Exploiting the High Availability features in SMB 3.0 to support Speed and Scale

James Westland Cain, Ph.D.
Quantel Limited

Biographies

- James:
  Fulltime Software Architect (aka Coder :-)
  Occasional Academic
- Quantel:
  40 years old technology innovator in TV & Film.
- Built bespoke disk systems for nearly 30 years
- Customers include: ESPN, BBC, Fox Sports
- Films include: Star Wars, Lord of the Rings, Slumdog Millionaire, Avatar, Titanic 3D & Oz

Agenda

- Analysing SMB3 architecture & Windows client implementation for speed and scale features.
- Background: Filling Pipes & BDP.
- Background: SMB 1/2/3 Feature History w.r.t. Speed.
- Single vs. Multiple Machines & Shared State.
- Demos
- Analysis
- Conclusion

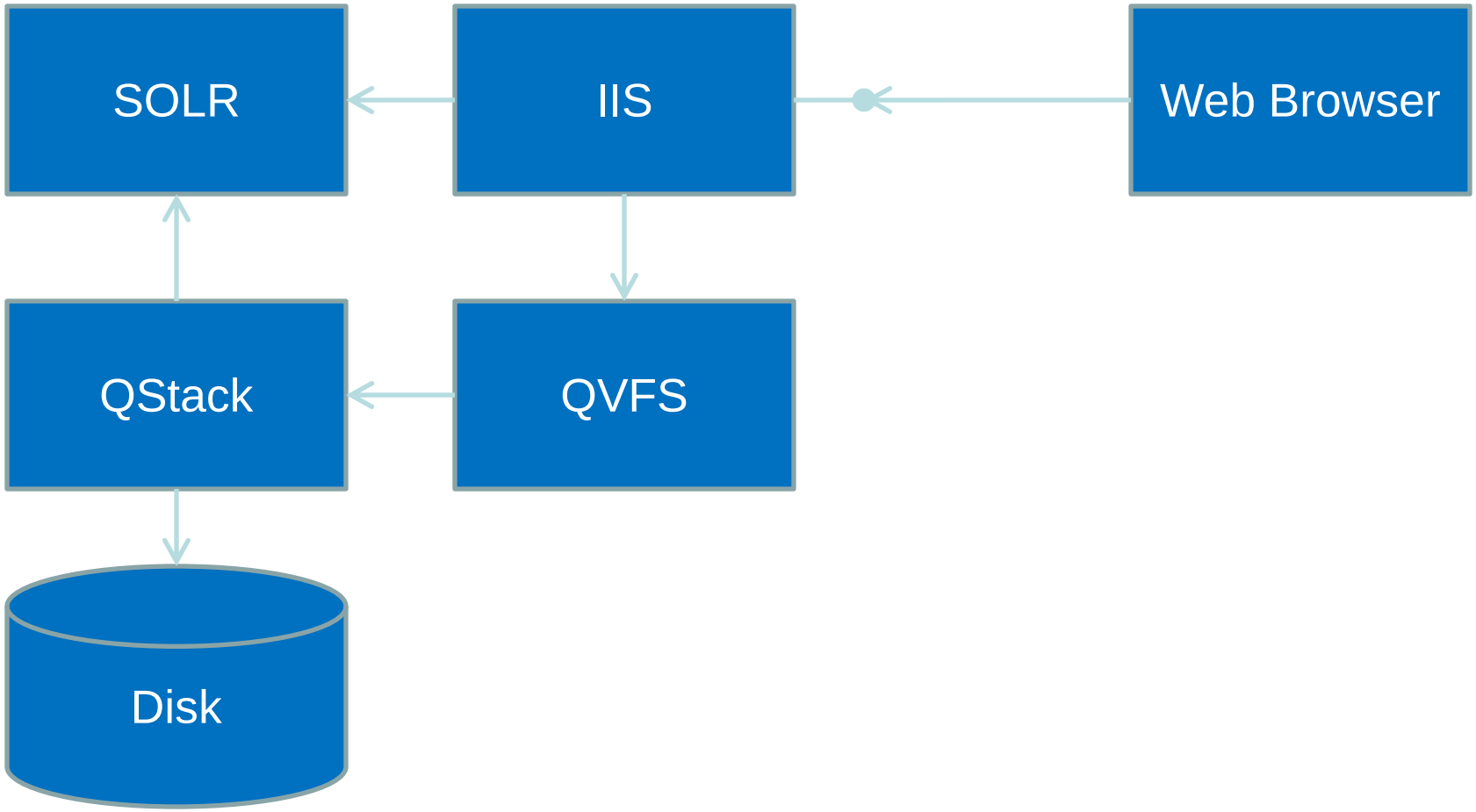
RESTful Recap

- "RESTful Filesystems" SDC 2010
- What is REST?
  - Resources
  - Self contained requests
- The path to the file contains instructions (application state)
- Allows for massive scale be storing state in URLs (ie clients - but they don't know it) instead of in servers.

Demo

- RESTful SMB3
- A read request is a command
- User mode SMB3 server running on Windows
- Written using Visual C++

Demo Components



Filling Pipes

- Speed (throughput) vs Bandwidth vs Latency
- Bandwidth Delay Product (BDP)
  - Bandwidth (b/s)
  - Delay (s)
  - Product: Bandwidth * Delay
    - (b/s) * (s) = b.
- Units of BDP is Bytes
    - it's a measure of size (not time!)

Filling Pipes

- Keeping packets back to back on the wire.
- However: TCP Window size used to *limit* number of packets on the wire (avoiding dropped packets).
- Mitigate by allowing concurrent tcp sessions on one wire.
- Mitigate by using multiple wires.
  - Multiple wires can connect the same machines (multiple NICs)
  - Multiple wires can connect to multiple servers

Filling Pipes: SMB 1 – XP era

- Remember “NT LM 0.12”?
- Filling pipes required application participation.
- To fill a Gb Ethernet connection I ran 4 read requests in parallel.
