts
- Design Goals
- Architecture

CONCEPTS

- Distributed File System
  - Users share information in a set of files without using same machine
  - Allows processing load to be shared across a set of machines without losing access to data
- Client-Server Model
  - Server machines are those that have permanent copies of file systems
  - Client machines are those that request access to files in file systems stored on server machines

WHERE DFS FITS IN DCE ARCHITECTURE



Applications

Personal Computer Integration

DFS

RPC

Threads

Operating System

LOCUS

DESIGN GOALS

- Scalability
- Name Transparency
- Replication
- Simplicity of Administration
- Heterogeneity of Hardware and Operating Systems
- Security
- Performance
- Availability
- Flexibility
- Interoperability with other Distributed File Systems
- Diskless Operation

Locus

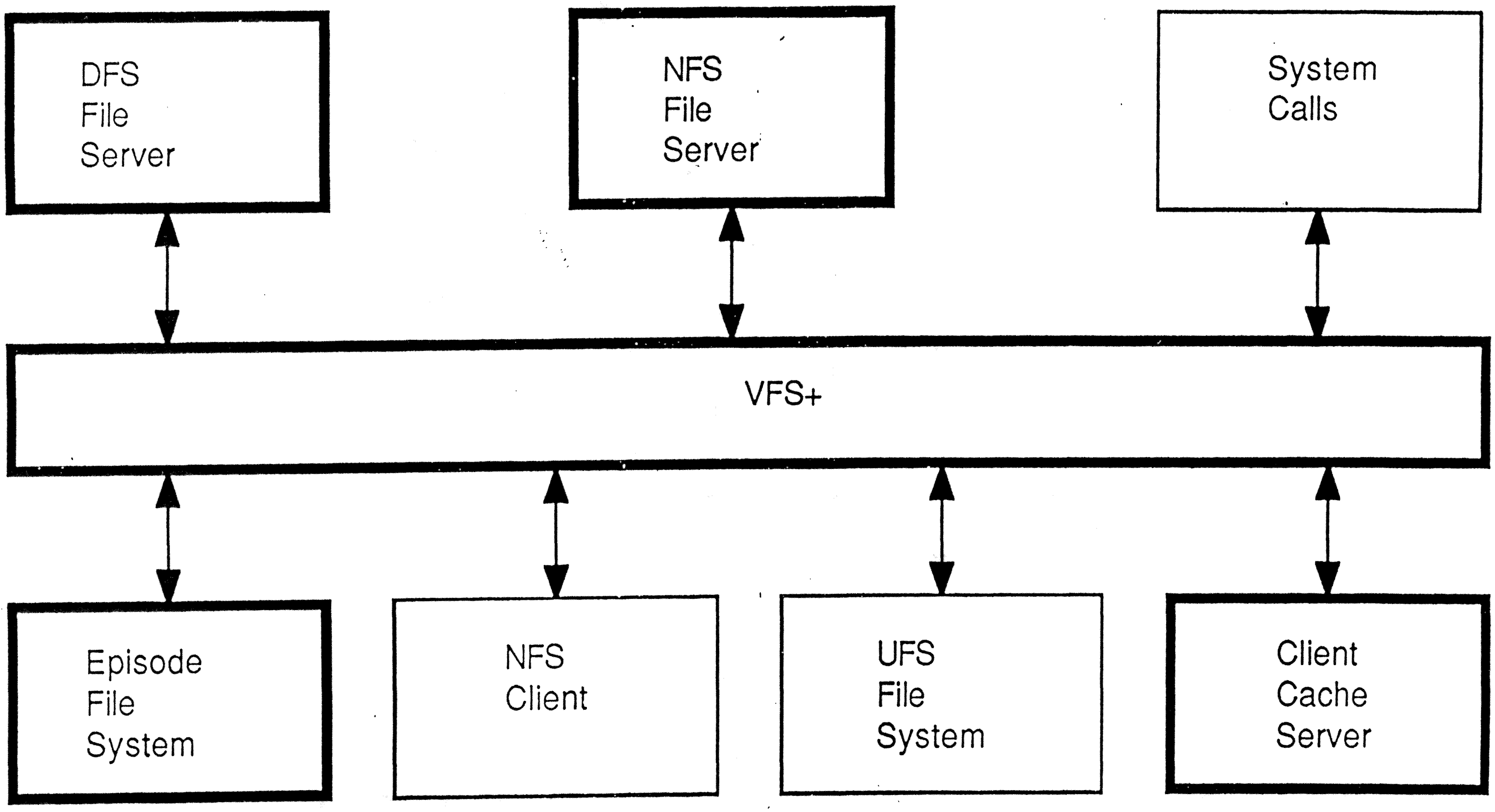
ARCHITECTURE

- Cells
- Modular Structure
- Filesets
- Cache Manager
- Consistency Mechanism
- Security
- Protection
- VFS+

CELLS

- Group of one or more machines comprising an administrative domain
- Common administration for machines
- A degree of trust between machines
- Examples:
  - Company
  - Division of Company
  - Department of University

