HP WORLD 2004
Solutions and Technology Conference & Expo

Introduction to NFS Version 4

Dave Olker
Advanced Technology Center
System Networking and Security Lab

Agenda

- ➢ **NFS Version 4 Design Goals**
- NFS Version 4 Features
  - Improved Access & Internet Performance
  - Strong Security with Negotiation
  - Better Cross-platform Interoperability
  - Designed for Protocol Extensions
- For More Information

NFS Version 4 Design Goals

- Improved **Access** and Good **Performance** on the **Internet**
- Strong **Security** with **Negotiation** Built into the Protocol
- Better Cross-platform **Interoperability**
- Designed for **Protocol Extensions**

Improved Access & Internet Performance

- Easy **transmission** through **Firewalls**
  - Merge **many** protocols into one, all using port **2049**
  - Use of **public** and **root** file handles
- Perform well in **high** latency, **low** bandwidth networks
  - **TCP** or other congestion management protocol required
  - **COMPOUND** procedure used to combine requests
  - **LOOKUP** processes entire path, not just a component
  - **READDIRPLUS** functionality added into **READDIR**
- **Scale** to a large number of clients per server
  - **Client caching** and **Delegation**

Strong Security with Negotiation

- Supports the **RPCSEC_GSS** protocol and **GSS-API**
  - Strong **Authentication**, **Integrity** and **Privacy** options
  - **New security flavors** may be added **without redesigning** the RPC or NFS protocol layers
- Clients and servers **negotiate** supported security flavors on a per file system basis
- **ACL** support is integrated in NFSv4
- Security concerns from **NFSv2/v3** addressed
  - Elimination of the **MOUNT** protocol
  - User and group **strings** used instead of UID and GID

Cross-platform Interoperability

- Feature set **does not favor** one **file system** architecture or **operating system** over another
- Flexible **attributes** and **ACL** support (POSIX and NT)
- Progressive **file** and **file system browsing** using NFS server **namespace**
- **File Locking** Interoperability
- Windows **SHARE** Semantics (i.e. **OPEN** and **CLOSE**)
- **Persistent** and **Volatile** File Handles
- Support for a **Universal Character Set**

Designed For Protocol Extensions

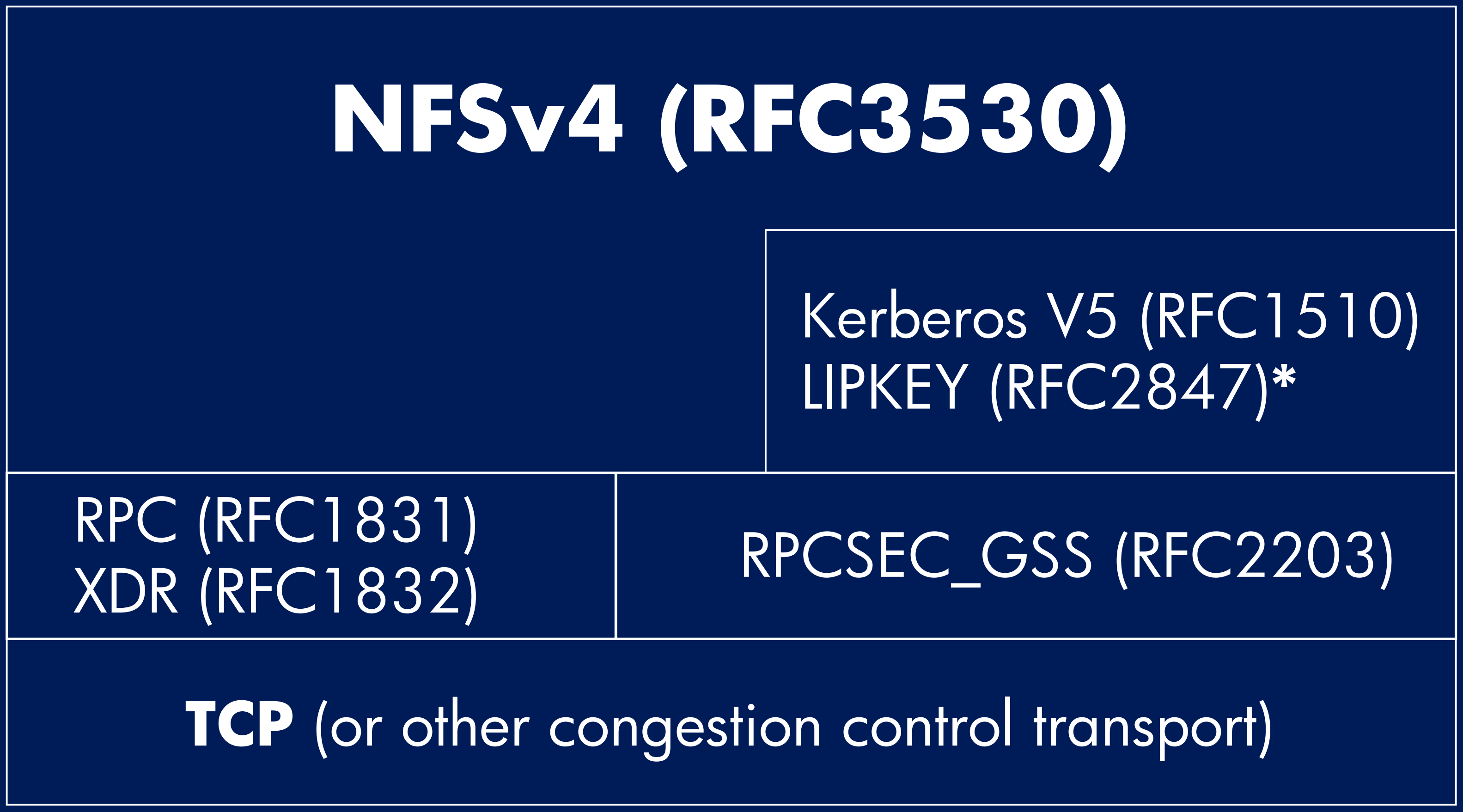
- Protocol supports **minor versions** (i.e. NFSv**4.1**)
- Ability to add **new functionality** into the NFSv4 protocol without compromising **backwards compatibility** with previous minor versions
- NFSv4 protocol was **designed** for **extensibility**
  - **COMPOUND** Procedure
  - **GSS-API** Security Framework
  - **Flexible Attribute** Mechanism

