SMB 2.2 : Bigger, Faster, Scalier (Part 1)

David Kruse
Mathew George
Microsoft

State of the Protocol

- SMB 2.002
  - Simplified command set
  - Uniformity (UNICODE, timestamps, etc.)
  - Expanded identifier space (UINT64)
  - HMAC-SHA256 signing
  - Dynamic crediting
  - Async notifications for long running requests
  - Unrestricted compounding of requests
  - Symbolic Link support
  - Durable opens for handling disconnects

State of the Protocol

- SMB 2.100
  - Frame reduction for common workloads and WAN
    - SMB Leasing
    - Branch Cache extensions
  - Large MTU support (throughput)
  - Resilient Handles

Problem Space

Availability

- Enable transparent client recover in the presence of
  - Network Failure
  - Server Failure
- Minimize failover time to reduce application stalls

Performance

- Enable clients to aggregate available bandwidth across adapters transparently
- Continue to increase efficiency on high bandwidth networks

Traffic Reduction

- Continue improving user perceived latency when working in a WAN environment

SMB 2.2

- Multichannel
- SMB over RDMA
- Scale-Out Awareness
- Persistent Handles
- Witness Notification Protocol
- Clustered Client Failover
- Directory Leasing
- Branch Cache v2
- Support for Storage Features (TRIM, etc)

SMB 2.2 – Advancements for WAN

Branch Cache v2 and Directory Leasing

Wednesday: 1:00-1:50

Molly Brown, Mathew George

Architecture Terminology

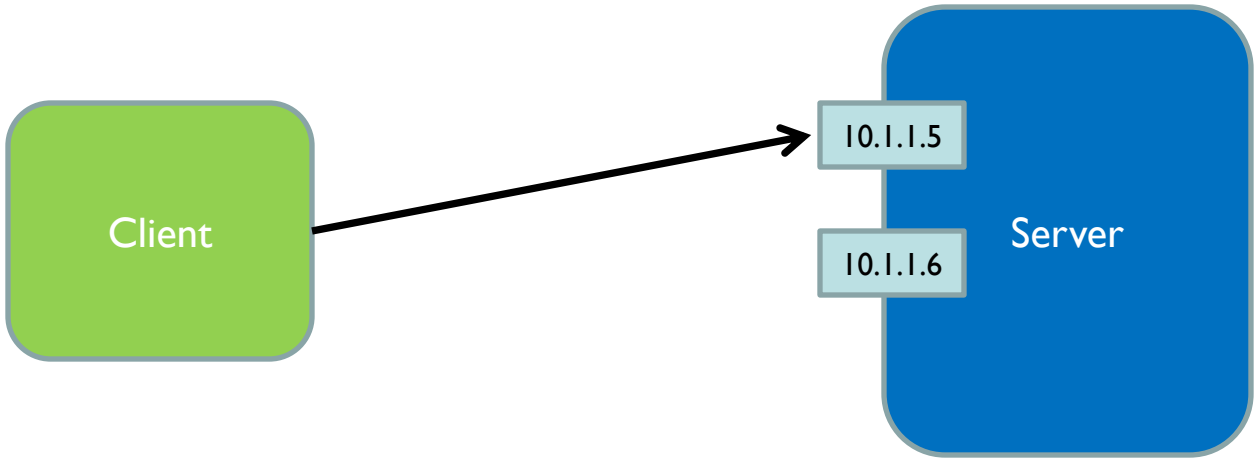
Architecture(s)

Stand Alone File Server

DNS

smbsrv.test.com

- 10.1.1.5
- 10.1.1.6



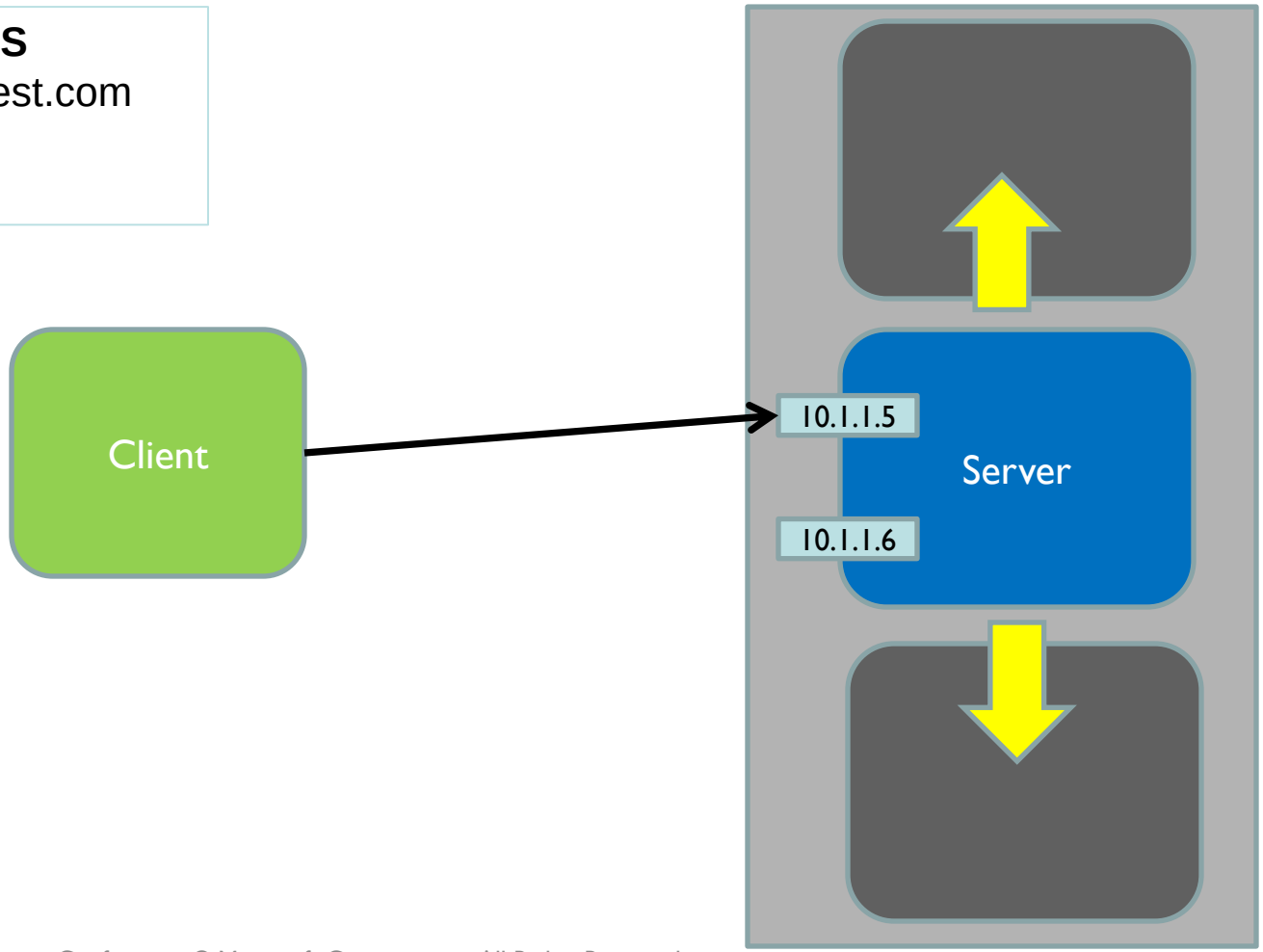
Architecture(s)

"Traditional" Clustered File Server

DNS

smbsrv.test.com

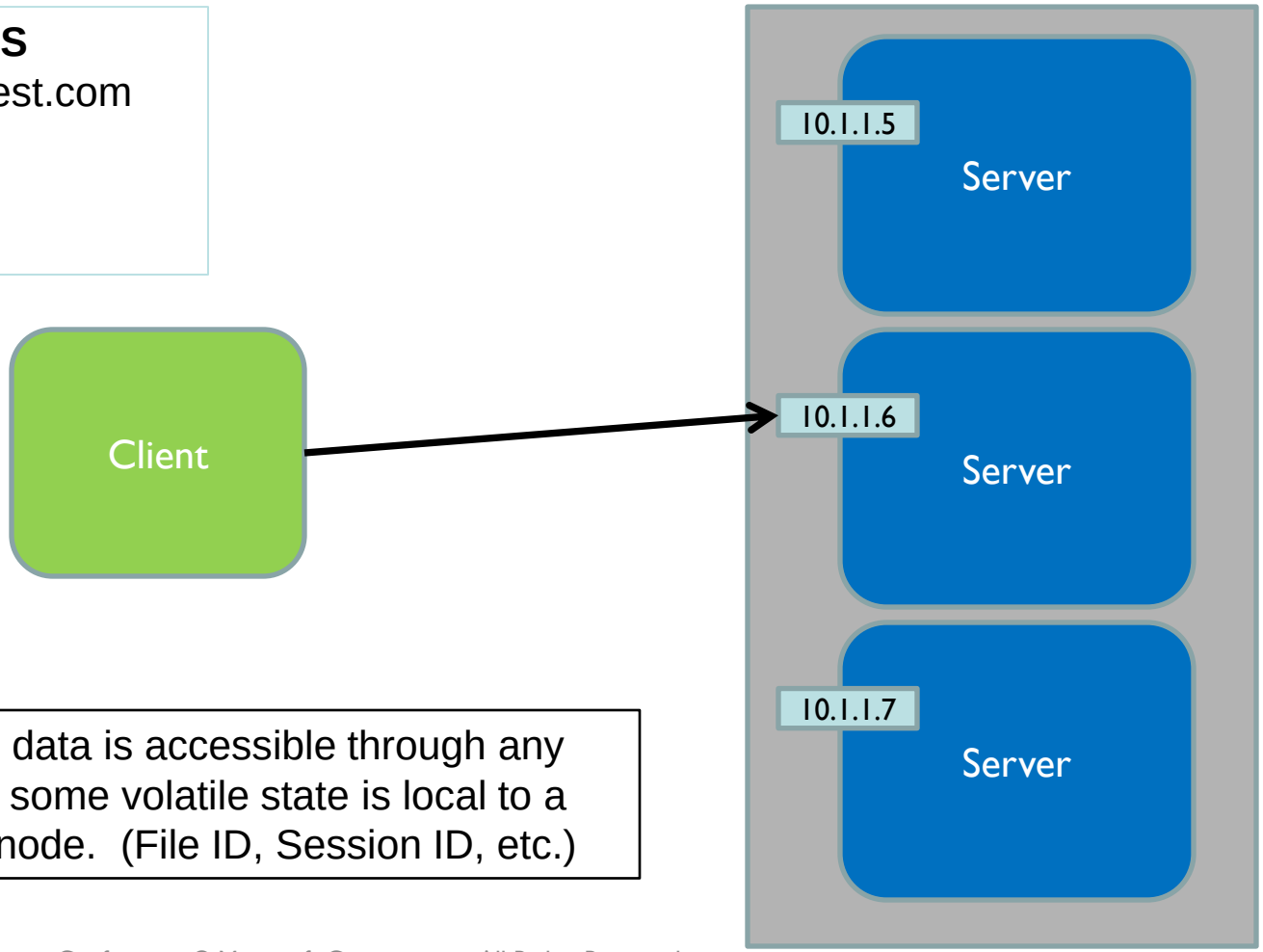
- 10.1.1.5
- 10.1.1.6



Architecture(s)

"Scale-Out" Clustered File Server

DNS
smbsrv.test.com
- 10.1.1.5
- 10.1.1.6
- 10.1.1.7



While data is accessible through any node, some volatile state is local to a given node. (File ID, Session ID, etc.)

