Unit 8: File System

8.4. NTFS Recovery Support

Roadmap for Section 8.4

- The Evolution of File Systems
- Recoverable File System
- Log File Service Operation
- NTFS Recovery Procedures
- Fault-Tolerance Support
- Volume Management – Striped and Spanned Volumes

NTFS Recovery Support

- Transaction-based logging scheme
- Fast, even for large disks
- Recovery is limited to file system data
  - Use transaction processing like SQL server for user data
  - Tradeoff: performance versus fully fault-tolerant file system
- Design options for file I/O & caching:
  - **Careful write**: VAX/VMS fs, other proprietary OS fs
  - **Lazy write**: most UNIX fs, OS/2 HPFS

Careful Write File Systems

- OS crash/power loss may corrupt file system
- Careful write file system orders write operations:
  - System crash will produce predictable, non-critical inconsistencies
- Update to disk is broken in sub operations:
  - Sub operations are written serially
  - Allocating disk space: first write bits in bitmap indicating usage; then allocate space on disk
- I/O requests are serialized:
  - Allocation of disk space by one process has to be completed before another process may create a file
  - No interleaving sub operations of the two I/O requests
- Crash: volume stays usable; no need to run repair utility

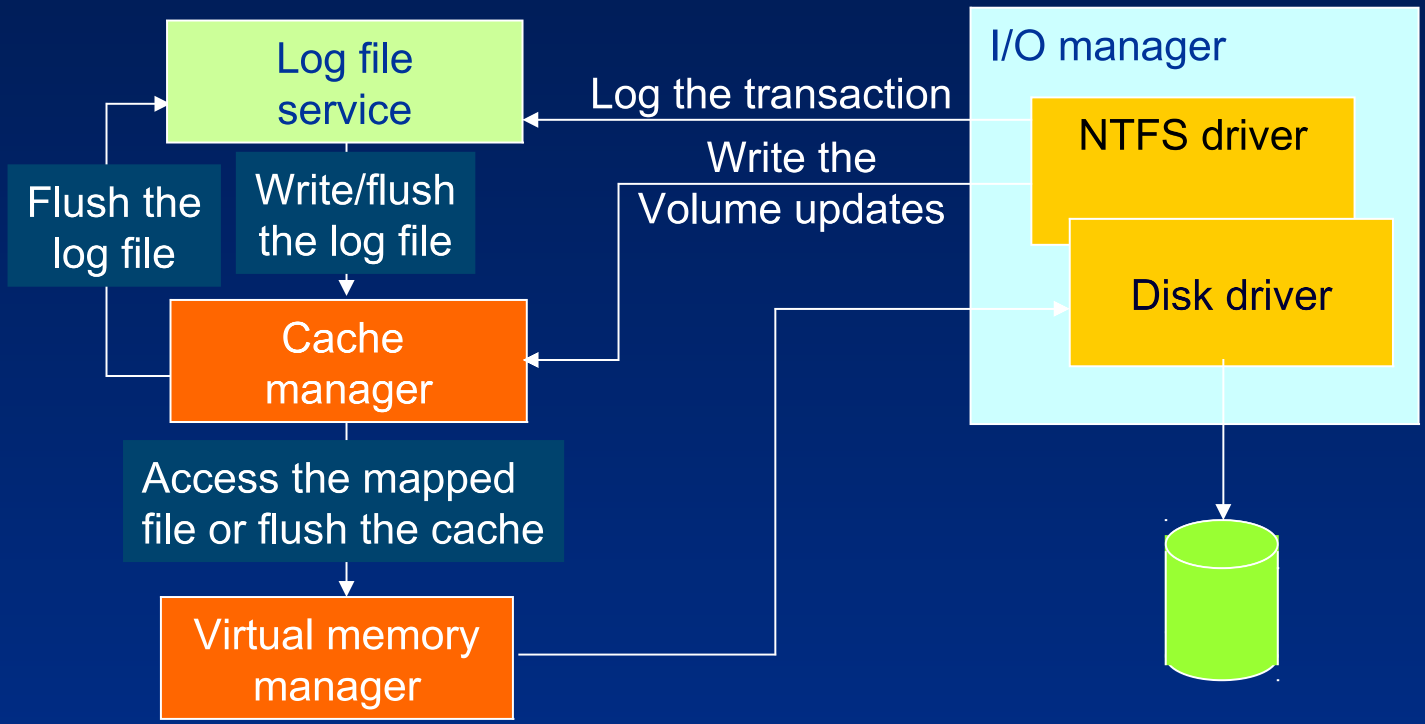
Lazy Write File Systems

- Careful write sacrifices speed for safety
- Lazy write improves performance by write back caching
  - Modifications are written to the cache;
  - Cache flush is an optimized background activity
- Less disk writes; buffer can be modified multiple times before being written to disk
- File system can return to caller before operation is completed
- Inconsistent intermediate states on volume are ignored
- Greater risk / user inconvenience if system fails

Recoverable File System
(Journaling File System)

- Safety of careful write fs & performance of lazy write fs
- Log file + fast recovery procedure
