Windows Kernel Internals II

Advanced File Systems

University of Tokyo – July 2004

Dave Probert, Ph.D.
Advanced Operating Systems Group
Windows Core Operating Systems Division
Microsoft Corporation

Disk Basics

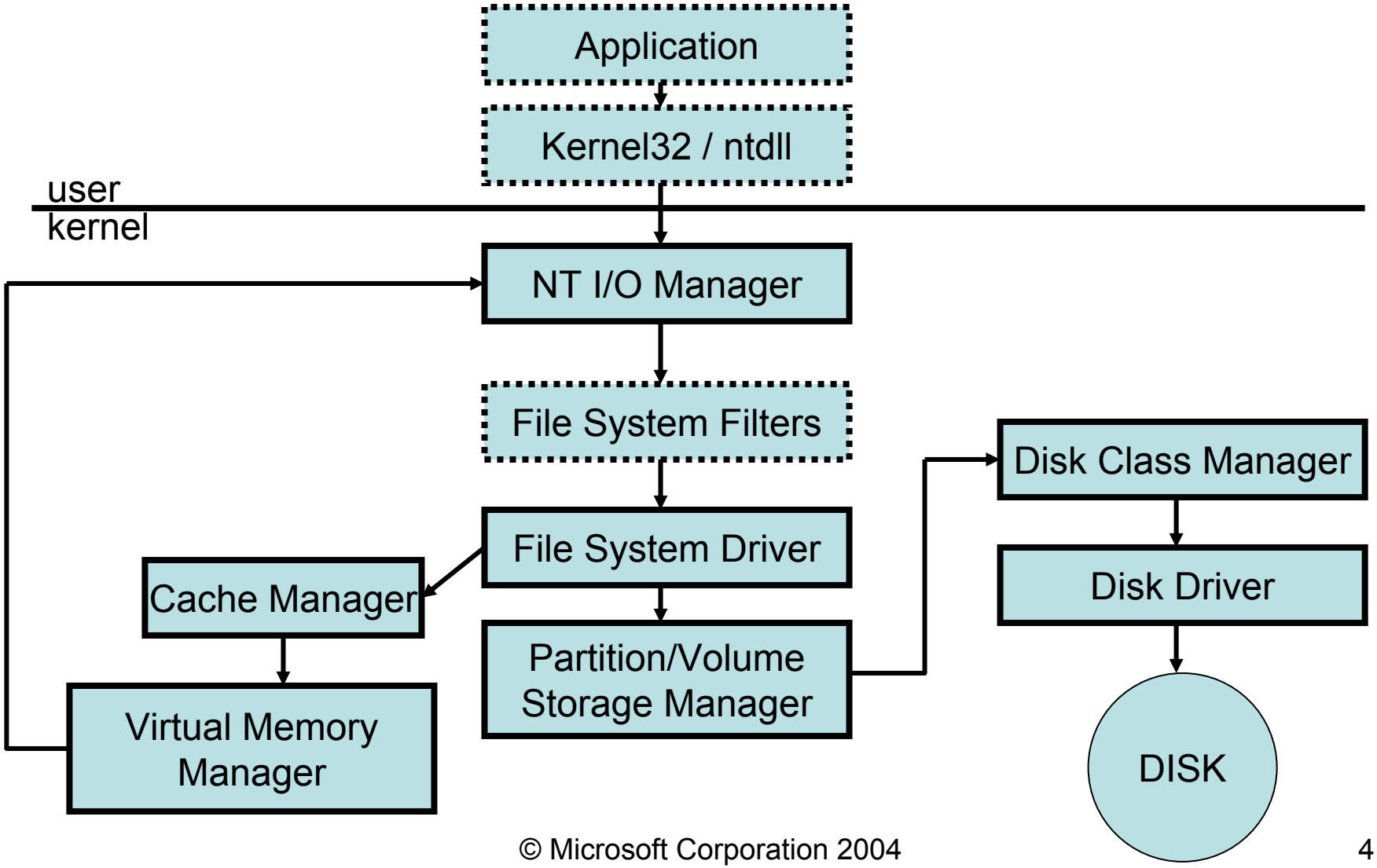
- Volume exported via device object
- Addressed by byte offset and length
- Enforced on sector boundaries
- NTFS allocation unit - clusters
- Round size down to clusters