Scale-Out Awareness

Scale-Out – Server Requirements

- File system semantics (data integrity, lease/oplock, byte range locks, etc.) must be coherent
- Persistent portion of FileID's must be unique across the cluster (including same-node reboot)
- Session invalidation (PreviousSessionID) must span nodes to locate and disconnect sessions on reconnect
- Server returns SMB2_SHARE_CAP_SCALEOUT in tree connect response to inform client of capability
- Lease keys are not shared across nodes

Scale-Out Client Changes

- Client will "prefer" current node on reconnect after abortive disconnect, but must be able to fall back to any node
- During reconnect, client may be forced to select new node after initial connect successful (based on auth or share failures) to permit stale DNS caching or node eviction
- Client may limit reconnect attempts in large scale-out scenarios (anti-DOS, app timeout concerns)
- A single Windows clients will operate against a single node at a time

Multichannel

Mathew George

Multichannel - Goals

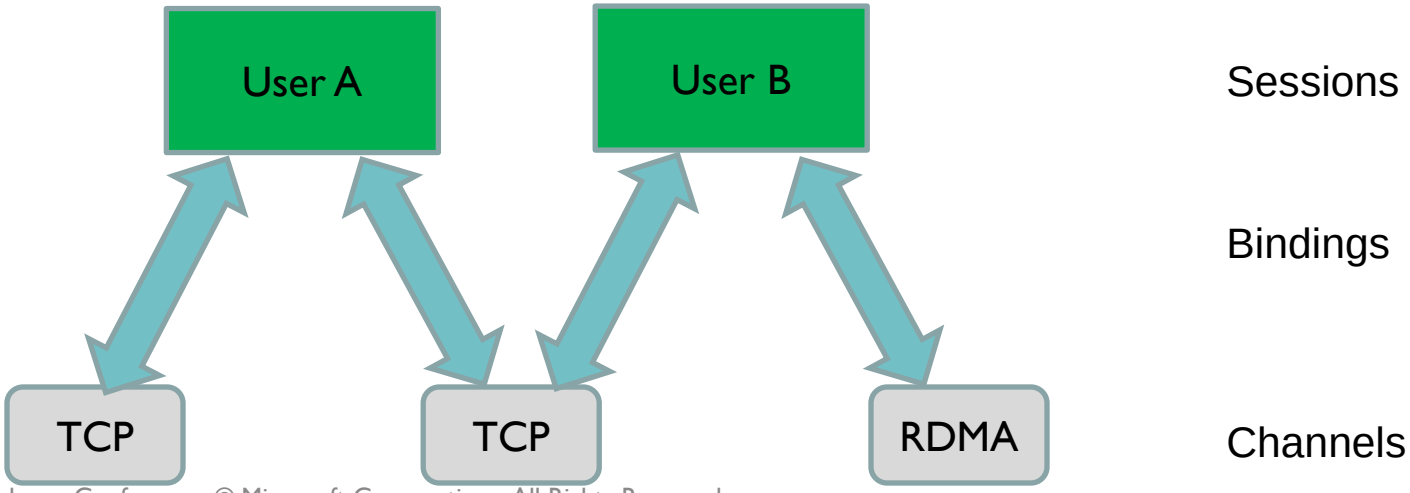
- Scalability
  - Use multiple interfaces if available.
    - non-homogeneous networks
  - Use multiple "streams/channels" on the same interface to get around I/O and CPU limitations
    - Exploit RSS capabilities in NICs.
  - Foundation for enabling SMB2 over newer ultra-high-performance network interconnects.

Multichannel – Goals

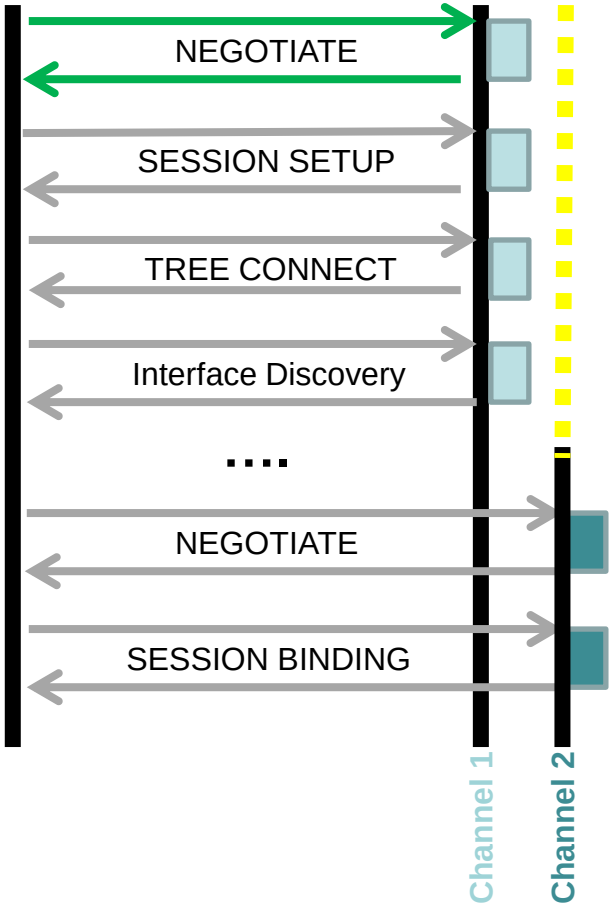
- Availability / Network fault tolerance
  - Make the SMB2 protocol resilient to interface, link or switch failures.
  - Move "link awareness" higher up the stack to enable more intelligent decision making.
    - Augment NIC teaming at the network layer.
    - Keep fallback paths ready, prioritize available links.
    - React quickly to changes to network availability.
- Manageability
  - (Nearly) ZERO configuration.
    - Interface discovery and capability exchange built in.

Multichannel - Terminology

- A **Channel** refers to an underlying transport connection between the client and the server.
- A **Session** refers to an authenticated user context.
- A **Session Binding** refers to a logical association between a Session and a Channel.
  - N:N relationship between Sessions and Channels



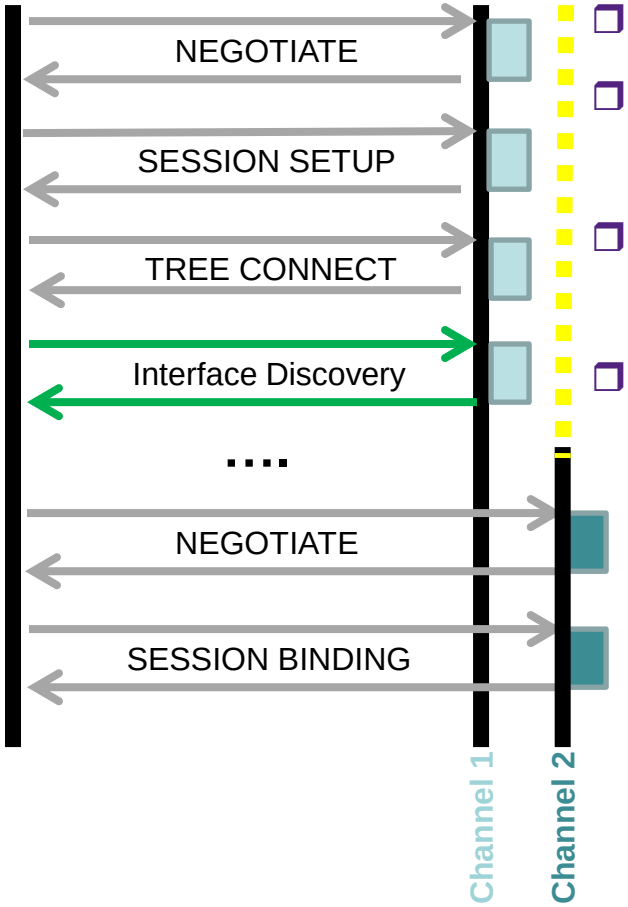
Multichannel – Capability Negotiation



- SMB 2.2 clients send client capabilities to server in the negotiate request.
- Server filters out capabilities it does not support and returns the effective capabilities in the negotiate response.
- New negotiate capability to indicate multichannel support.

```
#define SMB2_GLOBAL_CAP_MULTI_CHANNEL 0x8

typedef struct _SMB2_RESP_NEGOTIATE
{
    USHORT StructureSize;
    USHORT SecurityMode;
    USHORT DialectRevision;
    USHORT Reserved;
    GUID   ServerGuid;
    ULONG  Capabilities;
    …
    …
} SMB2_RESP_NEGOTIATE, *PSMB2_RESP_NEGOTIATE;
```