DFS ORGANIZATION



DFS File Server

NFS File Server

System Calls

VFS+

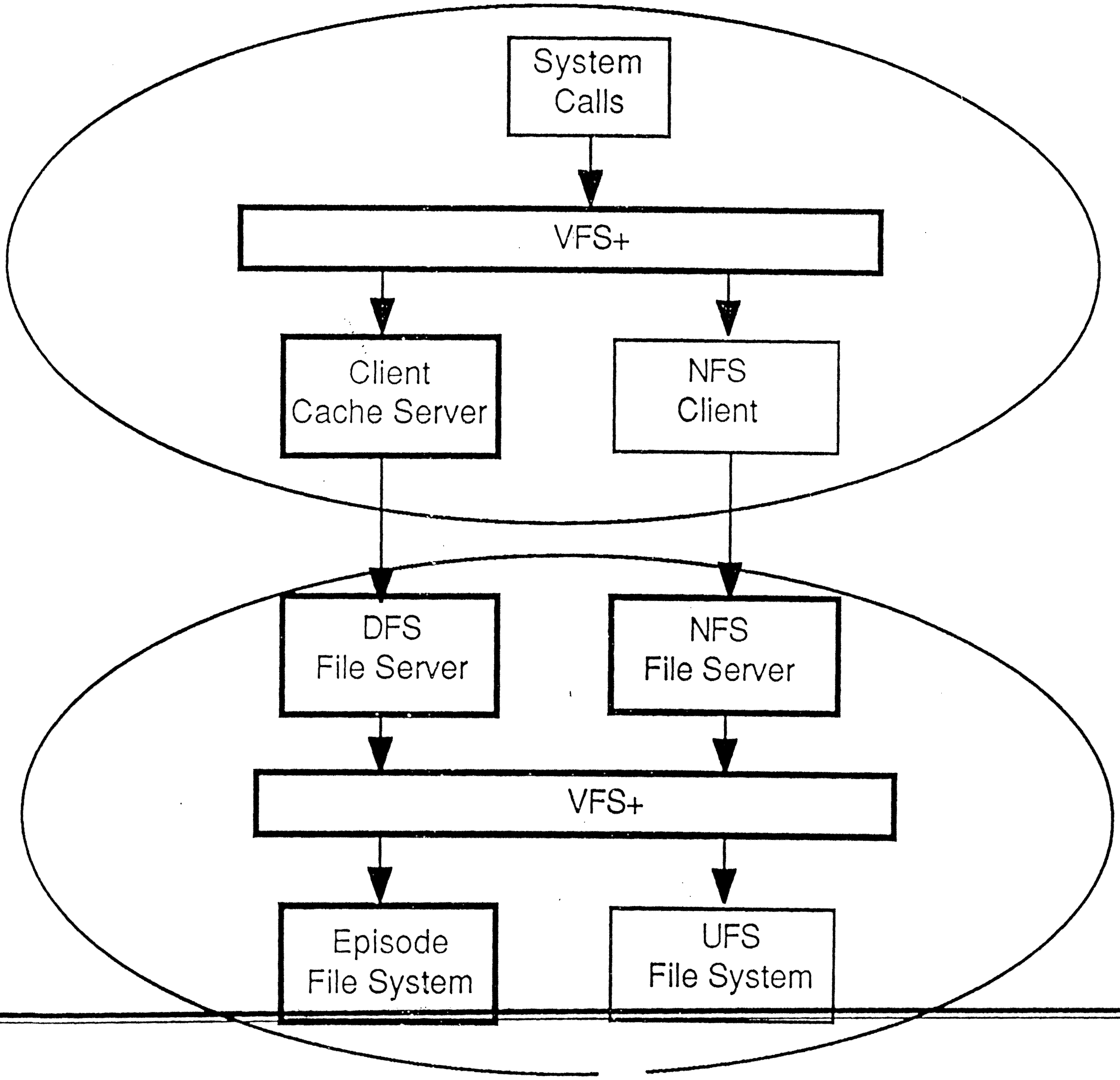
Episode File System

NFS Client

UFS File System

Client Cache Server

LOGICAL VIEW OF DFS ORGANIZATION



Locus

FILESETS

- Also known as volumes
- Related to traditional UNIX file system
- Handle to section of disk
  - corresponds to subtree of hierarchical file system
  - unit of administration
- Global name space is a collection of filesets
  - joined at mount points
- Volume Location Data Base (VLDB) tracks filesets

CLIENT CACHE SERVER

- Also known as "client" or "client server" or "cache manager"
- Used when accessing volumes not on caller's machine
- Interprets path names
  - finds mount points, accesses remote filesets
  - parses path name until file or directory leaf if reached
- Copies all or part of desired object to caller's machine

CONSISTENCY MECHANISM

- DFS uses token-passing
  - minimizes interaction between machines, yet maintains semantics of local file systems
- Types of tokens
  - Open
  - Lock
  - Data
  - File Synchronization
- Client only performs operations granted by token
- Tokens can be revoked by server or voluntarily given up by clients

SECURITY

- DFS uses Kerberos authentication
- Requires Kerberos tickets to be acquired
- Tickets presented to servers for operation

Locus

PROTECTION

- Standard UNIX mode bits
  - owner, group, other
  - read, write, execute permissions
- Also supports ACLs
  - specific set of rights for a specific set of users
  - much more flexible than standard UNIX modes
- Compatibility with other ACL schemes
  - POSIX ACLs
  - OSF/1 ACLs

Locus

VFS+

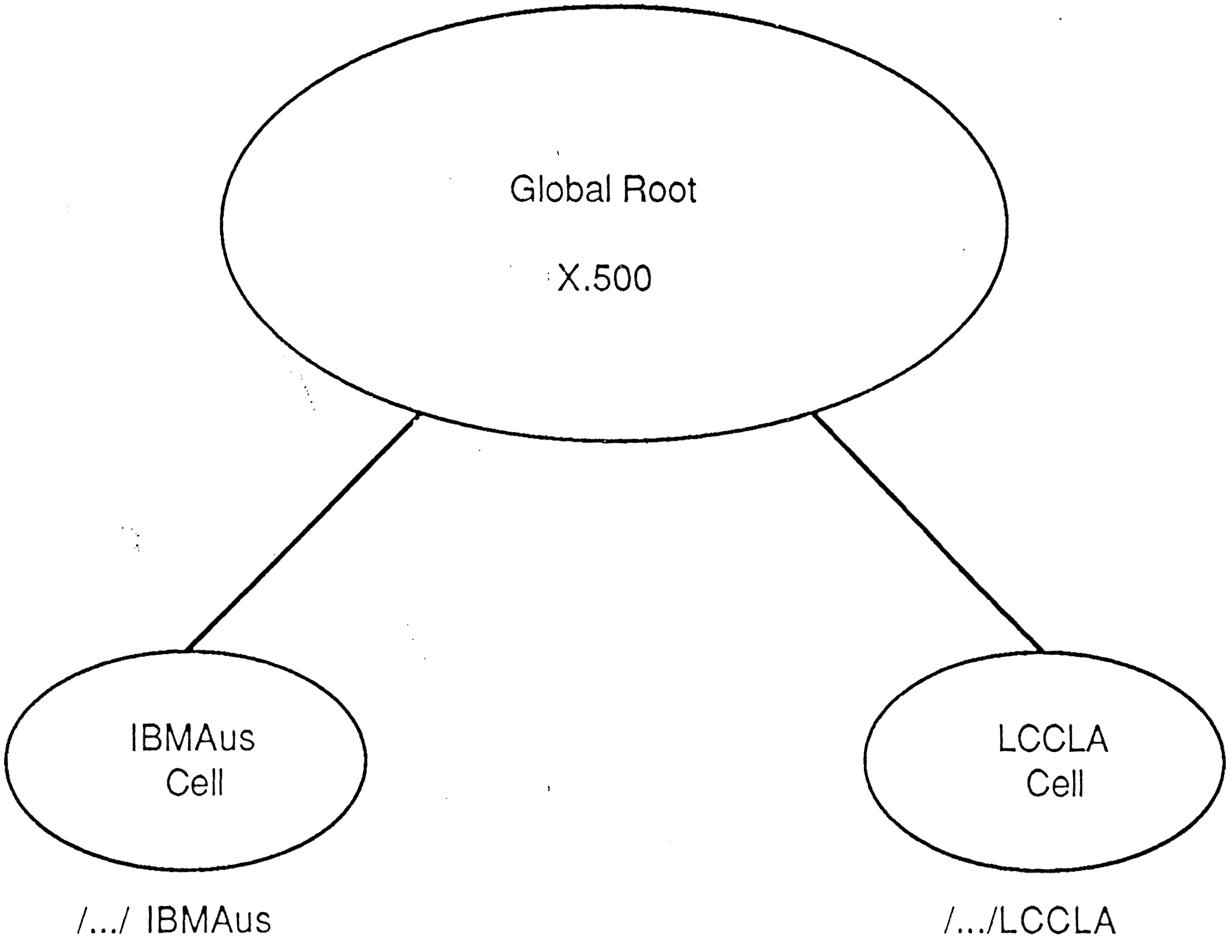
- VFS is standard file system switch for today's UNIX
- Allows multiple file system types to be supported in one operating system
- DFS provides extended version of VFS (called VFS+)
  - adds security and synchronization to standard VFS