Agenda

✓NFS Version 4 Design Goals

- **NFS Version 4 Features**
  - ➢ **Improved Access & Internet Performance**
  - – Strong Security with Negotiation
  - – Better Cross-platform Interoperability
  - – Designed for Protocol Extensions
- For More Information

NFS Version 4 Protocol "Stack"



NFSv4 (RFC3530)

Kerberos V5 (RFC1510)
LIPKEY (RFC2847)*

RPC (RFC1831)
XDR (RFC1832)

RPCSEC_GSS (RFC2203)

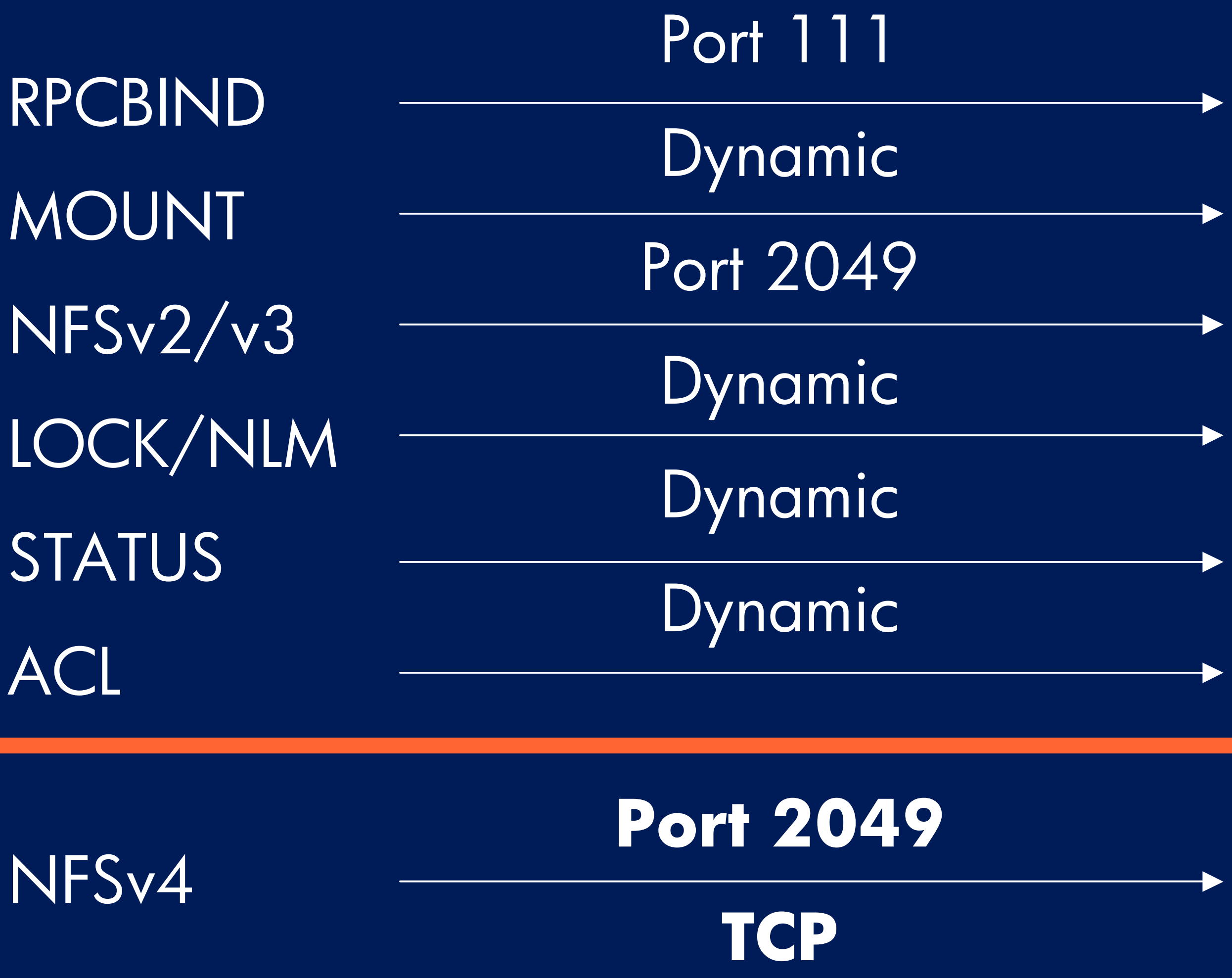
TCP (or other congestion control transport)

* Likely won't be included in initial NFS v4 implementations

Firewall “Friendly”

- Many Protocols **Merged** into NFSv4
  - MOUNT
  - Network Lock Manager
  - Network Status Manager
  - ACL
- Well-known port **2049** used for all NFSv4 traffic
  - Not necessary to contact **rpcbind**
- **TCP** support is **mandatory**
  - Use of a **congestion control** protocol is **required**
- Firewalls only need to allow port **2049/TCP**

Firewall "Friendly" – Merged Protocols



RPCBIND → Port 111

MOUNT → Dynamic

NFSv2/v3 → Port 2049

LOCK/NLM → Dynamic

STATUS → Dynamic

ACL → Dynamic

NFSv4 → **Port 2049** **TCP**

Combined Protocols == More Complex

- NFSv2 – **18** Procedures
- NFSv3 – **22** Procedures
- NFSv4 – **2 Procedures!!**
  - **NULL & COMPOUND**
  - ***36 COMPOUND Operations***