- Could transfer 110-112MBps sustained.
  - 1 Session would do about 60MBps.
- Getting FCP-7 to play over SMB1 was hard.
- Had no other choice than to reduce latency in the SMB1 layer using heuristics such as read ahead in the server.

Filling Pipes: SMB 2.0 – Vista era

- Re-write (hurrah!)
- Added credits. Thus the *protocol* acknowledged for the first time that latency could be overcome by overlapped IO.
- The SMB2 server can control overlapped requests using credits.
- Windows clients require a minimum of 4 credits*

* [MS-SMB2].pdf: Appendix A: Product Behavior: <79> Section 3.2.4.1.2

- Thus client application could make normal requests and yet fill the pipe.

Filling Pipes: SMB 2.1 – 7 era

- Add large MTU to enable filling faster pipes such as 10Gb Ethernet.
- Multi-credits only used for read, write, and directory enumeration*

* [MS-SMB2].pdf: Appendix A: Product Behavior: <87> Section 3.2.4.1.5

- For all other requests, a Windows-based client uses credit charge 1, even if the payload size of a request or the anticipated response is greater than 64 kilobytes*

* [MS-SMB2].pdf: Appendix A: Product Behavior: <87> Section 3.2.4.1.5

Filling Pipes: SMB3 – 8 era

- Added Multi-Path support (active/active)
- Enables:
  - RSS – multiple ports on one transport
  - Multiple Nics - multiple IP addresses
  - Multiple Machines - multiple IP addresses
  - Different Transports – RDMA

Filling Pipes: SMB3 – 8 era

- Added Multi-Path support (active/active)
- Enables:
  - RSS – multiple ports on one transport
  - **Multiple Nics - multiple IP addresses**
  - **Multiple Machines - multiple IP addresses**
  - Different Transports – RDMA

Filling Pipes: SMB3 – 8 era

- Added lots of other features too: mainly for high availability and fault tolerance…
- … but also security including forcing server to implement signing!
- Signing required for:
  - FSCTL_VALIDATE_NEGOTIATE_INFO
  - FSCTL_QUERY_NETWORK_INTERFACE_INFO