Multichannel – Interface discovery



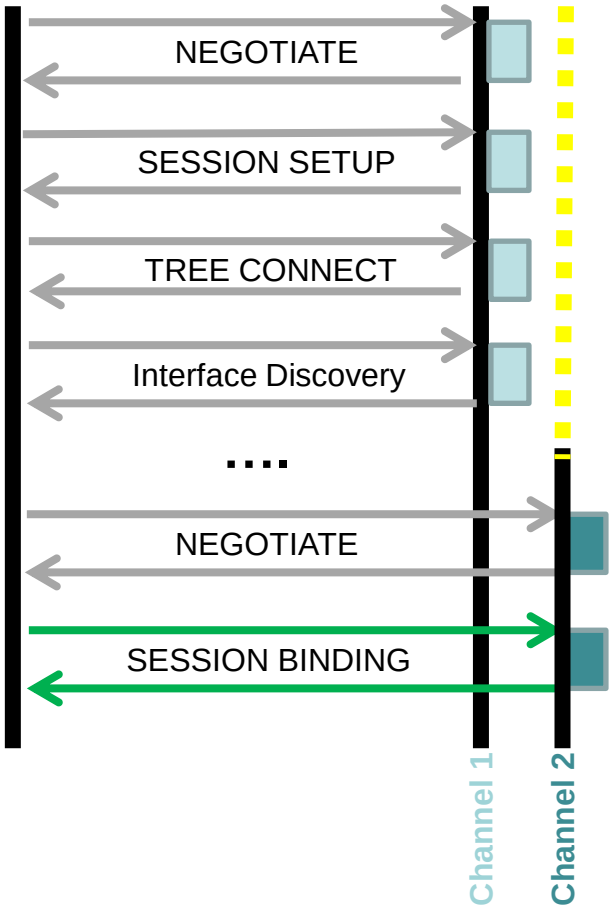
- Client establishes initial authenticated session to server
- Client queries for additional interfaces on server using the new FSCTL_QUERY_NETWORK_INTERFACE_INFO query.
- Server returns list of available interfaces and capabilities.
- Client matches up server interfaces with local interfaces and builds an ordered list of "possible channels".

```
#define NETWORK_INTERFACE_CAPABILITY_RSS    0x00000001L
#define NETWORK_INTERFACE_CAPABILITY_RDMA   0x00000002L

typedef struct _NETWORK_INTERFACE_INFO {
    ULONG    Next;
    ULONG    IfIndex;
    ULONG    Capability;
    ULONG    RssQueueCount;
    ULONG64  LinkSpeed;
    UCHAR    SockAddr[1];       // SOCKADDR_STORAGE
} NETWORK_INTERFACE_INFO;
```

Multichannel – Establishing Session Bindings.

- Client establishes additional channels based on the ordered list of addresses.
  - Connect to target address and NEGOTIATE
- Client binds existing sessions to one or more of the newly established channels.
  - New flag in session setup request to indicate session binding.
  - *SessionId* is initialized with UID from existing session.
  - Request MUST be signed using the original session signing key.
  - Authentication is done using SPNEGO as usual.
  - The new session key must be queried and saved on the binding structure.
    - If signing is active, this key MUST be used to sign requests for this UID on this channel.

Multichannel – Establishing Session Bindings



- Session remains active as long as there is at least one active binding.
- Kerberos re-authentication only needs to be done on a single channel.

```
#define SMB2_SESSION_FLAG_BINDING      0x00000001

typedef struct _SMB2_REQ_SESSION_SETUP {
    USHORT StructureSize;
    UCHAR  Flags;
    UCHAR  SecurityMode;
    ULONG  Capabilities;
    ULONG  Reserved;
    USHORT SecurityBufferOffset;
    USHORT SecurityBufferLength;
    UINT64 PreviousSessionId;
    UCHAR  Buffer[1];
} SMB2_REQ_SESSION_SETUP;
```

Multichannel – Sequencing and Correctness

- Classes of I/O operations
  - Operations with "exactly once" semantics
    - Replay detection is required.
    - State changing CREATE operations, byte range locks.
  - Operations resulting in "write-write" conflicts.
    - Full sequencing and replay detection is not required.
    - A "barrier" semantic can handle these conflicts.
  - Operations which are safe to replay.
    - Non-modifying, non-state changing. (read, queries, enumeration.)
- REPLAY flag added to the protocol(*)

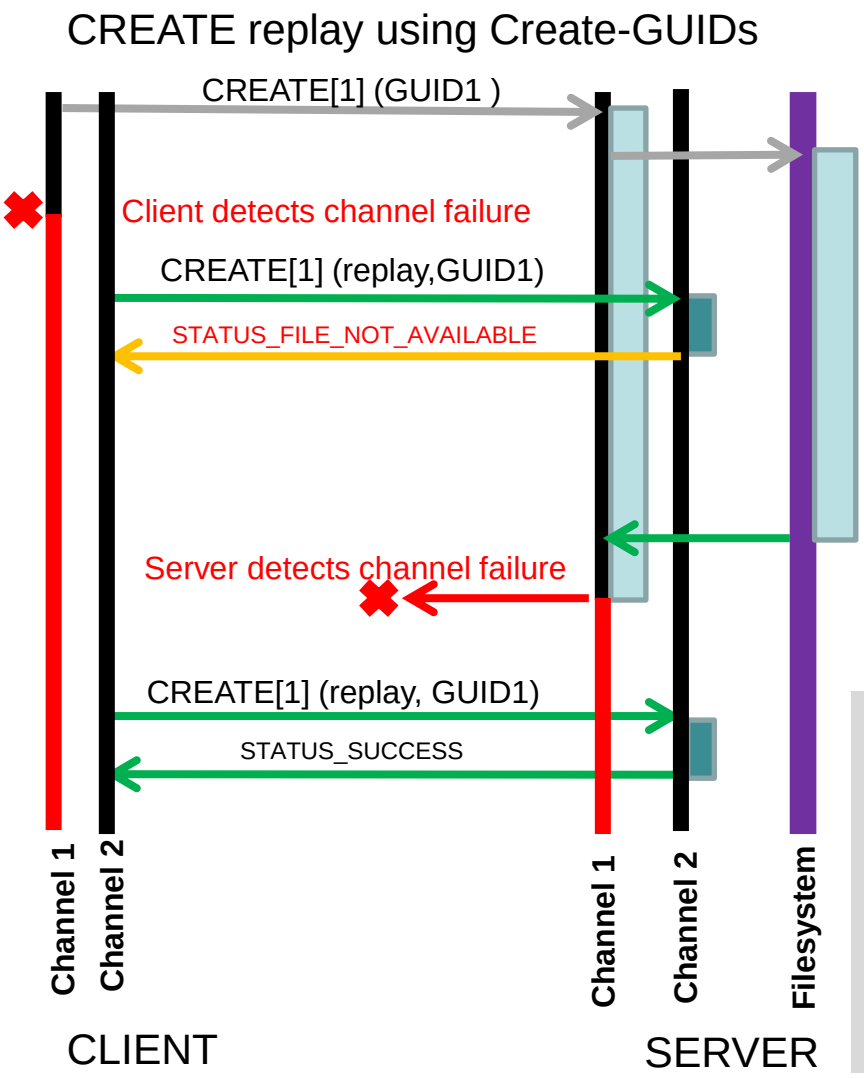
(*) - Upcoming change not in current builds or initial SMB 2.2 protocol documentation.

Multichannel – Lock Sequence Numbers

- Introduced in SMB 2.1 (resilient handles)
- Consists of a 28-bit bucket# and a 4-bit sequence#.
  - Windows Client / Server only supports 64 buckets. (6 bits)
  - Sequence# is incremented when the bucket is re-used.
  - Parallel (un)lock requests will use distinct bucket numbers.
  - Server remembers lock sequence numbers associated with active byte-range locks to detect lock replay.

```
typedef struct _SMB2_REQ_LOCK {
    USHORT StructureSize;    // = sizeof(SMB2_REQ_LOCK)
    USHORT LockCount;
    ULONG  LockSequence;     // bits 0..3 seq#, bits 4..9 bucket#
    SMB2_FILEID FileId;      // Identifier of the file being (un)locked
    SMB2_LOCK Locks[1];      // Array of (LockCount) lock structures
} SMB2_REQ_LOCK, *PSMB2_REQ_LOCK;
```

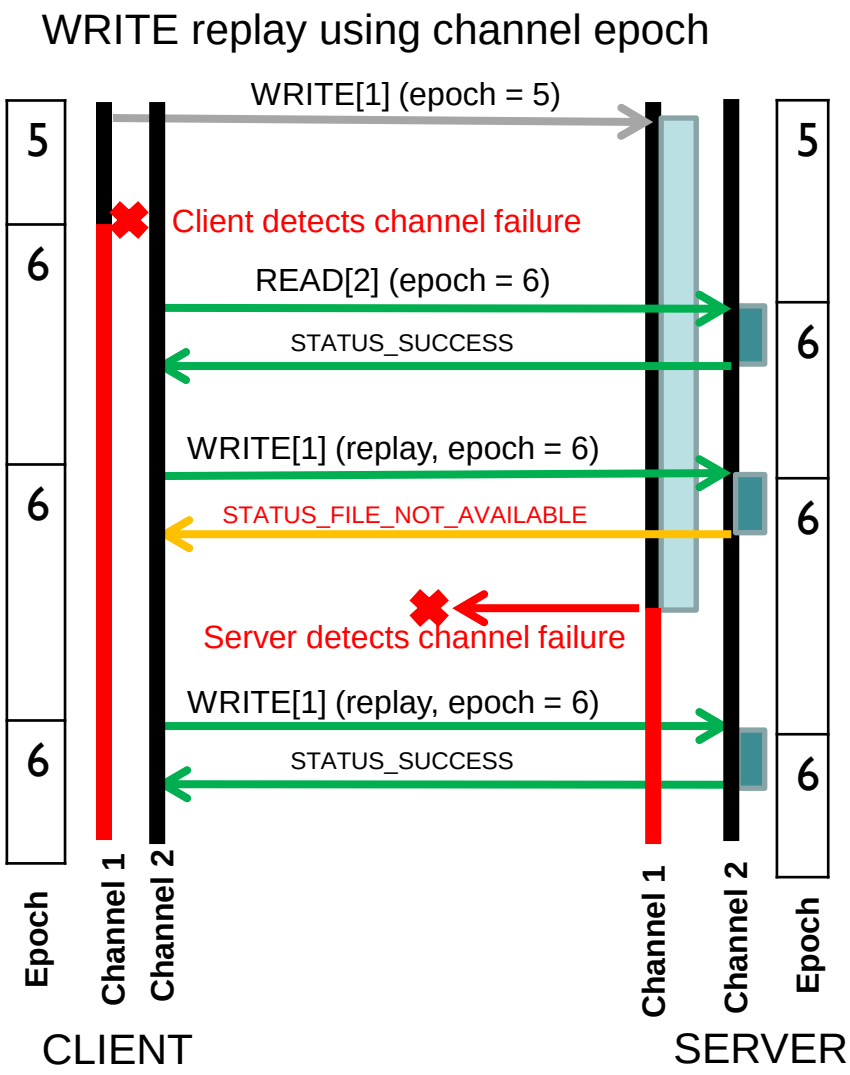
Multichannel – CREATE Replay



- Client supplies a handle ID (GUID) to the server with every create.
  - Unique per handle, per client.
  - Unique per handle per share.
- Server can detect replays by looking up the GUID.
- New durability V2 create context.

```
typedef struct {

    ULONG  Timeout;     // msec
    ULONG  Flags;
    UINT64 Reserved;    // MUST be zero.
    GUID   CreateGuid; // client supplied ID

} SMB2_DURABLE_HANDLE_REQUEST_V2;
```