NAME SPACE

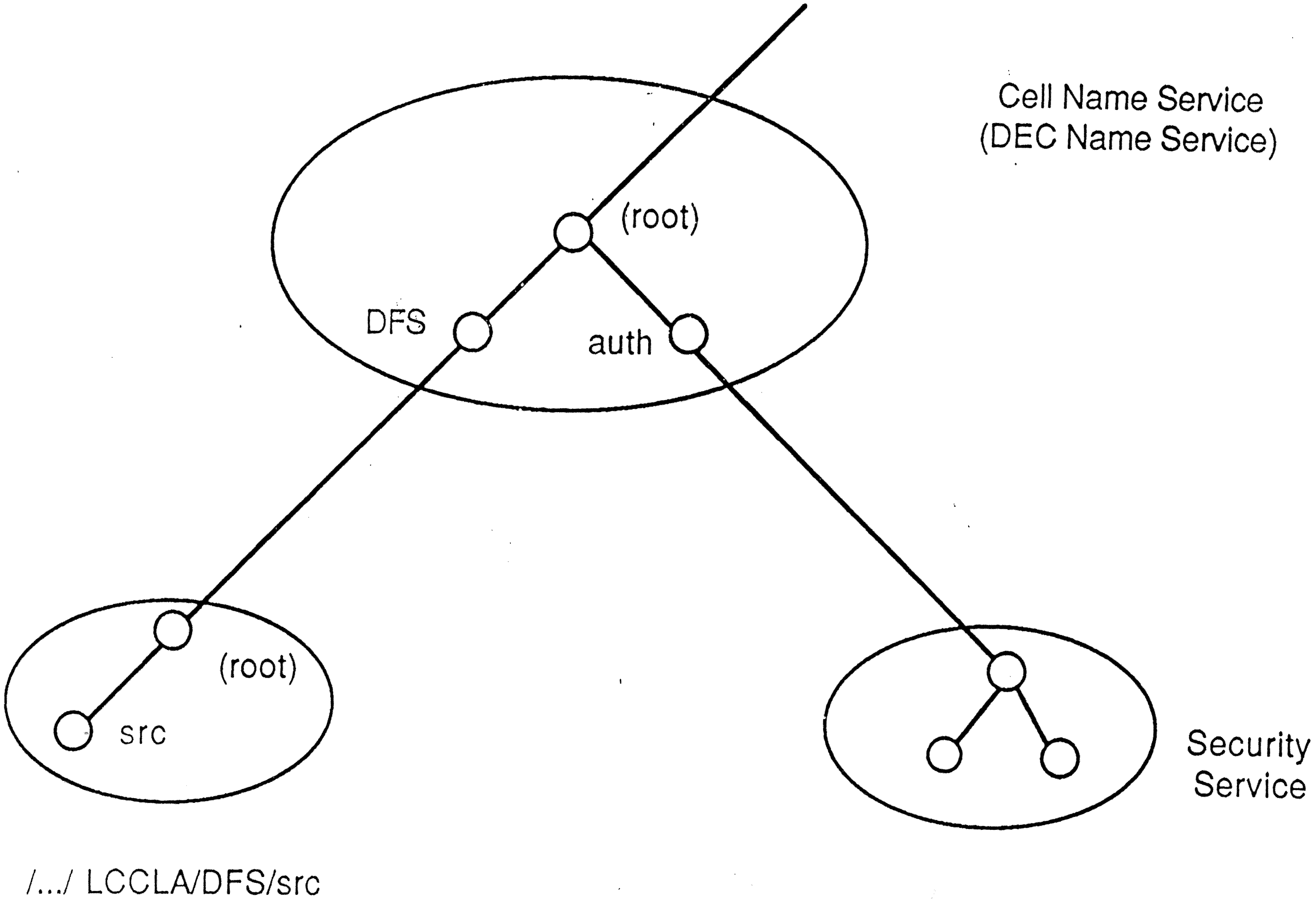
- Global Name Space
- Cell Name Space
- DFS Mounts
- Filesets

