

Filesystem Driver



What to Expect

- What is File System
- Three levels of File Systems
- FS relation with a Hard Disk Partition
- FS relation with /
- Writing a FS module

What is Filesystem?

- Three things at three levels
 - Hardware Space – The Physical Organization of Data on the Storage Devices
 - Kernel Space – Drivers to decode & access the data from the Physical Organization
 - User Space – All what you see from / - The User View
- Which ever it be, it basically does
 - Organize our data
 - For easy access by tagging
 - For fast maintenance by optimizing

Why we need a File System?

- Preserve Data for Later
 - Access
 - Update
 - Discard
- Tag Data for
 - Categorizing
 - Easy & Fast Lookup
- Fast & Reliable Data Management by
 - Optimized Operations

The Three File Systems

- Hardware (Space) File System - Storer
- Kernel (Space) File System - Decoder
- User (Space) File System - Show-er
- Storer: Partition for the File System (/dev/*)
 - Creating the File System (mkfs.*)
- Decoder: Driver for the File System (*.ko)
- Show-er: Root File System (/xyz)
- Connector of the 3: mount
 - Let's try an example

Recall

Understanding a Hard Disk

- Example (Hard Disk)
 - Heads (or Platters): 0 – 9
 - Tracks (or Cylinders): 0 – 24
 - Sectors: 1 – 64
- Size of the Hard Disk
 - $10 \times 25 \times 64 \times 512 \text{ bytes} = 8000\text{KiB}$
- Device independent numbering
 - $(h, t, s) \rightarrow 64 * (10 * t + h) + s \rightarrow (1 - 16000)$

Partitioning a Hard Disk

- First Sector – Master Boot Record (MBR)
 - Contains Boot Info
 - Contains Physical Partition Table
- Maximum Physical Partitions: 4
 - At max 1 as Extended Partition
 - Rest as Primary Partition
- Extended could be further partitioned into
 - Logical Partitions
- In each partition
 - First Sector – Boot Record (BR)
 - Remaining for File System / Format
 - Extended Partition BR contains the Logical Partition Table

