

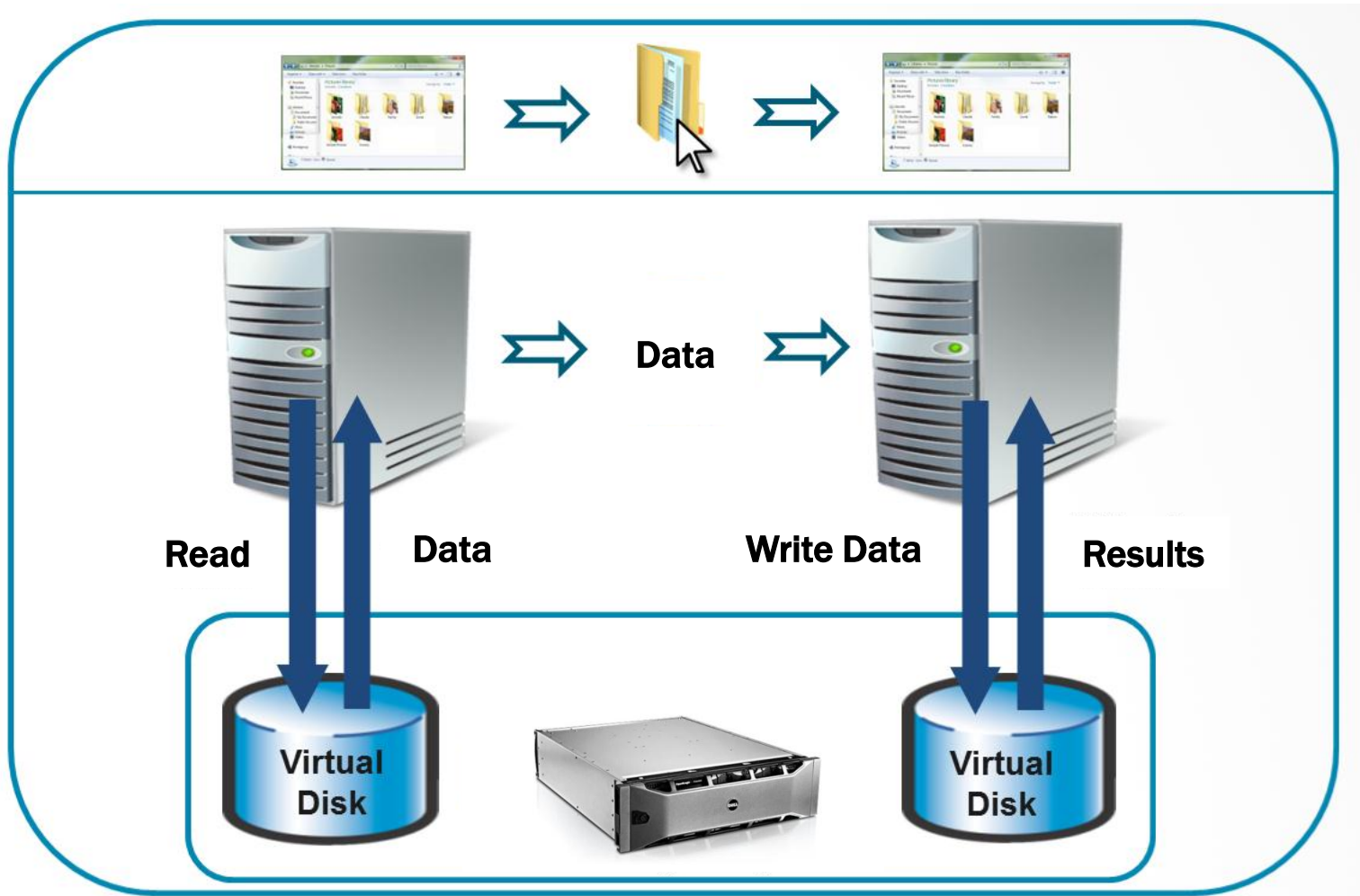
# **Windows 8 File System Performance and Reliability Enhancements in NTFS**

**Neal Christiansen**  
**Principal Development Lead**  
**Microsoft**

- ❑ Offload Data Transfers (ODX)
- ❑ CHKDSK Overhaul
- ❑ Reliability using Flush instead of FUA (Forced Unit Access)
- ❑ Changes in Short Filename Support
- ❑ Trim Enhancements
- ❑ Intelligent Error Correction
- ❑ Storage Optimizer
- ❑ Questions?