Storage Management

Volumes may span multiple logical disks

Partitioning	Description	Benefits
spanned	logical catenation of arbitrary sized volumes	size
striped (RAID-0)	interleaved same-sized volumes	read/write perf
mirrored (RAID-1)	redundant writes to same-sized volume, alternate reads	reliability, read perf
RAID-5	striped volumes w/ parity	reliability, size, read perf

File System Device Stack



NTFS Deals with files

Partition is collection of files

Common routines for all meta-data

Utilizes MM and Cache Manager

No specific on-disk locations

CacheManager overview

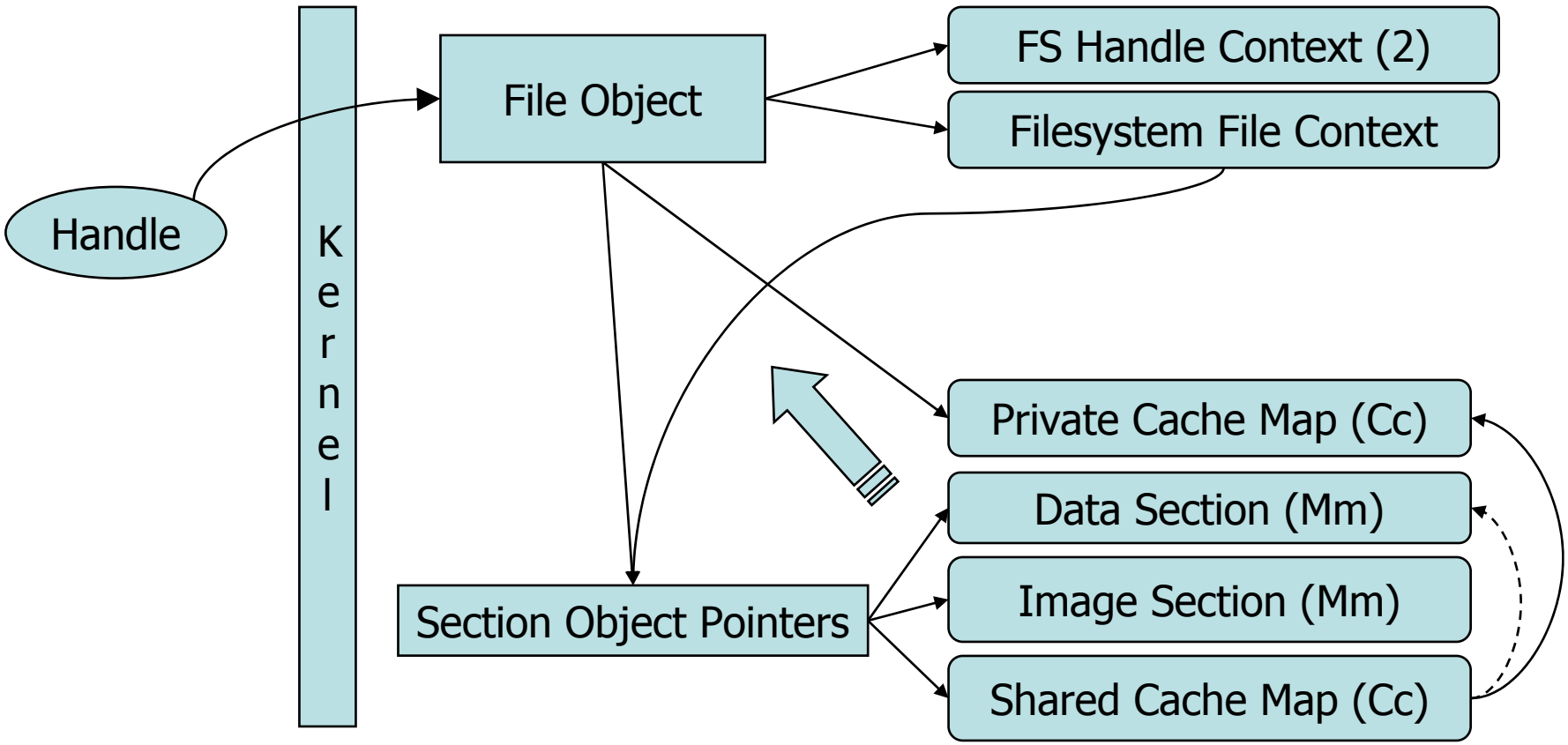
Cache manager

- kernel-mode routines
- asynchronous worker routines
- interface between filesystems and VM mgr