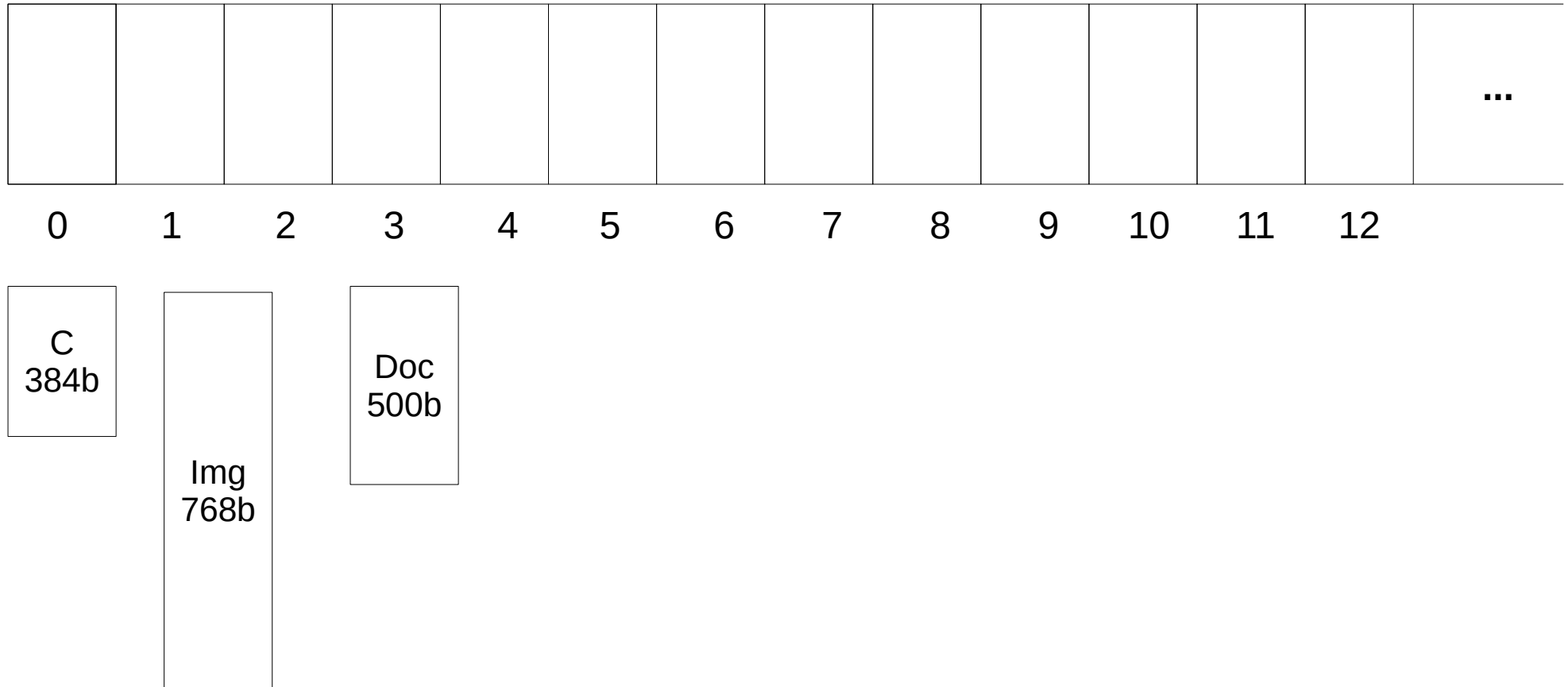
Placement of a Hardware FS

- Each Partition contains a File System
 - Raw (no FS)
 - Organized
- Organized one is initialized by the corresponding Formatting
- “Partition Type is to OS”
 - W95*, DOS*, BSD*, Solaris*, Linux, ...
- “Format Type is to File System”
 - ext2, ext3, vfat, ntfs, jffs2, ...
- Particular Partition Types support only Particular File System Types

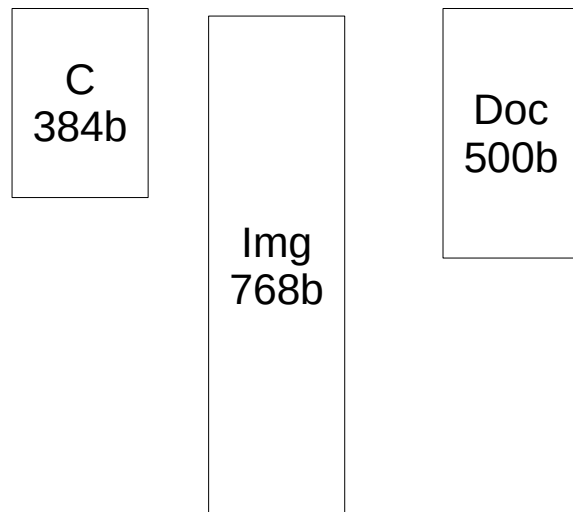
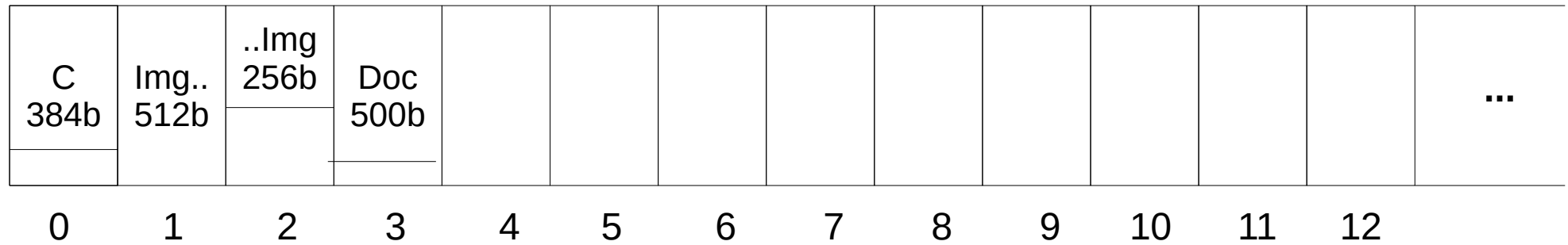
Design a FS