# Offload Data Transfers (ODX)

# Data Movement Today



# Data Movement Today

- ❑ Reads & Writes well understood
- ❑ Works well with OS Security Model
  - ❑ Security checks occur at open time
- ❑ Works well with application programming model

# Inefficiencies with Today's Model

- ❑ Data flowing out and back into the same storage system
- ❑ Data movement bottlenecked by the network transport
- ❑ Data movement consumes CPU and Memory on host(s)
- ❑ **There must be a better way to do this!**

# Offload Data Transfer (ODX)

- ❑ Takes advantage of advanced capabilities present in many of today's storage arrays to enable efficient data movement
- ❑ Rather than pass the data around, passes around a token which represents a point in time view of the data
- ❑ Support cross-machine and cross-subsystem data movement, while not constrained by protocol, transport, or geo-boundaries
- ❑ Maintains well understood OS-defined security framework
- ❑ Offers an easy & familiar programming model for developers
- ❑ Enable (even untrusted) applications to participate in efficient data movement

# Reading the Data: FSCTL\_OFFLOAD\_READ

- ❑ Instructs Storage to generate and return a “Token” which represents an immutable point-in-time view of the DATA in the desired region of the LUN
  - ❑ Token completely managed by Storage (Opaque to OS)
- ❑ Functionally equivalent to a normal “read” operation:
  - ❑ Operation behaves like a non-cached read (must be sector aligned)
  - ❑ Breaks oplocks using the same rules as read
  - ❑ Processes byte range locks using the same rules as read
  - ❑ Generates an IOCTL\_STORAGE\_MANAGE\_DATA\_SET\_ATTRIBUTES specifying DeviceDsmAction\_OffloadRead

# Writing the Data: FSCTL\_OFFLOAD\_WRITE

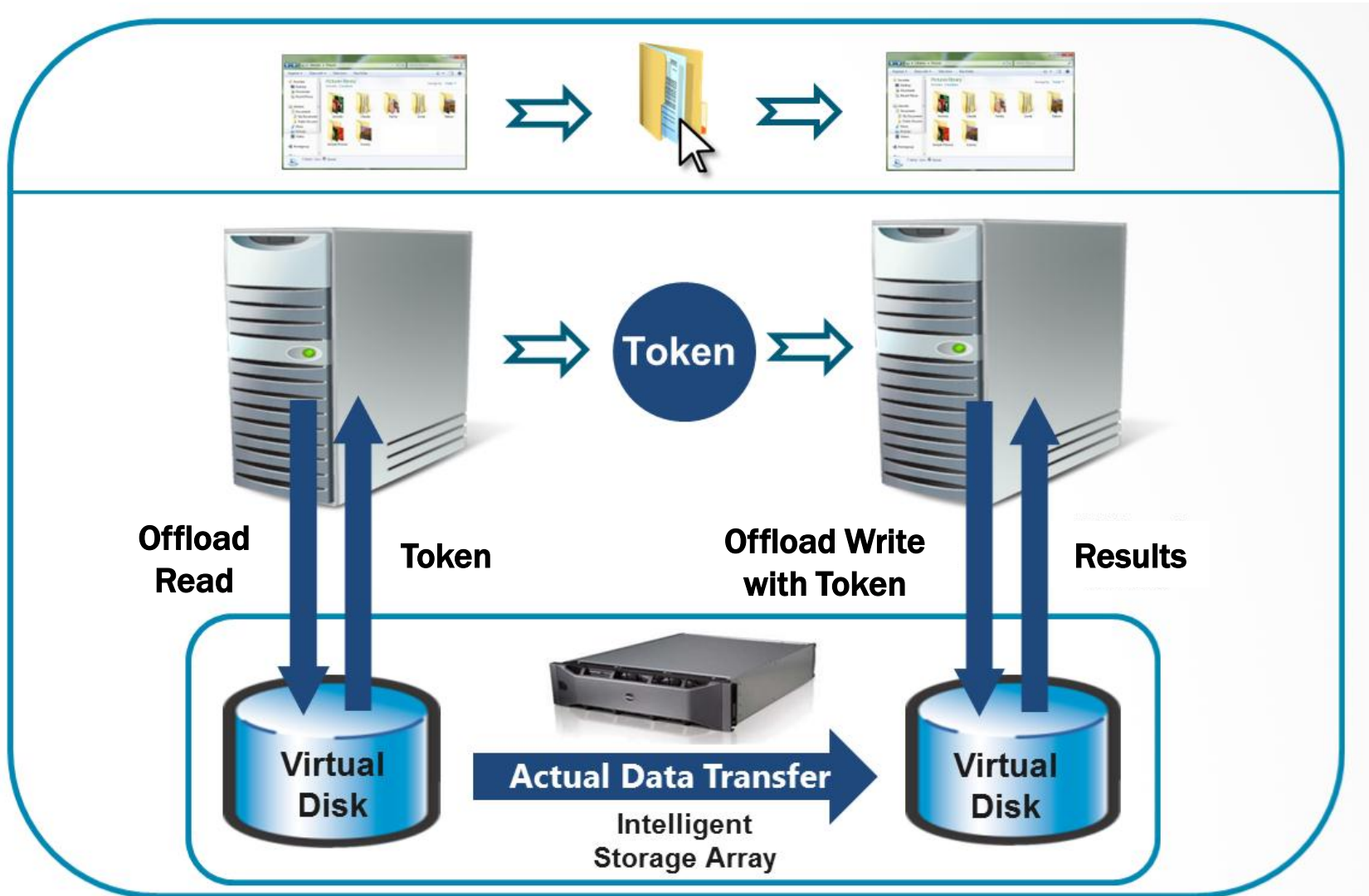
- ❑ Given a Token, the Storage attempts to independently execute data movement to the desired region of the LUN
  - ❑ Attempts to recognize Token
  - ❑ Determines where the DATA for the Token is located
  - ❑ Determines if the data movement is possible
  - ❑ Performs the data movement
  - ❑ All of this happens without OS intervention