		ACCESS	**CLOSE**
		COMMIT	CREATE
		DELEGPURGE	**DELEGRETURN**
		GETATTR	**GETFH**
		LINK	**LOCK**
		LOCKT	**LOCKU**
LOOKUP	**LOOKUPP**	**NVERIFY**	**OPEN**
OPENATTR	**OPEN_CONFIRM**	**OPEN_DOWNGRADE**	**PUTFH**
PUTPUBFH	**PUTROOTFH**	READ	READDIR
READLINK	**RELEASE_LOCKOWNER**	REMOVE	RENAME
RESTOREFH	**SAVEFH**	**SECINFO**	SETATTR
SETCLIENTID	**SETCLIENTID_CONFIRM**	**VERIFY**	WRITE

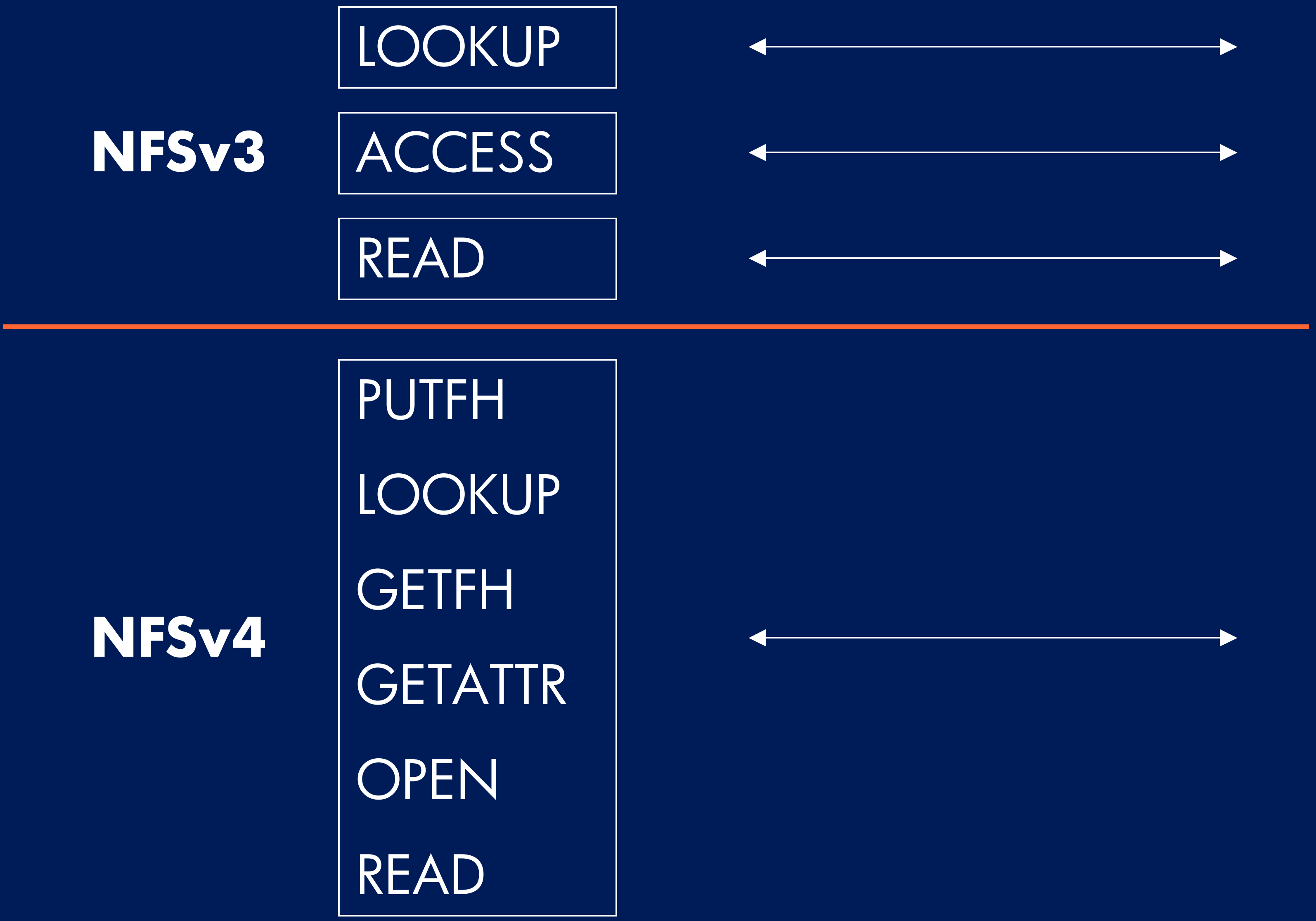
COMPOUND Procedure – Overview

- NFS Version 2/3 use **“traditional”** RPC
  - Client issues a single procedure call to the server
  - Server sends back a reply to the single procedure
- NFSv4 uses **COMPOUND** to build **sequences** of **related** operations into a **single** RPC
- The server evaluates each operation contained in the COMPOUND request **in order** until **completion** or an **error** occurs; then returns a reply for **all** operations
- **Latency** is **greatly reduced** with fewer over-the-wire roundtrips between client and server systems
- Decreases **transport** and **security overhead**

COMPOUND Procedure – Continued

- This design helps make the PV4 protocol **Extensible** – it is easier to add new COMPOUND **Operations** than entire new **Procedures**
- A single COMPOUND call may contain **any number** of operations – depends on the sophistication of the NFS **client** implementation
- Client COMPOUND **Implementation Dilemma**: *How long do you wait to build a COMPOUND call?*
  - Example: “ls -l” – The *stat()* call for each file arrives at the VFS layer independently. *How long should the client’s kernel wait before building the COMPOUND call?*

