Lecture 12: File Systems

Bo Tang @ 2021, Spring

Unix File System

- Original iNode format appeared in BSD 4.1
  - Berkeley Standard Distribution Unix
  - Similar structure for Linux Ext2/3
- File Number is index into iNode arrays
- Multi-level index structure
  - Great for little and large files
  - Asymmetric tree with fixed sized blocks
- Metadata associated with the file
  - Rather than in the directory that points to it
- UNIX Fast File System (FFS) BSD 4.2 Locality Heuristics:
  - Block group placement
  - Reserve space
- Scalable directory structure

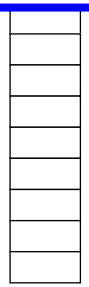
An "Almost Real" File System

- Pintos: `src/filesys/file.c, inode.c`

```c
/* An open file. */
struct file
  {
    struct inode *inode;        /* File's inode. */
    off_t pos;                  /* Current position. */
    bool deny_write;            /* Has file_deny_write() been called? */
  };
```

irect Data
ocks Blocks

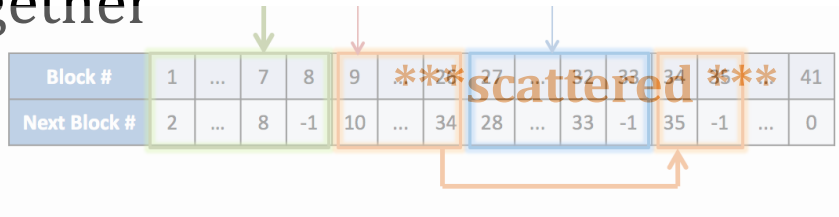
```c
/* In-memory inode. */
struct inode
  {
    struct list_elem elem;              /* Element in inode list. */
    block_sector_t sector;              /* Sector number of disk location. */
    int open_cnt;                       /* Number of openers. */
    bool removed;                       /* True if deleted, false otherwise. */
    int deny_write_cnt;                 /* 0: writes ok, >0: deny writes. */
    struct inode_disk data;             /* Inode content. */
  };
```

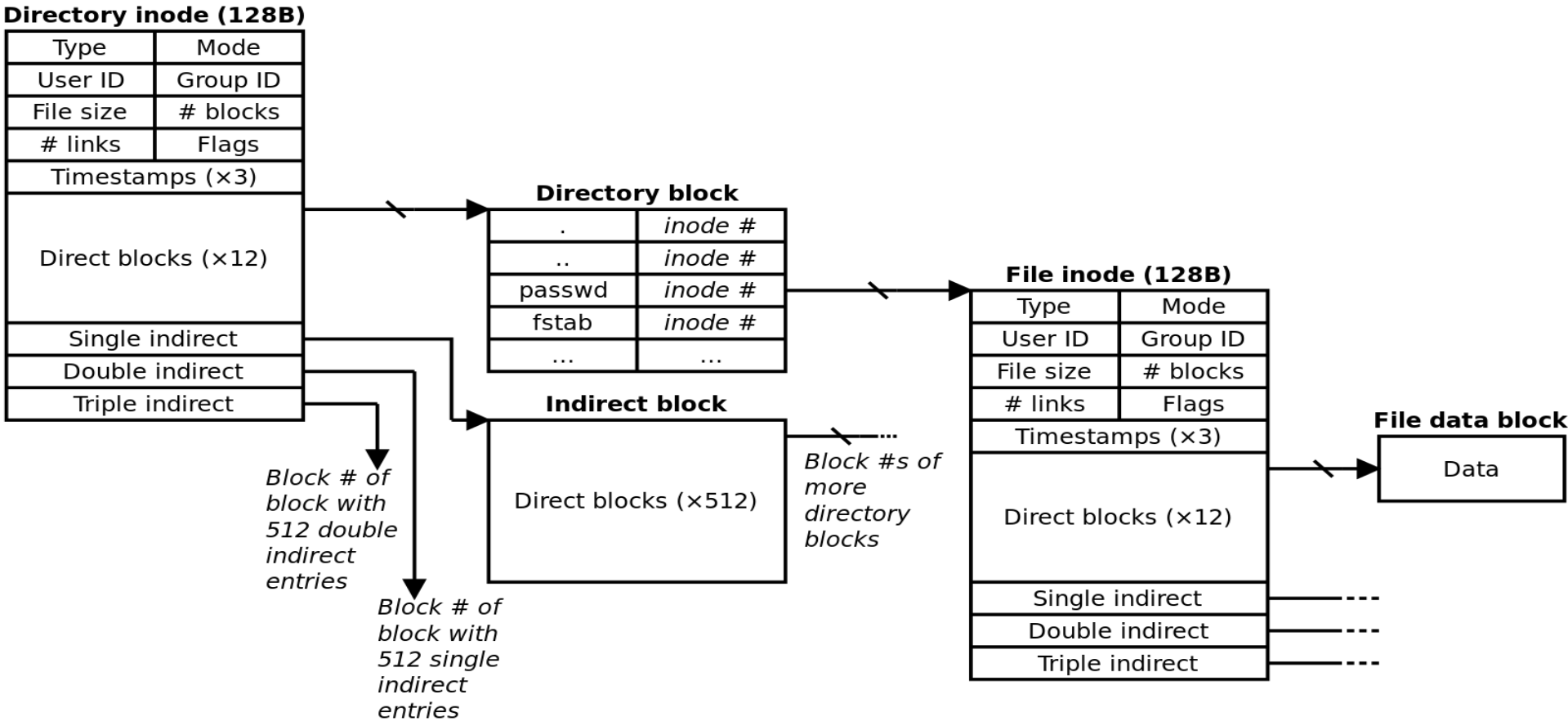


```c
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;               /* First data sector. */
    off_t length;                       /* File size in bytes. */
    unsigned magic;                     /* Magic number. */
    uint32_t unused[125];               /* Not used. */
  };
```