# Writing the Data:

## FSCTL\_OFFLOAD\_WRITE

- ❑ Functionally equivalent to a normal “write” operation
  - ❑ Operation behaves like a non-cached write (must be sector aligned)
  - ❑ Breaks oplocks using the same rules as write
  - ❑ Processes byte range locks using the same rules as write
  - ❑ Updates the USN Journal with a USN\_REASON\_DATA\_OVERWRITE record
  - ❑ Limitation: does not allocate disk space (space must be pre-allocated)
  - ❑ Generates an IOCTL\_STORAGE\_MANAGE\_DATA\_SET\_ATTRIBUTES specifying DeviceDsmAction\_OffloadWrite

# Overview



- ❑ Enables offloaded transfers between LUNs, arrays, or data centers
  - ❑ Supported to the same volume on the same machine
  - ❑ Supported across different volumes on the same machine
  - ❑ Supported across different volumes on different machines via SMB
  - ❑ Supported by Hyper-V
- ❑ Integrated into the Win32 CopyFile API
  - ❑ Any component that uses this API will automatically use ODX when available
  - ❑ If ODX is not supported, normal read/write copy semantics are used
  - ❑ Supported by copy, xcopy, robocopy, as well as Explorer drag and drop

# ODX Limitations

- ❑ Only supported by NTFS
- ❑ Not supported on NTFS compressed files
- ❑ Not supported on NTFS encrypted files
- ❑ Not supported on sparse files
- ❑ Not supported by BitLocker

# File System Filter Opt-in

- ❑ Implemented a generic feature opt-in model for file system filters
  - ❑ Mitigates compatibility issues with file system filters that are not aware of an impactful new feature (ODX in this case)
  - ❑ Filters identify at installation time if they are aware of a given feature by setting state in the new “SupportedFeatures” registry key value
  - ❑ When an ODX operation is issued the IoManager checks all of the filters on the file system stack to determine if the given feature is supported or not
    - ❑ If not the operation is failed
    - ❑ This state is cached and is reset when a filter loads or unloads
    - ❑ Supported for both mini and legacy file system filters
    - ❑ “fltmc instances” will show the opt-in state of all filters for a given volume

