NFS Version 4

Spencer Shepler
spencer.shepler@sun.com

Sun microsystems
We make the net work.

NFSv4

- History of Protocol Development
- Protocol Description
- Existing Implementations
- Performance
- Future IETF Work

When did this all start?

- First revision, NFSv2, was published in 1985
  - It exports basic POSIX 32-bit filesystems
  - Slow, particularly for writing files
- NFSv3, was published in 1994
  - Extended to 64-bit files & improved write caching
  - The most commonly used protocol for sharing files on *NIX/ Linux LANs today
- NFSv4
  - Secure, firewall friendly, performance
  - Extensible
  - IETF standard published in 2003

IETF and NFSv4

- IETF BOF - Fall 1996
- IETF WG formed - Spring 1998
- Design considerations - June 1999
- RFC3010 - December 2000
- RFC3510 - April 2003
- Minor version work continues

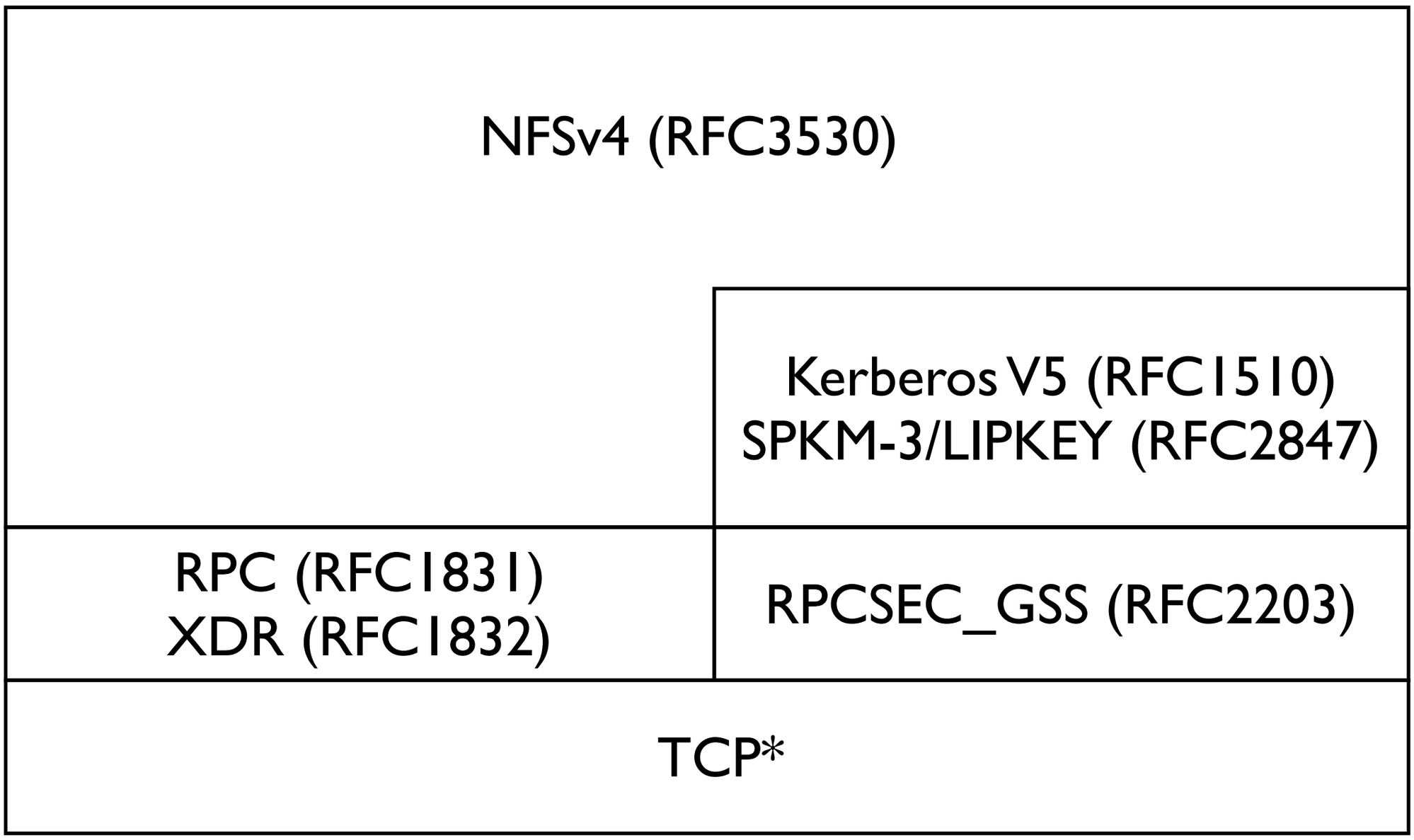
NFSv4 - Design Considerations

- Improve access and good performance on the Internet
- Strong security with negotiation built into the protocol
- Better cross-platform interoperability
- Designed for protocol extensions