COMPOUND vs. “Traditional” RPC – READ



NFSv3

LOOKUP

ACCESS

READ

NFSv4

PUTFH
LOOKUP
GETFH
GETATTR
OPEN
READ

COMPOUND – Example of Call Packet

```
Network File System
  Program Version: 4
  Procedure: COMPOUND (1)
  Tag: do_lookup()
    length: 11
    contents: do_lookup()
    fill bytes: opaque data
  minorversion: 0
  Operations (count: 4)
    Opcode: PUTFH (22)
      filehandle
        length: 12
        hash: 0x498a1402
        type: unknown
        data: 0000000113000800B3C51200
    Opcode: LOOKUP (15)
      Filename: allfiles
        length: 8
        contents: allfiles
```

```
    Opcode: GETATTR (9)
          attrmask
            mand_attr: FATTR4_TYPE (1)
            mand_attr: FATTR4_CHANGE (3)
            mand_attr: FATTR4_SIZE (4)
            mand_attr: FATTR4_FSID (8)
            recc_attr: FATTR4_FILEID (20)
            recc_attr: FATTR4_MODE (33)
            recc_attr: FATTR4_NUMLINKS (35)
            recc_attr: FATTR4_OWNER (36)
            recc_attr: FATTR4_OWNER_GROUP (37)
            recc_attr: FATTR4_RAWDEV (41)
            recc_attr: FATTR4_TIME_ACCESS (47)
            recc_attr: FATTR4_TIME_MODIFY (53)
        Opcode: GETFH (10)
```

COMPOUND – Example of Reply Packet

```
Network File System
    Program Version: 4
    Procedure: COMPOUND (1)
    Status: NFS4_OK (0)
    Tag: do_lookup()
        length: 11
        contents: do_lookup()
        fill bytes: opaque data
    Operations (count: 4)
        Opcode: PUTFH (22)
            Status: NFS4_OK (0)
        Opcode: LOOKUP (15)
            Status: NFS4_OK (0)
        Opcode: GETATTR (9)
            Status: NFS4_OK (0)
            obj_attributes
                attrmask
                    mand_attr: FATTR4_TYPE (1)
                        nfs_ftype4: NF4REG (1)
                    mand_attr: FATTR4_CHANGE (3)
                        changeid: 3948251870035276
                    mand_attr: FATTR4_SIZE (4)
                        size: 9192740
…
            attr_vals: <DATA>
                length: 100
                contents: <DATA>
        Opcode: GETFH (10)
            Status: NFS4_OK (0)
            Filehandle
                length: 24
                hash: 0x7096825a
                type: unknown
                data:
 0200000113000800B3C5120064C61200
               CA932D2CB3C51200
```

Client-side Caching

- NFSv4 clients cache the same data as previous clients
  - **File Data** is cached in a memory-based cache (i.e. UFC)
  - **Directory** data is cached in a READDIR cache
  - File and directory **Attributes** are held in attribute cache
- **Client** determines the validity **duration** of the **attribute** and **directory** caches
- Client checks **file data** cache validity at file **OPEN**
  - Sends query to the server to see if the file has **changed**
  - Determines if the cached data should **kept** or **released**
  - **Modified** data is written to the server at file **CLOSE**

Delegation – Overview

- Similar to **OPLOCKS** used by CIFS/Samba servers
  - The **client** system requests an **OPLOCK**
  - The **server** decides when to grant **delegation** to clients
- Server may provide delegation in response to **OPEN**
- Delegation support is **not required** for correct protocol operation (i.e. *optional* feature)
- Delegations are the **only** time when a server **initiates** contact with the client (when **recalling** a delegation)
- A **valid callback path** to the client must exist