- ❑ Implemented using new XCOPY LITE SCSI command
  - ❑ Part of T10 11-059r8 specification
  - ❑ Review Frederick Knight's presentation from earlier today on "Storage Data Movement Offload"
- ❑ Requires an advanced storage array that supports this new functionality
  - ❑ Currently working with several array vendors
  - ❑ Talk with your array vendor to determine their support for this command
  - ❑ If you are an array vendor and are interested in supporting this please contact me

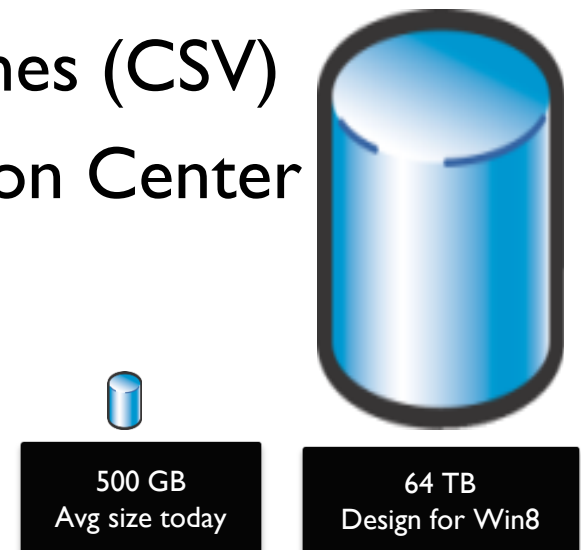
# CHKDSK Overhaul

# NTFS Volume Scalability

- ❑ NTFS supports volumes up to 256TB in size
- ❑ But the practical volume size is smaller based on CHKDSK execution time
  - ❑ CHKDSK scales based on the number of files on the volume (not the size of the volume)
- ❑ CHKDSK execution time has improved (decreased) with every windows release since Windows 2000
  - ❑ But there is a limit to what additional improvements could be made with the current execution model

# A new approach for detecting and repairing NTFS corruptions

- ❑ Improved detection and handling of corruptions in NTFS
- ❑ Change the CHKDSK execution model
  - ❑ Separate analysis and repair phases
- ❑ Integrated with Cluster Shared Volumes (CSV)
- ❑ File system health monitored via Action Center and Server Manager