  - Log file imposes some overhead
  - Optimization over lazy write: distance between cache flushes increased
- NTFS supports *cache write-through* and *cache flushing* triggered by applications
  - No extra disk I/O to update fs data structures necessary: all changes to fs structure are recorded in log file which can be written in a single operation
  - In the future, NTFS may support logging for user files (hooks in place)

Log File Service (LFS)



- LFS is designed to provide logging to multiple kernel components (clients)
- Currently used only by NTFS

Log File Regions

- NTFS calls LFS to read/write restart area
  - Context info: location of logging area to be used for recovery
  - LFS maintains 2nd copy of restart area
  - Logging area: circularly reused
  - LFS uses logical sequence numbers (LSNs) to identify log records
- NTFS never reads/writes transactions to log file directly
- During recovery:
  - NTFS calls LFS to read forward; recorded transactions are redone
  - NTFS calls LFS to read backward; undo all incompletely logged transactions

Operation of the LFS / NTFS

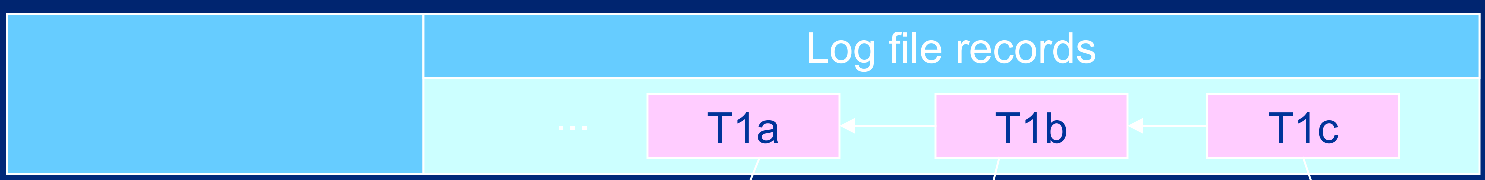
1. NTFS calls LFS to record in (cached) log file any transactions that will modify volume structure
2. NTFS modifies the volume (also in the cache)
3. Cache manager calls LFS to flush the log file to disk
   (LFS implements flushing by calling cache manager back, telling which page(s) to flush)
4. After cache manager flushes the log file, it flushes volume changes

-> Transactions of unsuccessful modifications can be retrieved from the log file and un-/redone

Recovery begins automatically the first time a volume is used after system is rebooted.

Log Record Types

- Update records (series of ...)
  - Most common; each record contains:
  - **Redo information**: how to reapply one sub operation of a committed transaction
  - **Undo information**: how to reverse a partially logged sub operation
- Last record commits the transaction (not shown here)