SignKey derivation


```
std::vector<unsigned char> signData; //Make signData for SignKey derivation.
{
  std::string label = "SMB2AESCMAC";
  std::string context = "SmbSign";

//KDF in Counter Mode, 'r' of 32, 'L' of 128, PRF HAMC-SHA256.
  unsigned char val1[4] = {0,0,0,1};   //unsigned int 1;
  unsigned char val2[1] = {0};         //unsigned char 0;
  unsigned char val3[4] = {0,0,0,128}; //unsigned int 128;

//unsigned int 1; [i]2, where i = n, where n = [L/h], = 1. h = 128, L = 128.
  signData.insert(signData.end(), &val1[0], &val1[4]);
  signData.insert(signData.end(), &label[0], &label[label.size()]); //label
  signData.insert(signData.end(), &val2[0], &val2[1]);//Nul terminator for label
  signData.insert(signData.end(), &val2[0], &val2[1]); //Separator
  signData.insert(signData.end(), &context[0], &context[context.size()]); //context
  signData.insert(signData.end(), &val2[0], &val2[1]); //Nul terminator for context
//unsigned int 128; [L]2, where L =128
  signData.insert(signData.end(), &val3[0], &val3[4]);
}
```

References: 3.2.5.3.3 & 3.1.4.2 in MS-SMB2.pdf, NIST SP800-108 section 5.1

SignKey derivation

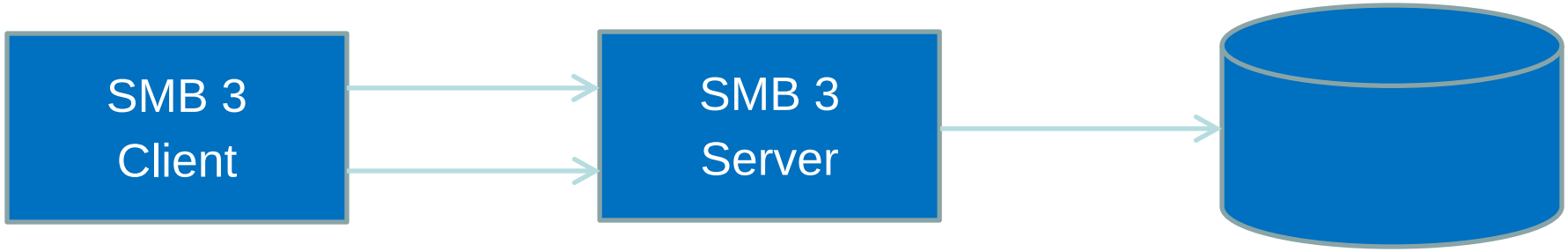

```
Buffer hashValue;
{
  AlgorithmProvider algorithmProvider;
  NT_VERIFY(algorithmProvider.Open(BCRYPT_SHA256_ALGORITHM,
    0, // implementation
    BCRYPT_ALG_HANDLE_HMAC_FLAG)); // flags

  Hash hash;
  NT_VERIFY(hash.Create(algorithmProvider,
    &sessionKey[0], // secret
    sessionKey.size(), // secret size
    0)); // flags
  NT_VERIFY(hash.HashData(&signData[0],
    signData.size(),
    0)); // flags

  ULONG hashValueSize = 0;
  NT_VERIFY(hash.GetHashSize(hashValueSize));
  NT_VERIFY(hashValue.Create(hashValueSize));
  NT_VERIFY(hash.FinishHash(hashValue.GetData(),
    hashValue.GetSize(),
    0)); // flags
}
```

Reference: Applying Cryptography Using The CNG API In Windows Vista, Kenny Kerr, MSDN Magazine, July 2007. http://msdn.microsoft.com/en-us/magazine/cc163389.aspx