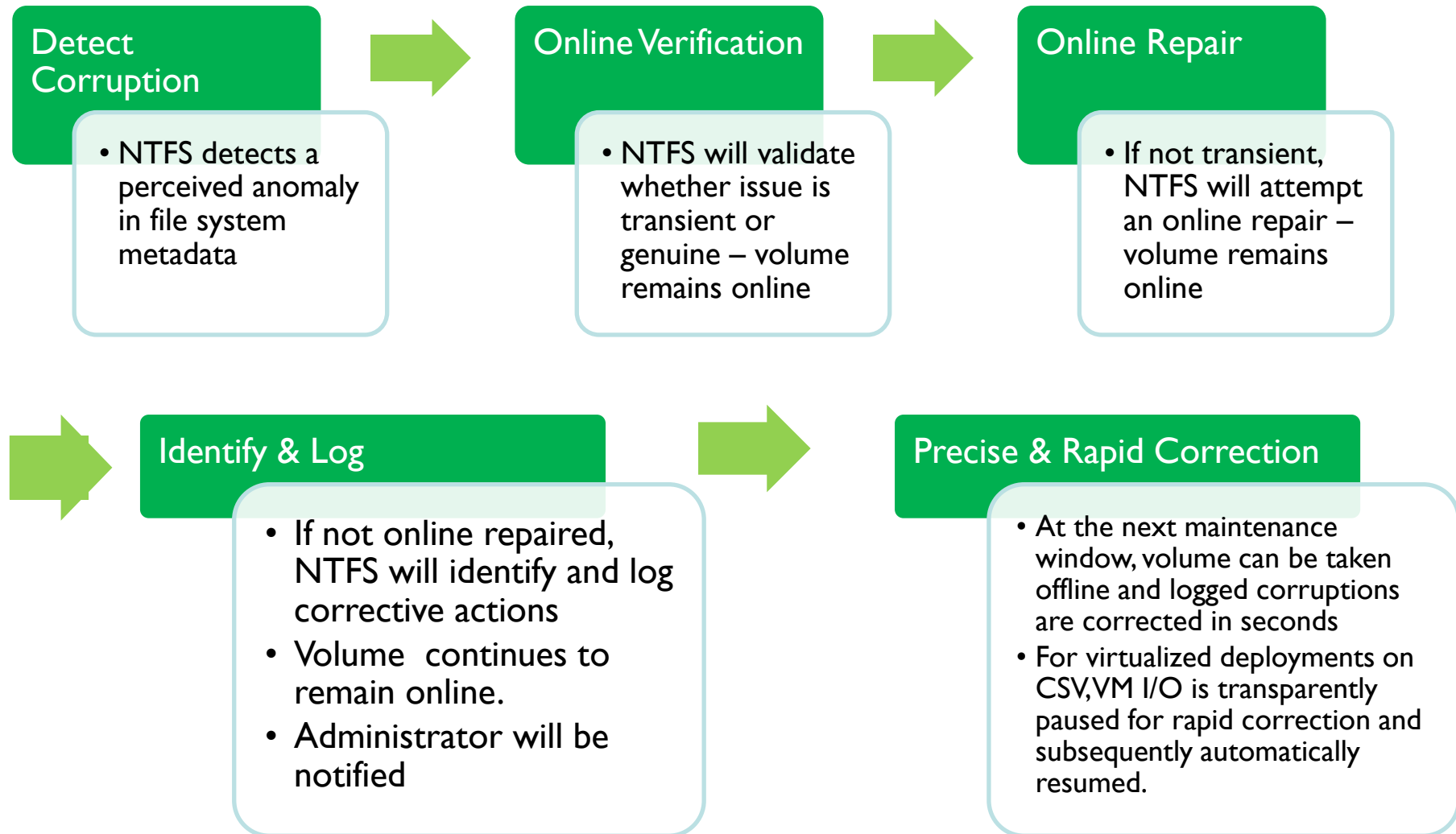
# Enhanced NTFS Corruption Handling

- ❑ NTFS has always been able to detect corruptions but its response in the past was to mark the volume “dirty” (corrupt)
- ❑ It now logs information on the nature of a detected corruption
  - ❑ Maintained in new metadata files
  - ❑ Enhanced event logging which includes more detailed information
  - ❑ New “Verification” component which confirms the validity of a detected corruption
    - ❑ Eliminates unnecessary CHKDSK runs
- ❑ Enhanced on-line repair
  - ❑ Self-healing feature introduced in Vista
    - ❑ Was limited to MFT related corruptions
  - ❑ Enhanced to handle a broader range of corruptions across multiple metadata files
    - ❑ Can do on-line repair of most common corruption scenarios

# A new model for CHKDSK

- ❑ The analysis phase is done online on a volume snapshot which maintains volume availability
  - ❑ If a corruption is detected:
    - ❑ First attempt an online repair via the self-healing APIs
    - ❑ If self-healing can not do the repair the detected corruption is logged to a new NTFS metadata file: \$Corrupt
    - ❑ All logged corruptions are verifiable
- ❑ Offline repair phase (spot fixing) if needed
  - ❑ Volume can be taken offline at administrator's discretion
  - ❑ Only repairs logged corruptions to minimize volume unavailability
    - ❑ Normally takes seconds to repair
- ❑ Online spot-fixing of errors with no down-time enabled via integration with Cluster Shared Volumes