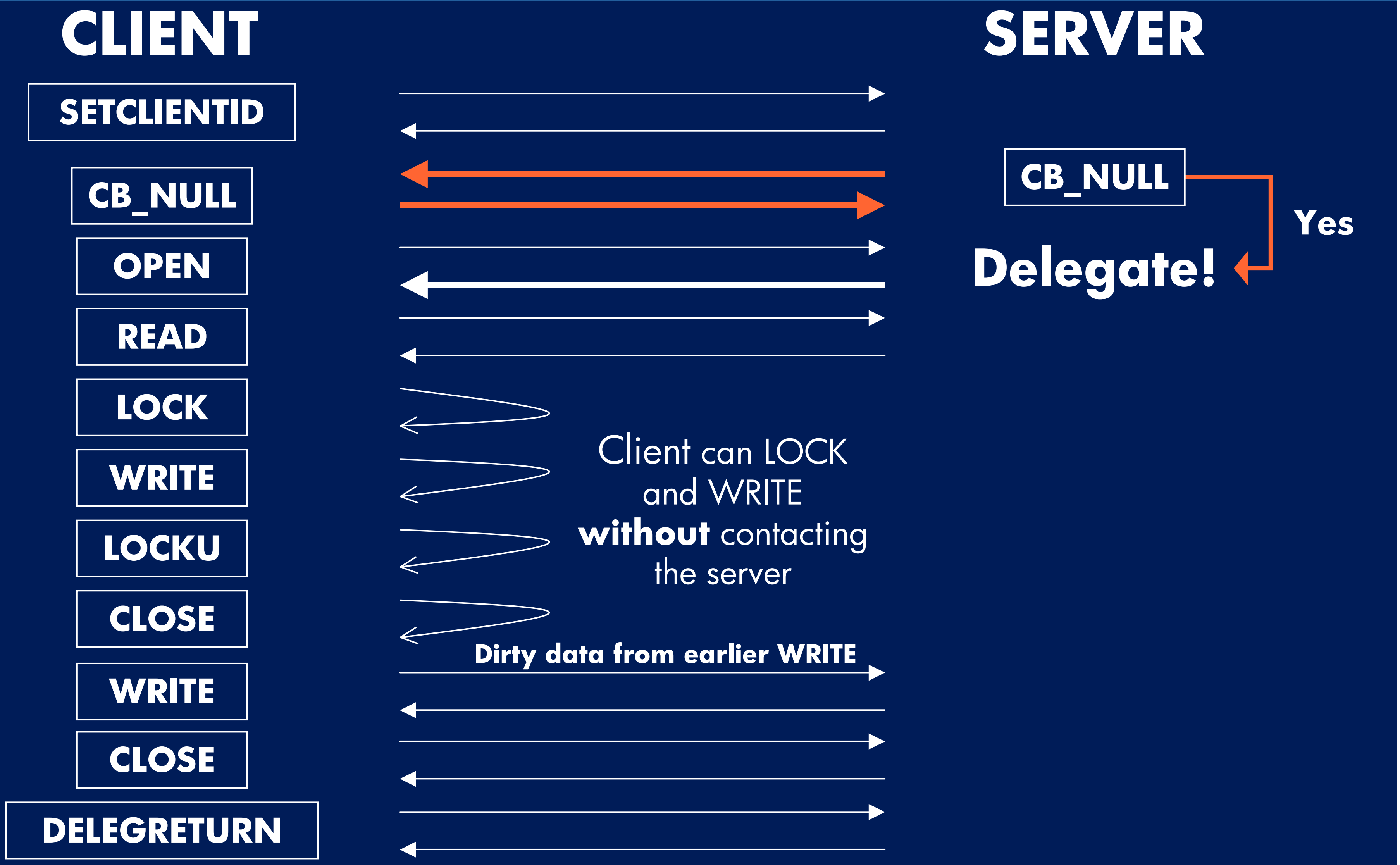
Delegation – Continued

- The server **confirms** the callback path to the client before delegating (i.e. server behind a **firewall**?)
- If delegation is provided, the client does not need to contact server for further **OPEN, LOCK, READ, WRITE,** and **CLOSE** operations for the file
- The server will **recall** a file delegation if a different client process sends an **OPEN** for the file with a **conflicting** mode (i.e. **READ** vs. **WRITE**)
- Multiple client applications may **share** a delegation providing they return **all state** to the server if revoked

Delegation – Continued

- The server **may** grant **multiple read-only** delegations for the same file to **different** clients
- Delegations are **lease-based**, just like file locking
- NFSv4 initially supports delegation of **files** – **directory delegation** is being considered for future

Delegation – Example



CLIENT

SERVER

SETCLIENTID

CB_NULL

OPEN

READ

LOCK

WRITE

LOCKU

CLOSE

WRITE

CLOSE

DELEGRETURN

CB_NULL

Yes

Delegate!

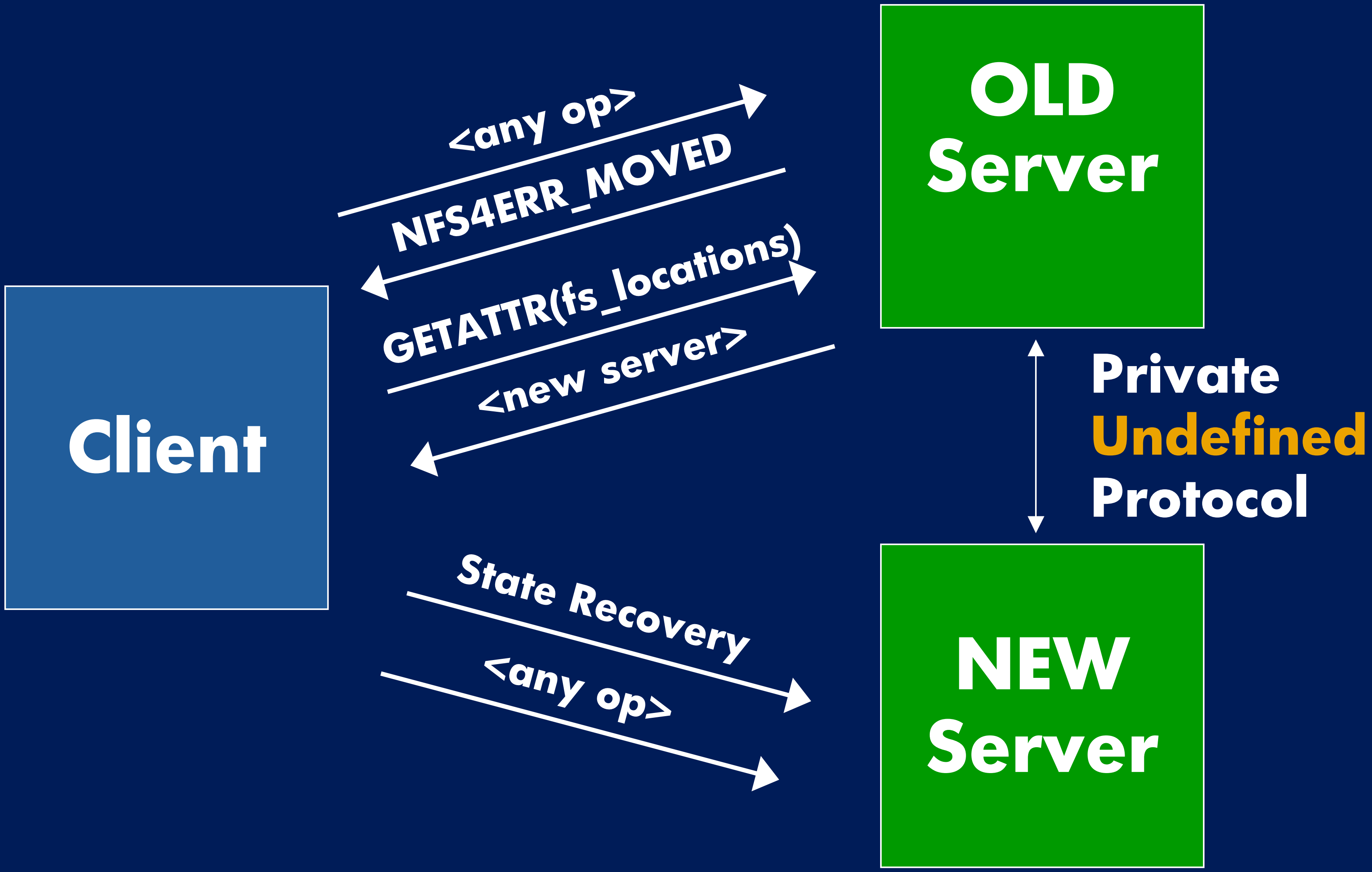
Client can LOCK and WRITE **without** contacting the server