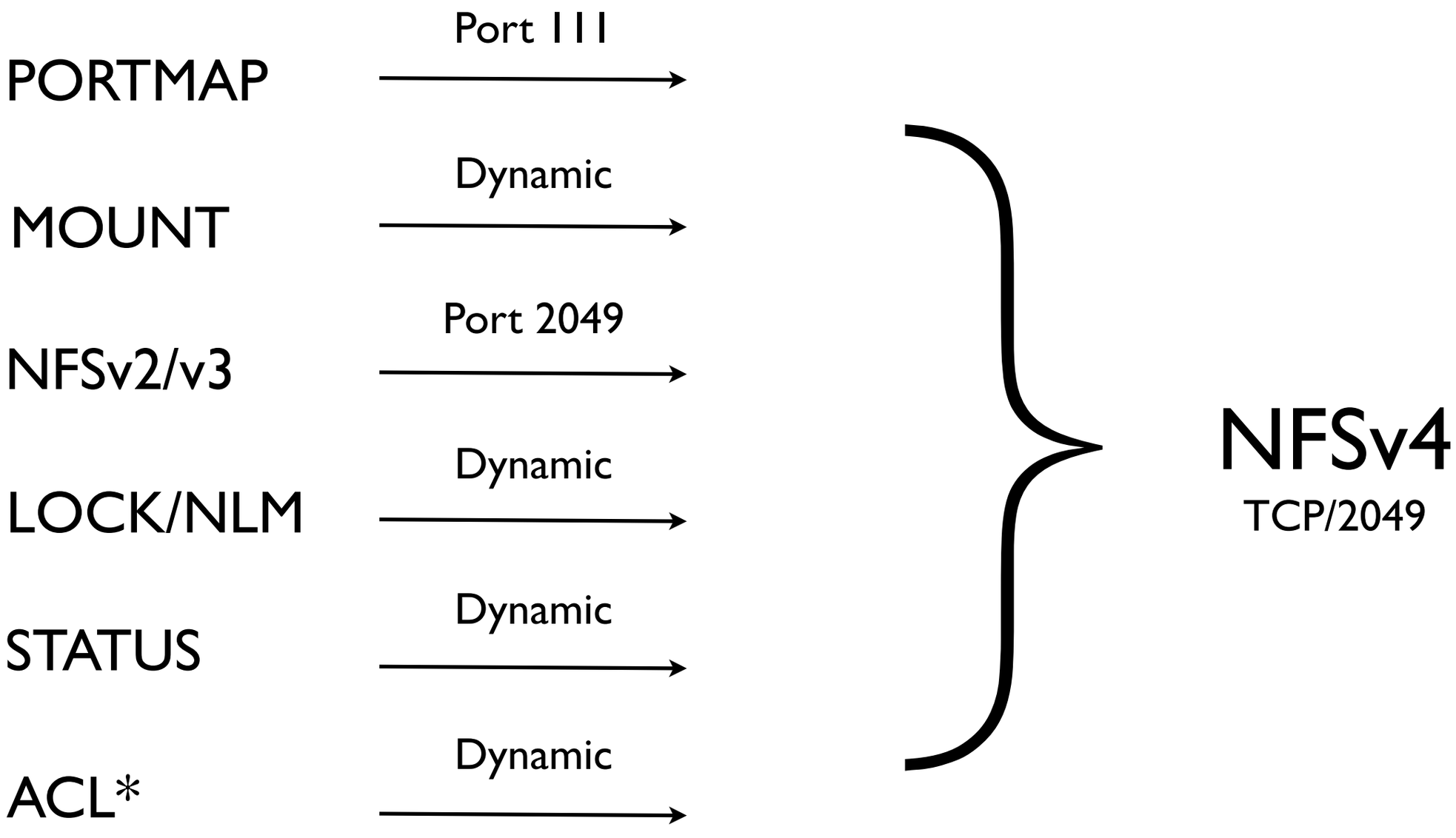
NFSv4 Strengths

- Openly defined distributed filesystem protocol!
- Moving forward, this is THE strength of NFSv4
- IETF process must continue to work well for NFSv4's minor versions to deliver needed functionality
- WG must drive to completion

NFSv4 Protocol "Stack"



Many To One



Operation Comparison

- NFSv2 - 18 operations
- NFSv3 - 22 operations
- NFSv4 - 38 operations
- v2/v3 use “traditional” RPC
- v4 COMPOUND procedure used to build operation sequences

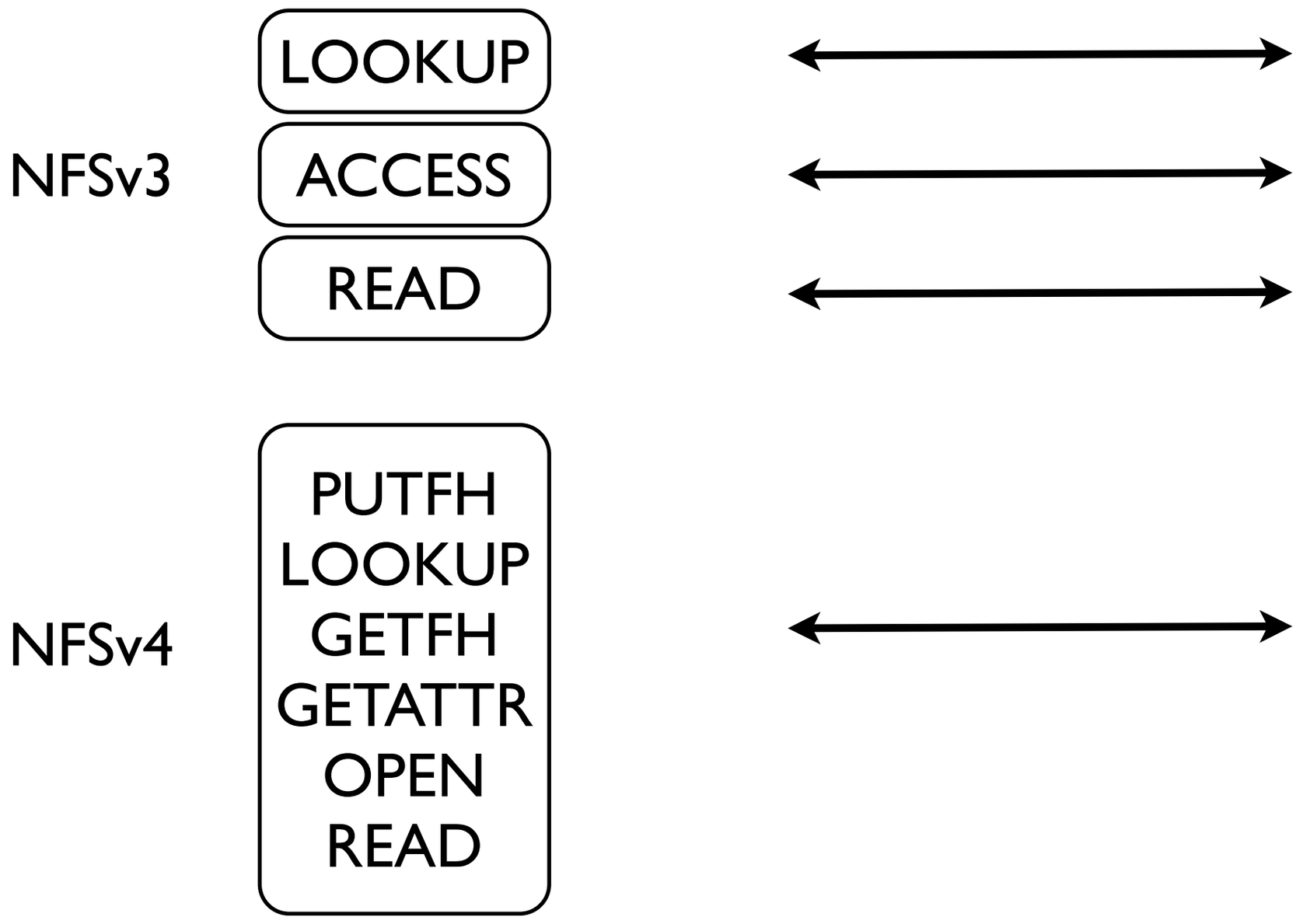
NFSv4 Operations

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
REMOVE
RENAME
RESTOREFH
SAVEFH
SECINFO
SETATTR
SETCLIENTID
SETCLIENTID_CONFIRM
VERIFY
WRITE
RELEASE_LOCKOWNER
CB_GETATTR
CB_RECALL