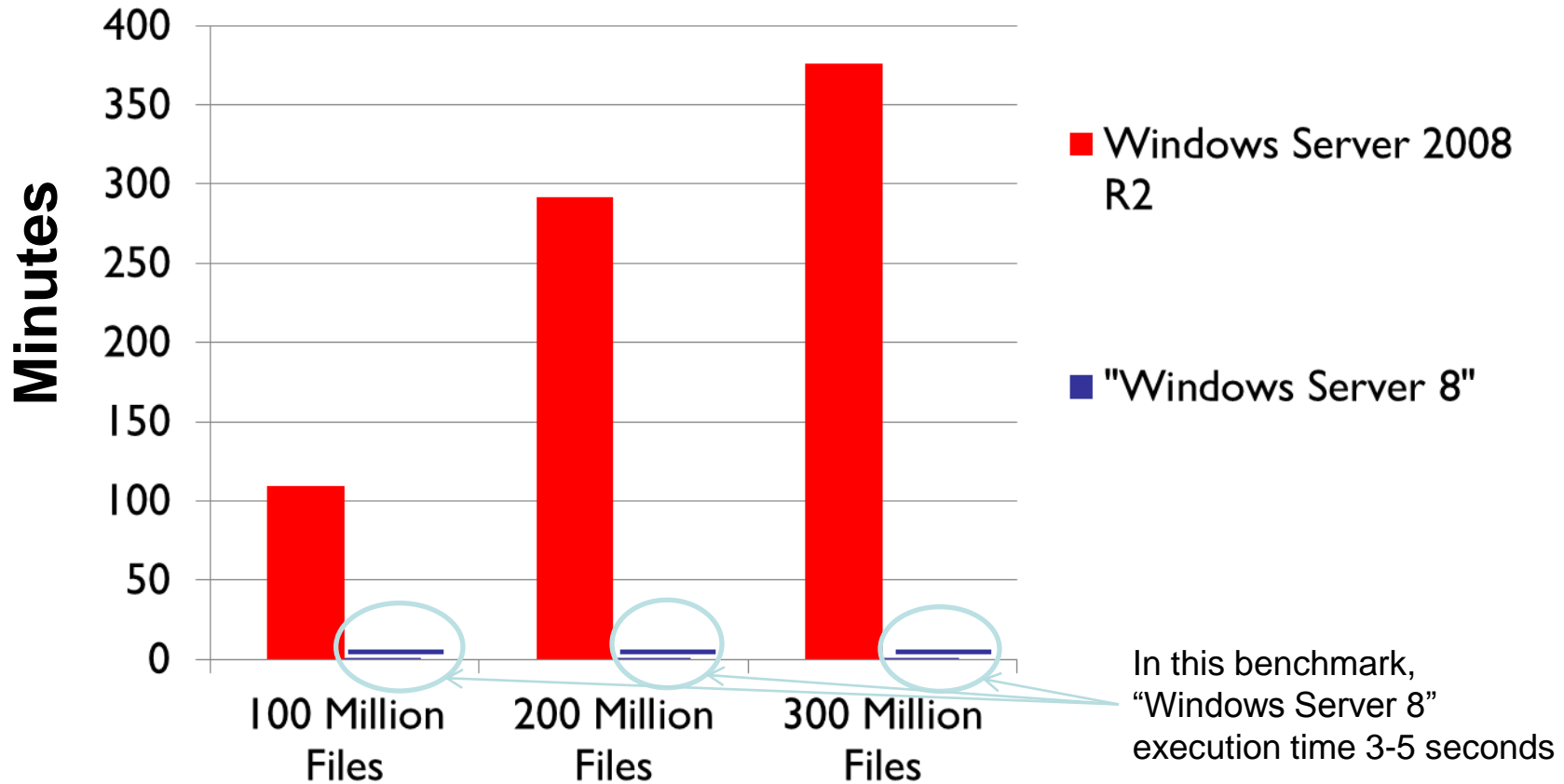
# Online scan & repair: execution flow



# Maximized File System Availability

## An illustrative example

Volume downtime to handle one corruption



- ❑ Confidently deploy 64TB NTFS volumes with Windows 8
- ❑ Explorer:
  - ❑ Check Now UX
  - ❑ Action Center
  - ❑ Server Manager
  - ❑ Systems Center
- ❑ “chkdsk” command line options:
  - ❑ chkdsk x: /scan - perform an online scan for corruptions
  - ❑ chkdsk x: /spotfix - perform an offline repair
  - ❑ chkdsk x: /f - still works as it always has
- ❑ “fsutil repair” command line options:
  - ❑ fsutil repair enumerate x: - list known verified corruptions
  - ❑ fsutil repair state - list corruption state of all volumes
  - ❑ Fsutil repair state x: - list corruption state of given volume
- ❑ powershell:
  - ❑ REPAIR-VOLUME -scan, -spotfix, -offlinescanandfix