Redo: Allocate/initialize an MFT file record
Undo: Deallocate the file record

Redo: Set bits 3-9 in the bitmap
Undo: Clear bits 3-9 in the bitmap

Redo: Add the filename to the index
Undo: Remove the filename from the index

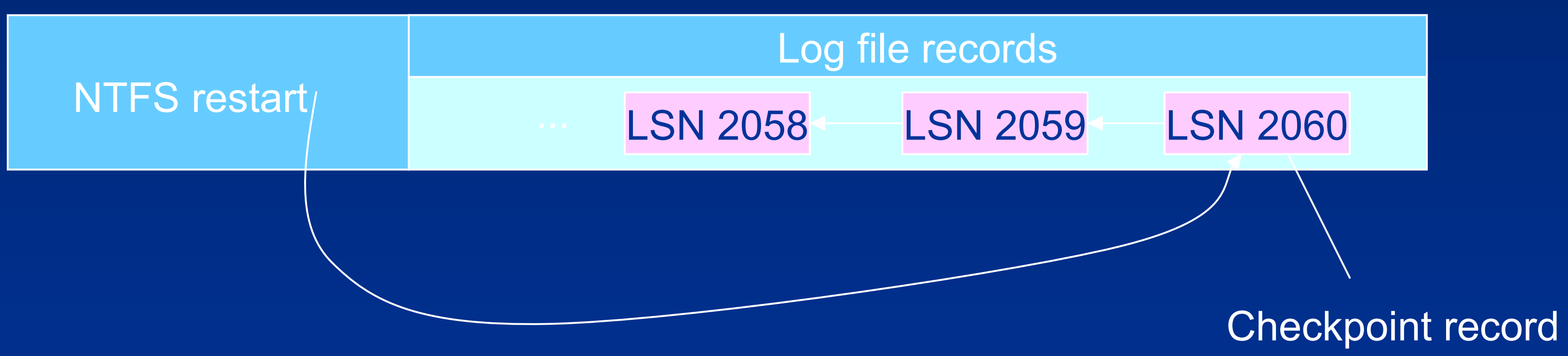
Recovery: redo committed/undo incompletely logged transaction

Log Records (contd.)

- Physical vs. logical description of redo/undo actions:
  - Delete file „a.dat“ vs. Delete byte range on disk
  - NTFS writes update records with physical descriptions
- NTFS writes update records (usually several) for:
  - Creating a file
  - Deleting a file
  - Extending a file
  - Truncating a file
  - Setting file information
  - Renaming a file
  - Changing security applied to a file
- Redo/undo operations must be idempotent (might be applied twice)

Checkpoint Records

- NTFS periodically writes a checkpoint record
  - Describes, what processing would be necessary to recover a volume if a crash would occur immediately
  - How far back in the log file must NTFS go to begin recovery
  - LSN of checkpoint record is stored in restart area



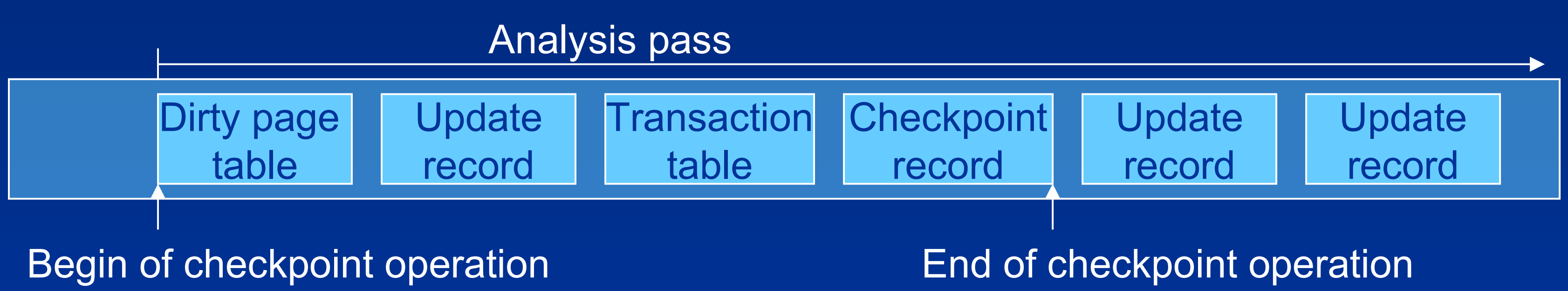
Log File Full

LFS presents the log file to NTFS as if it were infinitely large

- Writing checkpoint records usually frees up space
- LFS tracks several numbers:
  - Available log space
  - Amount of space needed to write an incoming log record and to undo the write
  - Amount of space needed to roll back all active (no committed) transactions, should that be necessary
- Insufficient space: „Log file full“ error & NTFS exception
  - NTFS prevents further transactions on files (block creation/deletion)
  - Active transactions are completed or receive „log file full“ exception
  - NTFS calls cache manager to flush unwritten data
  - If data is written, NTFS marks log file „empty“; resets beginning of log file
- No effect on executing programs (short I/O pause)