COMPOUND Procedure

- Group related operations in one RPC
- Evaluation stops at first error
- Reduce latency with fewer roundtrips
- Flexibility for client
- *Easy integration of new functionality

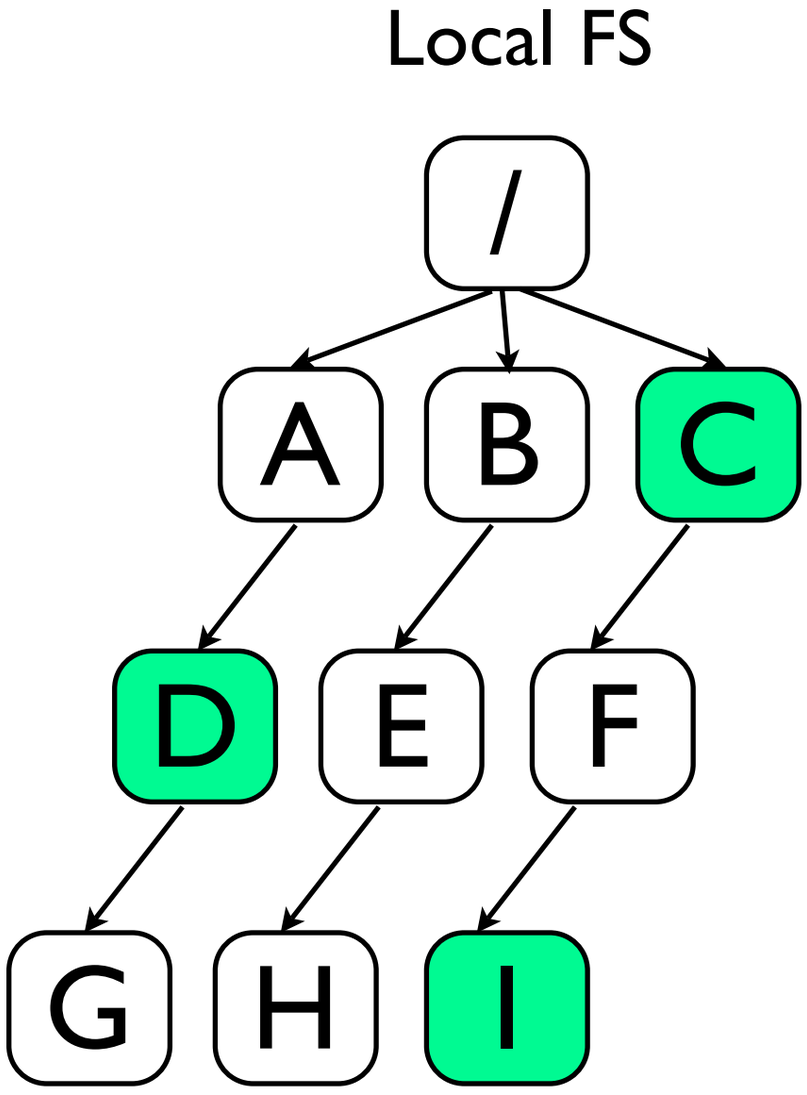
COMPOUND in Action



Namespace

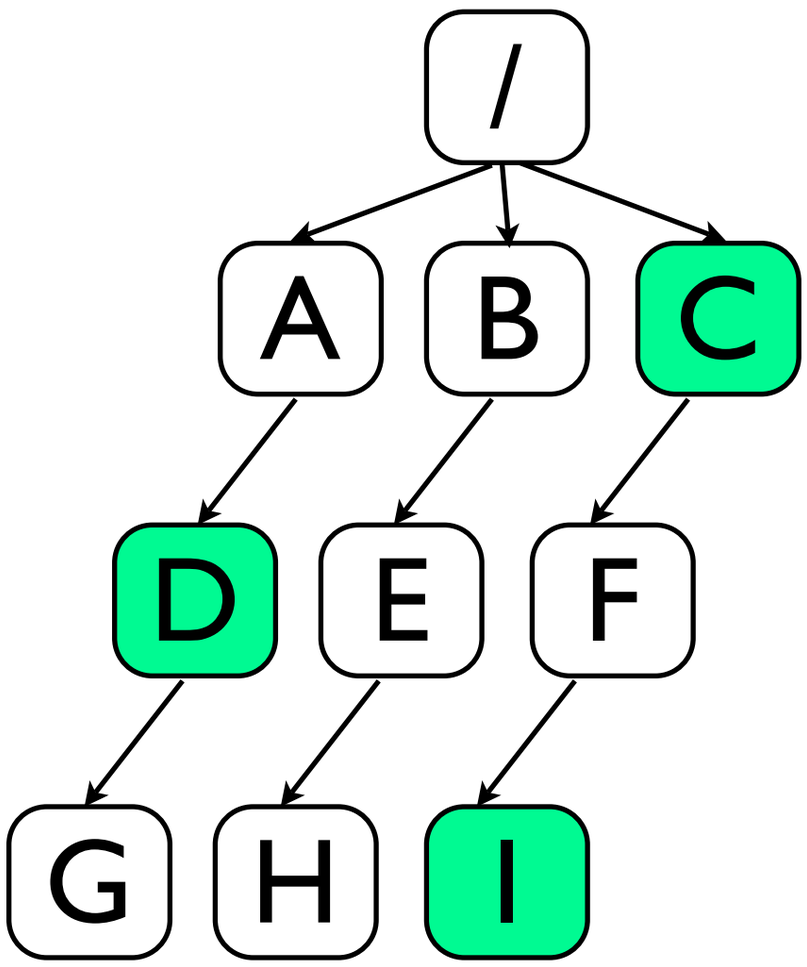
- Replaces use of MOUNT protocol
- Server provides access to filesystems from a "root filehandle"
- Server *pseudofs* joins exported subtrees with a read-only virtual filesystem
- Client traverses *pseudofs* with PUTROOTFH, LOOKUP, READDIR