# Reliability using Flush instead of FUA (Forced Unit Access)

- ❑ What is FUA (Forced Unit Access)
  - ❑ A flag originally implemented in the SCSI (T10) specification that indicates a given write should go directly to media, bypassing a devices write cache
- ❑ NTFS is a Journaled File System which uses FUA to guarantee write ordering to maintain its metadata integrity
- ❑ The ATA (T13) specification did not originally define FUA
  - ❑ FUA support was added to T13 in 2002 as part of the ATA7 spec
  - ❑ Since FUA has not been consistently implemented on ATA devices it has never been enabled on Windows platforms
- ❑ NTFS was designed to rely on proper FUA implementation to maintain robustness

# The switch to Flush

- ❑ To make NTFS robust on SATA devices it has switched in Windows 8 to issuing a flush of a drives write cache instead of relying on FUA
- ❑ Delivers improved reliability on industry standard SATA storage
  - ❑ Reduces possibility of corruption on power loss
- ❑ Improves performance on SCSI devices
  - ❑ Allows the disk to cache data for as long as safely possible

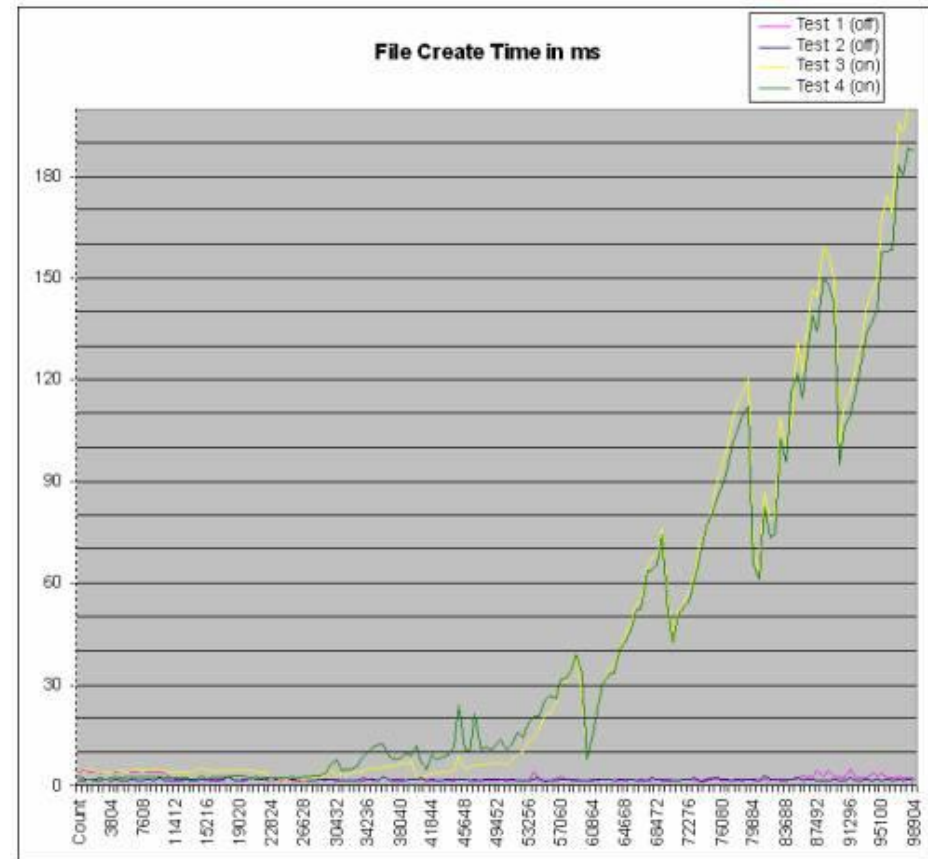
# Changes in Short Filename Support

- ❑ Windows 7 added the ability to enable/disable short filename generation on a per-volume basis
  - ❑ `fsutil 8dot3name set` - used to change state
    - ❑ Change takes effect immediately (no reboot required)
  - ❑ When disabled prevents short filename generation
    - ❑ Existing short filenames continue to function
  - ❑ Added the ability to strip short filenames
    - ❑ `fsutil 8dot3name strip`