iNode

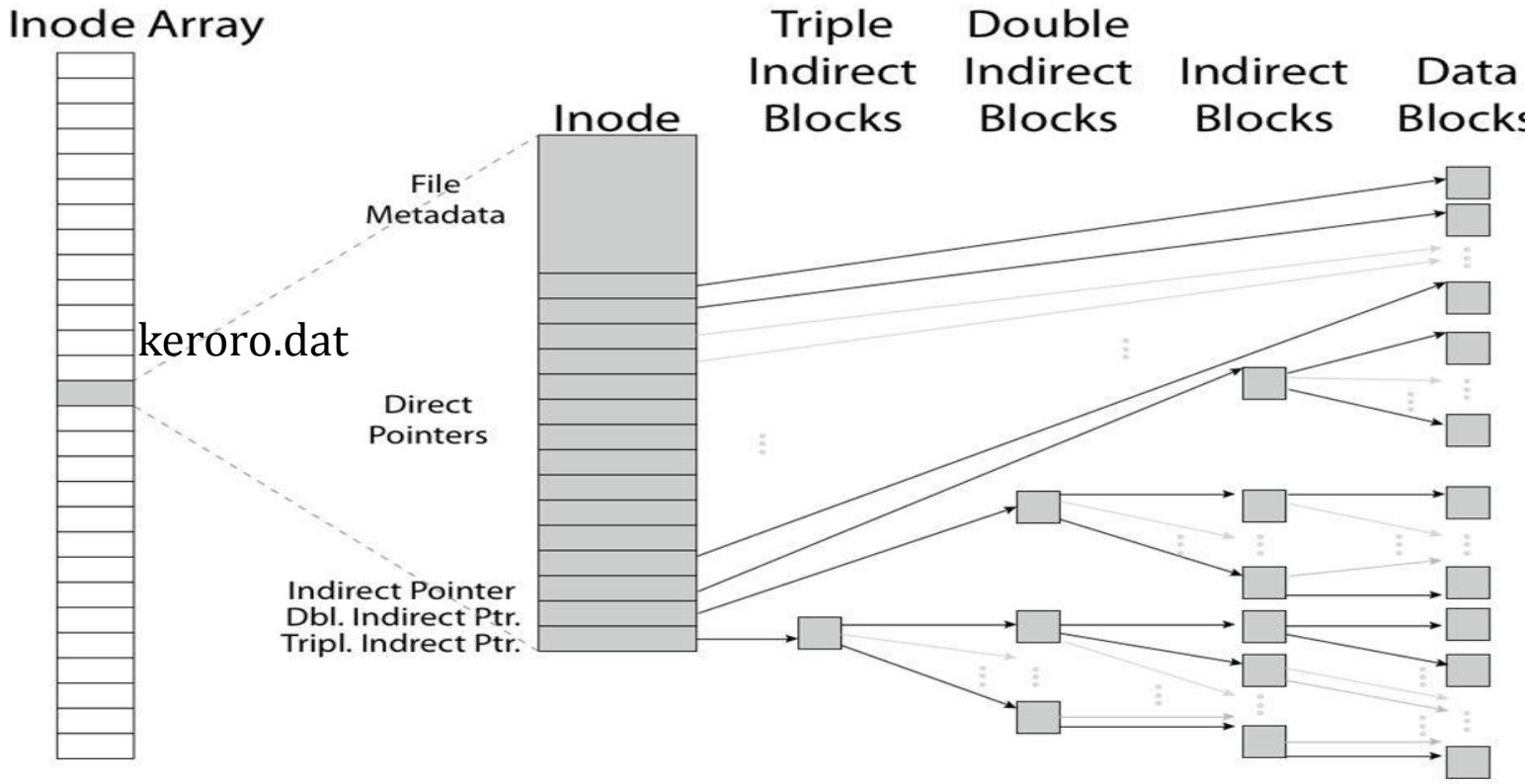
- All pointers of a file are located together
  - **VS. FAT: pointers of a file are scattered**
- One directory/file has one iNode