Recovery - Principles

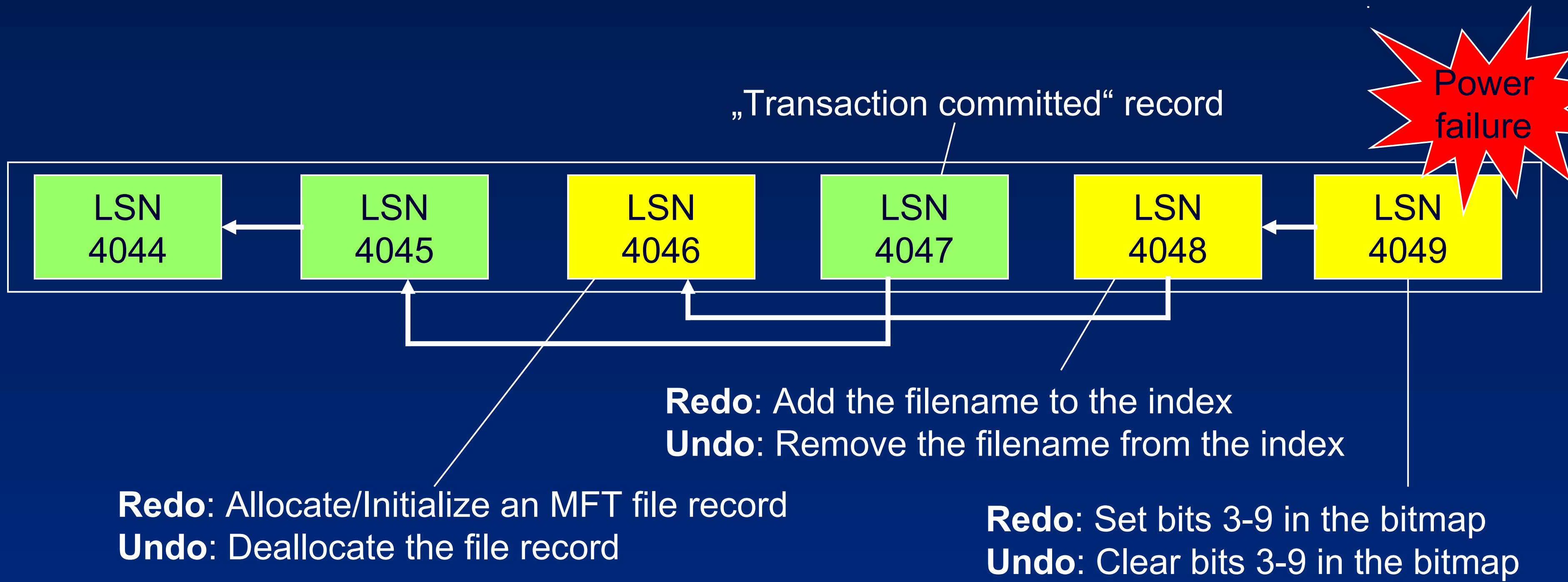
- NTFS performs automatic recovery
- Recovery depends on two NTFS in-memory tables:
  - *Transaction table*: keeps track of active transactions (not completed) Note: sub operations of these transactions must be removed from disk, in case of transaction abort
  - *Dirty page table*: records which pages in cache contain modifications to file system structure that have not yet been written to disk
- NTFS writes checkpoint every 5 sec.
  - Includes copy of transaction table and dirty page table
  - Checkpoint includes LSNs of the log records containing these tables



Recovery - Passes

1. Analysis pass
   - NTFS scans forward in log file from beginning of last checkpoint
   - Updates transaction/dirty page tables it copied in memory
   - NTFS scans tables for oldest update record of a non-committed transaction
2. Redo pass
   - NTFS looks for „page update“ records which contain volume modification that might not have been flushed to disk
   - NTFS redoes these updates in the cache until it reaches end of log file
   - Cache manager „lazy writer thread“ begins to flush cache to disk
3. Undo pass
   - Roll back any transactions that weren't committed when system failed
   - After undo pass – the volume is in a consistent state
   - Write empty LFS restart area; no recovery is needed if system fails now