Building *Pseudofs*

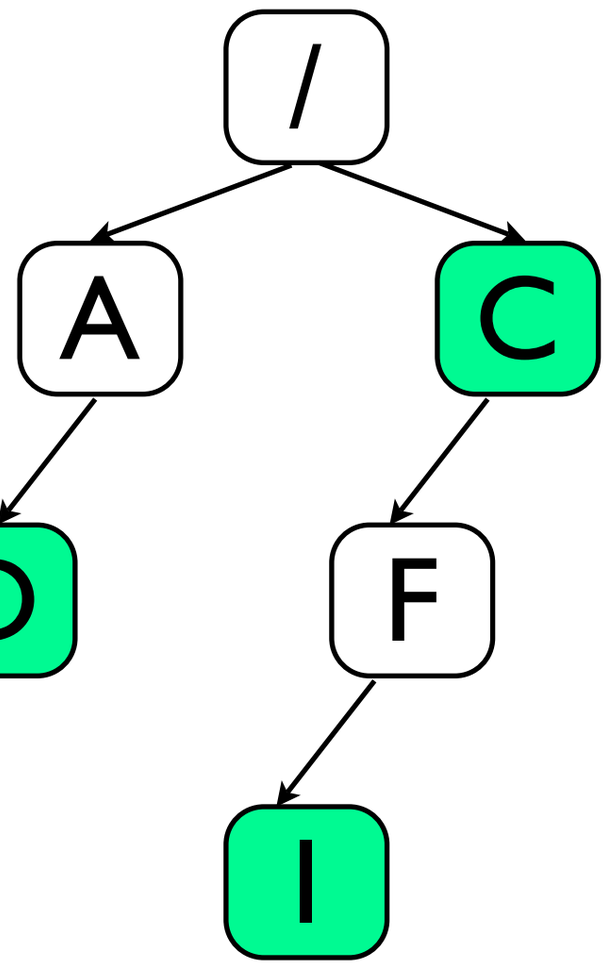


Building *Pseudofs*

Local FS



PseudoFS



Client Namespace

- Starts at root filehandle of server
- Inspects *fsid* attribute to determine when new filesystem is found
- At each new filesystem, client automatically mounts filesystem into its namespace

Security

- RPCSEC_GSS framework is basis for various security mechanisms
- Provides for Authentication, Integrity, and Privacy
- Kerberos V5 and SPKM/LIPKEY

Security Negotiation

- Policy set at each filesystem
- Access root filehandle with secure channel
- If client mismatches on security mechanism, server returns an NFS4ERR_WRONGSEC error
- Client will use SECINFO operation to enumerate available mechanisms

Filehandle Types

- Shorthand reference to file
- Persistent filehandles same as NFSv2/v3
- Volatile filehandles are new
- Filesystem types like FAT or user-level server implementations are examples of use
- Filehandle may become invalid and *expire*
- Different than traditional ESTALE
- Expiration at server restart
- Increases implementation burden on client

Volatile Filehandles

- Upon expiration, client attempts recovery
- At initial LOOKUP, client saves path of file
- Pathname used at recovery; client traverses pathname to find new filehandle
- Other expiration event may include RENAME or migration of filesystem

NFSv4 and its *"State"*

- Hierarchy of NFSv4 state begins with association between a single client and server
- This hierarchy is important because the LEASE period and recovery represents all client-server state

Lease Management

- Lease timeouts used to manage recovery
- Server determines lease period
- Period is for all state generated by client
- Lease renewal occurs at explicit RENEW or by any operation that uses *stateid*
  - CLOSE, DELEGRETURN, LOCK, LOCKU, OPEN, OPEN_CONFIRM, READ, RENEW, SETATTR, WRITE

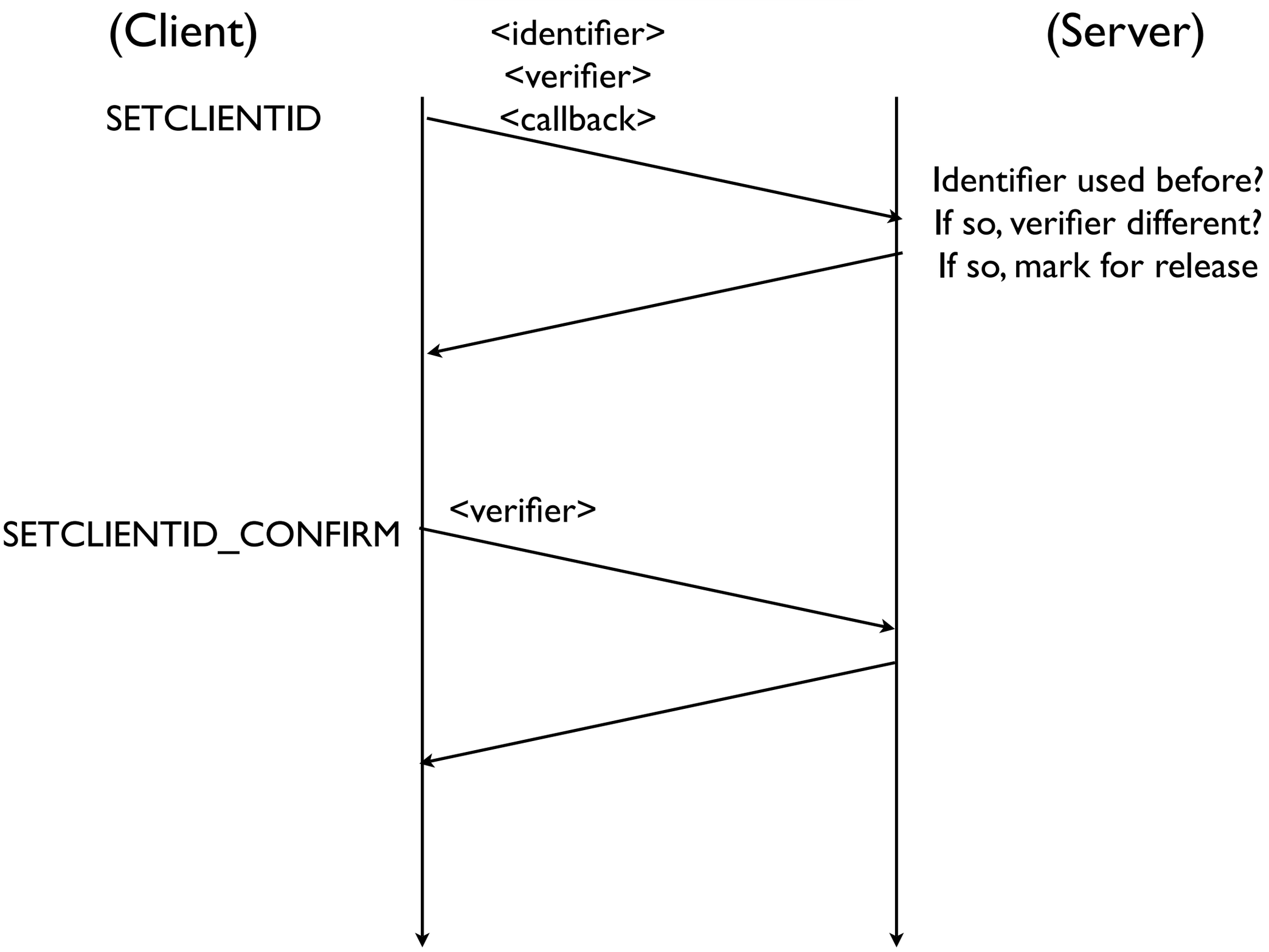
Lease Management Cont

- Lease timeouts adds to complexity of implementation
- Client tracks implicit lease renewal
- Must be prepared for network partitions and various forms of recovery
- Server has options to help in the event of network partition
  - Client's state may be released at lease timeout (or extended)
  - MUST be released if lease has expired and there is a conflict with existing state