Functionality

- access methods for pages of file data on opened files
- automatic asynchronous read ahead
- automatic asynchronous write behind (lazy write)
- supports “Fast I/O” – IRP bypass

Datastructure Layout



File Object == Handle (U or K), *not one per file*

Section Object Pointers and FS File Context shared/stream

Datastructures

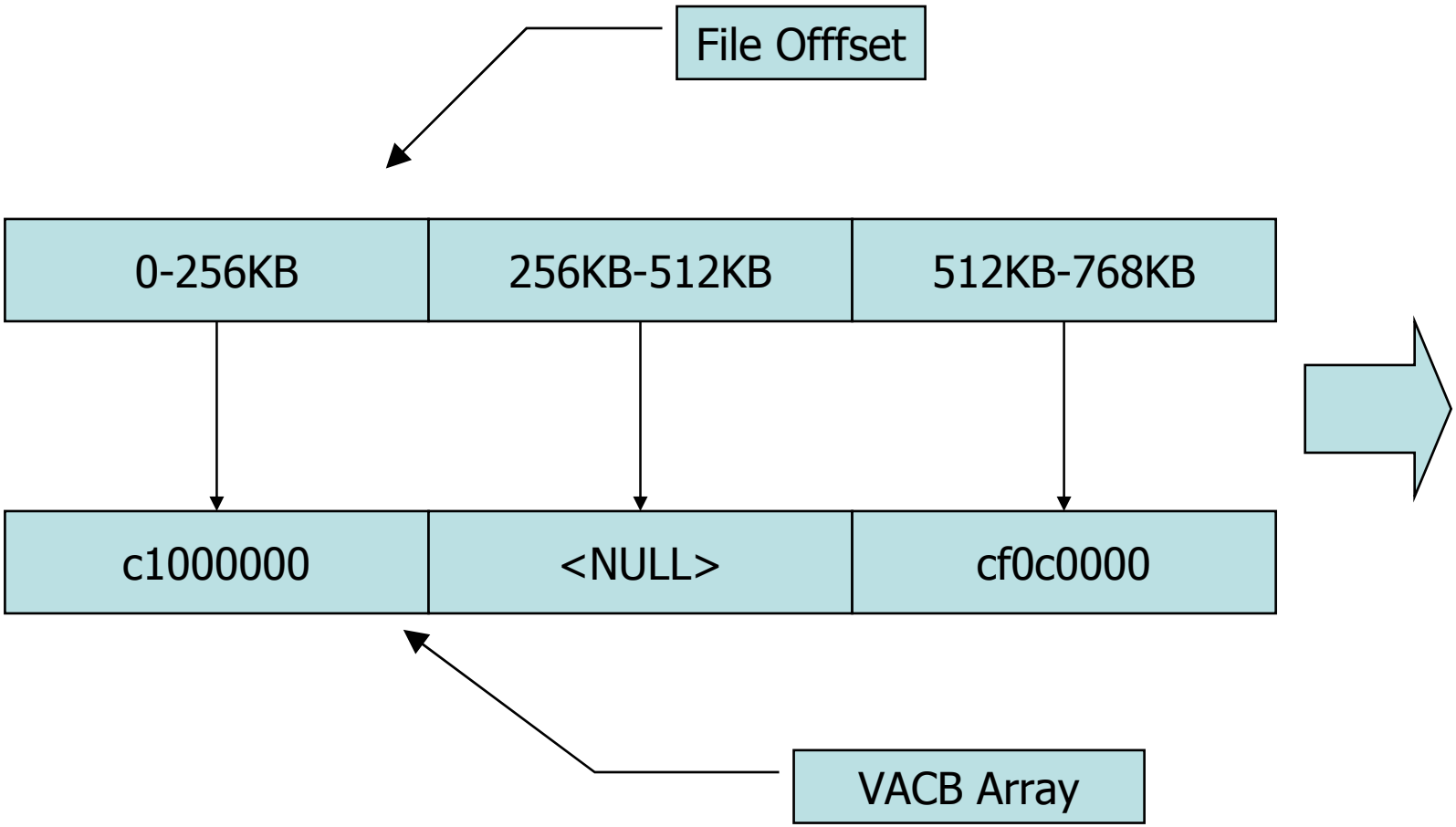
File Object

- FsContext – per physical stream context
- FsContext2 – per user handle stream context, not all streams have handle context (metadata)
- SectionObjectPointers – the point of “single instancing”
  - DataSection – exists if the stream has had a mapped section created (for use by Cc or user)
  - SharedCacheMap – exists if the stream has been set up for the cache manager
  - ImageSection – exists for executables
- PrivateCacheMap – per handle Cc context (readahead) that also serves as reference from this file object to the shared cache map