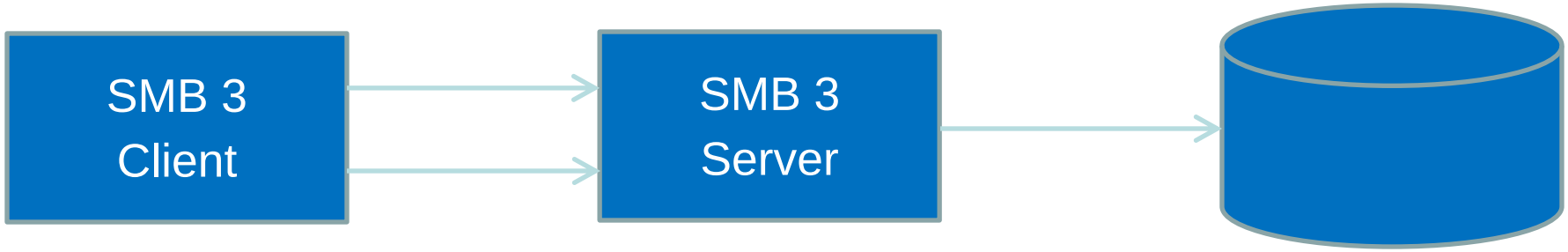
SMB3 Multi-Path



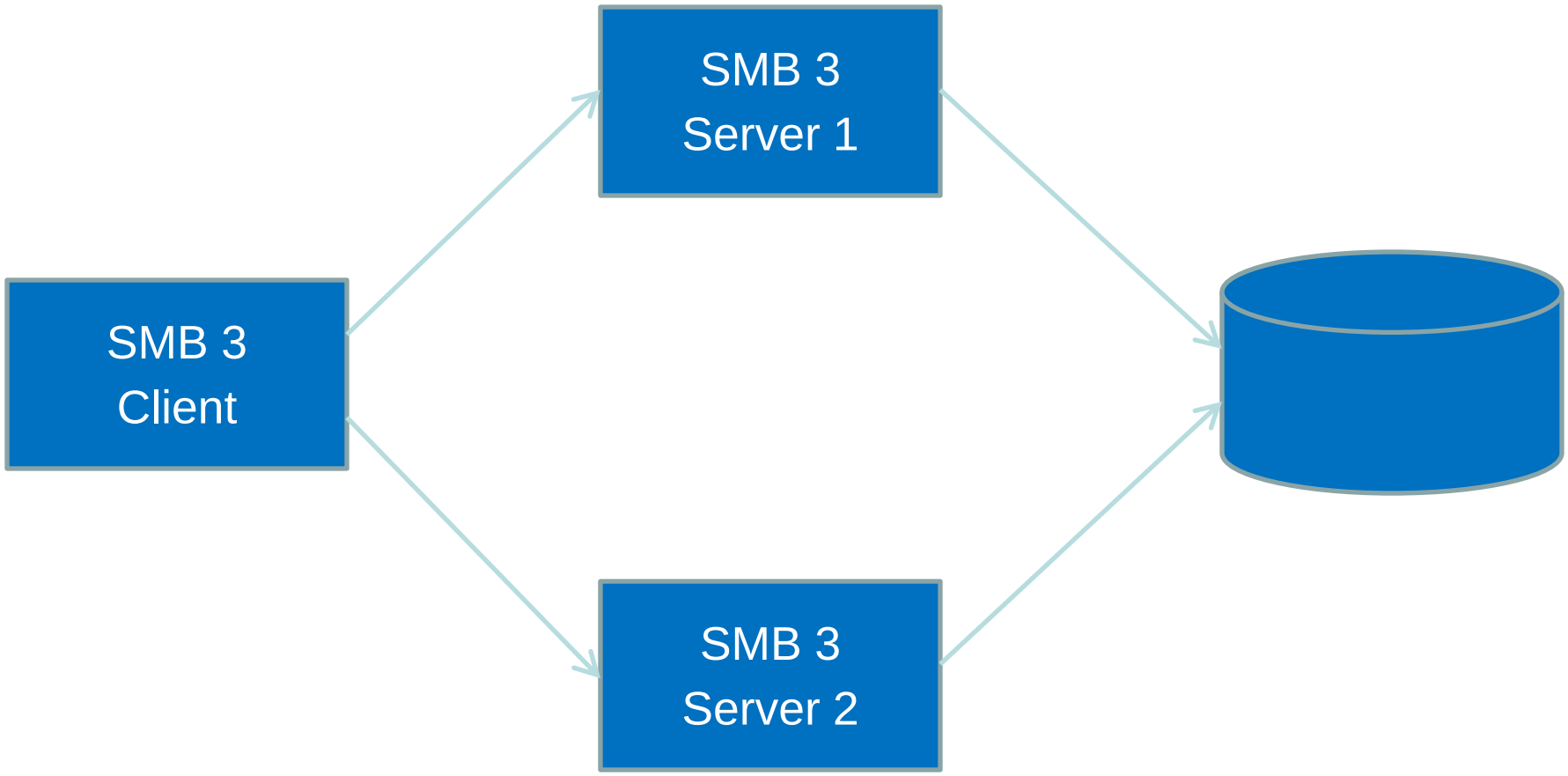
Demos – Multi Path Requests

- The Filesystem has DirectX GUI (it's part of a video editor – but with a filesystem on top instead of a GUI :-)
- We use GPUs to render, so our SMB3 server has Cuda compute built in too.
- Realtime Visualisation Tool used for optimising
  - "Software Oscilloscope"