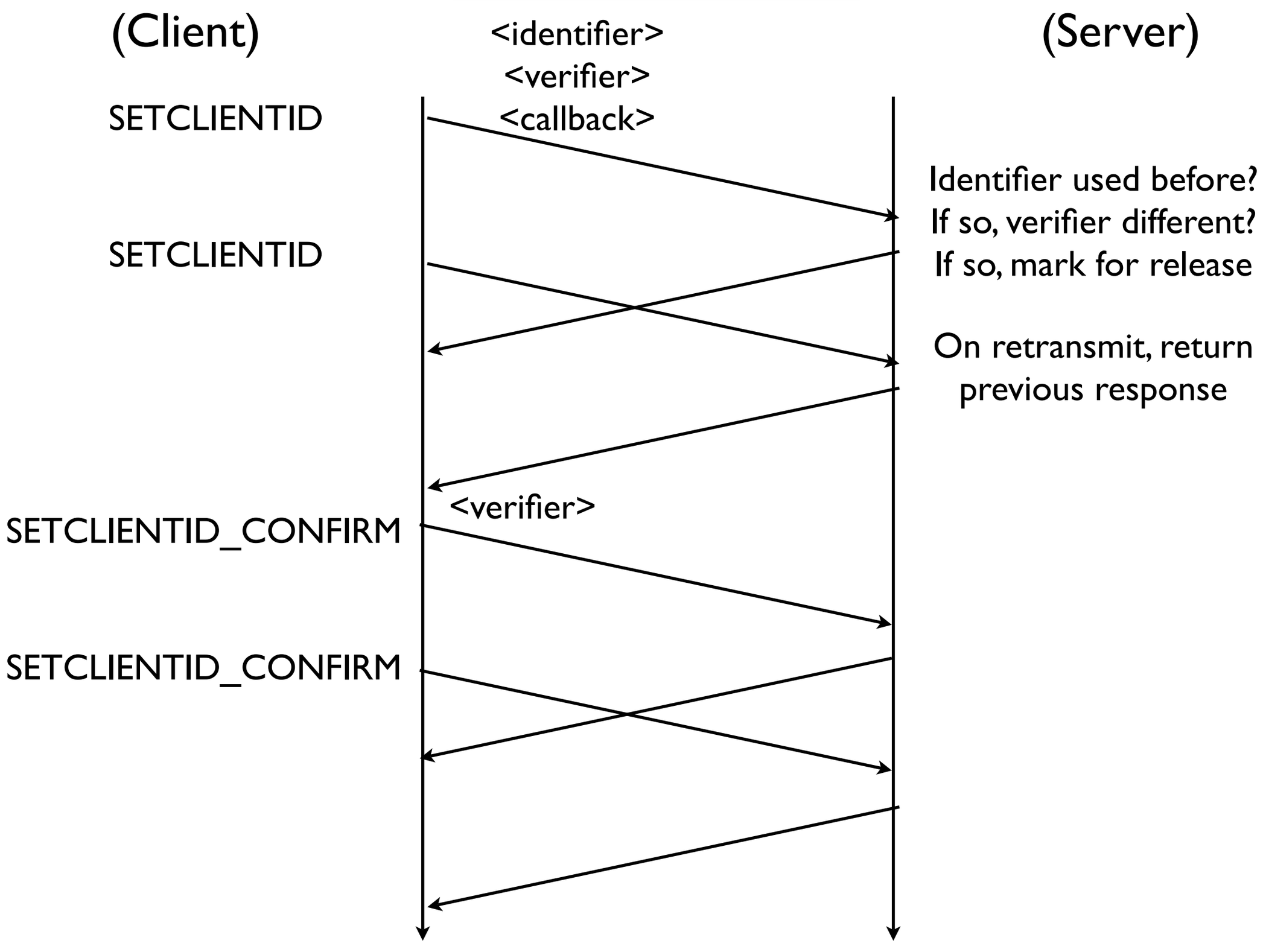
Creation of CLIENTID

- With SETCLIENTID the client chooses an opaque identifier and a verifier
- Uniquely identifies client and client instance (reboot detection)
- Server assigns a *shorthand* CLIENTID
- Client confirms use with SETCLIENTID_CONFIRM
- Server uses RPC authentication to verify requests (saves principal for future reference)