Undo Pass - Example



- Transaction 1 was commited before power failure
- Transaction 2 was still active when power failure occured
- NTFS must log undo operations in log file!
  - Power might fail again during recovery;
  - NTFS would have to redo its undo operations

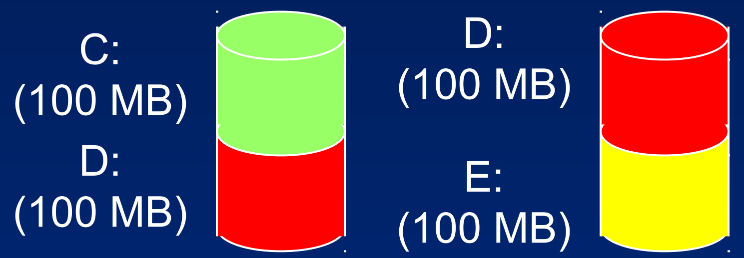
NTFS Recovery - Conclusions

- Recovery will return volume to some preexisting consistent state (not necessarily the state before the crash)
- Lazy commit algorithm: log file is not immediately flushed when a „transaction committed“ record is written
  - LFS batches records;
  - Flush when cache manager calls or check pointing record is written (once every 5 sec)
  - Several parallel transactions might have been active before a crash
- NTFS uses log file mechanisms for error handling
- Most I/O errors are not file system errors
  - NTFS might create a MFT record and then detect that disk is full when allocating space for a file in the bitmap
  - NTFS uses log info to undo changes and returns „disk full“ error to caller

Fault-Tolerance Support

- NTFS's capabilities are enhanced by the fault-tolerant volume managers FtDisk / DMIO
  - Lies above hard disk drivers in the I/O system's layered driver scheme
  - FtDisk – for basic disks
  - DMIO – for dynamic disks
- Volume management capabilities:
  - Redundant data storage
  - Dynamic data recovery from bad sectors on SCSI disks
- NTFS itself implements bad-sector recovery for non-SCSI disks

Volume Management Features – Spanned Volumes



Volume set D:
occupies half of
two disks

Spanned Volumes:

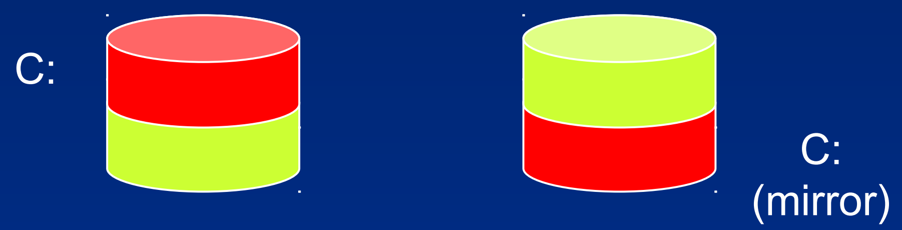
- single logical volume composed of a maximum of 32 areas of free space on one or more disks
- NTFS volume sets can be dynamically increased in size
(only bitmap file which stores allocation status needs to be extended)
- FtDisk & DMIO hide physical configuration of disks from file system
- Tool: Windows Disk Management MMC snap-in
- Spanned volumes were called volume sets in Windows NT 4.0

Striped Volumes

- Series of partitions, one partition per disk (of same size)
- Combined into a single logical volume
- FtDisk & DMIO optimize data storage and retrieval times
  - Stripes are narrow: 64KB
  - Data tends to be distributed evenly among disks
  - Multiple pending read/write operations will operate on different disks
  - Latency for disk I/O is often reduced (parallel seek operations)