Dirty data from earlier WRITE

File System Migration

- Enables **load balancing**, increased **availability** or server **reorganization**
- Client receives **NFS4ERR_MOVED** at migration event
- ***fs_locations*** attribute provides new location
- Server-to-server transfer mechanism is **undefined**
  - Possibly will be addressed in a **minor version**

File System Migration Event



File System Replication

- Only available for **read-only** file systems
- Increases **availability** of file system resources
- **Client's policy** determines when to switch to another replica server (i.e. after a certain number of tries)
  - Similar to **client-side failover** provided with ONC 2.3
- ***fs_locations*** attribute enumerates available replicas
- **Client** needs to **reconstruct** state at new server
- Server-to-server replication mechanism is **undefined**
  - Possibly will be addressed in a **minor version**

Agenda

✓NFS Version 4 Design Goals

- **NFS Version 4 Features**
  - ✓Improved Access & Internet Performance
  - ➢ **Strong Security with Negotiation**
  - – Better Cross-platform Interoperability
  - – Designed for Protocol Extensions
- For More Information

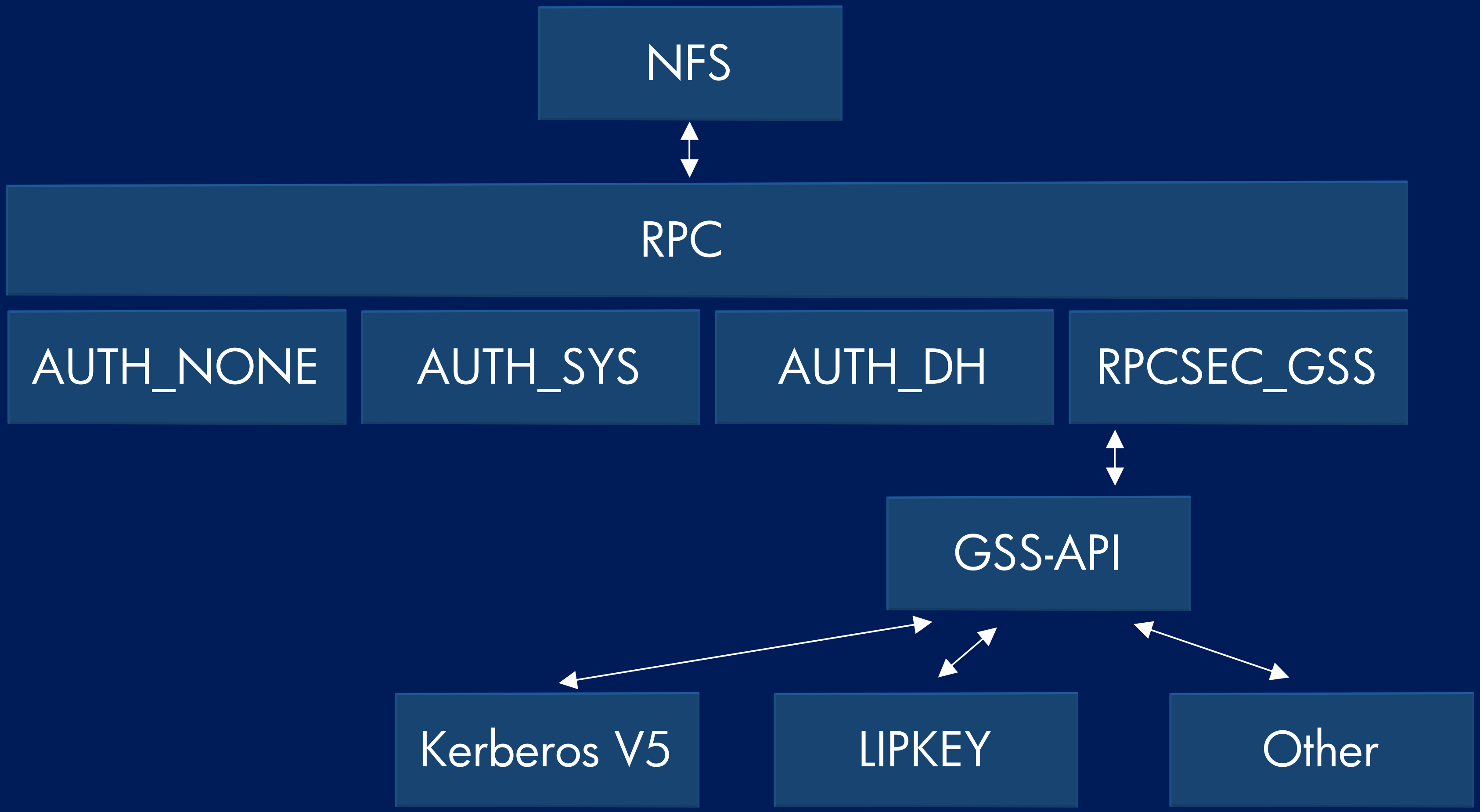
Security – Overview

- A strong security model was **mandated** by the **IETF**
- The **RPCSEC_GSS / GSS-API** framework allows NFSv4 to use various security mechanisms for **Authentication, Integrity,** and **Privacy**
- Security mechanisms are **negotiated** between systems on a per file system basis
- **ACL** support is included in the NFSv4 protocol
- **Character strings** used instead of UID and GID
  - Example: **user@domain** or **group@domain**
- Removal of the **MOUNT** protocol

Security – RPCSEC_GSS / GSS-API

- The **RPCSEC_GSS** security flavor allows RPC protocols to access the Generic Security Services Application Programming Interface (**GSS-API**)
- Supports **multiple** underlying security mechanisms and provides access to their services though a common API
  - **Kerberos V5**
  - **LIPKEY** (Low Infrastructure Public Key – similar to SSL)
  - Others
- **Extensible** – new security flavors can be added, provided they conform to the GSS-API model, **without** requiring NFS to be redesigned

Security – GSS-API Stack



NFS

RPC

AUTH_NONE

AUTH_SYS

AUTH_DH

RPCSEC_GSS

GSS-API

Kerberos V5

LIPKEY

Other

Using the GSS-API allows for the use of varying security mechanisms by the RPC layer without the additional implementation overhead of adding RPC security flavors.