Locus

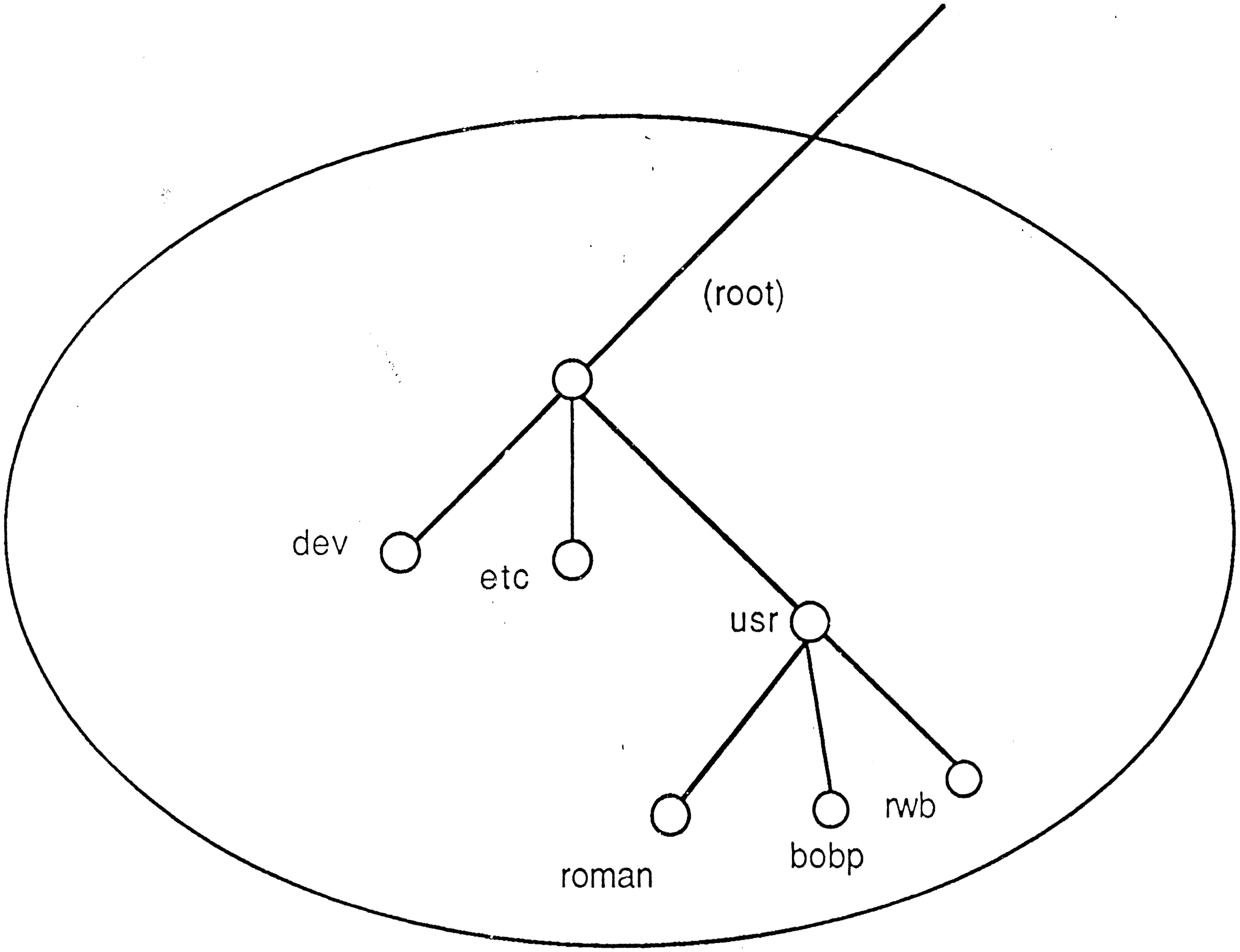
GLOBAL NAME SPACE



CELL NAME SPACE



DFS NAME SPACE

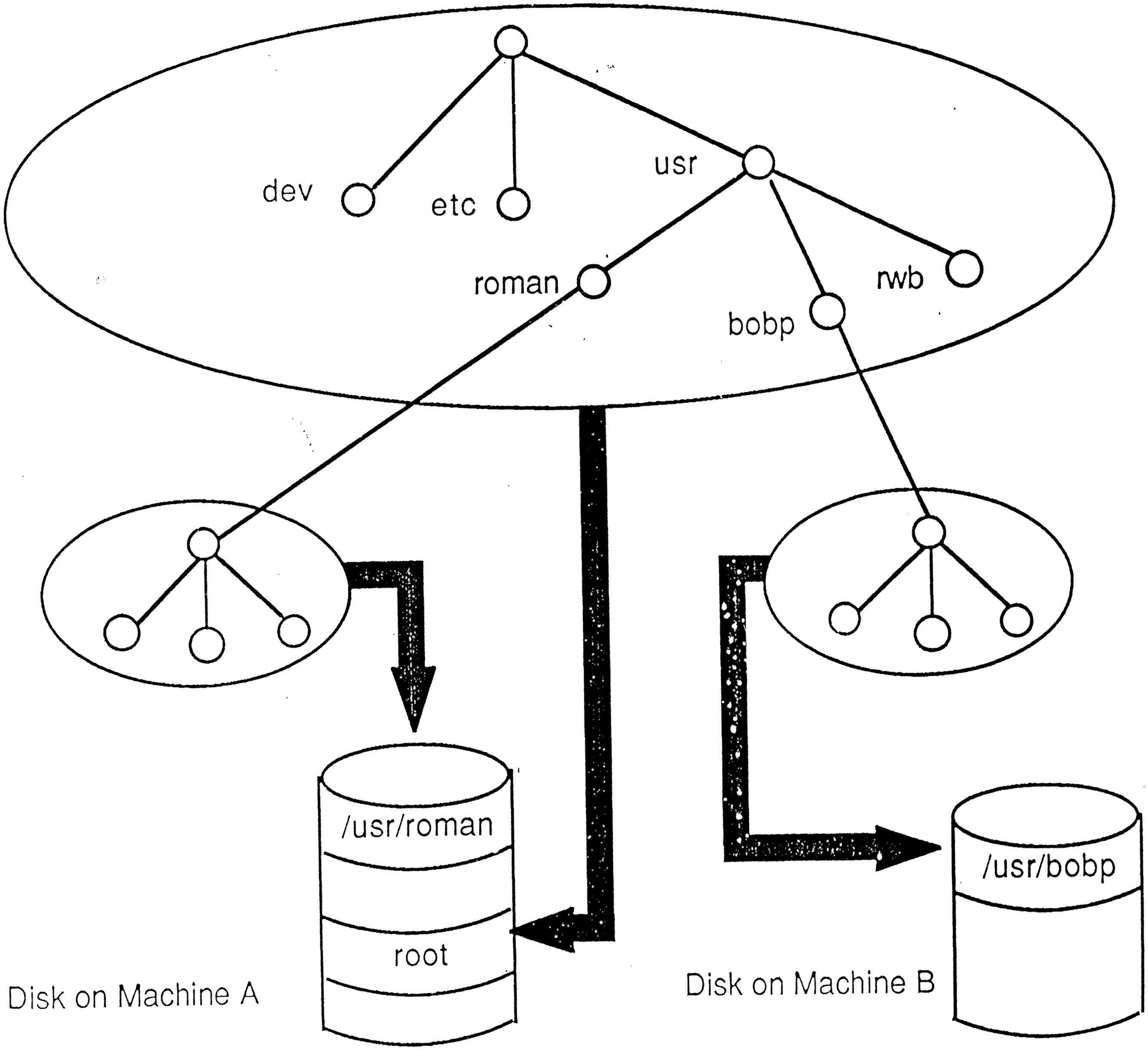


Locus

DFS MOUNTS

- Name space created by grafting subtrees using mounts
- Two types of mounts
  - In-memory mounts
    - traditional UNIX mount model
    - private to machine
  - On-disk mounts
    - mount is exported to cell (and thus globally)
    - special symbolic link to fileset to be mounted
- Standard DFS on-disk mount merely has name of fileset to be mounted
  - fileset name looked up in VLDB
  - VLDB has information about where fileset is stored

FILESETS



Locus

VFS+

- VFS Overview
- VFS+ Overview
  - Generalized Credentials
  - Synchronization
  - Operations on Filesets and Aggregates

VFS OVERVIEW

- VFS interface proposed in 1986
- VFS interface appears in most UNIX kernels today
- Architecture for accommodating multiple physical file systems on one machine
  - some file system types may be local, some may be remote
- All accesses to file system objects made through vnodes
- All operations on file systems done through vnode ops
- VFS has evolved over time
  - to be better isolated from the rest of the kernel
  - to allow different virtual memory implementations
  - to support distributed file systems better

Locus

VFS+ OVERVIEW

- Improvement to general VFS interface
- Principal improvements
  - Generalized credentials
  - Synchronization
  - Operations on filesets and aggregates
  - Operations on ACLs
- VFS+ can be added to base kernel, or added without modifying kernel
- VFS+ is part of OSF/1.1

GENERALIZED CREDENTIALS

- Credentials extended to allow Kerberos and other authentication methods
- Credential used by most VFS functions is generalized
  - Addition of "magic cookie" to credential
  - "Magic cookie" associated with a property list
  - Property list associated with user identification info.
    - NFS identity
    - Kerberos identity
- File systems that don't understand magic cookie use standard UNIX permissions

SYNCHRONIZATION

- Most operations redefined to call a synchronization package
- Protocol exporters can make guarantees to clients, and be notified when any changes are made to an object

Locus

OPERATIONS ON FILESETS AND AGGREGATES

- Done through *pioctl* interface
- Allows multiple implementations of filesets and aggregates
- Operations on aggregates
  - Creating new filesets
  - Iterating through all filesets
  - Deleting filesets
- Operations on filesets
  - Take fileset off-line
  - Iterating through all files in a fileset
  - Putting fileset on-line
  - Dealing with fileset quotas

VOLUME LOCATION DATA BASE (VLDB)

- VLDB Overview
- Ubik Goals
- Ubik Programming Model
- Ubik Algorithm
- Ubik Coordinator
- Ubik Recovery

VLDB OVERVIEW

- Maintains information about every fileset in cell
- Maintains following information about filesets
  - Fileset name
  - Fileset ID of read/write replica
  - Fileset ID of backup clone
  - Fileset ID of read-only replicas
- VLDB used:
  - When cache manager encounters a mount point (fileset name)
  - When cache manager encounters a fileset ID with no cached entry
  - By administration commands for replication, backup/restore and other fileset operations
- VLDB not used frequently, but is replicated for high availability
- Uses Ubik as its data base and replication mechanism

UBIK GOALS

- Strong consistency guarantees on updates
- Small number of write transactions
- Higher number of read transactions
  - 10 to 100 times higher than writes
- Handle network partitions
- Simple programming abstraction
- High read-only availability
  - High read/write availability not a goal, but came "for free", so was done

Locus

UBIK PROGRAMMING MODEL

- Ubik supports the following operations:
  - BeginTrans
  - Read
  - Write
  - SetLock
  - AbortTrans
  - EndTrans

UBIK ALGORITHM

- For data base writes, Ubik uses a quorum completion algorithm
- Quorum of more than half replicated servers must be present to perform data base writes
- Data base reads need only any single server to be present
- Two basic algorithms
    - Election of coordinator
    - Performing data base commits
- Algorithms sensitive to clock skew

UBIK COORDINATOR

- An Ubik data base has a single coordinator site
- Coordinator elected by majority of sites
- Coordinator is elected for a fixed time period (called epoch)
- Coordinator attempts to extend epoch when fixed time period half over
- New coordinator makes sure all remote data bases up to date before allowing transactions

Locus

UBIK WRITE TRANSACTIONS

- Writes occur on coordinator site
- Commit message sent to each server
- After successful commit, unlock messages sent to all servers
- If commit makes it to one server, transaction may survive
  (depends on new coordinator)
- If a quorum of commits happen, transaction will survive

UBIK RECOVERY

- Recovery executed by a new coordinator, and also by existing coordinator when servers are lost
- Several phases to recovery
  - Begin new epoch
  - Determine latest version of data base for all sites
  - Update coordinator to latest version
  - Label this version as first version of epoch
  - Update all remote data bases
- Transactions allowed when recovery is complete

FILESET REPLICATION

- Fileset Replication Overview
- Replication Behavior
- Policy on Updating Replicas
- Mechanism for Updating Replicas

Locus

FILESET REPLICATION OVERVIEW

- Replication is important part of a distributed file system
  - Performance
  - Availability
- Several conceptual forms of replication
  - Tight read/write replication
    - All changes made immediately visible on all replicas
  - Lazy read/write replication
    - Changes are eventually visible on all replicas
  - Read-only replication
    - Changes are not allowed

REPLICATION OVERVIEW (CONT.)