Multichannel – Channel Epoch Numbers(*)



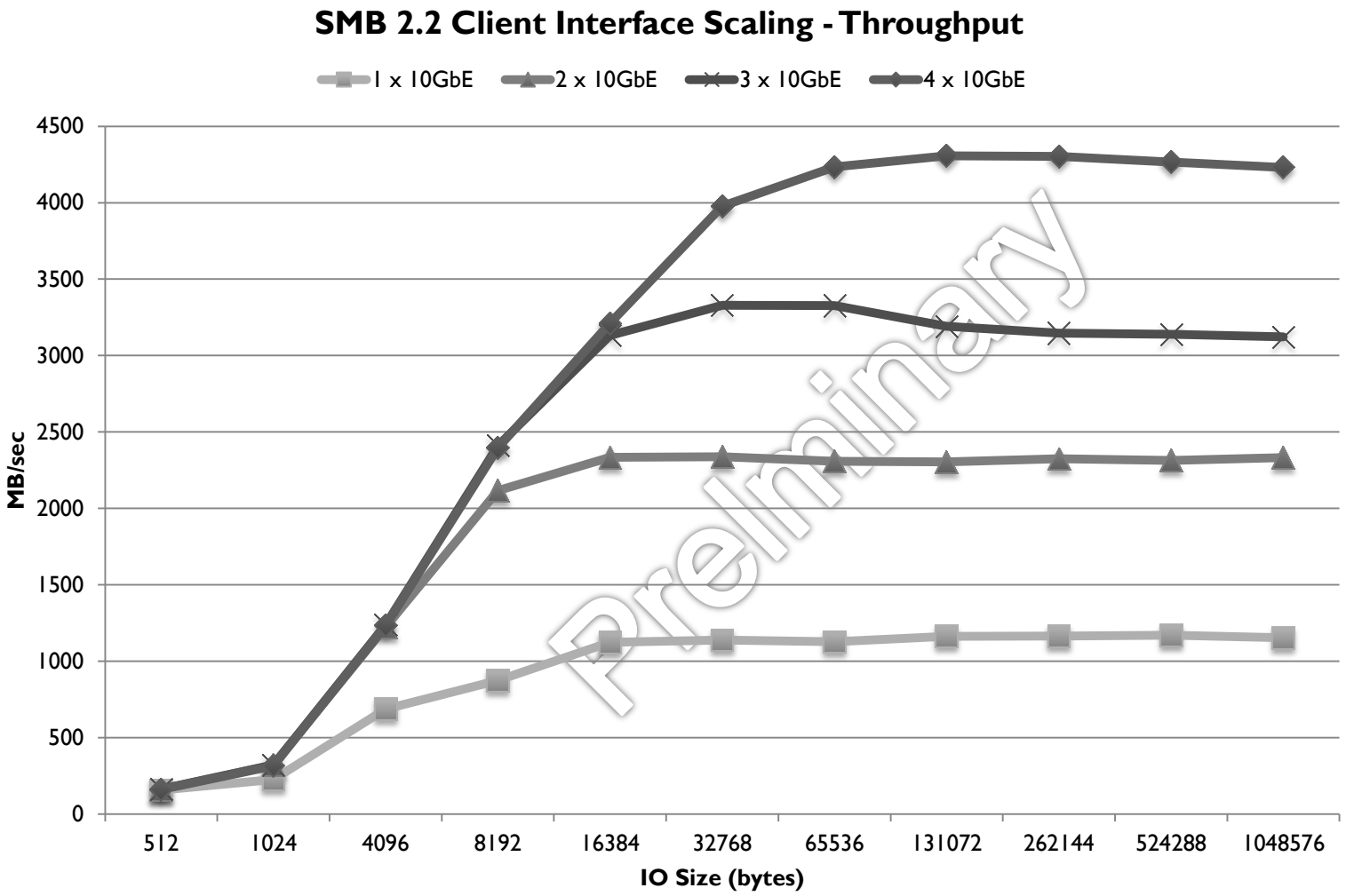
- Lightweight compared to full replay detection.
- Guarantees that all previous "instances" of an I/O are drained before the replay is executed.
- Client maintains 16-bit channel epoch number.
  - Incremented on a network failure.
  - Sent to server via unused Status field.
- Server fails "state changing" "non-replay" requests with stale epoch numbers.
- Server fails "state changing" "replay" requests when there are outstanding operations with older epoch numbers.
- New error - STATUS_FILE_NOT_AVAILABLE avoids blocking on the server and tells client to retry.
- Server can do epoch check at "handle" granularity.

(*) – Upcoming change. Not in current builds or initial SMB 2.2 protocol documentation.

Multichannel – Guidelines for Channel Selection & Load Balancing

- If multiple interfaces with similar characteristics are available, establish channels on each interface.
- For each interface, establish multiple channels if the NIC is RSS capable or is 10 Gbit or higher throughput.
- Server can do coarse grained load balancing by varying credits on each channel.
- Client can load balance using
  - round robin
  - Shortest queue length.
  - Processor / NUMA-node affinity based scheduling.
- Client may wait for sufficient data transfer before going multichannel.

SMB 2.2 Client 10GbE Interface Scaling



SMB 2.2 – Advancements in Server Application Performance

Thursday, 9:30-10:20

Dan Lovinger

SMB 2.2 over RDMA

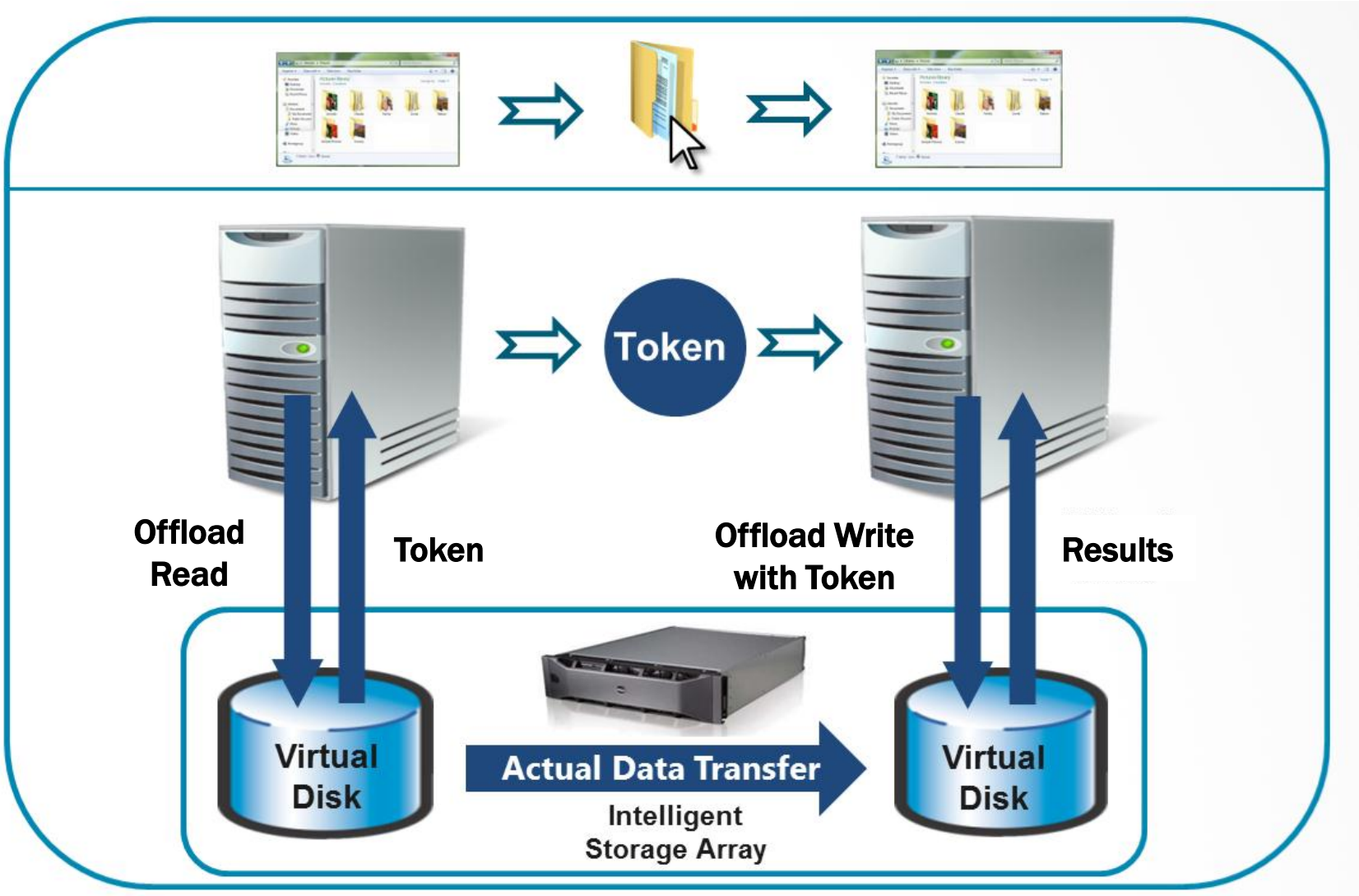
These go to eleven.