- Let's Take 1 Partition
- With $2N$ blocks of 1 sectors each
- Size = $2N \times 512 \text{ bytes} = N\text{KiB}$
- Given 3 data pieces
 - Source Code – 384 bytes
 - Image – 768 bytes
 - Document – 500 bytes
- Place optimally for further operations


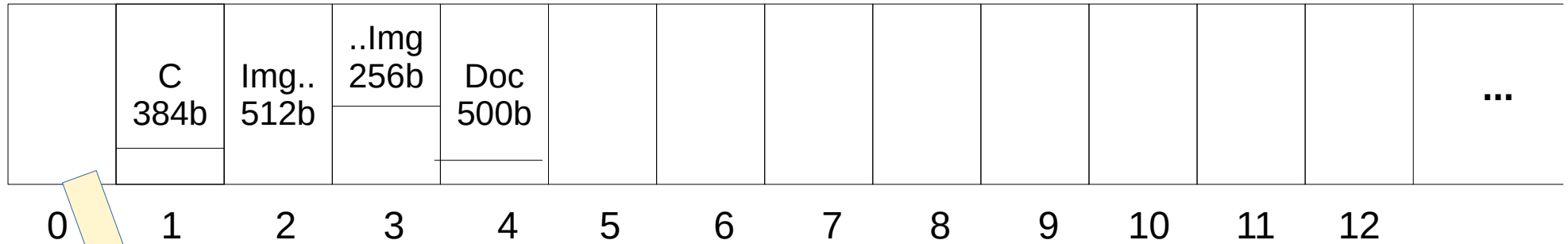
Filesystem Design



Filesystem Design



Filesystem Design

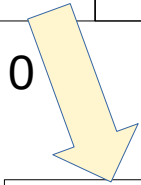


C	384	1	1
Img	768	2	3
Doc	500	4	4
XXX	X	X	X
XXX	X	X	X

▪
▪
▪

Filesystem Design

	C 384b	Img.. 512b	..Img 256b	Doc 500b									...
0	1	2	3	4	5	6	7	8	9	10	11	12	

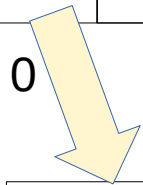


C	384	1	1
Img	768	2	3
Doc	500	4	4
0	0	0	0
0	0	0	0

▪
▪
▪

Filesystem Design

	C... 512b	Img.. 512b	..Img 256b	Doc 500b	...C 488b								...
0	1	2	3	4	5	6	7	8	9	10	11	12	

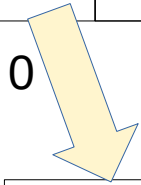


C	384	1	1
Img	768	2	3
Doc	500	4	4
0	0	0	0
0	0	0	0

▪
▪
▪

Filesystem Design

	C... 512b	Img.. 512b	..Img 256b	Doc 500b	...C 488b								...
0	1	2	3	4	5	6	7	8	9	10	11	12	



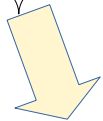
C	1000	1	5	0	0	0
Img	768	2	3	0	0	0
Doc	500	4	4	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

▪
▪
▪

Filesystem Design

				C... 512b	Img.. 512b	..Img 256b	Doc 500b	...C 488b					...
--	--	--	--	--------------	---------------	---------------	-------------	--------------	--	--	--	--	-----