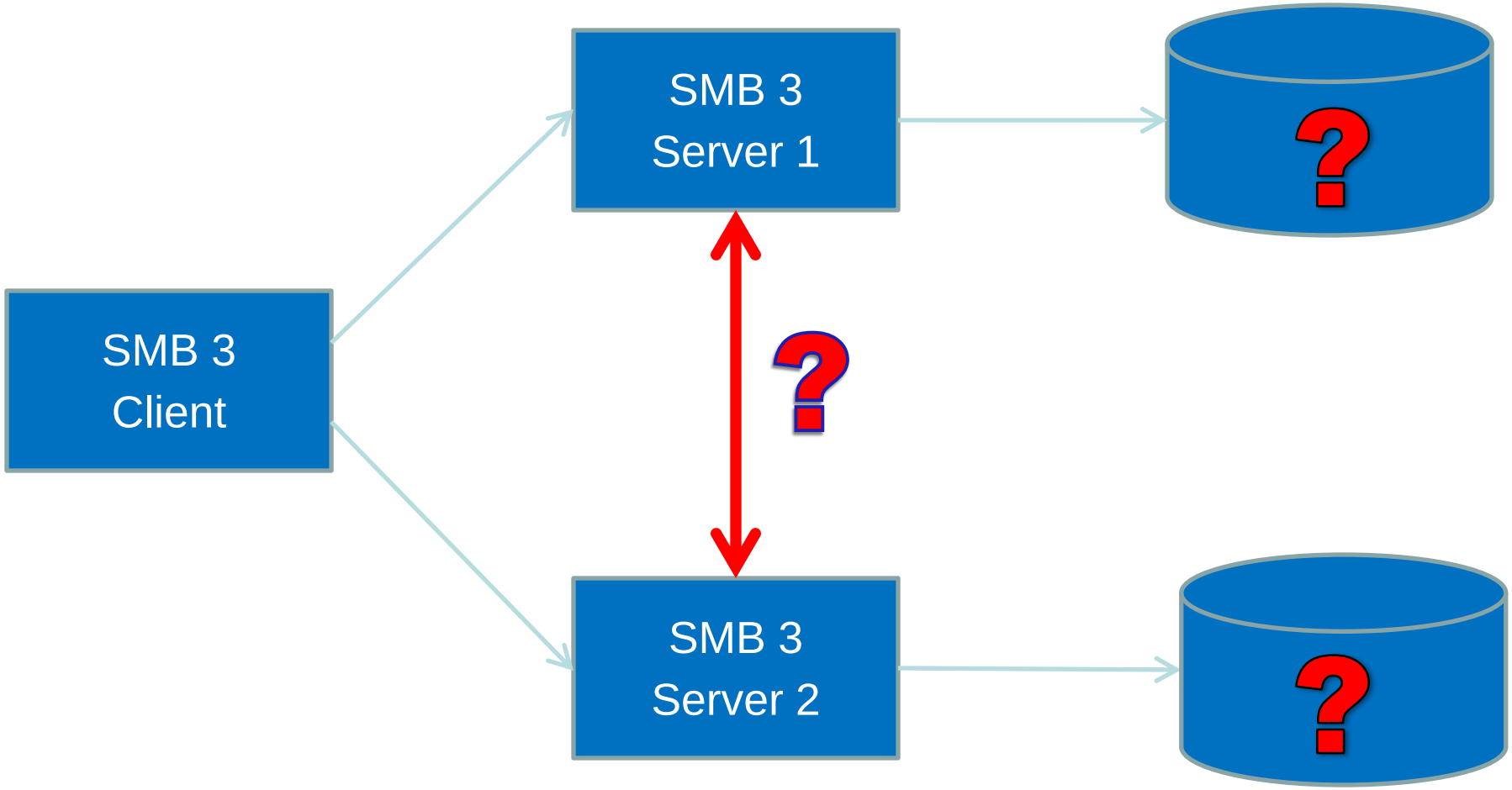
SMB3 Multi-Path



SMB3 Multi-Machine



Assumed Shared State in SMB3?