Today: 4:05-4:55

Tom Talpey, Greg Kramer

Support for Advanced Storage Features

File Copy Offload



File Copy Offload & Trim Support

- SMB 2.2 adds support for
  - FSCTL_OFFLOAD_READ, FSCTL_OFFLOAD_WRITE
    - Provides copy offload across different volumes on different file servers
  - FSCTL_FILE_LEVEL_TRIM
    - Allows a file system to tell an underlying storage device that the contents of specified sectors are no longer important
  - FileFsSectorSizeInformation
    - Returns both physical and logical volume sector info
- See NealCh's "Performance Enhancements in NTFS" talk from yesterday for more details

Pulling it Together

- Demo – Jose Barreto

Questions?

SMB 2.2 : Bigger, Faster, Scalier (Part 2)

David Kruse
Mathew George
Microsoft

SMB2 Continuous Availability Persistent Handles

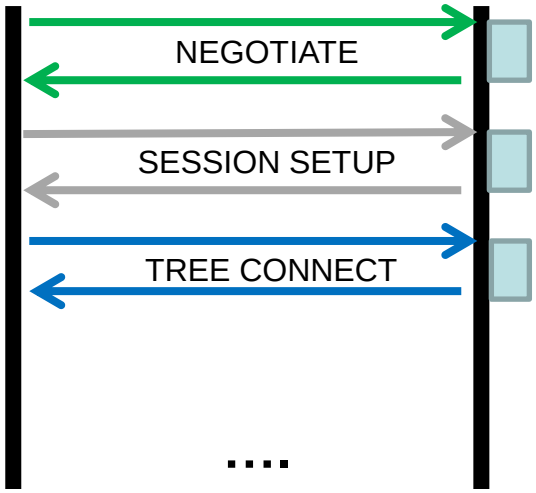
Continuous Availability – Server Application Expectations

- Server apps expect to be able to always access data on a continuously available file server / share.
- Transient network failures or server failures are completely hidden from the application.
  - Filesystem client is expected to transparently recover disconnected handles and retry I/O operations.
  - The application sees a small pause in the I/O, but no errors.
- I/O operations are bound by a specific timeout
  - Application requested OR
  - Server configured.
- Reliability on par with direct-attached storage / SAN.

Persistent Handles – The road to Continuously Available SMB

	Durability	Resilience	Persistence
SMB2 Protocol Revision	2.0 Windows Server 2008, Windows Vista	2.1 Windows Server 2008 R2, Windows 7	2.2 Windows 8
Strong guarantees on handle availability and I/O timeouts?	No (Best effort. Relies on H leases.)	Yes	Yes
Resilient to network glitches?	Yes	Yes	Yes
Resilient to server failures?	No	No	Yes (Shared state store)
Resilient to failures during CREATE?	No	No	Yes
Transparent to applications?	Yes	No	Yes. (Server configured or app specified timeouts.)
Can client cache data (if it has a lease)?	Yes.	No. (except for exclusive opens)	Yes.

Persistent Handles – Capability Negotiation.



- SMB 2.2 clients negotiate the new "persistent handle" capability.

```
#define SMB2_GLOBAL_CAP_PERSISTENT_HANDLES 0x10
typedef struct _SMB2_RESP_NEGOTIATE {
    USHORT StructureSize;
    USHORT SecurityMode;
    USHORT DialectRevision;
    USHORT Reserved;
    GUID   ServerGuid;
    ULONG  Capabilities;
    …
} SMB2_RESP_NEGOTIATE;
```

- Check for the "continuous availability" capability in the tree connect response.

```
#define SMB2_SHARE_CAP_CONTINUOUS_AVAILABILITY  0x10
#define SMB2_SHARE_CAP_CLUSTER                  0x40
typedef struct _SMB2_RESP_TREE_CONNECT {
    USHORT StructureSize;
    UCHAR  ShareType;
    UCHAR  Reserved;
    ULONG  ShareFlags;
    ULONG  Capabilities;
    ULONG  MaximalAccess;
} SMB2_RESP_TREE_CONNECT;
```

Persistent Handles – Durability V2 Create Context

- Clients request persistent handles on continuously available shares using the new durability V2 create context.
  - Clients can request optional timeout for which the server must reserve the handle.
  - Clients generate a unique 128-bit ID per handle (unique per share)
  - Clients set the "persistent" flag.
- Windows clients request persistence for
  - File handles opened for read, write, execute or delete access.
  - Directory handles opened for delete access or disposition != FILE_OPEN

```
#define SMB2_DHANDLE_FLAG_PERSISTENT 0x2

typedef struct _SMB2_DURABLE_HANDLE_REQUEST_V2 {

    ULONG  Timeout;      // in milliseconds. Value of ZERO indicates "use server default".
    ULONG  Flags;
    UINT64 Reserved;     // MUST be zero.
    GUID   CreateGuid;   // client supplied unique ID.

} SMB2_DURABLE_HANDLE_REQUEST_V2;
```

Persistent Handles – Durability V2 Create Context

- Servers indicate to the client that persistence was granted by setting the persistent flag in the durable handle V2 response.
- Windows Servers will grant persistence for
  - All handles opened for delete access.
  - All potentially state changing creates. (e.g. CREATE_NEW, OVERWRITE, OVERWRITE_IF)
  - File handles opened for read, write or execute.
- Granted timeout is decided by server based on the client supplied timeout and the server configured defaults / limits.
  - Client uses the timeout as a hint to determine how long to retry I/O.

```
#define SMB2_DHANDLE_FLAG_PERSISTENT 0x2

typedef struct _SMB2_DURABLE_HANDLE_RESPONSE_V2 {

    ULONG Timeout;     // timeout (ms) granted by the server.
    ULONG Flags;

} SMB2_DURABLE_HANDLE_RESPONSE_V2, *PSMB2_DURABLE_HANDLE_RESPONSE_V2;
```

Persistent Handles – Server Guarantees

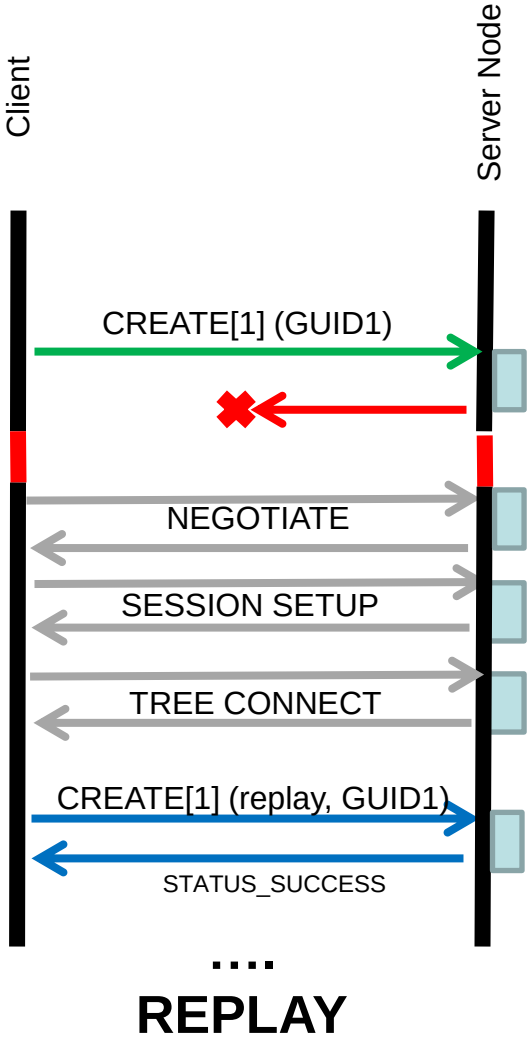
- The handle must be "reserved" for a disconnected client up to the timeout in the durable handle V2 response.
- All modifications made via the handle are persisted to stable storage before the I/O is completed.
- While a client is disconnected, all state changing operations affecting the file are blocked until the reservation has expired or the client has "resumed" the handle.
- All byte range locks taken on the handle are persisted by the server across network / server failures.
- Server must implement the correct replay semantics for state changing operations.

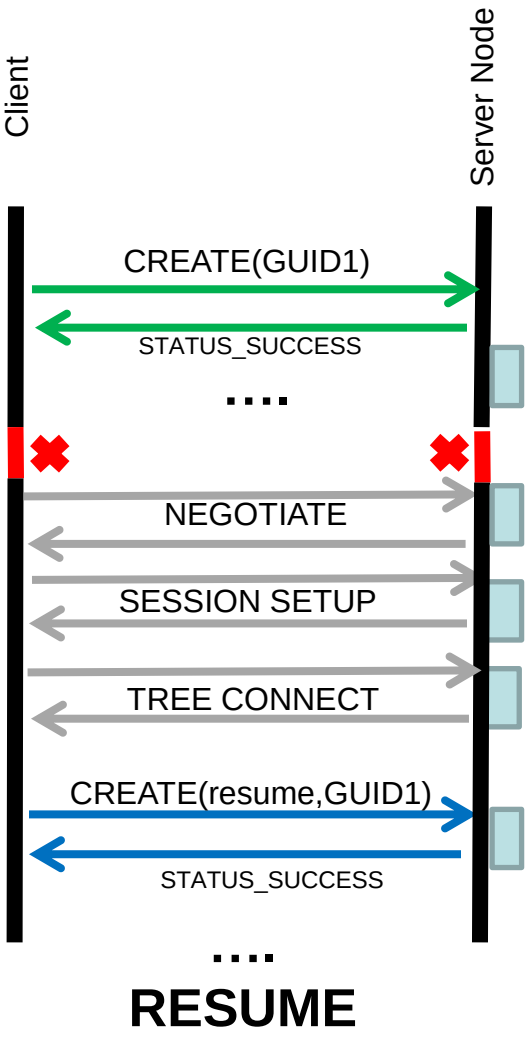
Persistent Handles – Create Replay vs. Resume

- Replay
  - Client did not get a response to its create request.
  - Client sets the replay flag in the SMB2 header and attaches a durable handle V2 request with the same CreateGUID used in the original create.
- Resume
  - Client is attempting to reconnect to a previously open handle
  - The attaches a durable handle V2 reconnect create context with the same CreateGUID used in the original create.
  - Server returns the same persistent FileID, but different volatile FileID.