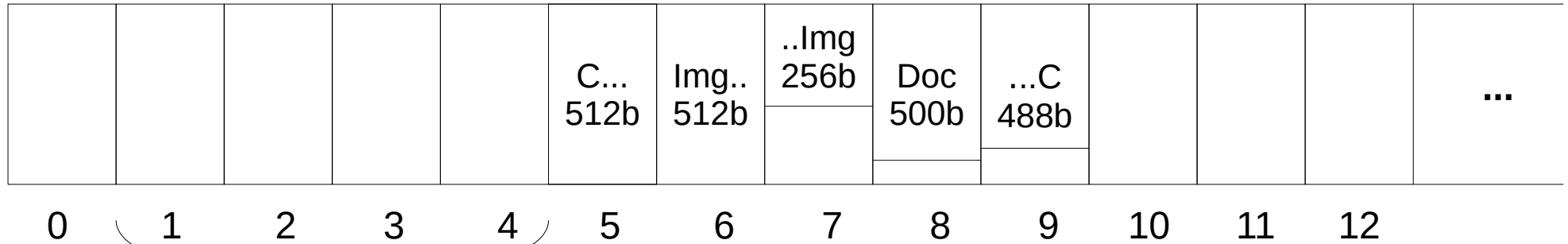
0 1 2 3 4 5 6 7 8 9 10 11 12



C	1000	4	8	0	0	0
Img	768	5	6	0	0	0
Doc	500	7	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

▪
▪
▪

Filesystem Design

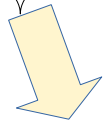


C	1000	5	8	0	0	0
Img	768	6	7	0	0	0
Doc	500	8	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

▪
▪
▪

Filesystem Design

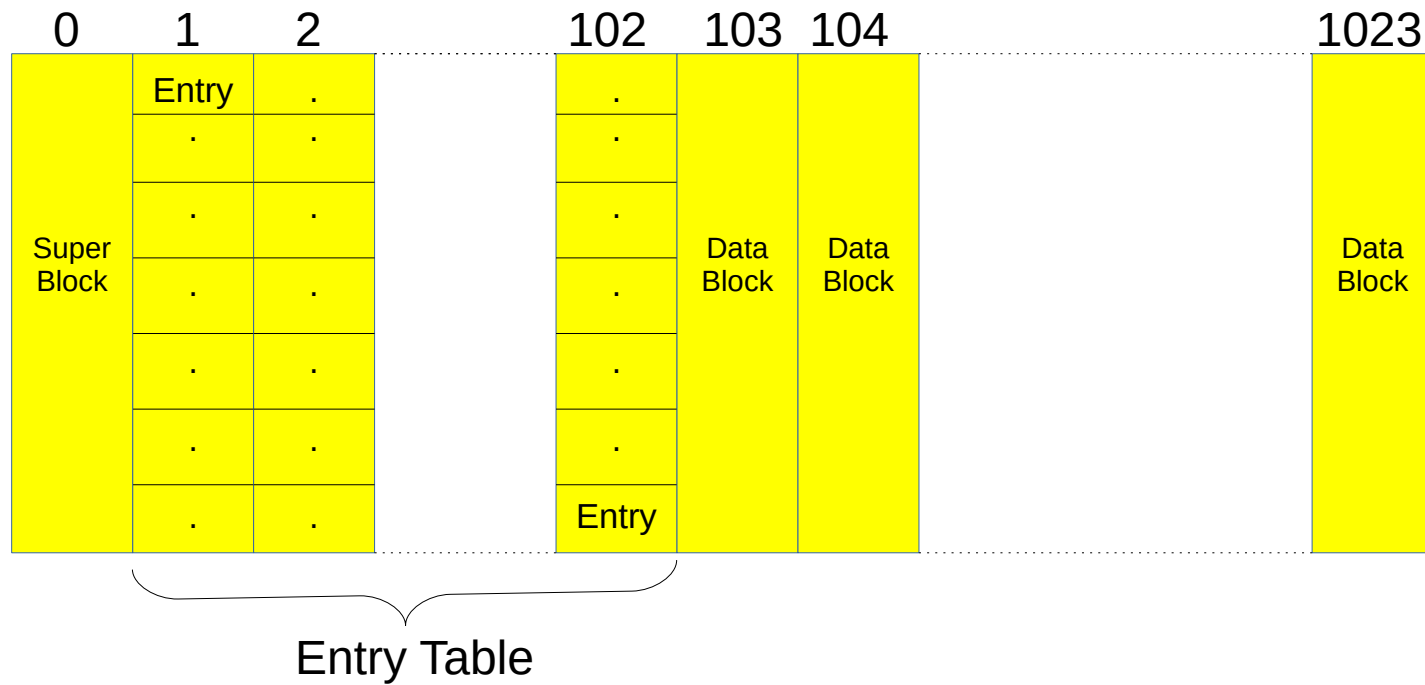
FST 4 ...					C... 512b	Img.. 512b	..Img 256b	Doc 500b	...C 488b				...
0	1	2	3	4	5	6	7	8	9	10	11	12	