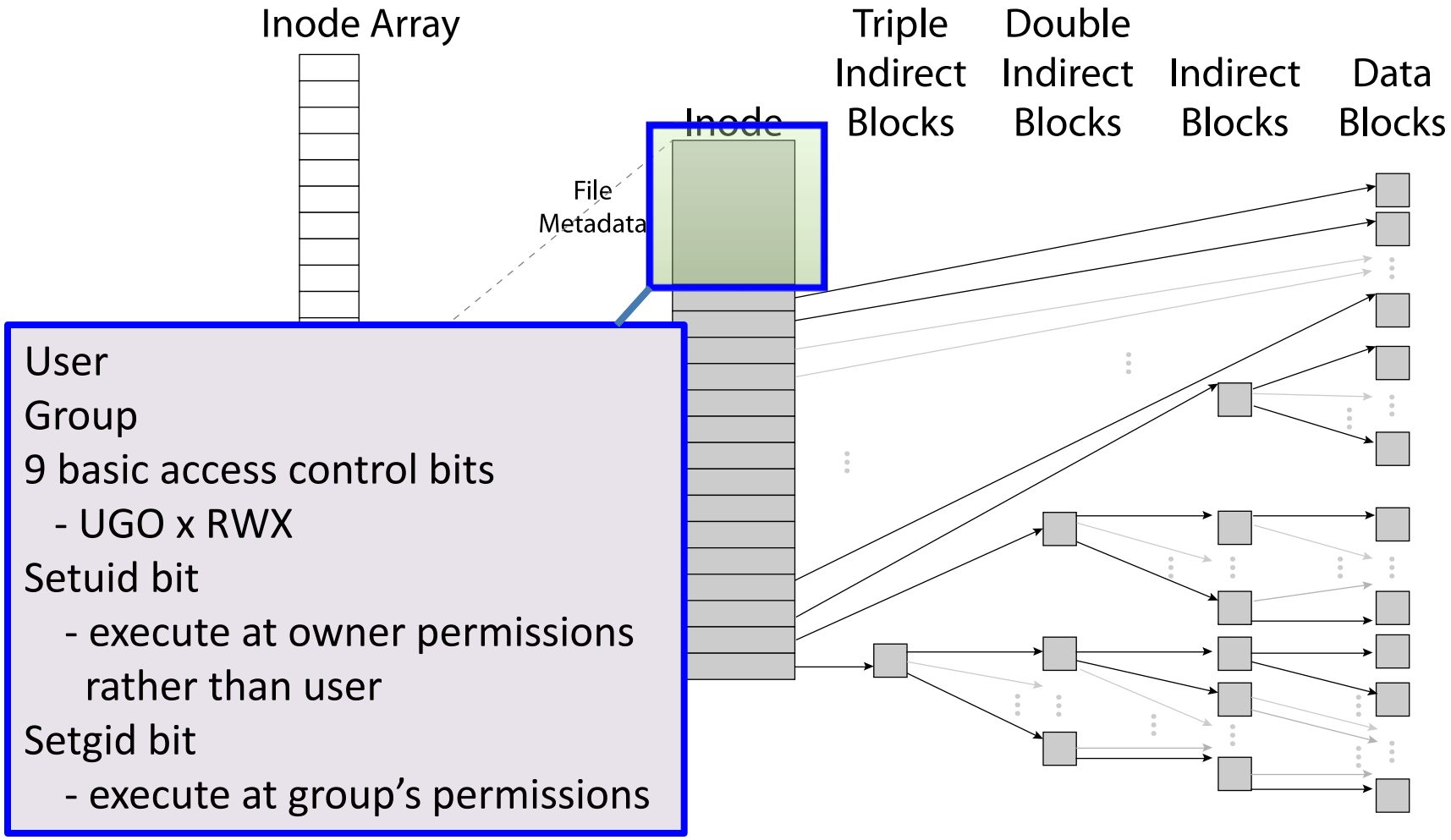
iNode

- iNode Table is an array of iNodes
- Pointers are unbalanced tree-based



File Attributes

- iNode metadata

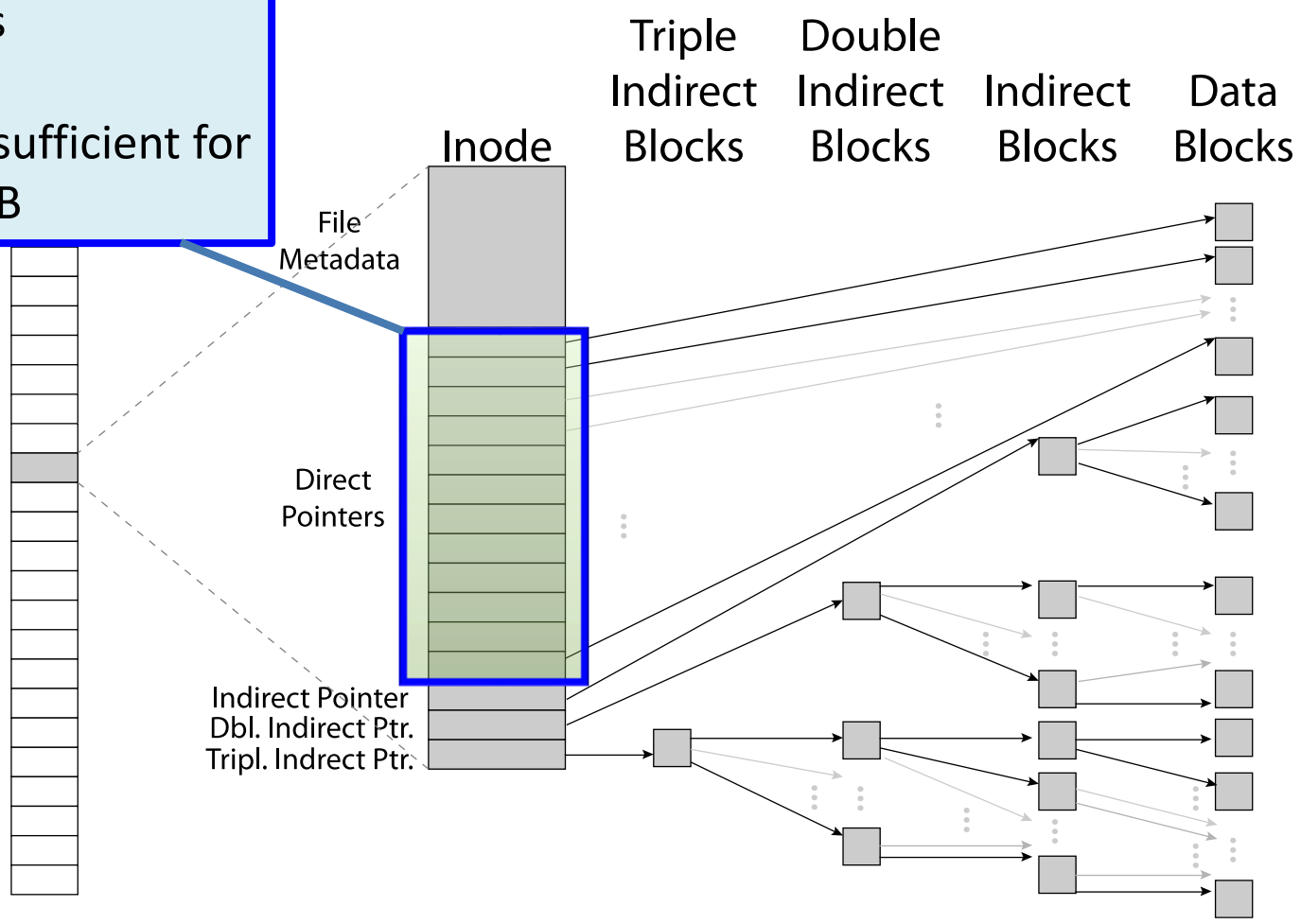


Data Storage

- Small files: 12 pointers direct to data blocks

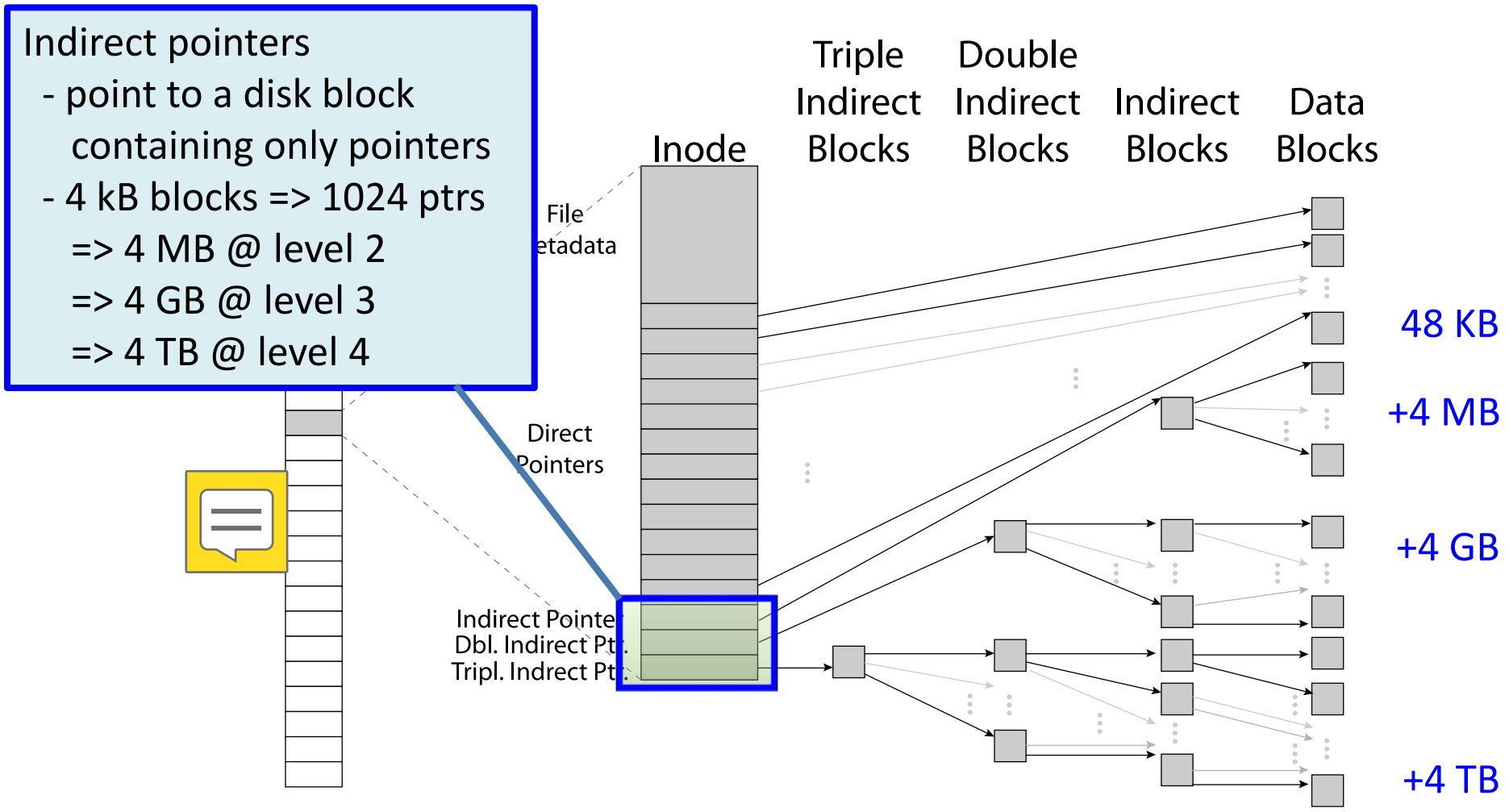
Direct pointers

4kB blocks ⇒ sufficient for files up to 48KB



Data Storage

- Large files: 1,2,3 level indirect pointers



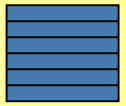
Index-node – file size

Reminder: Max file size != FS size

Number of direct blocks	12
Number of indirect blocks	1
Number of double indirect blocks	1
Number of triple indirect blocks	1
Block size	2^x bytes
Address length	4 bytes

$$12 \times 2^x + 1 \times 2^x/4 \times 2^x + 1 \times (2^x/4)^2 \times 2^x + 1 \times (2^x/4)^3 \times 2^x$$

File size = number of data blocks * Block size



contains "2^x / 4" addresses