CLIENTID creation



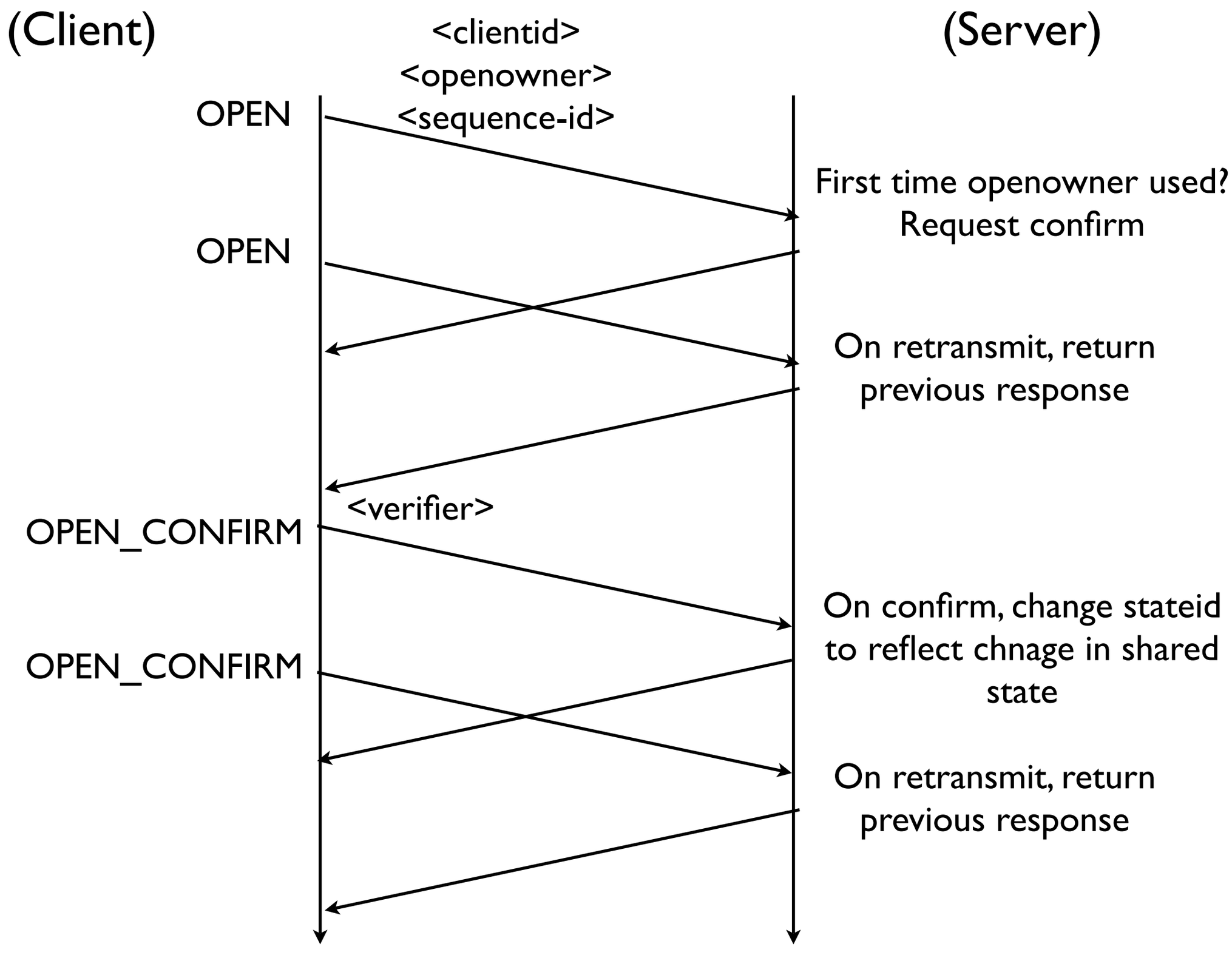
CLIENTID creation



OPEN and its state

- Combines regular file CREATE, LOOKUP, and share reservations
- Requires name of regular file object
  - CREATE operation is for non-file object types
- Identifies owner for OPEN and LOCK state
- Server may provide delegation* in response
- Returns stateid which is used for other operations

Openowner to openstateid



Stateid usage

- Shorthand referring to original OPEN, Delegation, of LOCK
- Operations that use stateid
  - LOCK / LOCKU
  - READ / WRITE
  - SETATTR
  - DELEGRETURN
  - OPEN_CONFIRM / CLOSE

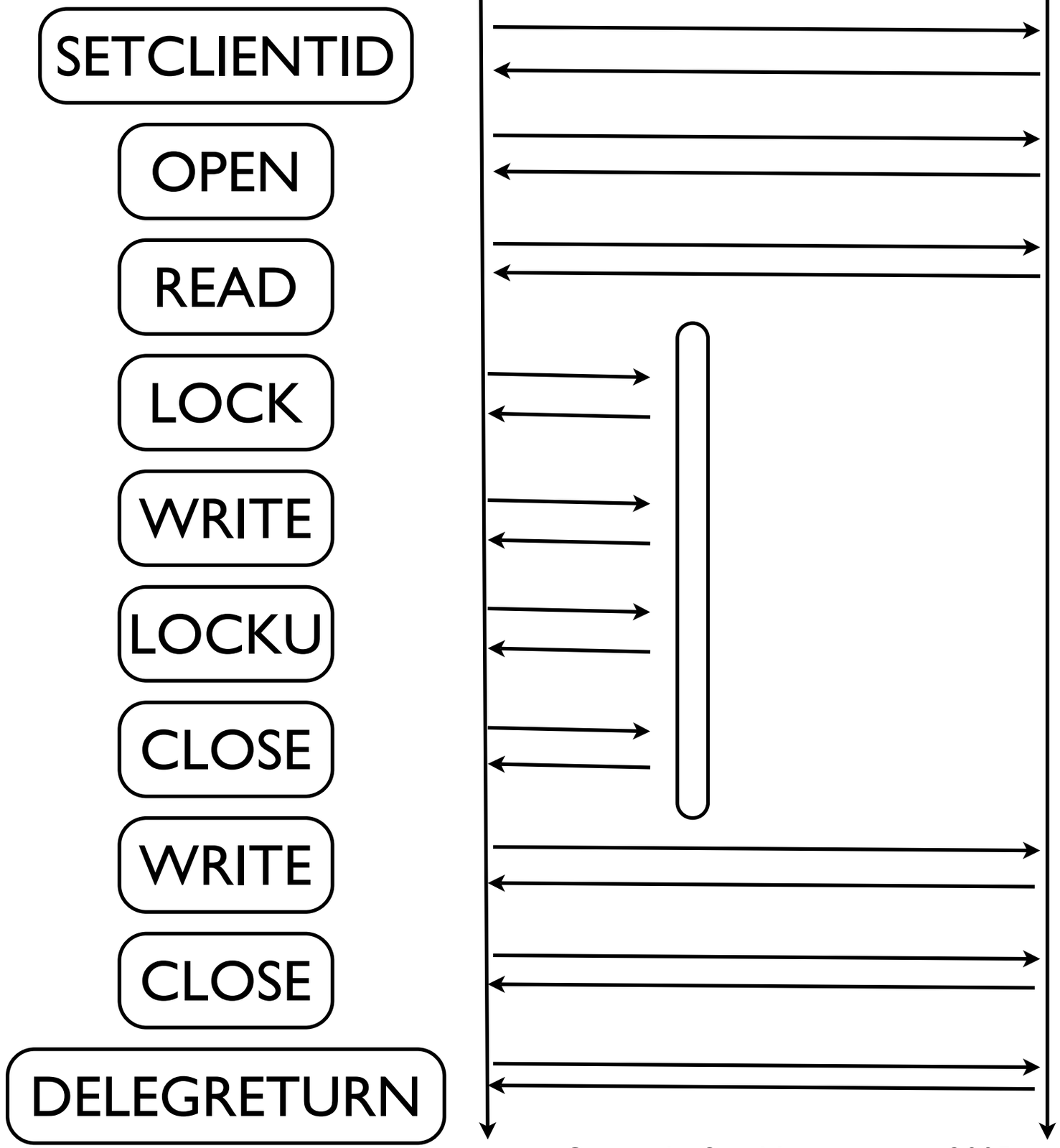
Delegation

- Intended for minimal sharing environments
- Server decides when to provide delegation
- Not required for correct protocol operation
- Callback path to client must exist
- If delegation provided, client does not need to contact server for further OPEN, LOCK, READ, WRITE, CLOSE
- Must use RENEW

(Client)

(Server)

Callback Path?
If yes, Delegate



File Locking

- Better implementation a result of protocol integration
- Byte-range locking: non-blocking, mandatory
- Callbacks not used*
- *Stateid* represents transitions of locking state
- *Sequence-id* preserves request ordering
- Locks released at lease expiration*
- Client does lock recovery at server restart