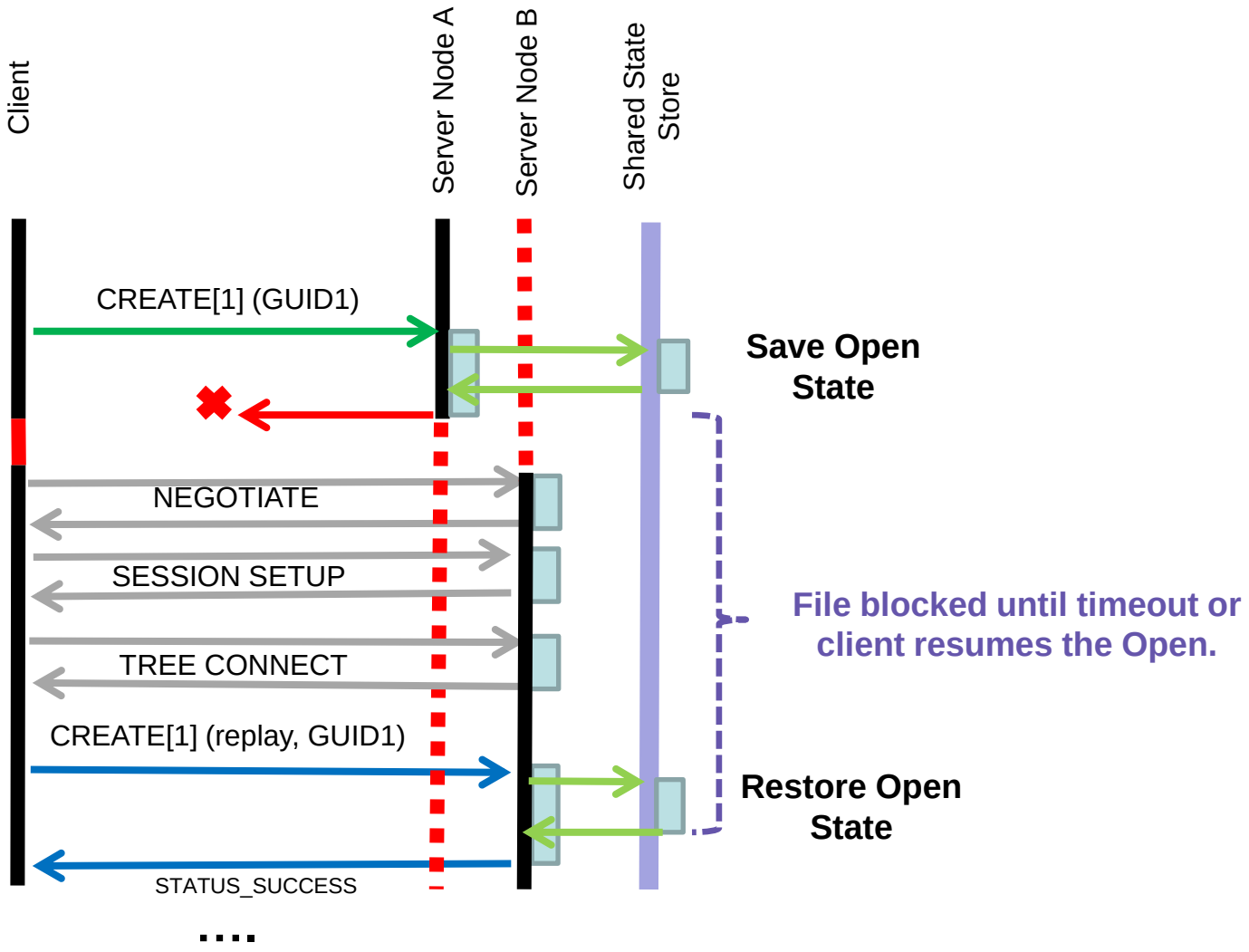
```
#define SMB2_DHANDLE_FLAG_PERSISTENT 0x2

typedef struct _SMB2_DURABLE_HANDLE_RECONNECT_V2 {

    SMB2_FILEID FileId;     // SMB2 FID returned by the server when opening the file.
    GUID        CreateGuid; // client supplied unique ID for this handle.
    ULONG       Flags;

} SMB2_DURABLE_HANDLE_RECONNECT_V2;
```

Persistent Handles – Replay vs. Resume (Network Failure)





Persistent Handles – Replay vs. Resume (Server Node Failure)



Persistent Handles – State associated with an "Open".

- All new opens which can potentially affect state must be blocked until clients have a chance to "resume" existing handles.
  - Windows implementation will block all opens except for read-attribute opens.
  - Side effect – prevents most H or W lease breaks.
- All other operations which cause W,H lease breaks are blocked.
  - parent directory renames, parent directory deletion.
- New error codes to indicate to the client a transient failure. Client is responsible for retrying the failed operation.
  - STATUS_SERVER_UNAVAILABLE, STATUS_FILE_NOT_AVAILABLE
  - Non SMB 2.2 aware clients will see STATUS_SHARING_VIOLATION error.
- Delete disposition state must be preserved on server.
- Byte range lock state must be preserved on the server.

Persistent Handles – Lease State

- Handle leases are preserved by virtue of handle reservation.
- Exclusive (W) leases are preserved via blocking new creates from other clients.
- R leases need not be preserved
  - Client can recover from a loss of R lease by simply discarding cached data.
- The server is not required to explicitly track the lease state across server failovers.
  - The client is expected to re-request its lease when resuming its persistent opens.

Persistent Handles – Lease State Validation using "Lease Epochs"

- A epoch/sequence number added to the protocol to accurately track lease state changes.
  - Incremented by the server every lease upgrade or downgrade.
  - Used by the client to detect unexpected lease state changes.
- New lease V2 create context.

```
#define SMB2_LEASE_FLAG_PARENT_LEASE_KEY_SET 0x04

typedef struct _SMB2_ECP_LEASE_v2 {

    GUID   LeaseKey;        // Unique ID which identifies owner of the lease.
    DWORD  LeaseState;      // The kind of lease the client is requesting
    DWORD  Flags;           // Optional:  flags.
    INT64  LeaseDuration;   // Not used. Must be ZERO.
    GUID   ParentLeaseKey; // Unique ID which identifies the lease owner for the parent
                            // directory.
    USHORT Epoch;           // Current lease epoch number.
    USHORT Reserved;

} SMB2_ECP_REQUEST_LEASE_v2, // Client->Server.
  SMB2_ECP_GRANTED_LEASE_v2; // Server->Client.
```

Persistent Handles – Lease State Validation on Client

- CREATE request typically retains or upgrades lease.
  - OK to lose (and regain) R lease.

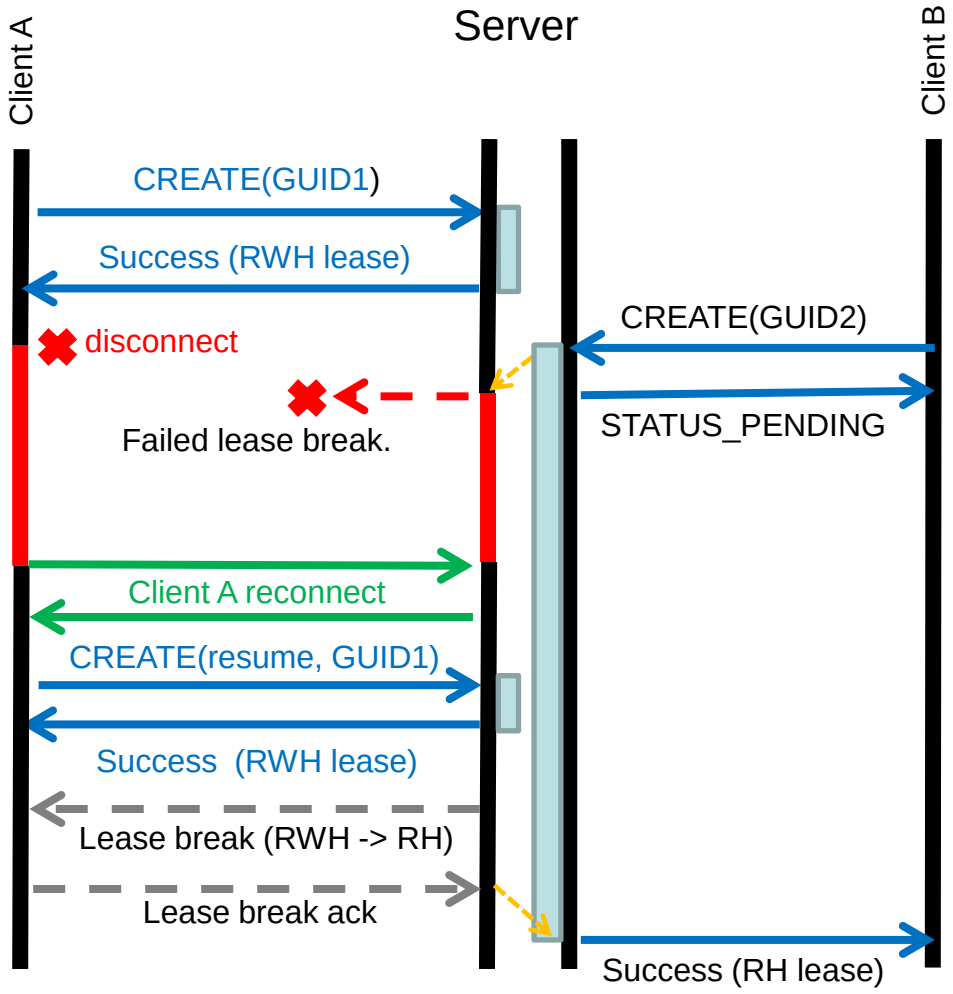
New Lease State →	R	RH	RWH	None
None	Δ_{epoch}=0 : Invalid Δ_{epoch}>0 : Upgrade	Δ_{epoch}=0 : Invalid Δ_{epoch}>0 : Upgrade	Δ_{epoch}=0 : Invalid Δ_{epoch}>0 : Upgrade	
R	Δ_{epoch}=0 : No change Δ_{epoch}>0 : Purge cache	Δ_{epoch}=1 : Upgrade Δ_{epoch}>1 : Upgrade & purge cache. Δ_{epoch}=0 : Invalid.	Δ_{epoch}=1 : Upgrade Δ_{epoch}>1 : Upgrade & purge cache. Δ_{epoch}=0 : Invalid	Δ_{epoch}>0 : Purge cache
RH	Not allowed.	Δ_{epoch}=0 : No change Δ_{epoch}>0 : Purge cache	Δ_{epoch}=1 : Upgrade Δ_{epoch}>1 : Upgrade & purge cache. Δ_{epoch}=0 : Invalid	
RWH	Invalid	Invalid	Δ_{epoch}!= 0 : Invalid	

Persistent Handles – Lease State Validation on Client

- CREATE-RESUME request typically retains existing lease state.
  - OK to lose R leases.