Block size 2^x	Max size
1024 bytes = 2^{10}	approx. 16 GB
4096 bytes = 2^{12}	approx. 4 TB

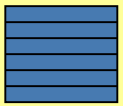
Index-node – file size

File size = number of data blocks x 2^x

Number of direct blocks	12
Number of indirect blocks	1
Number of double indirect blocks	1
Number of triple indirect blocks	1
Block size	2^x bytes
Address length	4 bytes

$$12 \times 2^x + 2^{2x-2} + 2^{3x-4} + 2^{4x-6}$$

The dominating factor.



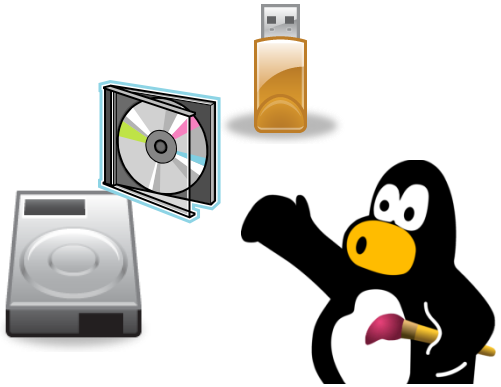
contains "2^x / 4" addresses

Block size 2^x	Max size
1024 bytes = 2^{10}	approx. 16 GB
4096 bytes = 2^{12}	approx. 4 TB

Reminder: Max file size != FS size

Ext 2/3/4

- Disk layout
- Directory
- Hard and Soft Links
- Consistency



File System Ext

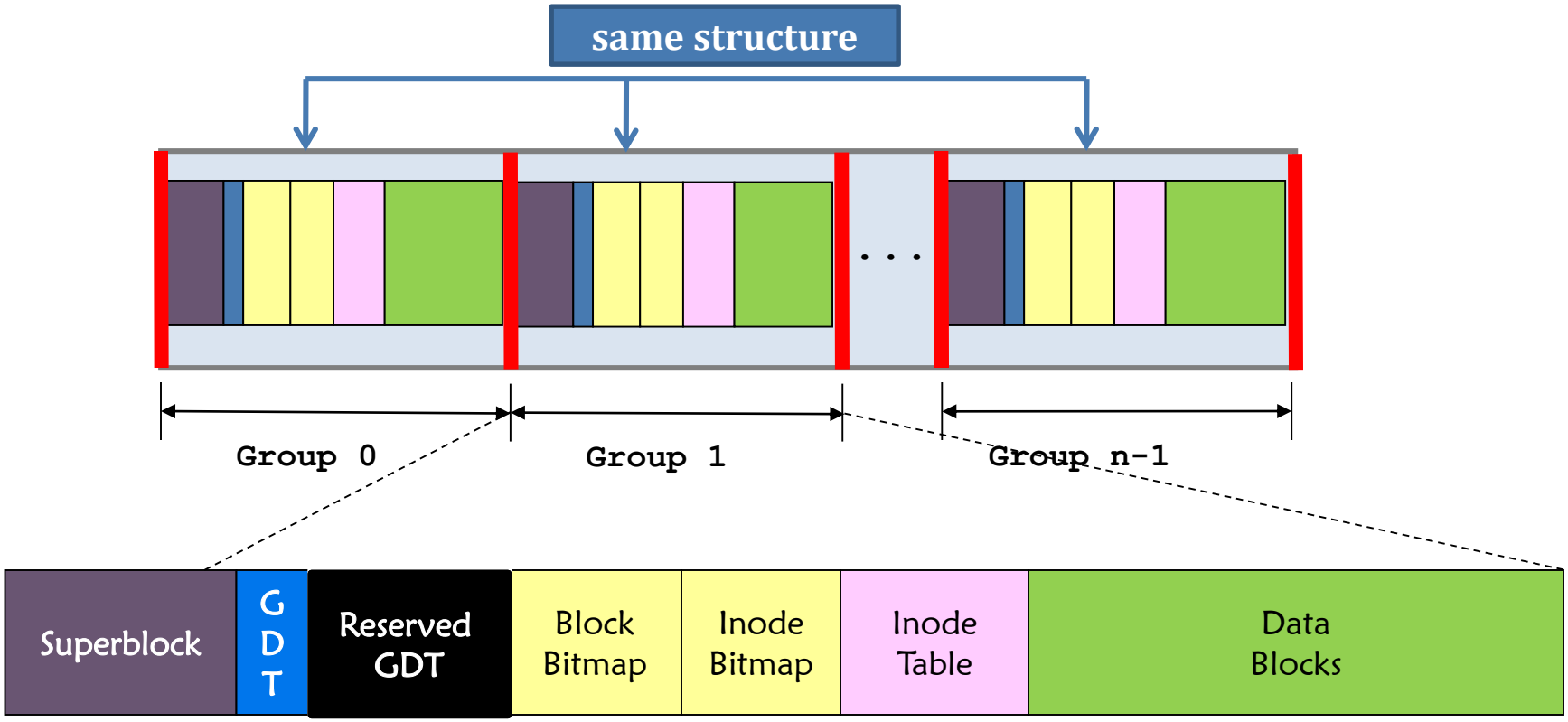
- The latest default FS for Linux distribution is the **Fourth Extended File System**, **Ext4** for short.