Attributes

- Mandatory, Recommended, and Named
- 52 mandatory / recommended attributes
- Extends beyond traditional Unix
- Encoding allow for extensibility at the cost of additional overhead

Mandatory Attributes

- type
- expiration type
- change
- size
- link?
- symlink?
- fsid
- lease duration

Named Attributes

- Support is optional
- OPENATTR operation returns directory filehandle of named attributes
- Once named attribute directory available, regular operations apply
- Named attributes may be recursive (not practical)
- Intended for application use
- Semantics are opaque to client and server

Access Control Lists

- Integrated into protocol
- Solved problem of 4+ ACL protocols
- ACLs may be manipulated by client
- Windows/NT ACL model (as of 2000)
- Combined with RPC security, provides for strong security model
- NFSv4 does full evaluation searching for allowance
- Differs from other models that stop at first denial

Owner / Group Attributes

- String based identifiers
- Take place of numeric uid / gid
- Identifer is: user@domain
- External mapping is not defined in NFSv4 protocol
- Mismatched *domains* may lead to “nobody”
- Solaris config:
  - /etc/default/nfs
  - DNS TXT RR
  - /etc/resolv.conf
  - domainname

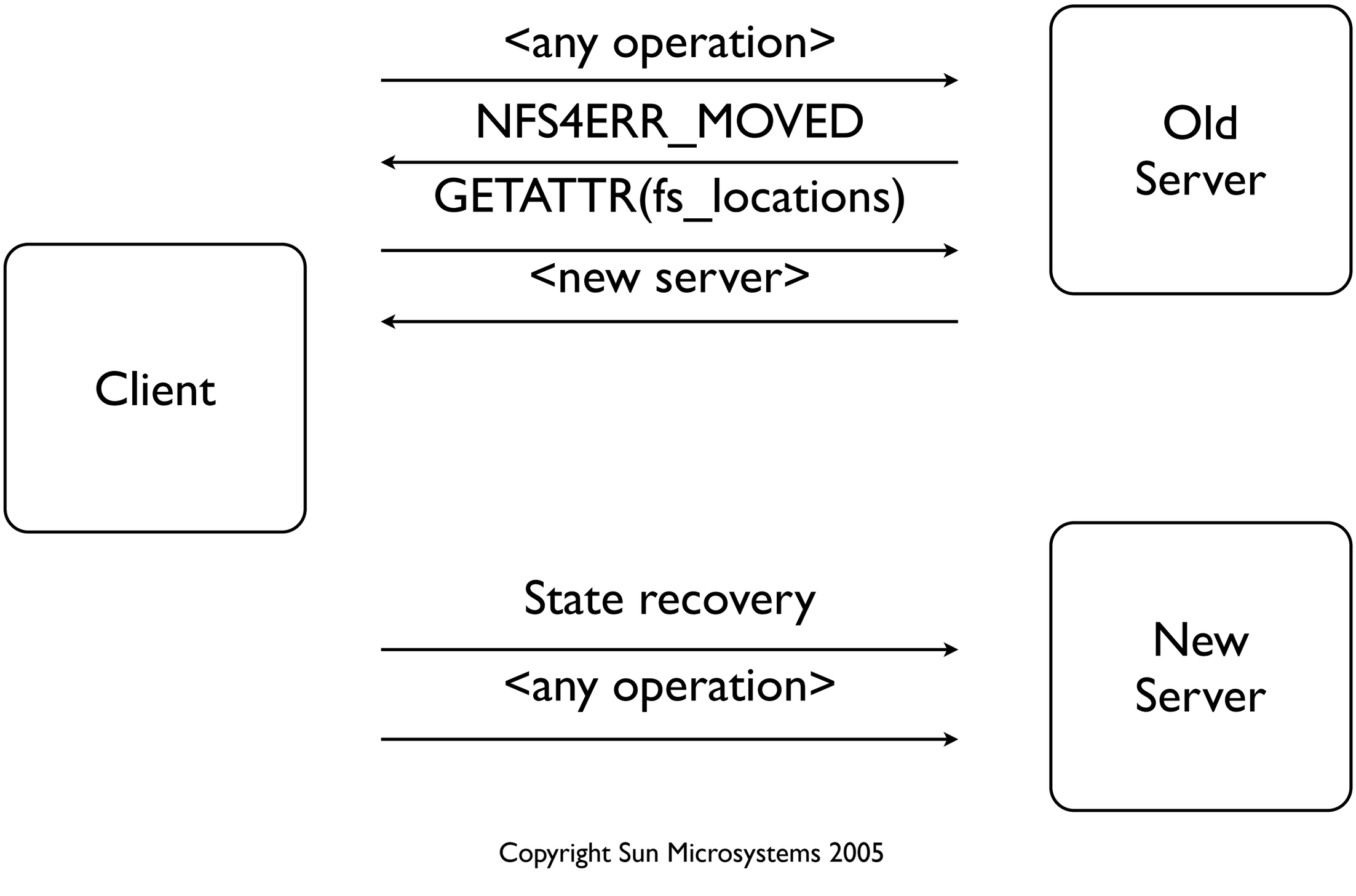
Filesystem Replication

- Intended for read-only filesystems
- Increases availability
- Client's policy directs when to switch to another replica
- *fs_locations* attribute enumerates replicas
- Client needs to reconstruct state at new server
- Server to server replication undefined

Filesystem Migration

- Enables load balancing or server reorganization
- Server-to-server transfer is undefined
- Client receives NFS4ERR_MOVED at migration event
- *fs_locations* attribute provides new location
- Mechanism may also be used for namespace construction

Migration Event



Minor Versioning

- Difficult to revise previous versions
- Protocol will not be perfect nor meet future needs
- Protocol must be allowed to evolve
- Changes allowed in minor versions
  - Operations (new argument types) may be added
  - Attributes may be added

Current Implementations

Sun Solaris 10

- NFSv4 client and server (NFSv4 is default)
- RPCSEC_GSS with Kerberos (Authentication, Integrity, Privacy)
- ACLs (limited by existing ACL APIs)
- Delegations (client and server)
- Named attributes

IBM AIX 5.3

- NFSv4 client and server (NFSv3 default)
- RPCSEC_GSS with Kerberos (Authentication, Integrity, Privacy)
- ACLs
- inter-NFSv4 domain identity mapping

Netapp NTAP 7.0.1

- NFSv4 server
- RPCSEC_GSS with Kerberos (Authentication, Integrity, Privacy)
- ACLs
- Delegations
- Named attributes

Hummingbird Windows

- NFSv4 client and server (all versions of Windows)
- RPCSEC_GSS with Kerberos (Authentication, Integrity, Privacy)
- ACLs
- Named attributes