- DFS supports either lazy read/write replication or read-only replication, depending upon point of view
  - Read-only replication because access for writing is via a different name than accessing replicas
  - Lazy replication because changes to read/write copy do make it out to replicas later
- DFS replication at level of filesets
  - No partial replication permitted

REPLICATION BEHAVIOR

- VLDB stores:
  - Information about read/write fileset
  - Information about cloned (backup) fileset
  - Information about all replicas of filesets
- Mount points identify fileset
  - If fileset name has "*.readonly*" appended to name, a replica is accessed rather than read/write copy
  - If a replica is local, that one is used
  - No clever algorithm if no local fileset is present
- Normal reading and writing can occur on read/write fileset

POLICY FOR UPDATING REPLICAS

- Replicas should be updated periodically
- In AFS 3.0, updates under administrative control
  - If administrator forgets, updates not done
  - Requires either notification of administrator or administrative privileges
- Current DFS policy is automatic periodic update
  - Never know when update has/will occur
  - Possible to get inconsistent update
- May change to be determined on a per-fileset basis

MECHANISM FOR UPDATING REPLICAS

- Clone of read/write fileset is created
- Read-only replicas marked temporarily out of date
- Fileset is copied from clone to replicas
- Read-only replicas marked accessible
- Clone of read/write fileset stays in existence until all read-only replicas updated

FILESET STORAGE

- Episode - DFS physical file system
- Episode Features
- Episode Concepts
  - Aggregate
  - Fileset
  - Anode
  - File
- Episode logging
- Episode Implementation
- Optimization

EPISODE FEATURES

- Multiple filesets per partition
- Ability to move filesets from partition to partition (including partitions on different machines)
- Access control lists on files and directories
- Recovery without fsck (journalling file system)
- Cloning filesets for backup

Locus

EPISODE CONCEPTS

- Aggregate
  - Similar to disk partition
  - Unit of storage that knows about filesets
- Fileset
  - Administrative entity of aggregate
  - Has access control and quota information
  - Restricted to single aggregate
  - Can be moved, and addresses are preserved
- Anode
  - Describes a data container
  - Data containers correspond to files, directories, etc.

EPISODE CONCEPTS (CONT.)

- Fragment
  - Smallest unit of disk space
  - Usually 1K
- Block
  - Contiguous group of fragments
  - Treated as unit for efficient I/O

AGGREGATE

- Unit of storage that contains an Episode File System
- Generally corresponds to UNIX disk partition
    - May correspond to "volume" in Logical Volume Manager
- Addressed via block number and offset
    - Maximum size is $2^{42} = 4.4 \times 10^{12}$ bytes
    - Unit of allocation is still fragments
- Aggregate has 3 "well known" containers/anodes
    - Aggregate fileset list
        - Information about aggregate itself
        - Array of information about filesets
    - Aggregate bitmap
    - Aggregate recovery log

FILESETS

- Fileset information stored in aggregate fileset list
  - Fileset name
  - Fileset ID
  - Version number
  - Quota
  - Creation date
  - Status
  - Root directory
  - Other stuff
- Quotas require cooperation with aggregates
  - Free pool of blocks aggregate-wide
  - Quotas are on a per-fileset basis
  - Fileset monitors and (possibly) denies storage requests

FILESETS (CONT.)

- Filesets can be cloned
  - Makes use of copy-on-write
  - Simple creation of block lists would make the clone the read/write copy, so all anodes must be copied
  - Care must be taken deleting a clone (blocks used by read/write copy must not be lost)

ANODE

- Describes data in an aggregate
- Information for files to be implemented and for fileset operation
- Duties of anodes
  - Keep track of blocks
    - Includes copy-on-write management
  - Keep track of access control object
    - Access control object has fileset itself
    - Can be of arbitrary size
    - Shared due to inheritance

ANODES AND DATA STORAGE

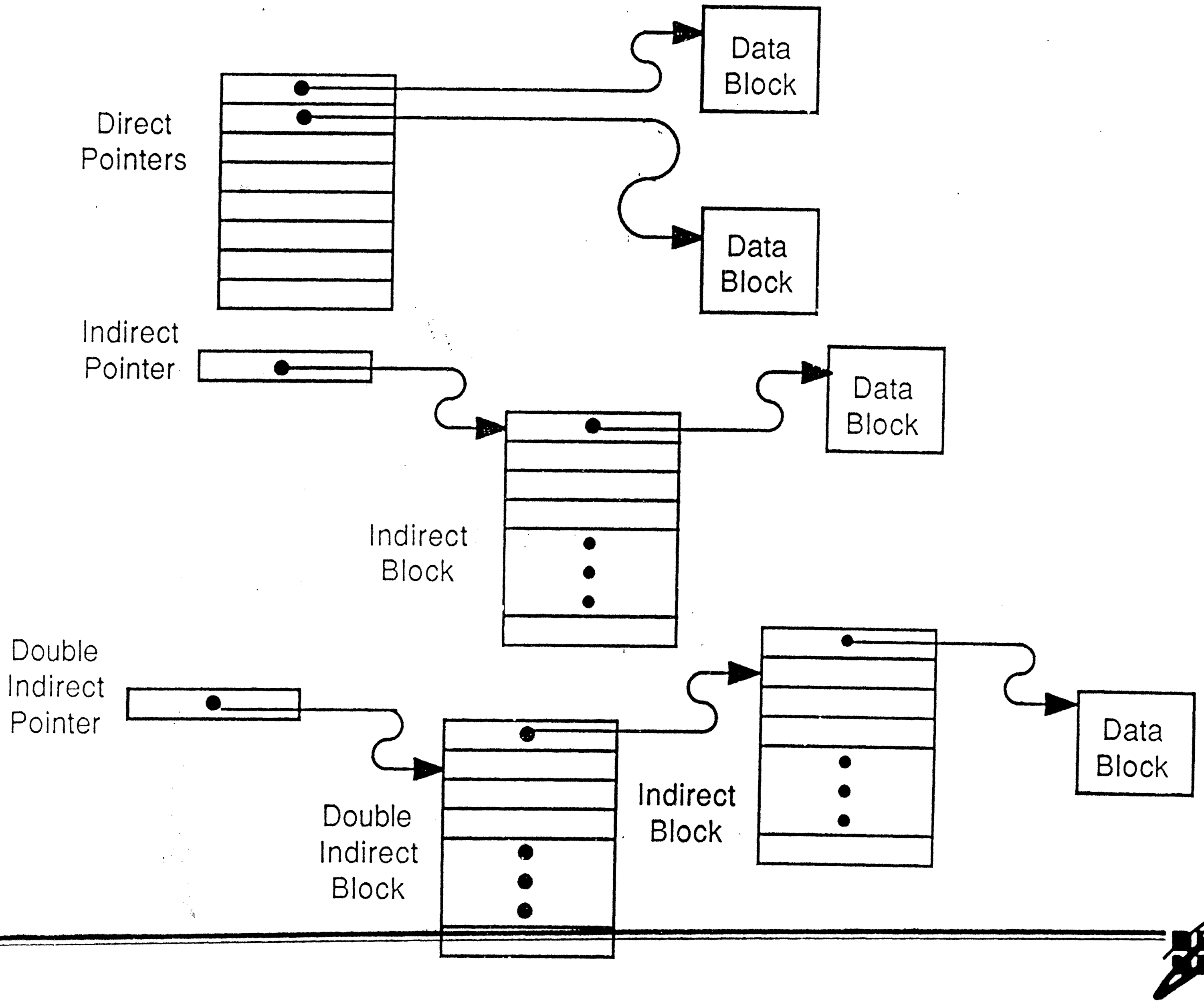
- Inline
  - Less than 150 bytes stored in anode itself
  - Useful for
    - Small files
    - Small access control list objects
    - Symbolic links
- Fragmented
  - Short files stored as continuous fragments in a block
  - If file grows, next fragment in block used if available
    - If not available, moved to different block

Locus

ANODES AND DATA STORAGE (CONT.)