- For Ext2 & Ext3:
  - Block size: 1,024, 2,048, or 4,096 bytes.
  - Block address size: 4 bytes => # of block addresses = 2^{32}

$2^x \times 2^{32} = 2^{32+x}$			
Block size	$2^x = 1024$	$2^x = 2048$	$2^x = 4096$
File System size	4 TB	8 TB	16 TB

Ext2/3 – Block groups

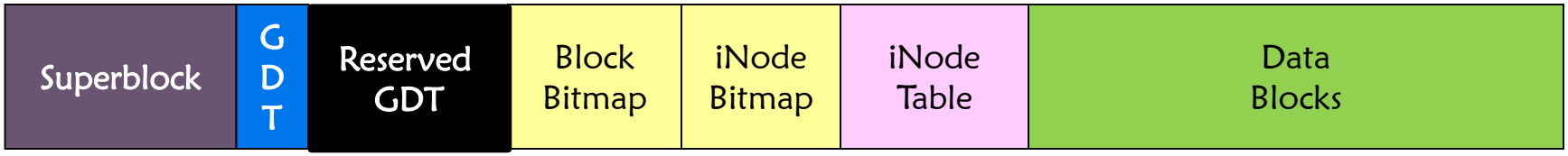
- The file system is divided into **block groups** and every block group has the **same structure**



Ext2/3 – FS layout

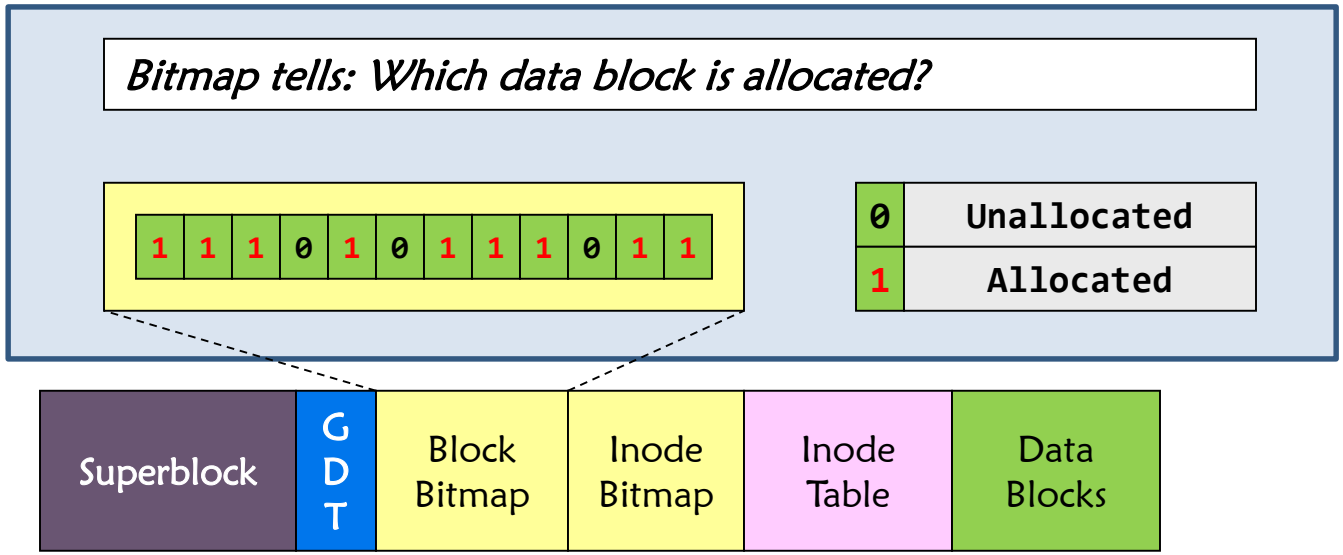
- Layout of one block group is as follows:

Superblock	Stores FS specific data. E.g., the total number of blocks, etc.
GDT – Group Descriptor Table	It stores: - The locations of the **block bitmap**, the **iNode bitmap**, and the **iNode table**. - Free block count, free iNode count, etc…
Block Bitmap	A bit string that represents if a block is allocated or not.
iNode Bitmap	A bit string that represents if an inode (index-node) is allocated or not.
iNode Table	An array of inodes ordered by the inode #.
Data Blocks	An array of blocks that stored files.