Security – Negotiation

- The server may specify a **different** security policy for **each file system**
- File system can exported with **multiple** security flavors
  - **AUTH_SYS** for Read-Only access to one set of clients
  - **RPCSEC_GSS** for R/W access to another set of clients
- Clients access root file handle via a **secure channel**
- If the client tries to use an *unsupported* security flavor, the server returns an **NFS4ERR_WRONGSEC** error
- Client can use **SECINFO** to enumerate the mechanisms **supported** by the server for the specific **file system**

Security – ACL Support Built into NFSv4

- ACL support has been tried for **NFSv2** and **NFSv3**
  - At last count there were more than **4** different ACL implementations, none of which **interoperate** well
- NFSv4 **standardizes** the ACL mechanism to guarantee **interoperability**
- ACLs may be **manipulated** from the **client**
- **Windows/NT** ACL compatible
  - **Does not** mean **all NT ACLs** are supported by NFSv4 clients and servers – **implementation specific** support
- Combined with RPC security, ACLs provide a **strong security** mechanism

Agenda

✓NFS Version 4 Design Goals

- **NFS Version 4 Features**
  - ✓ Improved Access & Internet Performance
  - ✓ Strong Security with Negotiation
  - ➢ **Better Cross-platform Interoperability**
  - – Designed for Protocol Extensions
- For More Information

File Handle Types – Overview

- **Persistent** file handles behave as with **NFSv2/v3**
  - Valid for the **lifetime** of the object
  - Survives **server reboots** or file system **migrations**
  - **Security** concern – can be **sniffed** and **spoofed**
- **Volatile** file handles are **new for NFSv4**
  - Provided for servers that **cannot** implement **Persistent**
  - File handle may become **invalid** and **expire**
  - **Client** needs to know how to handle **both types**, **server** is **not required** to provide both types
  - **New file attribute** informs the client which file handle type is being used by the server – **Persistent** or **Volatile**

File Handle Types – Continued

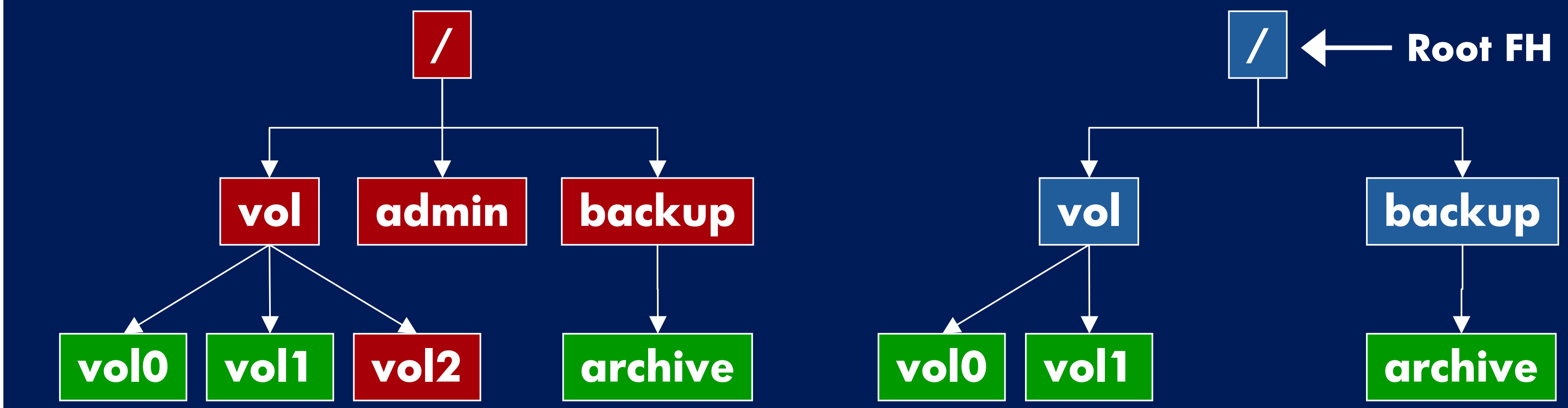
- **Root** file handle represents the conceptual root of the file system namespace on an NFS server
  - Client uses the **PUTROOTFH** operation to set the "current" file handle to the **root** of the server's file tree
  - Client can then traverse the entirety of the server's file tree with the **LOOKUP** operation
- **Public** file handle is used to bind or represent an **arbitrary** file system object on the server
  - **Server** decides what the public file handle references
  - May refer to the **ROOT** file handle, but **not necessarily**
  - **Client** cannot make assumptions about **public** location

Namespace – Overview

- Replaces the use of the **MOUNT** protocol
- The server provides access to its exported file systems from a single point called the **ROOT file handle**
- The server can designate a **Public file handle**, which may be **different** from the ROOT file handle
- The server constructs a **“pseudo file system”** or *logical* representation of its exported file systems
- The clients are able to browse the hierarchy of exported file systems by traversing the “pseudo file system” with **LOOKUP** calls

Namespace – Server's Pseudo File System

Local File System | **Pseudo** File System



Color Key: 🟥 **NOT Exported** 🟩 **Exported** 🟦 **Empty – R/O**

In this example, the server is exporting **/vol/vol0, /vol/vol1**, and **/backup/archive**
The server *does not* want clients to see /, **/vol**, **/vol/vol2**, **/admin**, or **/backup**

Namespace – Client's Perspective

- The client starts at the server's **ROOT file handle**, which is the **top** of the server's **pseudo** file system
- The client inspects the ***fsid*** attribute of each directory to determine when it encounters a **new file system** in the server's exported hierarchy
- At each new file system, the client **automatically mounts** the file system into its namespace
- The client's view of the pseudo file system is **limited** to those paths that lead to **exported** file systems