Cache View Management

- **A Shared Cache Map has an array of View Access Control Block (VACB) pointers which record the base cache address of each view**
  - promoted to a sparse form for files > 32MB
- **Access interfaces map File+FileOffset to a cache address**
- **Taking a view miss results in a new mapping, possibly unmapping an unreferenced view in another file (views are recycled LRU)**
- **Since a view is fixed size, mapping across a view is impossible – Cc returns one address**
- **Fixed size means no fragmentation …**

View Mapping



CacheManager Interface Summary

File objects start out unadorned

CcInitializeCacheMap to initiate caching via Cc on a file object

- setup the Shared/Private Cache Map & Mm if neccesary

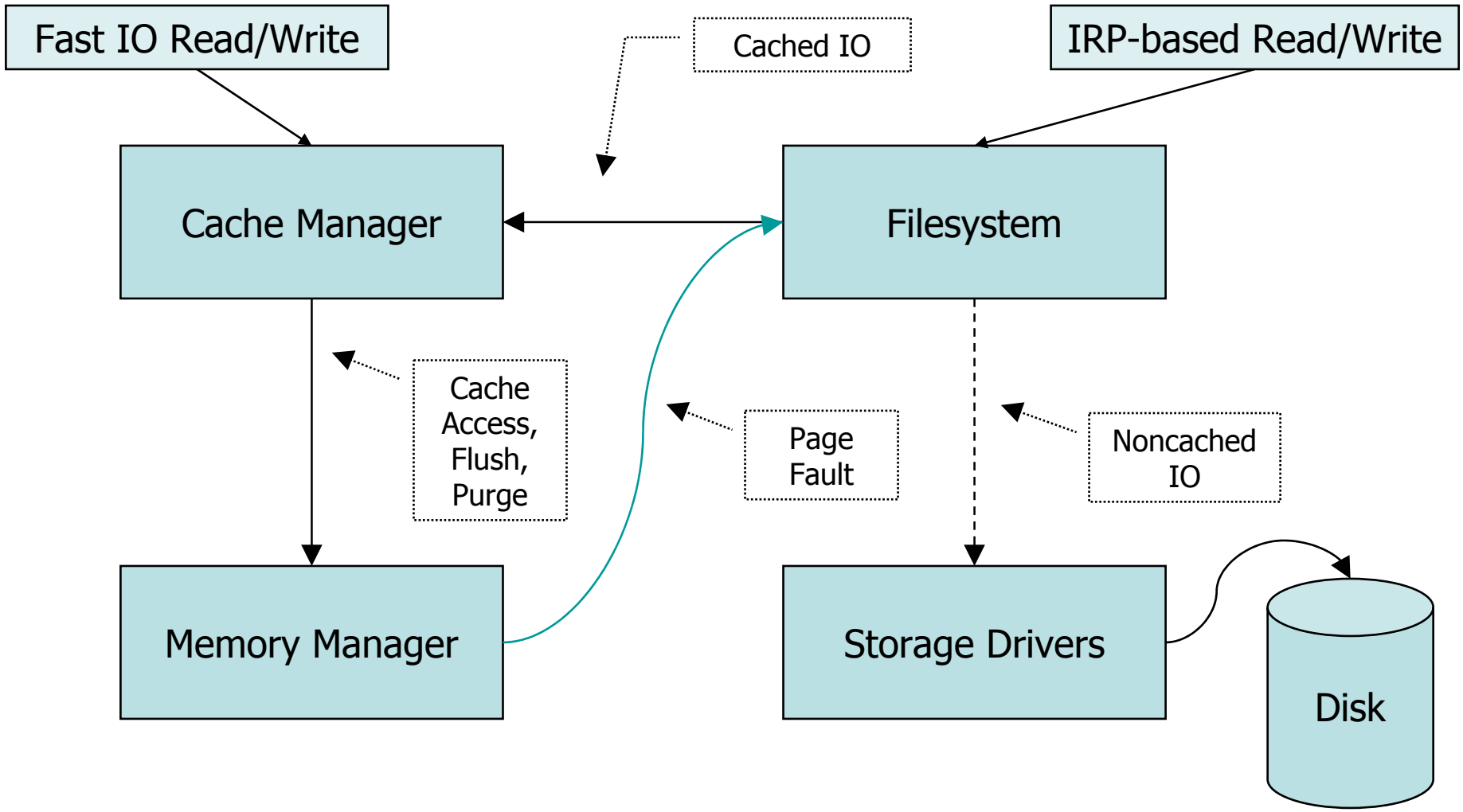
Access methods (Copy, Mdl, Mapping/Pinning)