https://www.ibm.com/developerworks/cn/linux/l-cn-filesrc5/

Ext2/3 – Block Bitmap & iNode Bitmap

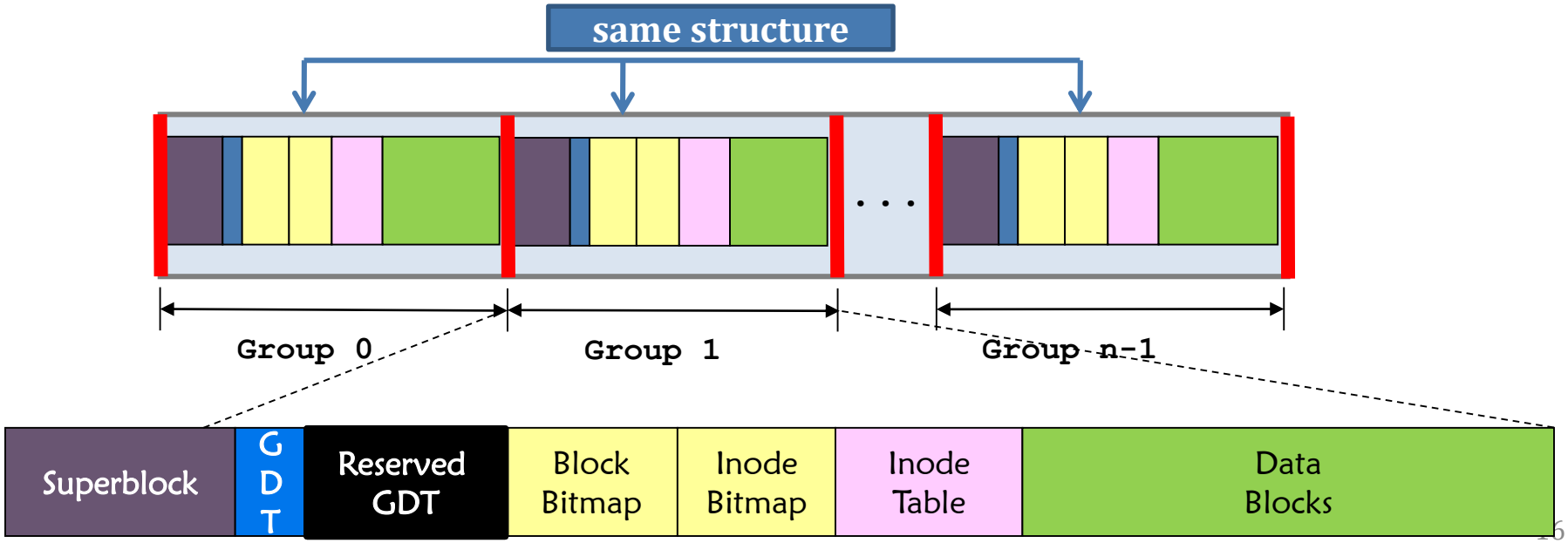


- iNode Bitmap
  - A bit string that represents if an iNode (index-node) is allocated or not

➔ implies that the **number of files in the file system is fixed**!

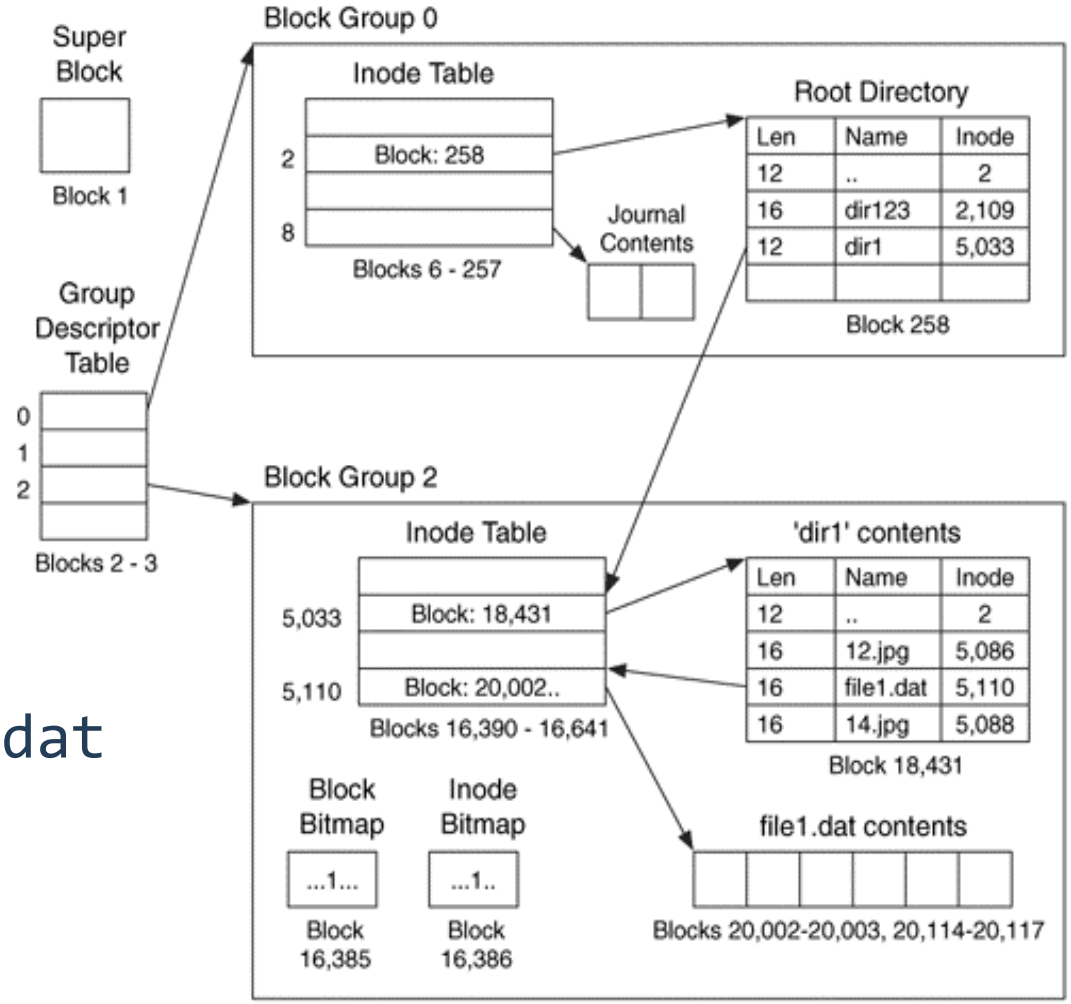
Ext2/3 – Block groups

- Why having groups?
- For **(1) performance** and **(2) reliability**
  - (1) Performance: spatial locality.
    - Group iNodes and data blocks of related files together
  - (2) Reliability: superblock and GDT are **replicated** in each block group (yes, very reliable!)