C	1000	TS	rw-	5	8	0	0	0
Img	768	TS	rw-	6	7	0	0	0
Doc	500	TS	rw-	8	0	0	0	0
0	0	0	...	0	0	0	0	0
0	0	0	...	0	0	0	0	0

▪
▪
▪

File System Layout



512KiB File system Layout

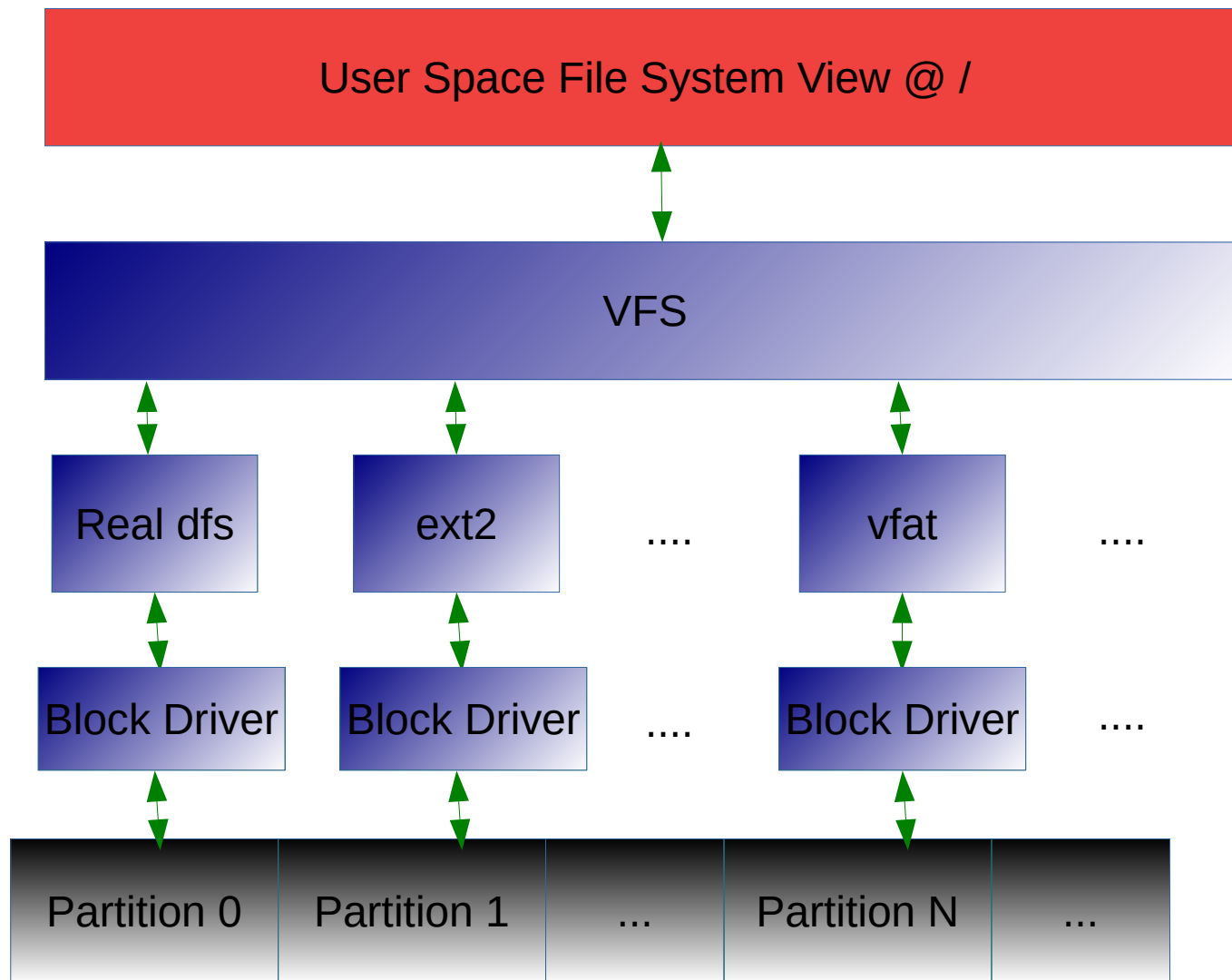
Kernel Space File System

- Has 2 Roles to Play
 - Decode the Hardware FS Layout to access data (by implementing your logic in s/w)
 - Interface that with the User Space FS to give you a unified view, as you all see at /
- First part: ext2, ext3, vfat, ntfs, jffs2, ...
 - Also called the File System modules
 - We will use a simplified version
- Second part: VFS
 - Kernel provides VFS to achieve the virtual user view
 - So, FS module should actually interact with VFS