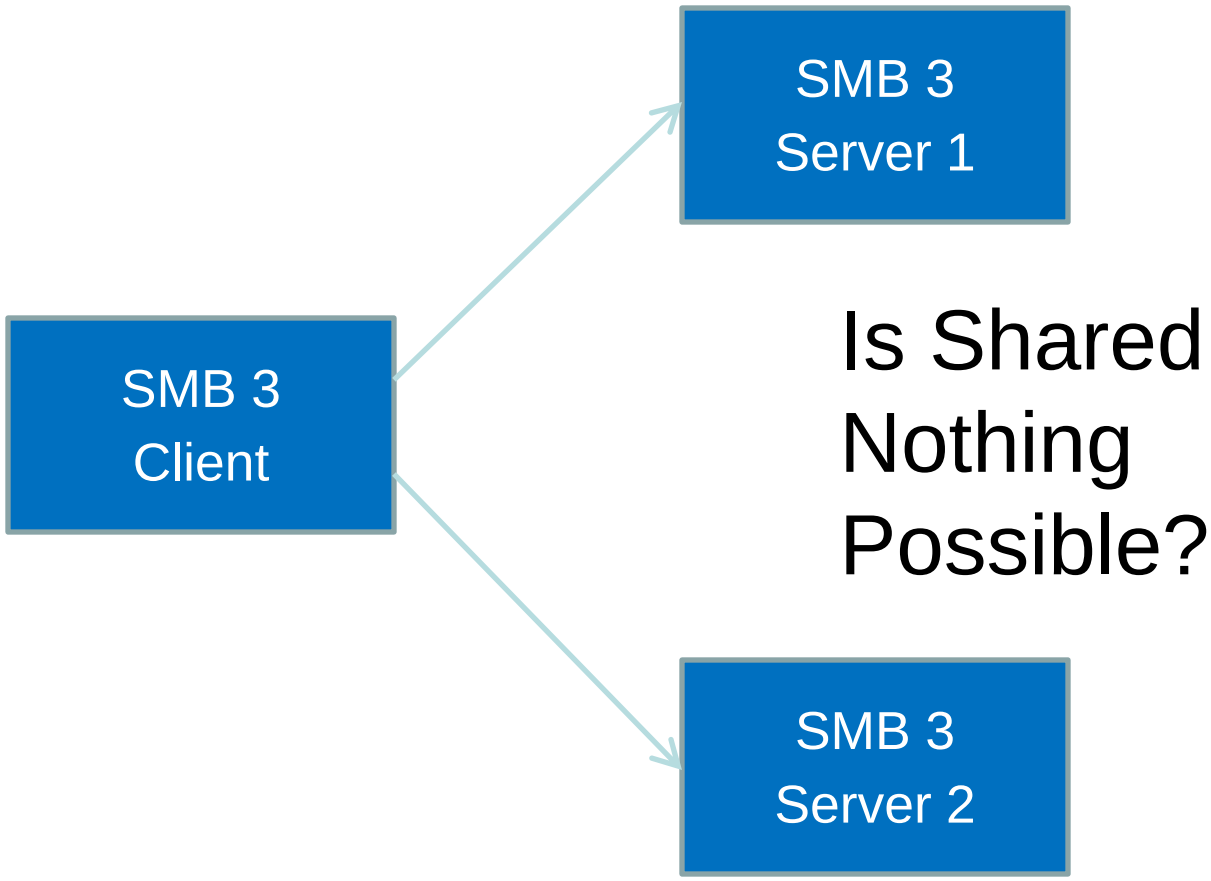
Second NIC Initiation

	Summary
	O Negotiate, ClientGuid = 605663fe-1e2a-11e3-be94-00155d000
	O Session Setup, SessionFlags = 0, Status = STATUS_MORE_PRO
	O Read, FID = 0x7, Read 65536 bytes from offset 53477376, S
	O Read, FID = 0x7, Read 65536 bytes from offset 53608448, S
	O Read, FID = 0x7, Read 65536 bytes from offset 53542912, S
	C Read, FID = 0x7, File = process\big.iso
	C Read, FID = 0x7, File = process\big.iso
	C Read, FID = 0x7, File = process\big.iso
	C Read, FID = 0x7, File = process\big.iso