Linux Example: Ext2/3 Disk Layout

- Disk divided into block groups
  - Each group has two block-sized bitmaps (free blocks/inodes)
  - Block sizes settable at format time: 1K, 2K, 4K, 8K…
  - Provides locality



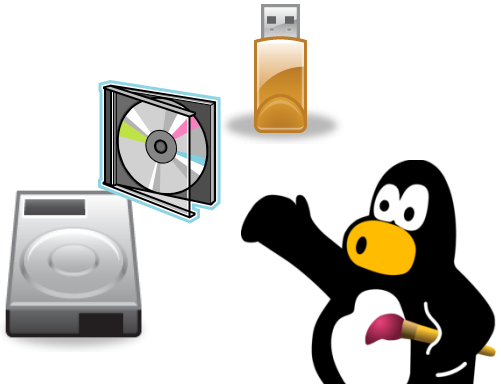
- Example: create a `file1.dat` under `/dir1/` in Ext3

Ext 2/3

- Disk layout;

- Directory;

- Hard and Soft Links.

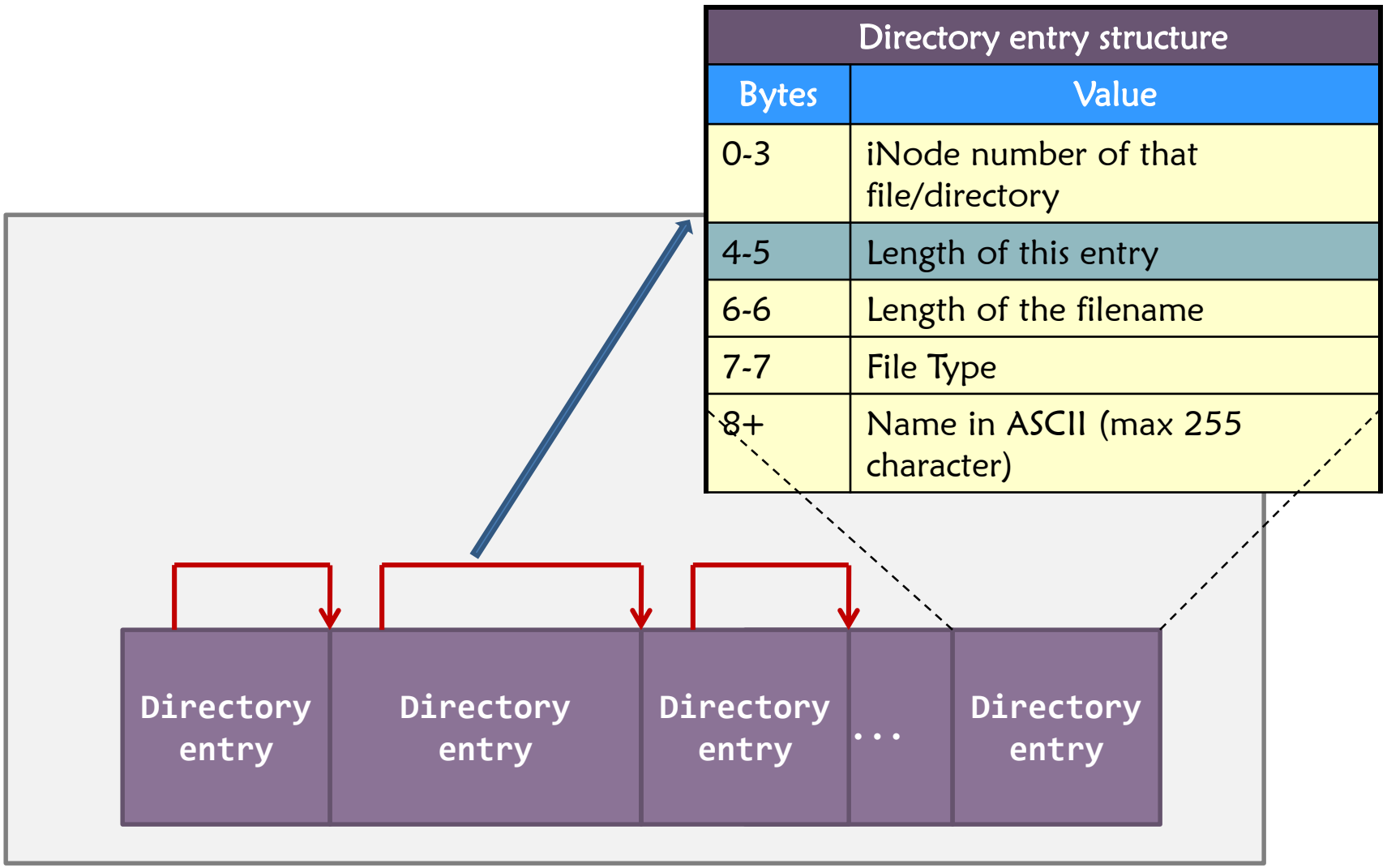


Ext2/3 – iNode structure (for 1 file)

iNode Structure (128 bytes long)	
Bytes	**Value**
0-1	File type and permission
2-3	User ID
4-7	Lower 32 bits of file sizes in bytes
8-23	Time information
24-25	Group ID
26-27	Link count (will discuss later)
…	…
40-87	12 direct data block pointers
88-91	Single indirect block pointer
92-95	Double indirect block pointer
96-99	Triple Indirect block pointer
…	…
108-111	Upper 32 bits of file sizes in bytes

The locations of the data blocks are stored in the inode.