Maintenance Functions

CcUninitializeCacheMap to terminate caching on a file object

- teardown S/P Cache Maps
- Mm lives on. *Its data section is the cache!*

CacheManager / FS Diagram



File System Notes

Three basic types of IO

- cached, non-cached, paging

Three file sizes

- file size, allocation size, valid data length

Three worker threads

- Mm's modified page writer (paging file)
- Mm's mapped page writer (mapped files)
- Cc's lazy writer pool (flushes views)

Cache Manager Summary

- Virtual block cache for files not logical block cache for disks
- Memory manager is the ACTUAL cache manager
- Cache Manager context integrated into FileObjects
- Cache Manager manages views on files in kernel virtual address space
- I/O has special fast path for cached accesses
- The Lazy Writer periodically flushes dirty data to disk
- Filesystems need two interfaces to CC: map and pin

NTFS on-disk structure

Some NTFS system files

- $Bitmap
- $BadClus
- $Boot
- . (root directory)
- $Logfile
- $Volume
- $Mft
- $MftMirr
- $Secure

$Mft File

Data is entirely File Records

File Records are fixed size

Every file on volume has a File Record

File records are recycled

Reserved area for system files

Critical file records mirrored in $MftMirr

File Records

‘Base’ file record for each file

Header followed by ‘Attributes’