- Large files
  - All files greater than one block in size
  - Described by tree of data blocks
  - First 8 blocks described by 8 block numbers
  - Next is first level indirect block number
  - Then a 2nd level indirect block number
  - Then a 3rd and 4th level indirect block number
  - Allows 3,921,937,758 blocks (or so)
  - Possible to have sparse allocation (address 0xffffffff)
    - Read of unallocated block produces 0
    - Write of unallocated block causes block allocation

LARGE EPISODE FILES



LOCUS

ANODES AND COPY-ON-WRITE

- Copy-on-write block number is 0xfffffffe, indicating inheritance
- Read of such a block goes to backing anode
- Write causes block to be allocated
- Inline files are never copy-on-write
- Fragmented files can be copy-on-write
  - If write occurs, entire file is copied

ANODES AND BLOCK ALLOCATION POLICY

- Hints about large file creation
- Keeping "logical" neighbors close by (cylinder groups)
- Allocation algorithm must be isolated and tunable

Locus

FILE

- Container for data
- Also has:
  - UNIX features like type and mode bits
  - Link count
    - Used for "hard" links to objects
  - DCE features like access control lists
    - Actually a "pointer" to another anode

Locus

EPISODE IMPLEMENTATION

- Episode is built in terms of layers
- Layers are (from lowest upwards):
  - Logging buffer package
    - Similar to UNIX buffer management
    - Makes many documented assumptions about underlying OS capabilities
  - First level anode layer
    - Read and write blocks of anodes
  - Layer for allocating and freeing anodes
    - Also has routines for allocating and freeing physical blocks
  - Layer for dynamically growing and shrinking anodes
  - Layer for maintaining directories
  - Layer for manipulating filesets and files

EPISODE LOGGING

- Meta-data changes logged (changes to anodes themselves)
  - Data is not logged
- After crash, log can be replayed to get consistent file system
  - Note that actual data can be lost
- Care must be taken to have meta-data never refer to unwritten blocks
  - Block must be written to disk before meta-data logged
  - Even more care must be taken for fragmented files
- Theoretically, logging causes better performance than standard UFS file system
  - This is because UFS has numerous forced writes to ensure system is recoverable after a system crash

OPTIMIZATION

- File that is written, referenced, deleted may never be written to disk
  - Must be enough buffer space for file
  - Queued I/O can simply be deleted
- Eliminating logged changes to meta-data is much harder, and not done

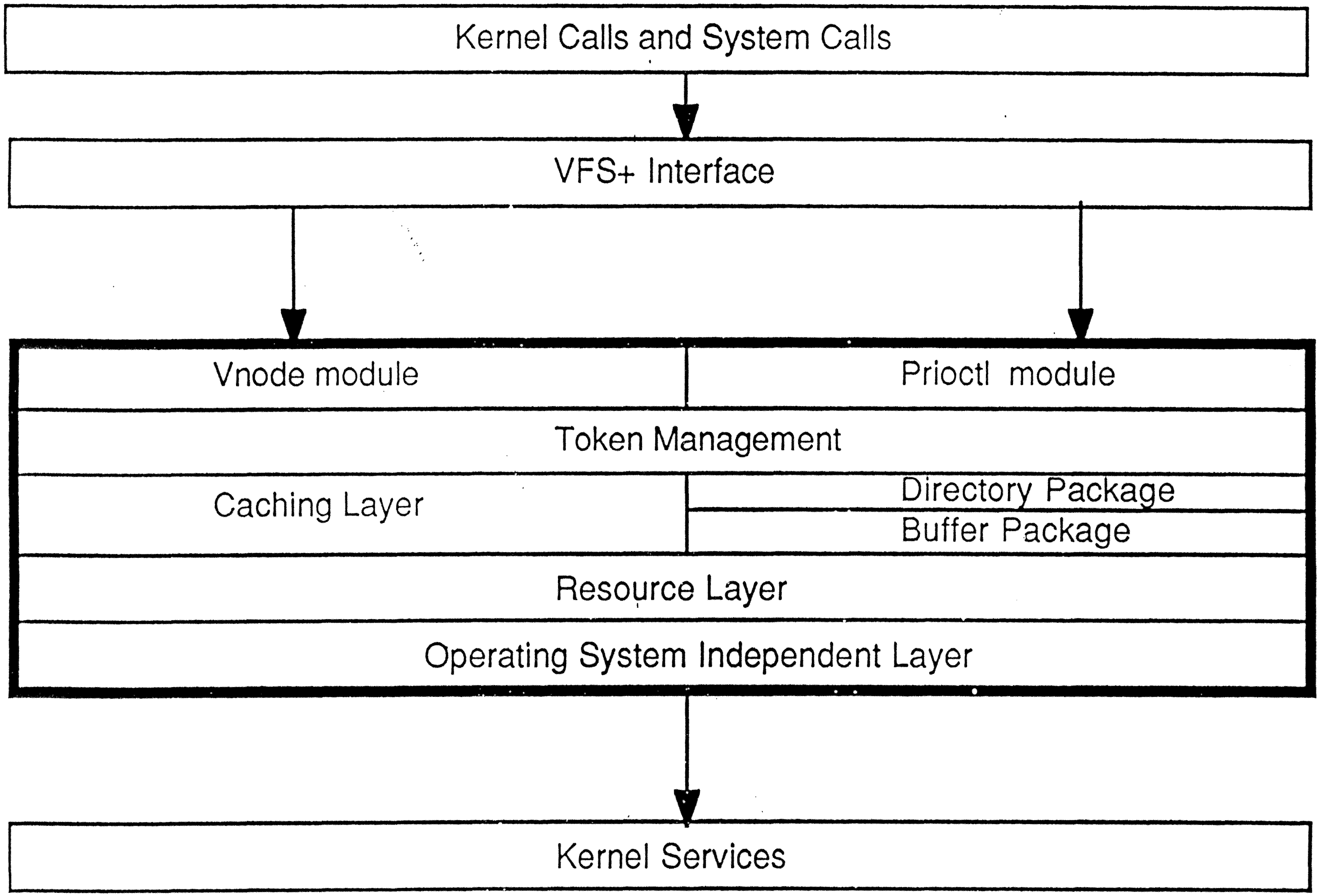
CLIENT CACHE SERVER

- Overview
- Architecture
  - Operating System Independent Layer
  - Resource Layer
  - Caching Layer
  - Directory Package
  - Token Management Layer
  - VFS/Vnode Module
- Performance

CLIENT CACHE SERVER OVERVIEW

- Sits physically below VFS layer (type of file system)
- Logically, sits between remote file server and rest of machine
- Client caches files from remote file servers
  - Also exports pioctl interface to allow local manipulations of access control lists and quotas
- Cache manager is what gives DFS its performance
  - Most accesses are from local cache
  - Very few writebacks from cache
  - Reason that DFS scales much better than NFS

CLIENT CACHE SERVER ARCHITECTURE



Locus

OPERATING SYSTEM INDEPENDENT LAYER

- Known as OSI layer
- Actually is one layer that is dependent on underlying UNIX operating system
- Responsibility of this layer to isolate rest of cache manager from underlying operating system
- Deals with
  - I/O to disk
  - network
  - file system
  - virtual memory

LOCUS

RESOURCE LAYER

- User Authentication Cache Module
  - Per-user Kerberos tickets
  - Per-user credentials
  - Mechanism for associating these credentials with VFS+ credential extension
- Server Module
  - Tracks recently contacted file servers
  - Status of servers - running or not, etc.
  - Maintains cache of RPC connections to servers

Locus

RESOURCE LAYER (CONT.)