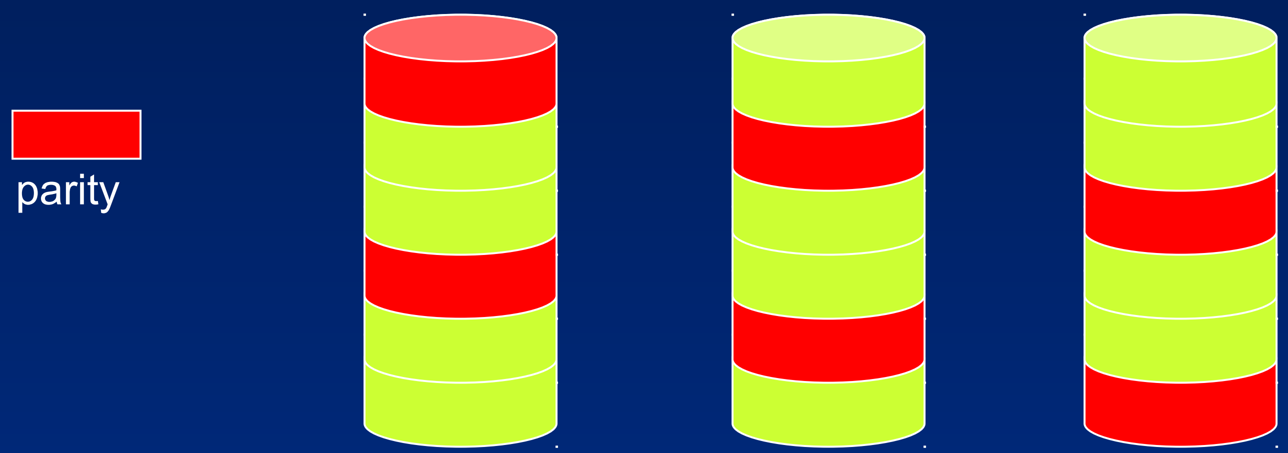
Fault-Tolerant Volumes

- FtDisk & DMIO implement redundant storage schemes
  - Mirror sets
  - Stripe sets with parity (RAID-5 volumes)
  - Sector sparing
- Tools: Windows Disk Management MMC snap-in



- **Mirrored Volumes:**
  - Contents of a partition on one disk is duplicated on another disk
  - FtDisk & DMIO write same data to both locations
  - Read operations are done simultaneously on both disks (load balancing)

RAID-5 Volumes



- Fault tolerant version of a regular stripe set
- Parity: logical sum (XOR)
- Parity info is distributed evenly over available disks
- FtDisk & DMIO reconstruct missing data by using XOR operations

Bad Cluster Recovery

- Sector sparing is supported by FtDisk & DMIO
  - Dynamic copying of recovered data to spare sectors
  - Without intervention from file system / user
  - Works for certain SCSI disks
  - FtDisk & DMIO return bad sector warning to NTFS
- Sector re-mapping is supported by NTFS
  - NTFS will not reuse bad clusters
  - NTFS copies data recovered by FtDisk & DMIO into a new cluster
- NTFS cannot recover data from bad sector without help from FtDisk & DMIO
  - NTFS will never write to bad sector (re-map before write)

Bad-cluster re-mapping

"Before":

Standard info	NTFS filename	Security desc.	Data		
			Starting VCN	Starting LCN	Number of clusters
			0	0	0