Additional file records as needed

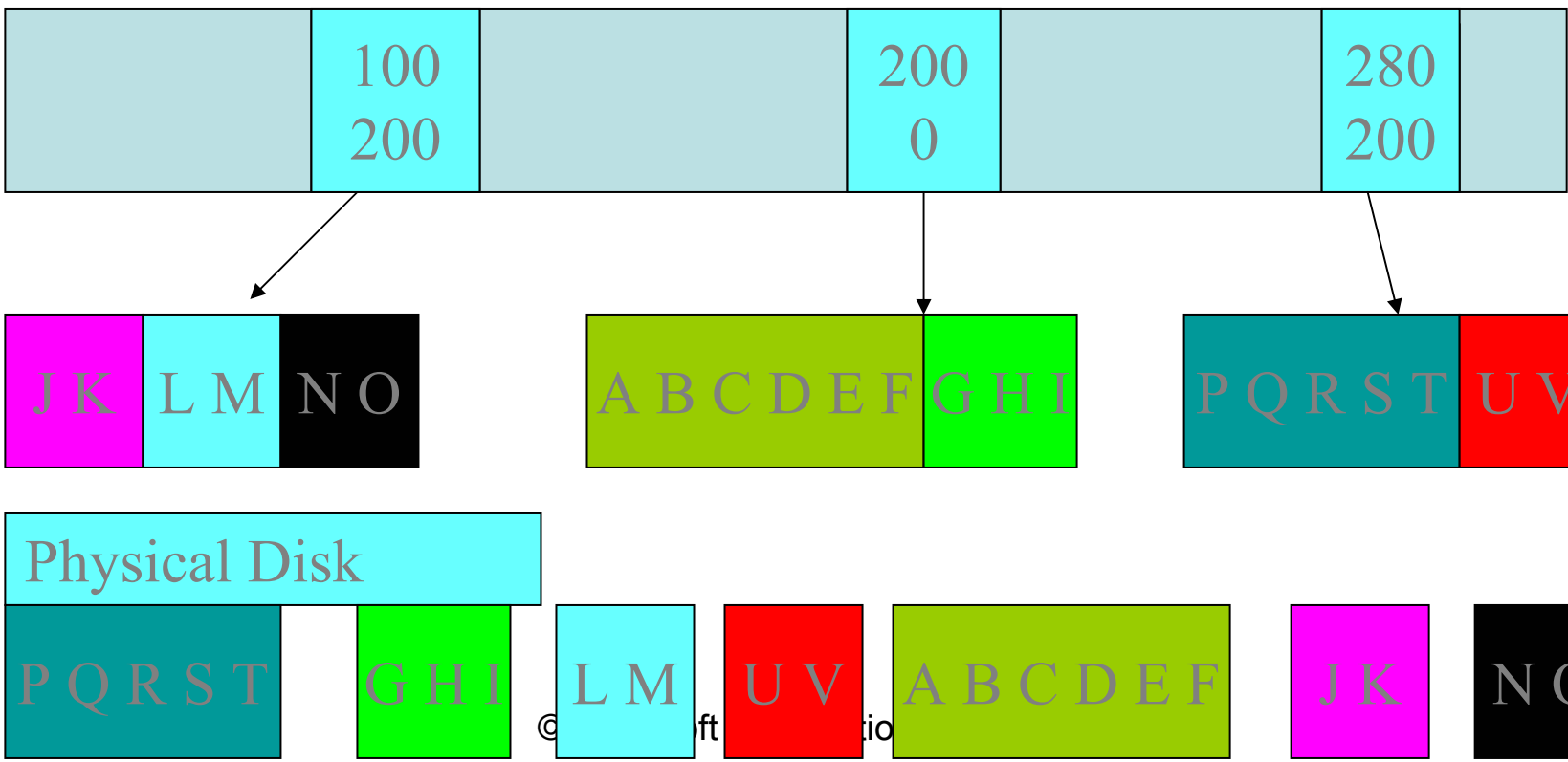
Update Sequence Array

ID by offset and sequence number

File D:¥Letters (File ID 0x200)

A B C D E F G H I J K L M N O P Q R S T U V

File ¥$Mft



Physical Disk

File Basics

Timestamps

File attributes (DOS + NTFS)

Filename (+ hard links)