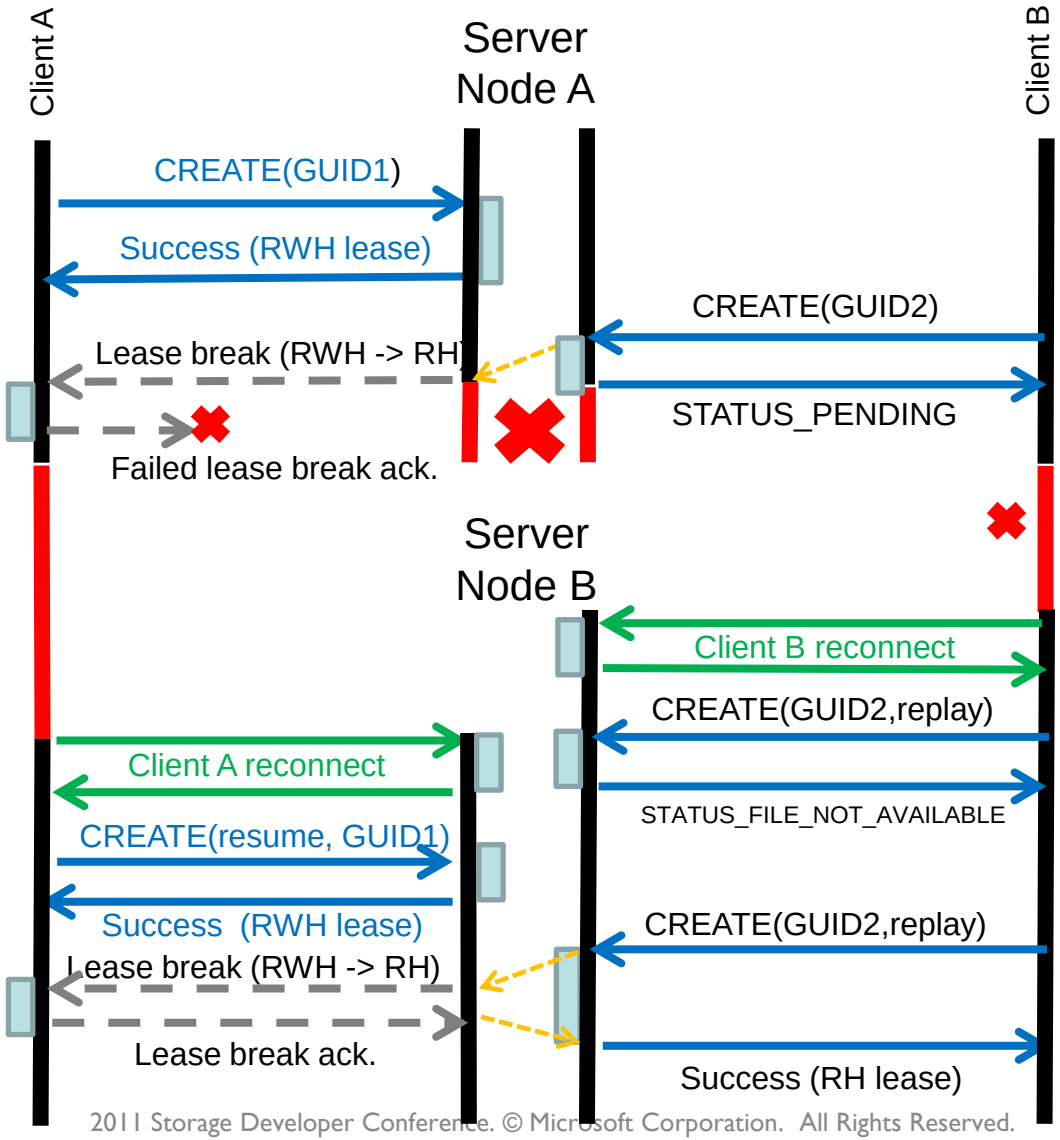
New Lease State→	R	RH	RWH	None
None	Invalid	Invalid	Invalid	
R	Δ_{epoch}=0 : No change Δ_{epoch}>0 : Purge cache	Invalid	Invalid	Δ_{epoch}>0 :Purge cache Δ_{epoch}=0 :Invalid
RH	Invalid	Δ_{epoch}=0 : No change Δ_{epoch}>0 : Purge cache	Invalid	Invalid
RWH	Invalid	Invalid	Δ_{epoch}=0 : No change Δ_{epoch}>0 : Invalid	Invalid

Persistent Handles – Handling Lease breaks : Network Disconnects



- Server holds the lease break until the client reconnects and resumes its Open.
  - Waits for up to the durable handle v2 timeout for the client to resume the handle;
  - Then waits for up to the oplock break timeout for the client to acknowledge the break.

Persistent Handles – Handling Lease breaks : Server Failures



- Server guarantees that any operations which break W or H leases are blocked.
- Clients request prior lease state when resuming its Opens.
  - Server MUST grant H and W lease if requested.
  - R lease may be denied forcing the client to purge cached data.
- When the operation which caused the lease break is replayed a new lease break is initiated by the server.

Should all Continuously Available Opens be Persistent?

- Persistent handles consume more server resources to keep persistent/volatile state
  - Sharing/Access modes and associated fencing info.
  - Byte range locks
  - Replay caches
- Opens which do not impact sharing, lease state or filesystem metadata need not be persistent.
  - Server "forgets" these opens when client disconnects.
  - Client re-opens the file before next use.
    - Best-effort.
    - No hard guarantees or reservations.

Should all Continuously Available Opens be Persistent?

- Metadata opens (READ_ATTRIBUTE)
  - Do not impact sharing, lease state or filesystem metadata. No fencing required.
  - Server can "forget" these opens when client disconnects.
  - Client re-opens the file before next use.
    - Best effort without any hard guarantees or reservations.

Should all Continuously Available Opens be Persistent?

- Directory enumeration handles
  - Have state (enumeration position and query template.)
  - Weak guarantees when directory content is changing.
  - Client re-opens directory, resets query template, restarts enumerations and skips entries.
    - Tricky when directory content is changing.
- Change notifications
  - We cheat ! STATUS_NOTIFY_ENUM_DIR

Witness Notification Protocol

Goals

- Accelerate client detection/notification of a server-side resource failure
- Notify client when relevant server resources have come online and ready for operation
- Provide a communication channel with the client for load balancing

Overview

- Optional additional service
- RPC based interactions
- Used in clustered environments for Continuously Available shares
- CA will function without witness, but witness should accelerate failover as well as enable new functionality

Witness Discovery


```
DWORD WitnessrGetInterfaceList(
    [in] handle_t Handle,
    [out] WITNESS_INTERFACE_LIST ** InterfaceList);
```

- Client requests list of interfaces from the file server node it is connected to
- Interfaces provide list of 3rd party "witness" that can provide notifications if this server interface experiences a failure

Witness Monitoring


```
DWORD WitnessrRegister(
    [in] handle_t Handle,
    [out] PCONTEXT_HANDLE_TYPE * ppContext,
    [in] [string] [unique] WCHAR * NetName,
    [in] [string] [unique] WCHAR * IpAddress,
    [in] [string] [unique] WCHAR * ClientComputerName);
```

- Client registers for resources of interest (network names, ip addresses)
- Client can unregister resources it is no longer interested in, or register additional resources as they become relevant
- Server returns registration key on successful register. Key is used for unregister operations.

Witness Notifications


```
DWORD WitnessrAsyncNotify(
    [in] handle_t Handle,
    [in] PCONTEXT_HANDLE_TYPE_SHARED pContext,
    [out] RESP_ASYNC_NOTIFY * pResp);
```

- Client posts one or more async notification requests
- If a resource for which the client has registered becomes unavailable or available, notification is returned to the client.

Client Operation

- On TREE_CONNECT to a continuously available share:
  - Register witness notification for netname (if not already monitored)
  - Register witness notification for IP(s)

Client Operation

- On notification of NetName offline
  - Disconnect/cancel pending operations and prepare for reconnect
- On notification of IP offline
  - Disconnect/cancel operations on that connection.
  - If multichannel, retry on alternate channel
  - If not multichannel, prepare for reconnect
- In both cases, start timer to retry if no online received within acceptable timeout (for witness server failure cases)

Client Operation

- On notification of NetName online
  - Initiate reconnect
  - If unsuccessful, enter normal reconnect retry loop
  - If successful, re-establish handles and retry pending operations.
- On notification of IP online
  - Same as above
  - Potentially rerun multichannel selection algorithm