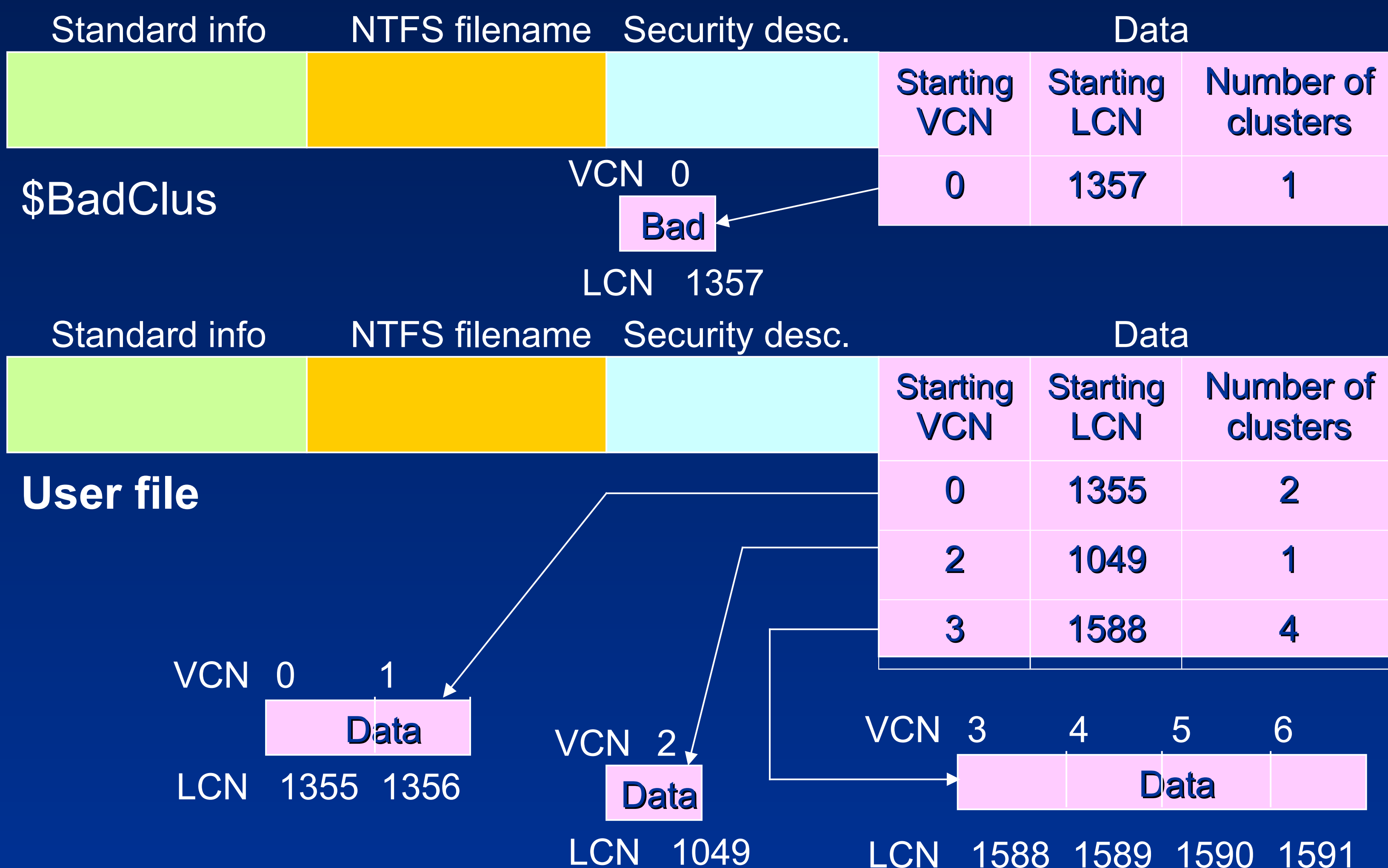
$BadClus: is "empty"

Standard info	NTFS filename	Security desc.	Data		
			Starting VCN	Starting LCN	Number of clusters
			0	1355	3
			3	1588	4

User file

Bad-cluster re-mapping

“After”:



Starting VCN	Starting LCN	Number of clusters
0	1357	1

Starting VCN	Starting LCN	Number of clusters
0	1355	2
2	1049	1
3	1588	4

Self-Healing

- A feature to repair damage while a volume remains online
  - When NTFS detects corruption, it prevents access to the damaged file or files and creates a system worker thread that performs chkdsk-like corrections to the corrupted data structures, allowing access to the repaired files when it has finished. Access to other files continues normally during this operation, minimizing service disruption.
- The *fsutil repair* commands:
  - The periodic automatic self-healing of NTFS volumes:
    - *fsutil repair set ...* : can be used to view or set a volume’s repair options
  - NTFS also supports manually initiated self-healing cycles:
    - *fsutil repair initiate/wait ...* : can be used to force the repair of a specific file and to wait until repair of that file is complete

Further Reading

- Mark E. Russinovich, David A. Solomon, and Alex Ionescu, "*Windows Internals*", 6th Edition, Microsoft Press, 2012.
  - Chapter 12 – File Systems (from pp. 391)
    - NTFS Recovery Support (from pp. 477)
  - Chapter 9 – Storage Management (from pp. 125)
    - Volume Management (from pp. 138)

Remark: these chapters will be in part 2 of 7th edition!