Data streams

ACL

Indexes

File Building Blocks

File Records

Ntfs Attributes

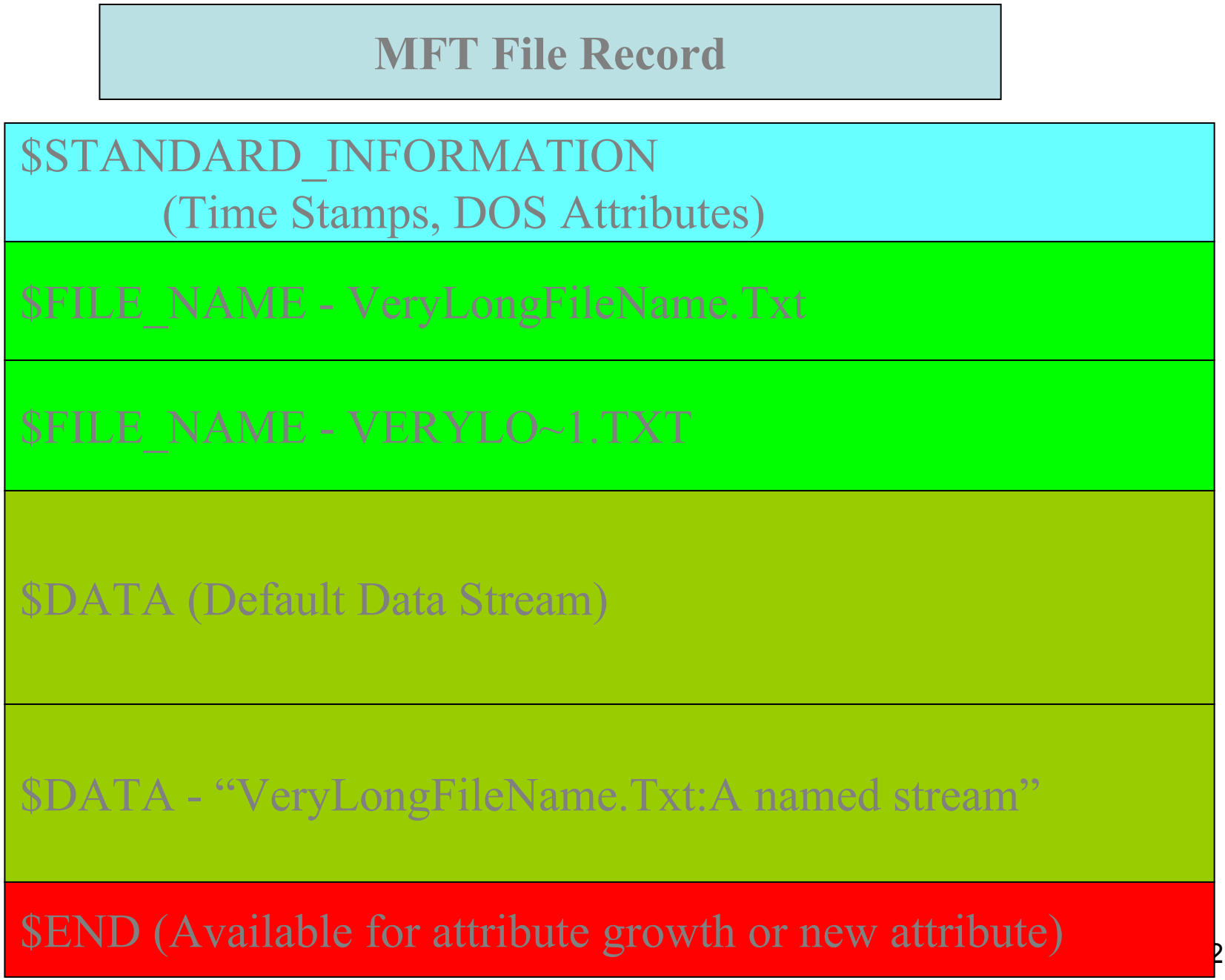
Allocated clusters

File Record Header

- USA Header
- Sequence Number
- First Attribute Offset
- First Free Byte and Size
- Base File Record
- IN_USE bit

NTFS Attributes

- Type code and optional name
- Resident or non-resident
- Header followed by value
- Sorted within file record
- Common code for operations


MFT File Record
$STANDARD_INFORMATION
(Time Stamps, DOS Attributes)
$FILE_NAME - VeryLongFileName.Txt
$FILE_NAME - VERYLO~1.TXT
$DATA (Default Data Stream)
$DATA - “VeryLongFileName.Txt:A named stream”
$END (Available for attribute growth or new attribute)

Attribute Header

Length

Form

Name and name length

Flags (Compressed, Encrypted, Sparse)