Linux

- NFSv4 client and server (started in linux-2.6 kernel)
- RedHat and derivatives have NFSv4 included (NFSv3 default)
- RPCSEC_GSS with Kerberos (Authentication, Integrity)
- Delegations
  - client only in RedHat
  - server recent addition in 2.6 kernel

BSD

- NFSv4 Server available
- RPCSEC_GSS with Kerberos (Authentication, Integrity, Privacy)
- ACLs - exist but need more work
- Delegations
- Named attributes
- Ported to: OpenBSD3.6, FreeBSD5.3, xnu-517.3.7
- http://snowhite.cis.uoguelph.ca/nfsv4
- “solid beta”

NFSv4 Performance

- No industry standard
- SPEC SFS unlikely to be extended to support NFSv4
- Outside of SFS, throughput is the focus
- Throughput, latency, efficiency measurements are needed
- Workload variety

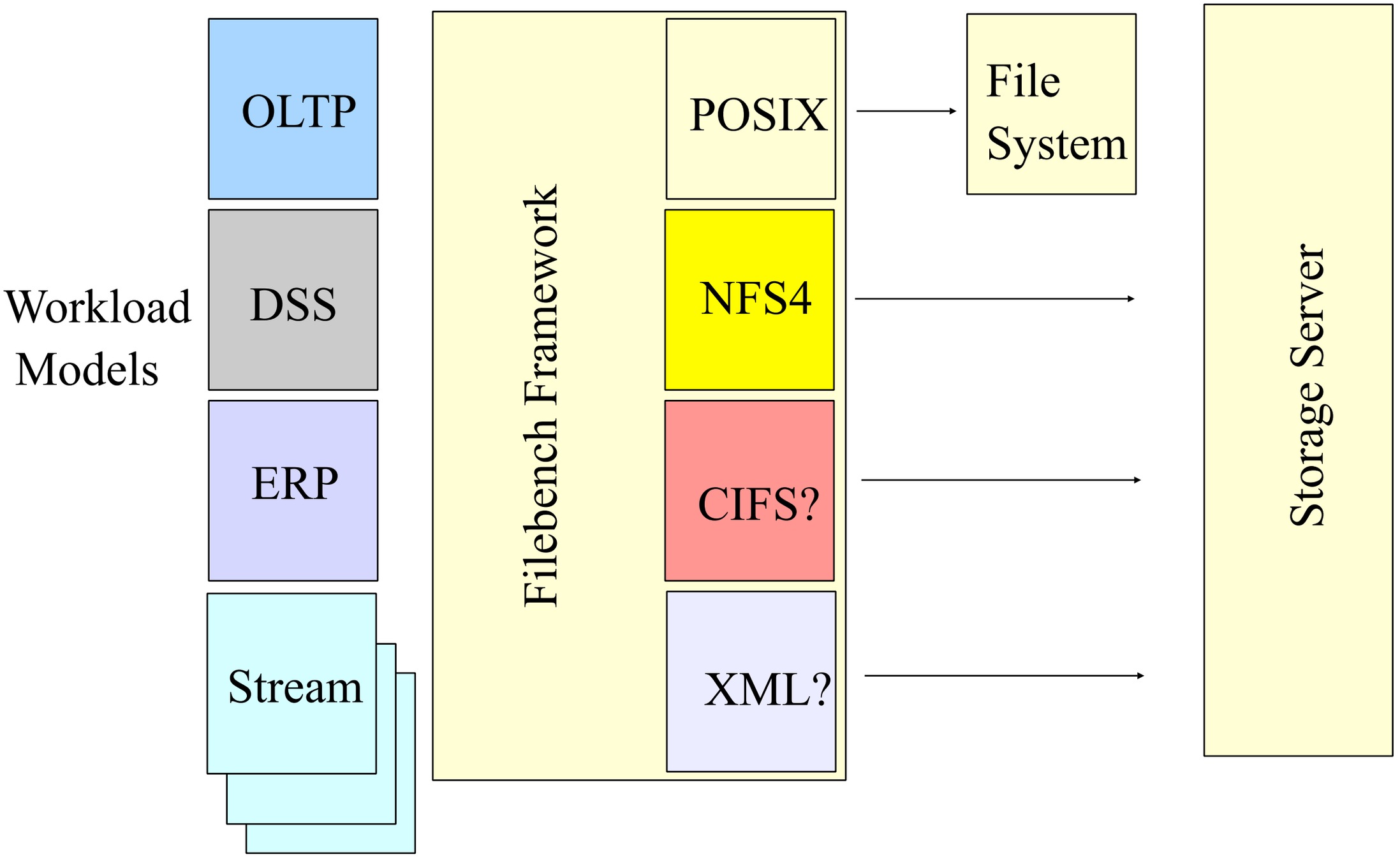
Invest in performance framework?

- We need complete test coverage for file level applications
- Current test coverage is mostly via “micro benchmarks”: Bonnie, iozone, mongo
- Test coverage was very limited (less than 10% of important cases covered)
- The current approach is to use benchmark full application suites: e.g. Oracle using TPC-C: expensive, labor intensive
- Up to 100 different benchmarks are required to accurately report on filesystem performance today

Filebench: Application Level File System Measurement

- FileBench is a configurable file level workload synthesis and measurement framework
- FileBench is an application simulator
  - Facilitates easy reproduction of complex applications
  - Applications are pre-defined by "workload descriptions"
- Workloads closely mimic real applications
  - Unique model-based approach can emulate complex applications – for example Oracle RDBMS
  - Workloads are defined using a model-language "f"
- Framework is highly extensible

Filebench Achitecture



FileBench Pre-defined Workloads

- "File Macro"
  - Small Database
  - Large Database
  - Multi-threaded web server
  - Multi-threaded proxy server
  - Home directory server
  - NFS Mail Server (postmark)
  - DB Mail Server
  - Video Server

"File Micro"

Sequential Read/Write

Multistream Read/Write

Allocating Writes

Reallocating Writes

Random Read/Write

MT Random Read/Write

File Create/Delete

File meta-data ops

I/O Types: O_DSYNC etc

Directory size scaling

Filebench Status

- Porting Status
  - Completed: S8, 10, x86, SPARC, Linux (2.6/Fedora)
  - Binary packages for Solaris 8/9/10 for x86/SPARC avail.
- Open Source
  - Intested in community development
  - Linux port phase 1 complete: requires NPTL threads
  - Framework will be co-ordinated via sourceforge

IETF Futures

- CCM
- NFS direct data placement (RPC/RDMA/RDDP)
- ACL clarifications
- Replication/Migration
- Directory Delegations
- Namespace (“global”)
- Multi-”realm” user/group mappings
- pNFS

GETATTR, CLOSE

- spencer.shepler@sun.com
- blogs.sun.com/shepler

- www.nfsv4.org
- www.ietf.org