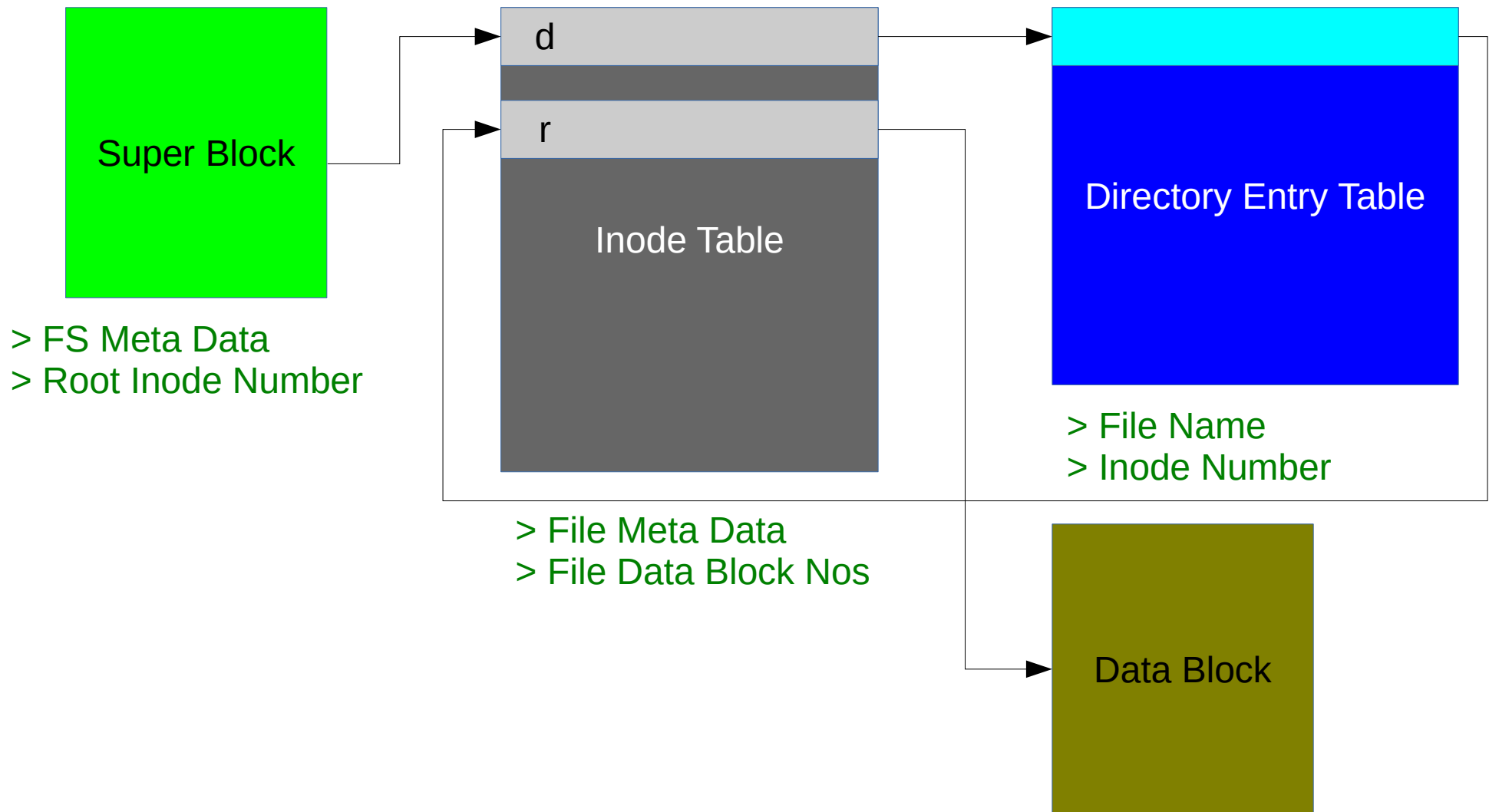
VFS Specifics

- Inherited from the Linux specific FS
- Promoted to a super set of various FS
- Providing a unified view to the User
- Default values for various attributes
 - owner, group, permissions, ...
- File Types
 - Same seven types as in ext2/ext3