Resident Attributes

Data follows attribute header

‘Allocation Size’ on 8-byte boundary

May grow or shrink

Convert to non-resident

Non-Resident Attributes

Data stored in allocated disk clusters

May describe sub-range of stream

Sizes and stream properties

Mapping pairs for on-disk runs

Some Attribute Types

$STANDARD_INFORMATION

$FILE_NAME

$SECURITY_DESCRIPTOR

$DATA

$INDEX_ROOT

$INDEX_ALLOCATION

$BITMAP

$EA

Mapping Pairs

Stored in a byte optimal format

Represents allocation and holes

Each pair is relative to prior run

Used to represent compression/sparse

Indexes

File name and view indexes

Indexes are B-trees

Entries stored at each level

Intermediate nodes have down pointers

$INDEX_ROOT

$INDEX_ALLOCATION

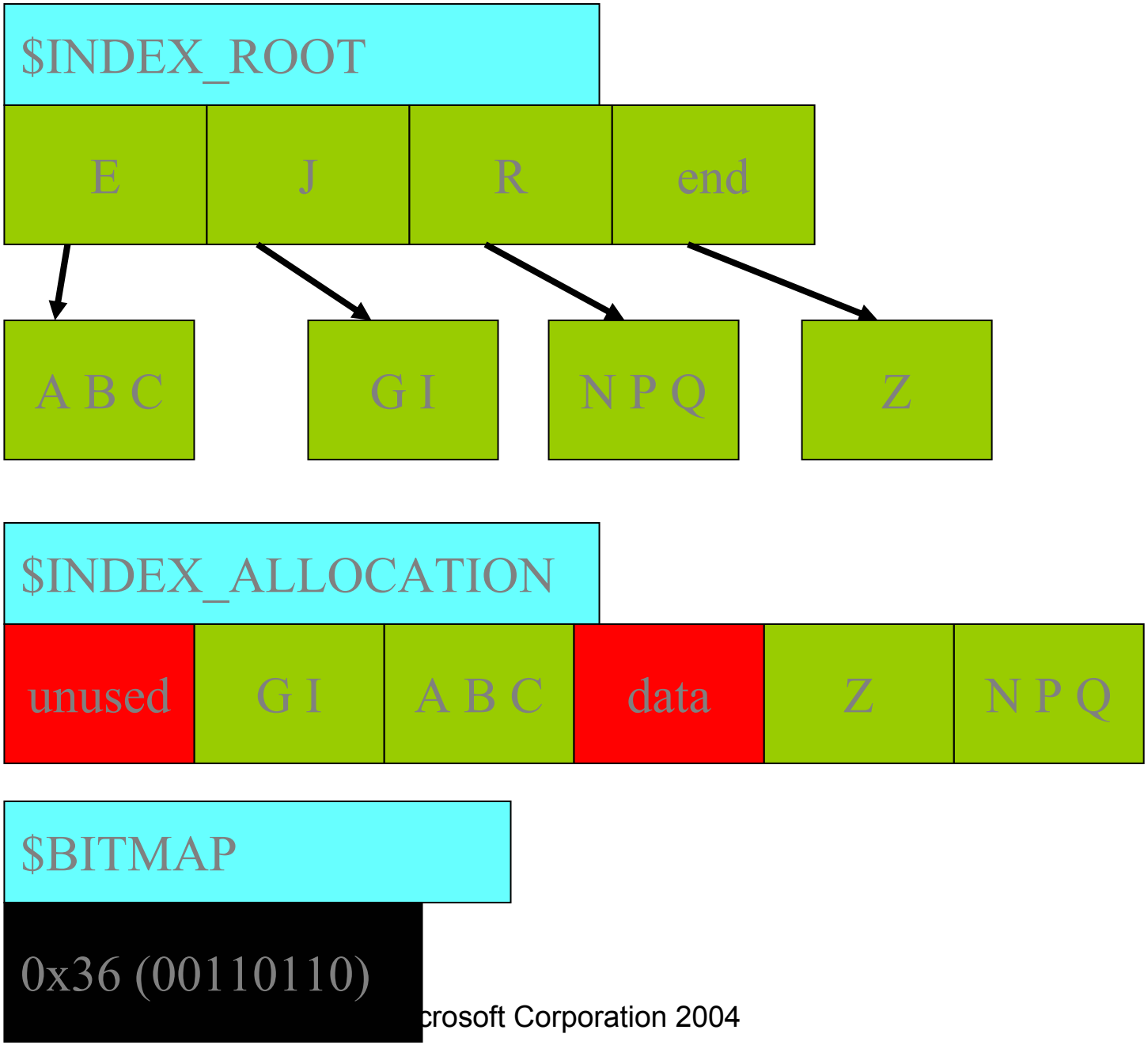
$BITMAP

Index Implementation

Top level - $INDEX_ROOT

Index buckets - $INDEX_ALLOCATION

Available buckets - $BITMAP


$INDEX_ROOT
E
J
R
end
A B C
G I
N P Q
Z
$INDEX_ALLOCATION
unused
G I
A B C
data
Z
N P Q
$BITMAP
0x36 (00110110)

$ATTRIBUTE_LIST

Needed for multi-file record file

Entry for each attribute in file

Resident or non-resident form

Must be in base file record

Attribute List (example)

- Base Record - 0x200
- 0x10 - Standard
- 0x20 - Attribute List
- 0x30 - FileName
- 0x80 - Default Data
- 0x80 - Data1 “Owner”

- Aux Record - 0x180
- 0x30 - FileName
- 0x80 - Data “Author”
- 0x80 - Data0 “Owner”
- 0x80 - Data “Writer”

Attribute List (example cont.)

Code	FR	VCN	Name	(Not Present)
0x10	0x200			$Standard
0x30	0x200			$Filename
0x30	0x180			$Filename
0x80	0x200	0		$Data
0x80	0x180	0	“Author”	$Data
0x80	0x180	0	“Owner”	$Data
0x80	0x200	40	“Owner”	$Data
0x80	0x180		“Writer”	$Data

Discussion