- ❑ Windows 8 disables short filename generation on all volumes except the boot volume.
  - ❑ Only affects volumes formatted under Windows 8
    - ❑ `format x: /s:enable` - to enable at format time
  - ❑ Volumes migrated from downlevel versions of windows will maintain their existing short filename generation policy
  - ❑ Still have the ability to enable/disable short filename generation policy on a per-volume basis

# Short Filename Generation Performance Impact

- ❑ Short filename generation does have a performance impact
  - ❑ Small impact for directories with < 30,000-40,000 files
  - ❑ Beyond this threshold the performance impact continues to increase



# Trim Enhancements

- ❑ What is Trim?
  - ❑ Allows a file system to tell an underlying storage device that the contents of specified sectors are no longer important
- ❑ Trim support was added to NTFS in Windows 7
  - ❑ A Trim command is generated every time clusters are freed
    - ❑ File deletion, file truncation, file defrag, etc.
  - ❑ Trim is only supported by ATA drivers
  - ❑ There is no SCSI driver support for Trim
  - ❑ No other Windows inbox file systems support Trim

# Windows 8 Trim Enhancements

- ❑ Trim is now supported by SCSI drivers
  - ❑ Generates a SCSI unmap command
  - ❑ Important for thin provisioned volumes
- ❑ NTFS now supports file level trim
  - ❑ Allows an application to tell the underlying storage device that the contents of specified ranges of a file are no longer important
  - ❑ Semantically operates like a write command
    - ❑ Breaks oplocks using the same rules as write
    - ❑ Processes byte range locks using the same rules as write
    - ❑ Updates the USN Journal with a `USN_REASON_DATA_OVERWRITE` record
    - ❑ Trimmed ranges of the file are purged from the cache
  - ❑ Not supported on compressed or encrypted files
  - ❑ File level trim requests are rounded to page size boundaries (4K)
  - ❑ Trim requests to a mounted VHD or inside Hyper-V are now propagated to the underlying storage device

# Intelligent Error Correction

# NTFS support for Intelligent Error Correction

- ❑ Windows Virtualized Storage: Storage Spaces
  - ❑ Supports sector level mirroring
  - ❑ For more information on Storage Spaces see Shiv Rajpal's presentation at 1:00pm on Tuesday entitled "Storage Stack Evolution in Windows"

# NTFS support for Intelligent Error Correction

- ❑ Applications that maintain their own data integrity information (ex: checksums or CRCs) can detect and repair data inconsistencies when their data is stored on a mirrored space
  - ❑ An explicit copy of a mirrored set can be read using `FSCTL_MARK_HANDLE` to indicate which copy to read for a given file handle
    - ❑ Use normal read operations to retrieve the specified copy of the data
    - ❑ Only supports non-cached IO
  - ❑ Each copy of the data is read and compared against existing integrity information
  - ❑ If data corruption is detected and at least one copy of the data is correct, `FSCTL_REPAIR_COPIES` can be used to indicate which copy is good and which copy(s) need to be repaired
    - ❑ Spaces will handle the necessary synchronization to repair the specified copy(s), if still needed, even as the sectors are being read or written

# Storage Optimizer

also known as Defrag

# New Features of Defrag

- ❑ Slab Consolidation
  - ❑ Efficiently defrags files to minimize the number of allocated slabs
    - ❑ A slab is the unit of allocation on a thin provisioned volume
    - ❑ Requires support for `IOCTL_STORAGE_QUERY_PROPERTY` requesting a property ID of: `StorageDeviceLBP provisioningProperty`
      - ❑ Retrieves a volumes slab size
- ❑ ReTRIM
  - ❑ Generates Trim commands for all free space on the given volume
  - ❑ Supported on live volumes
- ❑ Fast Analysis of Optimizations
  - ❑ Significantly faster analysis phase by using new NTFS interface: `FSCTL_QUERY_FILE_LAYOUT`

# New Features of Defrag Continued

- ❑ Media-aware optimization
  - ❑ Performs the proper optimization based on the media type of the given volume:
    - ❑ HDD – Defrag + ReTRIM
    - ❑ SSD – ReTRIM only
    - ❑ VirtualDisks (Spaces) – Slab Consolidation + ReTRIM
    - ❑ Thin Provisioned Arrays – Slab Consolidation + ReTRIM
    - ❑ Dynamic VHDs – Slab Consolidation + ReTrim

**Questions?**