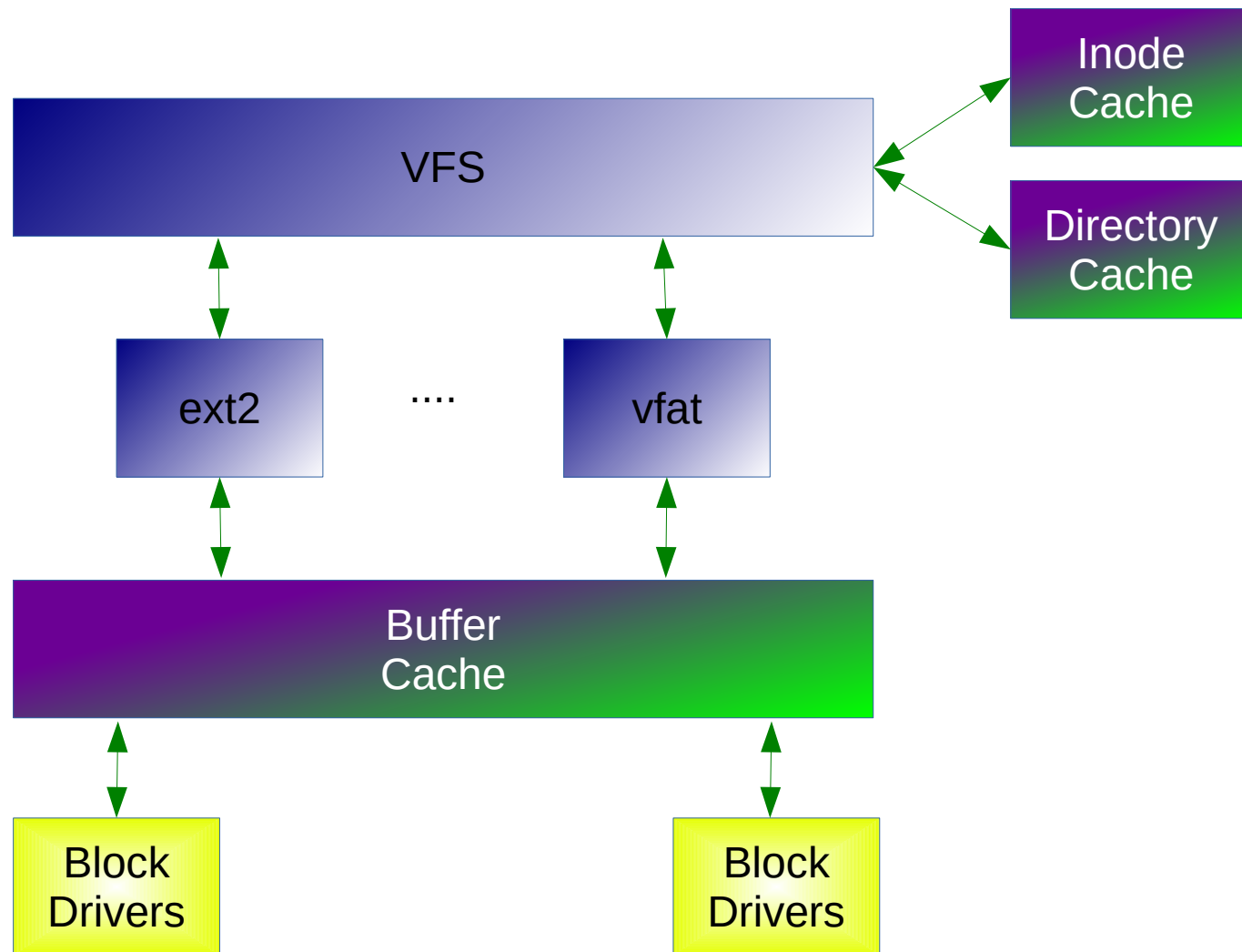
Virtual File System Interactions



Virtual File System Interactions



VFS Interaction Internals



VFS Translation Callbacks

- Five categories
 - File System level
 - Super Block Operations
 - Super Block level (Meta Meta Data)
 - Inode Operations
 - Inode level (Meta Data)
 - Directory Entry Operations
 - File level (Data)
 - File Operations through the Buffer Cache
 - Buffer Cache level
 - Actual Data Operations

File System Registration

- Header: <linux/fs.h>
- APIs
 - int register_filesystem(file_system_type *)
 - int unregister_filesystem(file_system_type *)
- File System Operations
 - get_sb/mount: (Invoked by mount)
 - get_sb_bdev/mount_bdev -> Fill the Super Block
 - kill_sb: (Invoked by umount)
 - kill_block_super

Fill the Super Block

- Block Size & Bits
 - Magic Number
- Super Block Operations
 - File System type
- Root Inode
 - Inode Operations
 - Inode Mode
 - File Operations

Super Block Operations

- Header: <linux/fs.h>
- APIs
 - write_inode: Update File Meta Data
 - statfs: Get the File Status (Invoked by stat)
 - simple_statfs (Handler from libfs)

Inode Operations

- Header: <linux/fs.h>
- APIs
 - lookup
 - Search the file name by absolute path
 - Traverse the inode chain
 - if found matching inode, then populate the dentry with the inode information
- Inode Information
 - Size
 - Mode
 - File Operations
 - ...

File Operations

- Header: <linux/fs.h>