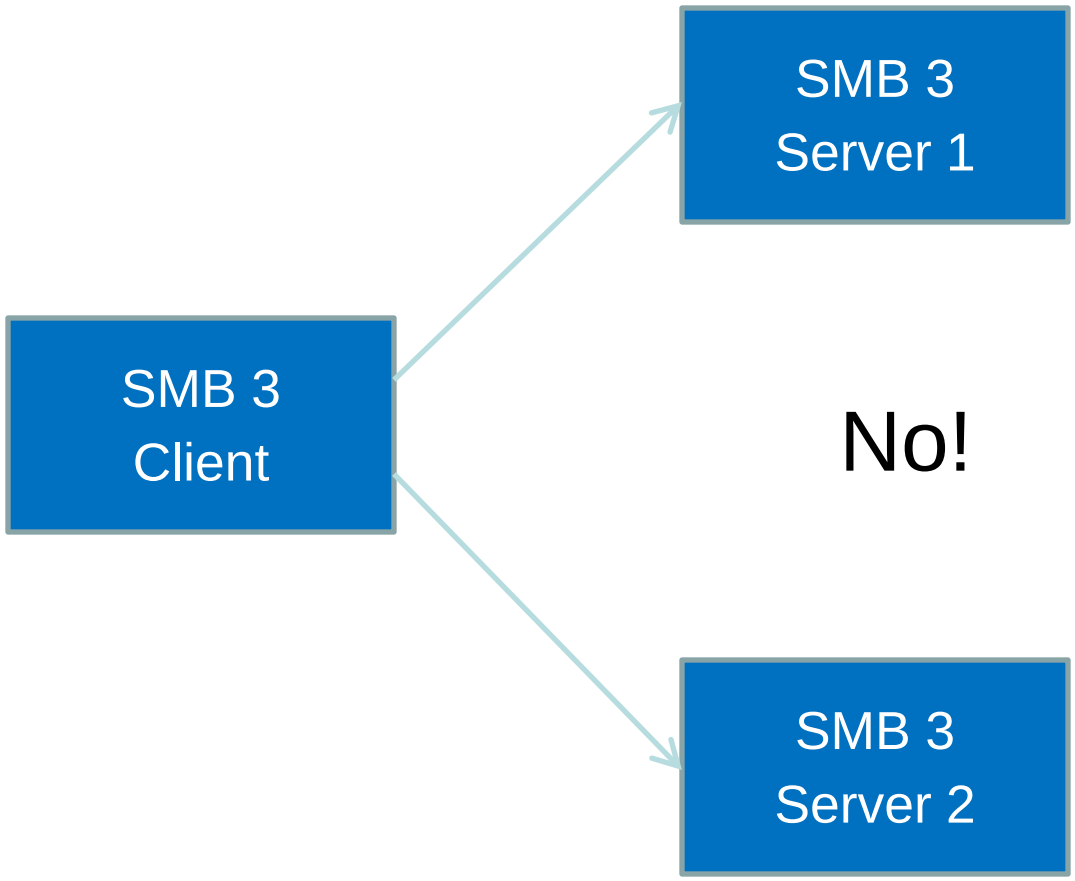
SMB3 Protocol

- SMB3 Multi-channel makes no distinction between NICs on a single server and NICs on multiple servers.
- On opening a new channel over a second NIC there is an assumption that all session state is available – as it is on a single process.
- Session state includes all file handles.