- Fileset Module
  - List of accessed filesets
  - Mounted position
  - Physical locations
  - Basically a cache of VLDB information
- Cell Module
  - List of cells accessed by cache manager

CACHING LAYER

- Information about files, not directories
- NOT just in-memory cache, also uses local disk
- State kept across reboots
- Two types of information
  - Status of vnode
  - Data itself for vnode
- Data dealt with in terms of chunks
  - Current chunk size 64K
- This layer used to fetch and store data
- Uses LRU algorithm on cached data
- Currently uses UFS for storing cached data on disk

DIRECTORY PACKAGE

- Same directory operations as on server allowed
- Keeps cached directories synchronized with file server directories
- Separate package so that cached directories can be updated without fetching whole directory
- Uses special buffer package to interface to underlying operating system for caching pages of directories

TOKEN MANAGEMENT LAYER

- Determines what operations can be performed on cached files and when
- Deals with server token revocation requests
- Handles token probes
- Token management covered elsewhere

VFS/VNODE MODULE

- Deals with all lower levels
  - Makes them look like a true file system
- Vnode module deals with standard VFS calls
- Pioctl module deals with pioctl extension to VFS
  - Mostly, requests are passed on to file server
  - Deals with:
    - Cache configuration
    - Access control list operations
    - Kerberos management
    - Mount point maintenance
    - Cell operations
- Vnode layer checks if fileset is local
  - If so request directly passed to local file server

PERFORMANCE

- Key to client cache server is performance
- Extensively uses local memory and disk as cache
- Minimizes network traffic and delays
  - Minimizes communication between caching machine and file server
- Resides in kernel
- Designed to deal with thousands of clients
  - High client/server ratio

Locus

DFS FILE SERVER

- DFS File Server Overview
- Host Module
- Fileset Registry Module
- Token Manager Module

Locus

DFS FILE SERVER OVERVIEW

- Also known as "DFS protocol exporter"
- Resides on site where fileset (file system) is stored
- Responsible for interaction between remote cache client servers and actual physical file systems
  - Physical file systems may be Episode filesets, UFS file systems, or other local physical file systems
- For each client cache request file server executes the following:
  - Obtain the *principal* structure (from the host module) for the request
  - Perform all token management operations (using the TKM), including revocations from other remote clients
  - Using fileset registry, convert file identifiers to vnodes
  - Perform function using vnode layer
  - Release everything and return to caller

Locus

HOST MODULE

- Maintained on all machines that are DFS file servers
- Maintains state of all clients using file server
- Executed early so authentication expiration can be checked, spotting of down client cache managers can occur, etc.

- Executed early so authentication expiration can be checked, spotting of down client cache managers can occur, etc.

- Two structures returned for every call - *host* structure and *principal* structure
- *Host* structure
  - State of client cache manager making request
  - Includes whether all token revocations have been delivered
- *Principal* structure
  - Kerberos identity of individual making call
  - Also per-user cached information

FILESET REGISTRY MODULE

- Maintained on all machines that are DFS file servers
- Keeps list of all filesets on local file server
  - Translates DFS file IDs to pointer to fileset structure along with pointer to vnode structure

Locus

TOKEN MANAGER MODULE (TKM)

- TKM maintains guarantees about operations that can be performed by clients
- TKM discussed elsewhere

TOKEN MANAGEMENT

- Token Management Overview
- Token Types
    - Open Tokens
    - Lock Tokens
    - Synchronization Tokens
    - Data Tokens
- Failure Cases

Locus

TOKEN MANAGEMENT OVERVIEW

- Key concept of DFS is "data consistency"
  - Also known as "Single System Image"
- Ability to access files from multiple sites as if all accesses from one site
  - Contrast with NFS and RFS
- Tokens used by file servers and client cache managers
  - Tokens exist on a per-file basis
- Requests intercepted, and interaction occurs with Token Manager (TKM)
  - TKM is background daemon
  - TKM for a file is on machine where file resides

Locus

TOKEN MANAGEMENT OVERVIEW (CONT.)

- Tokens are guarantees that client can perform certain actions without permission of server
  - Key to high performance of DFS
  - Key to "statefulness" of DFS
- Tokens naturally expire
  - Must be renewed prior to expiration
- When token needed, requested from TKM
  - TKM responsible for revoking conflicting tokens from other clients
  - Revocation can be refused

OPEN TOKENS

- Open tokens represent right to open a file
- Several types of open tokens:
  - Read (TKM_OPEN_READ)
  - Write (TKM_OPEN_WRITE)
  - Exclusive (TKM_OPEN_EXCLUSIVE)
  - Shared (TKM_OPEN_SHARED)
- Attempts to get open token can fail:
  - Exclusive if any other open outstanding
  - Shared if write or exclusive open outstanding
  - Read if exclusive open outstanding
  - Write if shared or exclusive open outstanding
- Exclusive token implies write lock on whole file
- Shared token implies read lock on whole file

Locus

LOCK TOKENS

- Lock tokens represent right to lock a file
  - Used for System V style record and file locking
- Can lock subrange of file
  - Subrange can be whole file
- Normally, initial request is to lock whole file
  - If revocation request occurs while lock is outstanding, whole file lock revoked and new lock tokens for subranges are granted
    - Can result in proliferation of lock tokens
- Two types of lock tokens
  - Read lock on subrange of file (TKM_LOCK_READ)
  - Write lock on subrange of file (TKM_LOCK_WRITE)

FILE STATUS TOKENS

- File status token represents right to access status of file
- Status includes:
  - Size of file
  - Last modification time
  - Access control lists
- Two types of file status tokens:
  - Read status (TKM_STATUS_READ)
  - Write status (TKM_STATUS_WRITE)
- Needed by most operations, in addition to other tokens
  - To write file, need TKM_STATUS_WRITE (to update modification time) in addition to TKM_DATA_WRITE (to actually modify data)

DATA TOKENS

- Data tokens represent right to read or write byte range in a file
- Can read or write subrange of file
  - Subrange can be whole file
- Normally, initial request is for lock of whole file
  - Subranges used if conflicts occur
- Two types of data tokens:
  - Read (TKM_DATA_READ)
  - Write (TKM_DATA_WRITE)
- Write token implies data in subrange may not have been written back to server
  - Revocation of write token means data must be written back first
  - File's version number not incremented until actual data makes it back (TKM_STATUS_WRITE token does not give right to change version number)

FAILURE CASES

- Of great interest is behavior of system when network partitions occur
- Current design unspecified to a large extent

Locus

ACCESS CONTROL LISTS

- Access Control Overview
- DFS Access Control Lists Overview
- ACL Rights
- ACL Commands
- ACL System Calls
- Summary

ACCESS CONTROL OVERVIEW

- Access Control is the principle of security
- Two types of policies
  - Discretionary
    - Access rights can be transferred to other users
    - Access authorized on user or group IDs
  - Mandatory
    - Access rights cannot be transferred
    - Access is based on subject and object labels
- DFS uses discretionary policy
- Standard POSIX permissions not enough
  - Read, write execute permissions
  - Single owner, multiple groups, everyone else
  - Distributed environment means more flexible access needed

Locus

ACCESS CONTROL OVERVIEW (CONT.)

- Access Control Lists are a general extension
  - Several definitions
    - POSIX
    - OSF/1 Security
    - AIXv3
    - UNIX International
    - DFS
  - Currently all subtly different
  - All iterating towards POSIX (hopefully)

DFS ACCESS CONTROL LISTS OVERVIEW

- DFS provides Access Control Lists
  - Provided for Episode files only
  - Both permissive and restrictive ACLs provided
- ACLs map users (OR groups of users) to rights
- Work in addition to POSIX mode bits
- Decision of what to do with mode bits
  - POSIX bits could be ignored
  - Permissions can be AND'ed together
  - Permissions can be OR'ed together
  - DFS chooses to AND permissions of ACL and POSIX bits
- Directory has ACL and "default ACL"
  - "Default ACL" is ACL used for any created nondirectory object in a directory
  - New directories inherit parent's ACL and default ACL

Locus

ACL RIGHTS

- *Read* right
  - For files, ability to read and/or stat files
  - For directories, ability to read names and IDs of files in directory, ability to stat directory
- *Write* right
  - For files, ability to write to a file
  - For directories, ignored
- *Execute/Search* right
  - For files, ability to execute a program
  - For directories, ability to examine individual entries
    - Unlike read, requires string name of entry to be presented

ACL RIGHTS (CONT.)