- APIs
 - open → generic_file_open
 - read → do_sync_read → generic_file_aio_read → do_generic_file_read → readpage
 - write → do_sync_write → generic_file_aio_write → generic_file_buffered_write → write_begin & write_end
 - release
 - readdir: Change Directory (cd). List Directory (ls)
 - Directory Entry
 - Fill Function
 - fsync: Sync the File (sync)
 - simple_sync_file

Address Space Operations

- The Buffer / Page Cache level Operations
- Header: `<linux/fs.h>`
- Page Operations
 - `readpage` (Invoked by `read`)
 - `write_begin`, `write_end` (Invoked by `write`)
 - `writpage` (Invoked by `pdflush` kernel threads)
- Page Functions
 - `PageUptodate`, `PageDirty`, `PageWriteback`, `PageLocked`
 - `SetPageUptodate`, `ClearPageDirty`, `unlock_page`, ...
 - `page_address`, ...

Dive into the Real Driver



File System Pieces – Set 1

- Hardware Space
 - File System Design (Logic) – `real_sfs_ds.h`
 - File System Creator (Application) – `format_real_sfs.c`
- Kernel Space
 - File System Decoder (Driver) – `real_sfs_ops.[hc]`
 - File System Translator (Driver) – `real_sfs.c`
 - File System Show-er (VFS)
- User Space
 - File System Connector (mount)

What all did we learn?

- W's of a File System
- Three levels of File Systems
 - And relation between them
- FS relation with a Hard Disk Partition
- FS relation with /
 - Role of VFS
- Writing a FS module
 - Simplified Interaction with the Block Driver
 - 5 Operation Sets with VFS