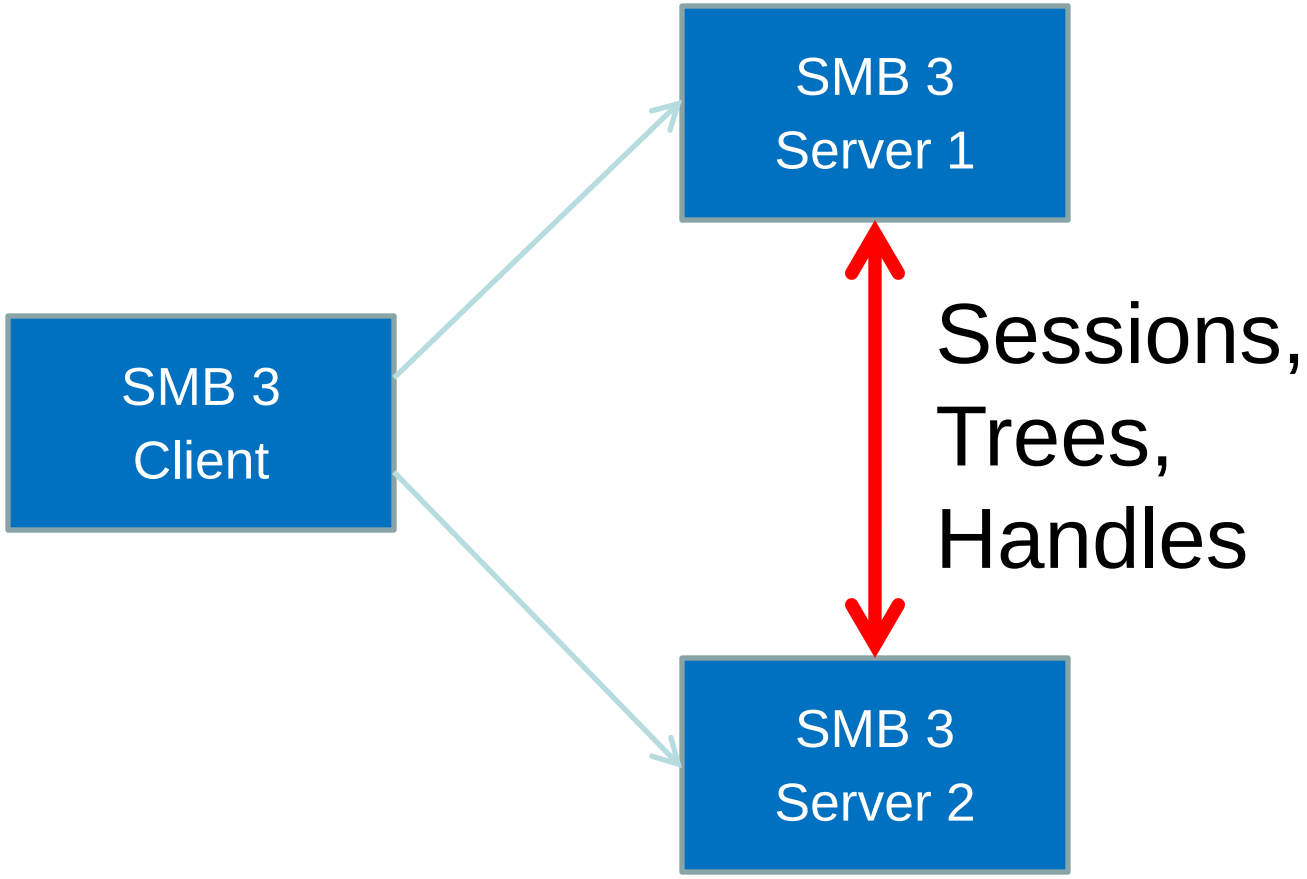
Shared Nothing in SMB3?



Shared Nothing in SMB3?



Shared State in SMB3



Conclusion

- SMB 2++ is getting massive traction in client OS redirectors – it's a great echo-system to be in.
- MS have pushed to make SMB 3 fit for big data in data centres
- Simple use cases (subset: non shared locking) are well supported by the protocol
- SMB 3 has a quite high cost of entry (but lower than writing an IFS in kernel mode)
- There are limits to how far SMB 3 can scale in a big data world

Learning Objectives

- Understanding SMB 3.0 at an architectural level.
- Practical insight into network topologies for NAS.
- Finding the minimal shared state between SMB servers in a cluster.
- Issues in writing an SMB server in User Mode running on Windows.