Lease Management

- A **lease** is a *time-bounded* **grant of control** of the **state** of a file, through **lock** or **delegation**
- **Server** determines the **duration** of the lease period
- **Client** is responsible for contacting the server to **refresh** the lease
- Lease management adds to the **complexity** of both the client's and server's implementation
  - Client has to deal with **lease timeouts** and **renewals**
  - Server has to decide whether to **keep** or **release** the client's **state information** at lease timeout

Locking – Overview

- **NFSv4 is STATEFUL**
  - **NFSv2/v3** are **stateless**, but **NLM** and **NSM** protocols did monitor state of locking participants
  - Integrating **locking** support into NFSv4 requires state
- **Superior** implementation as a result of **integration**
- **Byte-range** and **mandatory** locks are supported
- Compatible with **Windows SHARE** locking
- **Local** locking if **WRITE Delegation** is granted
- Stateful **OPEN** and **CLOSE** needed to support **SHARE** locking, **Exclusive Creates**, and **Delegation**

Locking – Lease Management (part 1)

- Locks are controlled by **leases** that need to be **RENEWED** at lease **expiration**
  - Avoids situation where a client locks files and **crashes without releasing** its locks on the server
- ***stateid*** represents the current locking state of a file
  - Used by **client** and **server** to maintain lock state
- The server's receipt of a valid ***stateid*** is a positive acknowledgement that **all locks** held by the sending client are still **valid** (i.e. implicit lease renewal)
- A refresh of **any** lock **validates all locks** held by the client to a particular server

Locking – Lease Management (part 2)

- Client **recovers** its locks when the server sends it a **lease expiration** notice or if the **server restarts**
- When the **server restarts** it a waits duration equal to a **lease interval** for all clients to **reclaim** their locks
- Lease renewal occurs at explicit **RENEW** or by operations that use ***stateid***
  - **CLOSE, DELEGRETURN, LOCK, LOCKU, OPEN, OPEN_CONFIRM, READ, RENEW, SETATTR, WRITE**
- ***sequence-id*** preserves request ordering
  - No more **out-of-order** lock request problems

Locking – Client ID and Lock Recovery

- Client issues the **SETCLIENTID** operation along with a unique value called a **verifier**
  - The **verifier** is similar to the WRITE verifier used for safe asynchronous writing – a unique value set at **boot time**
- Server responds to SETCLIENTID with a unique ***clientid***
- Server assigns new ***clientid*** following a client reboot
  - After client reboot it contacts server to obtain a ***clientid***
  - When the server sees the same client identity information with a **new verifier** (i.e. new **client boot time**), the server knows the client rebooted and will **free** its old locks

Attributes – Overview

- **Mandatory Attributes**
  - **MUST** be supported by every NFSv4 client and server in order to ensure a minimum level of interoperability
- **Recommended Attributes**
  - The attributes are understood well enough to warrant support in the NFSv4 protocol, but **may not be supported** by all clients and servers
- **Named (Extended) Attributes**
  - Allows a client to name, store and retrieve **arbitrary** data and associate it as an attribute of a file or directory

Attributes – Continued

- Extends beyond **traditional UNIX** attributes
- **52 Mandatory** and **Recommended** Attributes
  - Examples include: type, size, fsid, link?, symlink?, etc.
- **Named** (Extended) attribute support is **optional**
  - Associated with each file system object is a **hidden directory** containing all its **named attributes**
  - **OPENATTR** returns named attributes directory file handle
  - Intended for **application-specific** use
  - Named attribute data is **not interpreted by NFS**
  - Feature of file systems like **Windows NTFS**

Internationalization – UTF-8 Encoding

- All strings used for **file, directory** and **symbolic link** contents are encoded using **UTF-8**
- UTF-8 is a **Universal Character SET** (UCS)
- Client can determine what **language** a filename was created in and how to properly **display** it
- Supports mapping of **8** and **16** bit characters
- Supports **direct** mapping of previously stored objects
  - NFSv2/v3 use 7-bit **US ASCII** and 8-bit **ISO Latin 1**
- **Efficient** encoding for wire transfers
- Can expand **beyond** characters longer than 31-bits

Agenda

✓NFS Version 4 Design Goals

- **NFS Version 4 Features**
  - ✓ Improved Access & Internet Performance
  - ✓ Strong Security with Negotiation
  - ✓ Better Cross-platform Interoperability
  - ➢ **Designed for Protocol Extensions**
- For More Information

Protocol Extensions & Minor Versioning

- **Difficult** to revise previous versions of NFS (**v2 → v3**)
- Realize that protocol is **not perfect** and must **evolve**
- Examples of **Potential** Minor Version Features
  - **New** COMPOUND **Operations**
  - **New** Mandatory or Required **Attributes**
  - **SECINFO** Fixes
  - **Directory** Delegations
  - Support for **RDMA**
  - Server to Server **Replication** Mechanisms
  - **Global** File System **Namespace** Architecture

Agenda

- ✓NFS Version 4 Design Goals
- ✓NFS Version 4 Features
  - ✓Improved Access & Internet Performance
  - ✓Strong Security with Negotiation
  - ✓Better Cross-platform Interoperability
  - ✓Designed for Protocol Extensions
- ➢**For More Information**

To Learn More about NFS Version 4

- IETF now owns the NFS protocol (http://ietf.org)
  - Several RFCs available including RFC3530 (NFSv4)
- NFS Version 4 Protocol White Paper
  - http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf
- NFS Mailing List Archives
  - http://playground.sun.com/pub/nfsv4
  - https://www1.ietf.org/mail-archive/working-groups/nfsv4/current/maillist.html
- Connectation Conference (http://www.connectathon.org)
- U of M Linux Port (http://www.citi.umich.edu/projects/nfsv4)
- NFS Industry Conference (http://www.nfsconf.com)
- NFS Version 4 Homepage (http://nfsv4.org)

HP WORLD 2004

Solutions and Technology Conference & Expo

Co-produced by:

interex

shared knowledge • shared power

encompass

AN HP USER GROUP

RECOMMENDED TRAINING VENUE FOR THE

HP Certified Professional