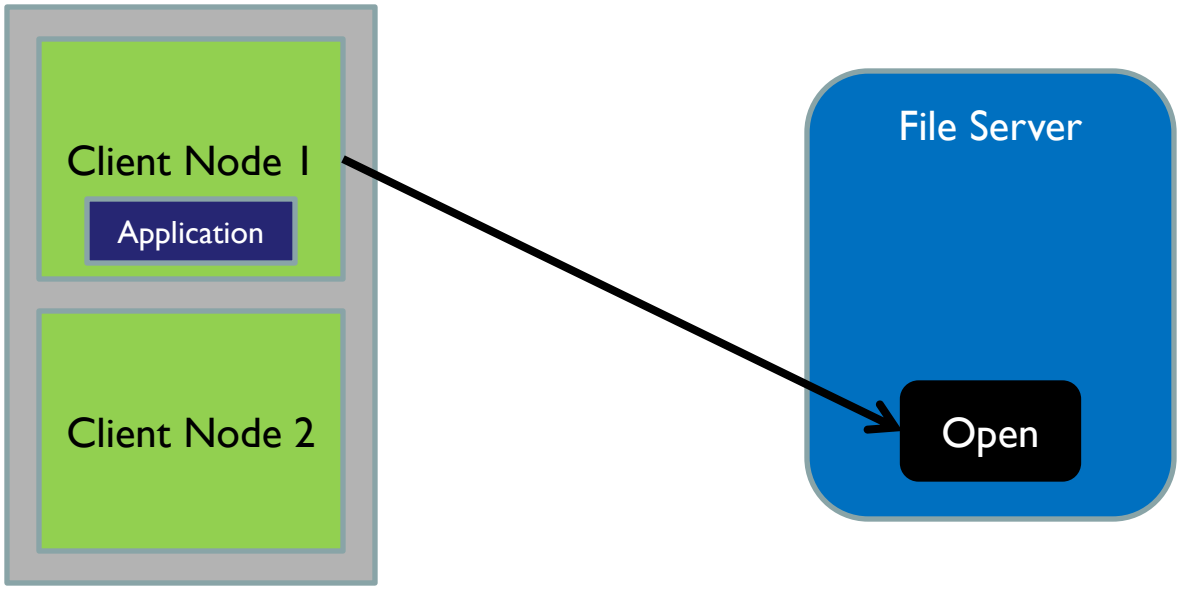
Load Balancing for Scale-Out

- Witness provides mechanism to request a client move from one interface to another
- For a scale-out server, local administrative actions permit dynamically moving clients between nodes.
- On move, client will:
  - Let existing operations complete
  - Disconnect
  - Connect to new target. If fails, reconnect to any available node.
  - Re-establish handles and resume operation

Clustered Client Failover

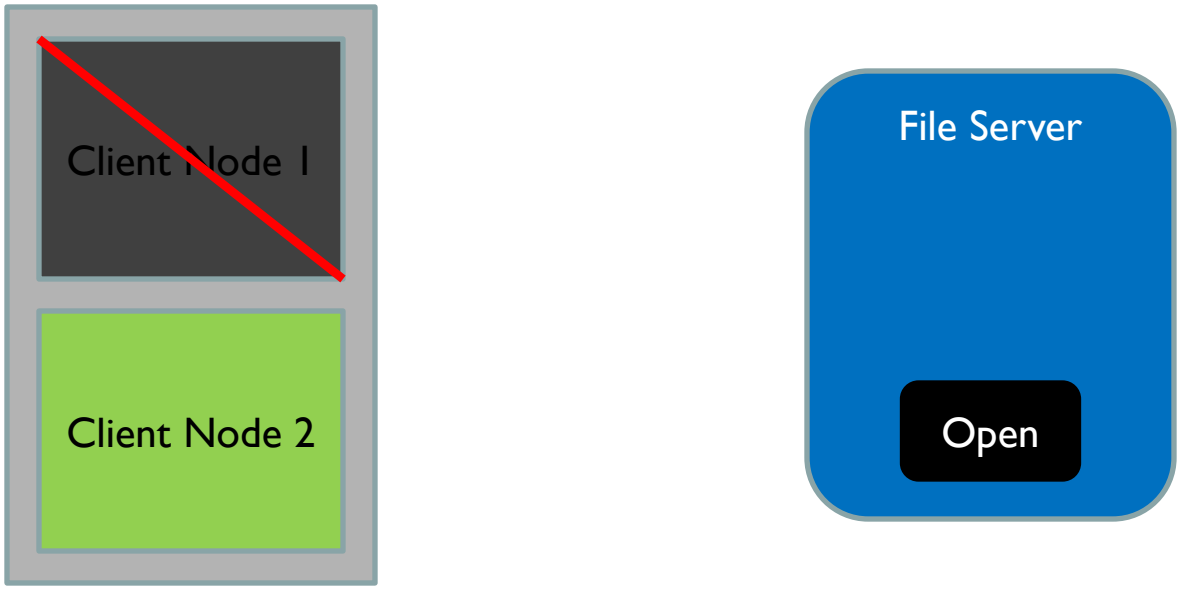
Scenario

- Highly Available application has a data file open exclusively on the file server



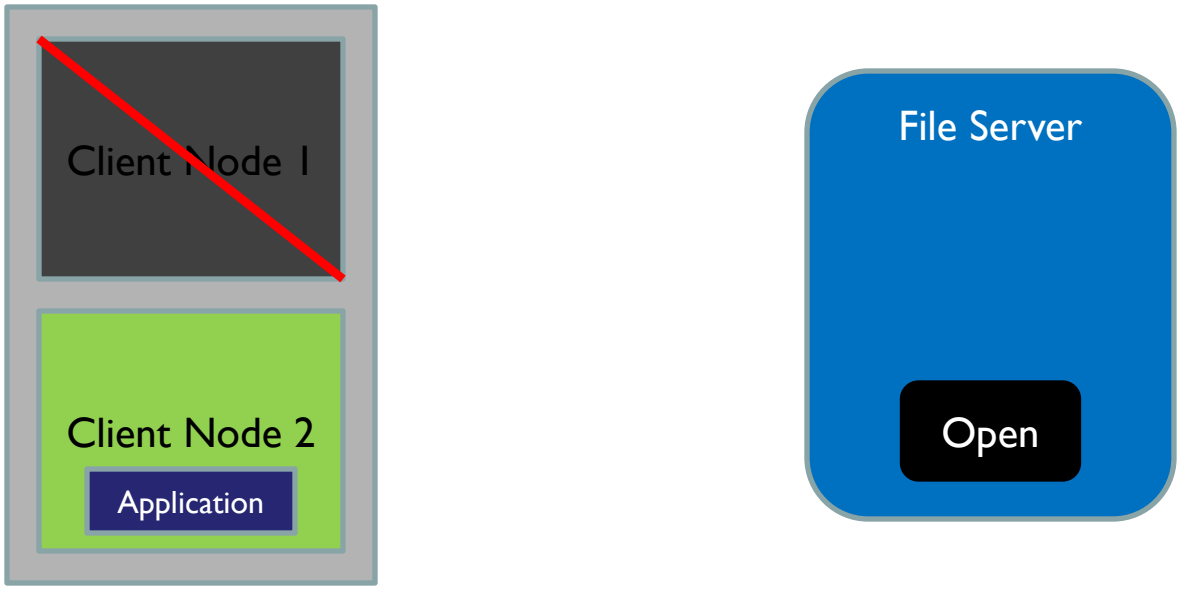
Scenario

- Client Node fails. Server holds reservation for file as client has disconnected.



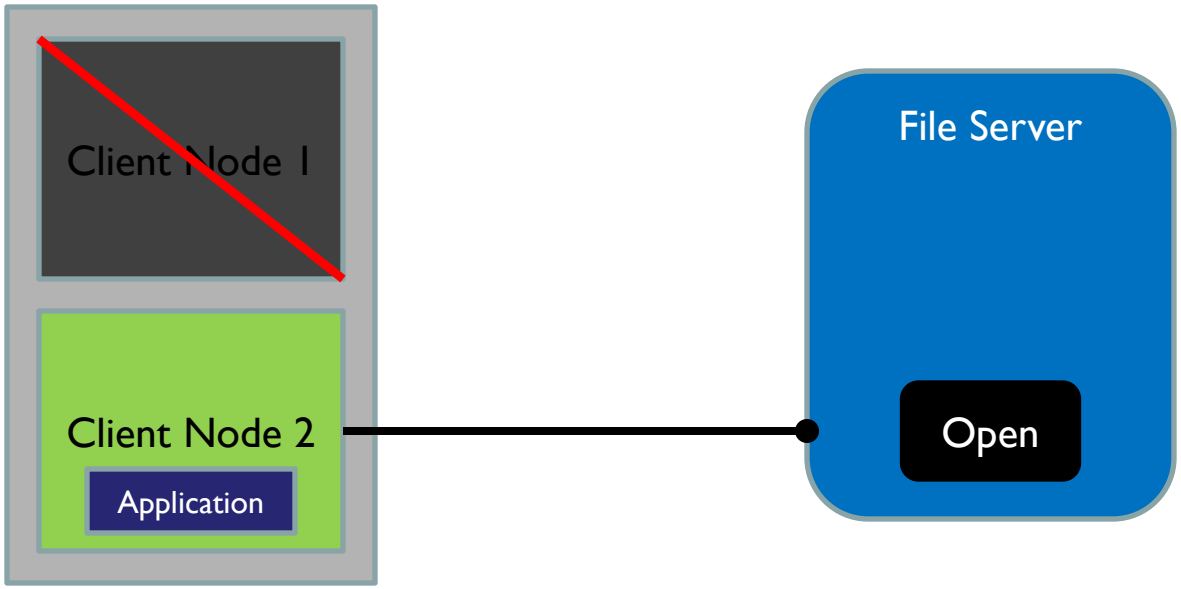
Scenario

- Application is relaunched on second node



Scenario

- Application attempts to open file, server rejects it as file is reserved for original client for timeout period.



Goals

- Optionally identify an open as associated with an instance of an application
- When a highly available application fails over, use of the instance identifier by a new client for a given file allows for accelerated teardown of existing opens to prevent delay in access

Instance Create Context


```
typedef struct _SMB2_CREATE_APP_INSTANCE_ID {
    //
    // This must be set to the size of this structure.
    //
    USHORT Size;

    //
    // This must be set to zero.
    //
    USHORT Reserved;

    //
    // The caller places a GUID that should always be unique
    // for a single instance of the application.
    //
    GUID AppInstanceID;
} SMB2_CREATE_APP_INSTANCE_ID;
```

- App instance is provided by application
- Application guarantees that instance ID is consistent as application moves

Server Side Invalidation


```
If (AppInstance Create Context present )
    If (User has access to the underlying file )
        If (Open exists with matching instance ID)
            Close previous open
        End If
    End If
End if
```

Application Requirements

- Application provides instance identifier (GUID) on a per-process or per-open basis
- Application guarantees that instance identifier remains consistent as application moves between client nodes
- Application guarantees single instance of application is running within the cluster

Windows Server 8 and SMB 2.2 – Advancements in Management Capabilities

Is there life beyond NetShareAdd?

Wednesday, 3:05-3:55

Jose Barreto

Pulling it Together

- Demo – Claus Joergenson

Questions?

Thank you!