- *Insert* right
  - For files, ignored
  - For directories, ability to add entries to directory
    - Ability to perform renames within directory
    - If target to rename exists, must have delete right also
- *Delete* right
  - For files, ignored
  - For directories, ability to remove entries from directory
    - Includes ability to rename to different directory
    - Ability to rename "on top of" another entry
- *Owner* right
  - Ability to execute chmod and utimes on object
  - Ability to change ACL list for object

ACL COMMANDS

- ls
  - Also displays information about ACLs
- acledit
  - Change an object's ACL
- aclget
  - Read an object's ACL
- aclput
  - Set an object's ACL

ACL SYSTEM CALLS

- access, faccess
  - Extended versions to query rights
- chacl, fchacl
  - New call to set an object's ACL
- statacl, fstatacl
  - Return an object's ACL information

SUMMARY

- ACL's subject to change
- All ACLs should iterate to POSIX functionality

NFS INTEROPERABILITY

- NFS Interoperability Overview
- Implementation
- Performance

NFS INTEROPERABILITY OVERVIEW

- Also known as "NFS protocol exporter"
- Allows any DFS machine to act as NFS server
- Machine accepts standard NFS protocol
  - includes full support of Sun RPC and XDR
- Has extensions to NFS protocol (optional)
  - *pioctl* interface for dealling with quotas and ACL's
  - Allows support of *fs* command on NFS clients

Locus

IMPLEMENTATION

- Kernel implementation of NFS server
- Based on Sun NFS reference port
  - requires Sun NFS license
- No assumptions about underlying file system made
  - all operations based on VFS layer functions

Locus

PERFORMANCE

- Performance should be at least as good as native NFS server
  - Typical NFS client accesses file cached on server
  - With NFS protocol, NFS client typically accesses DFS cache
- If DFS cache miss occurs
  - Data copied from DFS server to NFS protocol exporter
  - Data copied from NFS protocol exporter to NFS client
  - Standard DFS client behavior is to make data available before whole chunk arrives over network
  - This allows better than might be expected performance

ADMINISTRATION

- Basic Overseer Server (BOS)
  - BOS Overview
  - BOS Command
- Fileset Server
  - Fileset Server Overview
  - Vos Command
- Backup System
  - Backup System Goals
  - Backup System Components
  - Backup Operation
  - Backup Commands

Locus

BASIC OVERSEER SERVER (BOS) OVERVIEW

- Oversees DFS as a whole
- Shortens length of DFS outages
- Automates many administrative functions
- Main functions:
  - Monitor other servers
    - Restarts servers automatically
    - Also has administrative interface for manual monitoring and restarting
  - Interface for changing *server encryption keys*
  - Interface for managing configuration interface
  - Interface for managing list of administrators
- User interface through *bos* command

BOS CONFIGURATION MANAGEMENT

- bos addhost
  - Add host to cell data base
- bos removehost
  - Remove host from cell data base
- bos listhosts
  - List the current cell host llist
- bos setcellname
  - Set the cell's name
- bos create
  - Create a new server instance
- bos delete
  - Delete a server instance

BOS PROCESS MONITORING

- bos rebos
  - Restart all servers on a machine, including BOS server
- bos restart
  - Restart all servers on a machine. Also reboot kernel if in-kernel server is restarted
- bos shutdown
  - Shutdown all server processes on a machine
- bos setrestart
  - Set a time for an automatic restart of all servers on a machine
- bos getrestart
  - Get the automatic restart time

BOS PROCESS MONITORING (CONT.)

- bos startup
  - Start all configured servers on a machine
- bos start
  - Start an individual server on a machine
- bos stop
  - Stop a server on a machine
- bos status
  - Show status of one or more server processes on a machine

BOS ADMINISTRATOR MANAGEMENT

- bos users
  - List server administrators
- bos addusers
  - Add users as server administrators
- bos removeuser
  - Remove users from server administrator list

BOS SERVER ENCRYPTION KEY MANAGEMENT

- bos addkey
  - Add keys to key data base
- bos listkeys
  - List server keys in key data base
- bos removekey
  - Remove key from key data base

FILESET SERVER

- Interface for operators to manipulate filesets
  - Can create, delete, move, replicate filesets
- Creating and deleting done for adding and deleting users
- Moving filesets is done for load balancing
- Replication done for placement on multiple file servers
- Volume server updates the VLDB
- All done through *vos* command

VOS COMMAND

- vos create
  - Creates a new fileset in an aggregate
- vos backup
  - Creates a clone of a fileset, adds *.backup* as extension to name of fileset
- vos backupsys
  - Performs *vos backup* on a set of filesets
  - All filesets in aggregate, all filesets on server, or all filesets with name matching pattern
- vos addsite
  - Prepare fileset for replica

Locus

VOS COMMAND (CONT.)

- vos remove
  - Remove fileset
  - If fileset is Read/Write copy, backup is also removed
    - Replicas stay
    - If no replicas, VLDB entry removed
  - If replica is specified, replica is removed
  - Backup copy may be removed
- vos remsite
  - Remove replica site for fileset
- vos move
  - Move fileset from one aggregate to another
- vos rename
  - Rename a fileset
  - Clones renamed as well

VOS COMMAND (CONT.)

- vos dump
  - Convert a fileset to ASCII and put into a file called *dump*
  - If *read-only* or *backup* desired, must be specified
  - Possible to do incremental dump
    - Only dump changes from a certain time
- vos restore
  - Convert a previous dump file back to a fileset
  - May over-write existing fileset, or create new one
- vos listvldb
  - List information about filesets
  - All filesets in aggregate, all filesets on server,
    all filesets matching pattern

VOS COMMAND (CONT.)

- vos listpart
  - List aggregates on server
  - Does not consult with VLDB
- vos partinfo
  - List information about aggregates
  - Particular aggregates or all aggregates on server
  - Gives free space and total space in aggregate
- Other *vos* commands more useful for debugging

BACKUP SYSTEM GOALS

- Must be more flexible than standard UNIX
- Standard UNIX deals with partitions
- DFS deals with filesets, which move from one aggregate/partition to another
- Backup operations deal with filesets
- Restore operations normally deal with logical tree location
  - Operator must translate to fileset to be restored
- Restores must avoid searching large number of tapes to determine relevent fileset

Locus

BACKUP SYSTEM COMPONENTS

- Backup data base
  - Data base records information about filesets, tapes, and dumps of filesets
  - There so restore requests can be traced to tape or set of tapes
- Tape coordinator
  - Accepts requests to dump/restore to/from tape
- Backup coordinator
  - Process that accepts commands
  - Interacts with backup data base and tape coordinator
  - Optimizes save and restore
  - Can work on several backup operations in parallel

BACKUP COORDINATOR

- Backup coordinator performs 3 types of requests:
  - Saving filesets
  - Restoring filesets by date
  - Restoring aggregates or servers
- Saving filesets can be performed on a "schedule"
  - Filesets to be dumped, and days when dumps are done
  - Further automation of backups

SAVING FILESETS

- Receives request for filesets to be saved
- Filesets sorted by server and aggregate
- Tape coordinators are started
  - May start several in parallel
- Backup data base informed continuously about progress
- Tape coordinator uses fileset server to perform appropriate operations

RESTORING FILESETS BY DATE

- Backup coordinator determines where latest fileset dump is (from backup data base)
- Full dump history obtained
- History examined and sequence of dumps that need to be restored is determined
- Sequence of tape operations is created that do restore most efficiently
- Tape controller is requested to perform restored in specific sequence
  - Tape controller uses fileset server as necessary

RESTORING AGGREGATES OR SERVER

- Restore server or partition as of current time
- VLDB queried for set of volumes
- Backup coordinator proceeds as before

BACKUP COMMANDS

- backup adddump
  - Add a list of filesets to dump schedule
- backup addhost
  - Add a tape coordinator host to a configuration
- backup addvolentry
  - Add a new fileset to a fileset set
  - Fileset sets are for convenience of backup
- backup addvolset
  - Create a new fileset set
- backup deletedump
  - Delete a list of filesets from dump schedule

Locus

BACKUP COMMANDS (CONT.)

- backup deletehost
  - Delete a tape coordinator host from configuration
- backup deletevolentry
  - Delete a fileset from a fileset set
- backup dump
  - Start a dump to tape
- backup fullrestore
  - Restore an aggregate
- backup initcmd
  - Initialize the backup coordinator
- backup listdumps
  - List the dump schedules (when filesets are scheduled to be dumped)