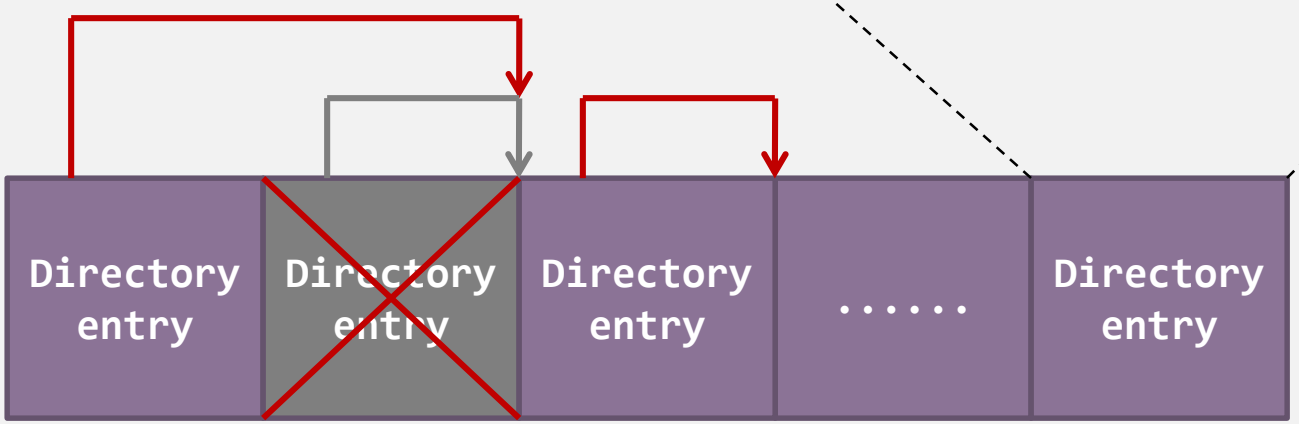
Ext2/3 –directory entry in a directory block



Ext2/3 – File Deletion

Directory entry structure	
Bytes	**Value**
0-3	Inode number for that directory
4-5	Length of this entry
6-6	Length of the filename
7-7	File Type
8+	Name in ASCII (max 255 character)

File deletion is just an update of the "entry length" of the previous entry.

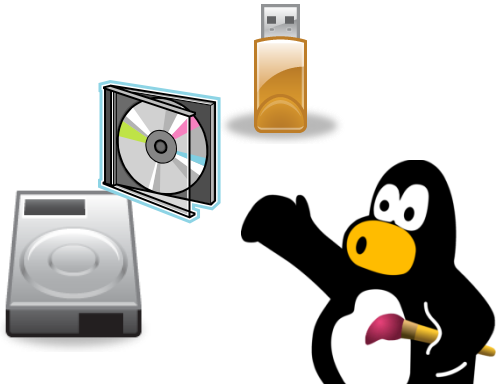


Ext 2/3

- Disk layout;

- Directory;

- **Hard and Soft Links.**



Ext2/3 – link file: what is a hard link

- A hard link is a **directory entry** pointing to the iNode of an existing file.

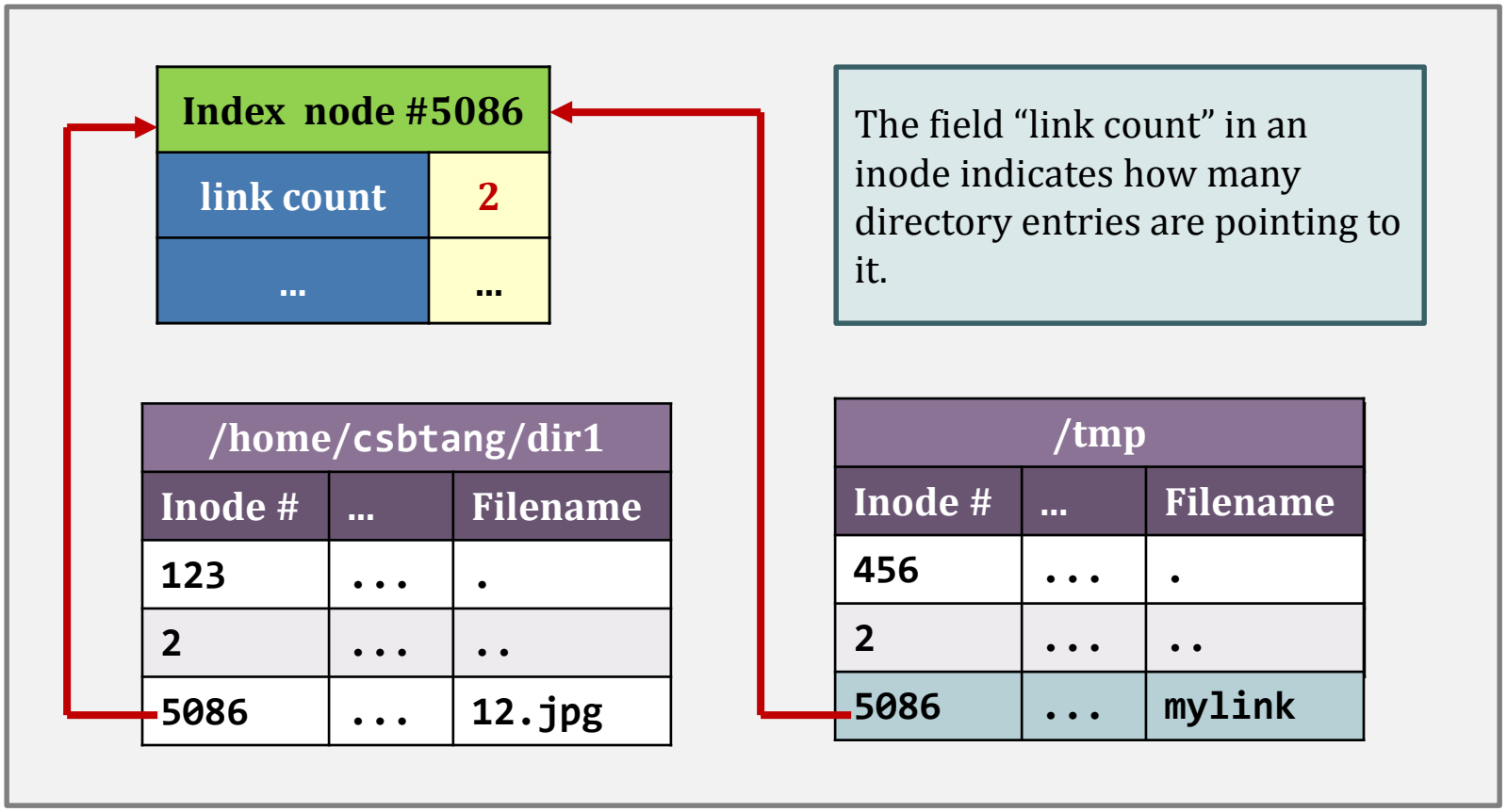
```
# ln /home/csbtang/dir1/12.jpg /tmp/mylink
```

/home/csbtang/dir1		
Inode #	...	Filename
123	...	.
2	...	..
5086	...	12.jpg

/tmp		
Inode #	...	Filename
456	...	.
2	...	..
5086	...	mylink

Ext2/3 – link file: what is a hard link

- That **file can accessed through two different pathnames**.

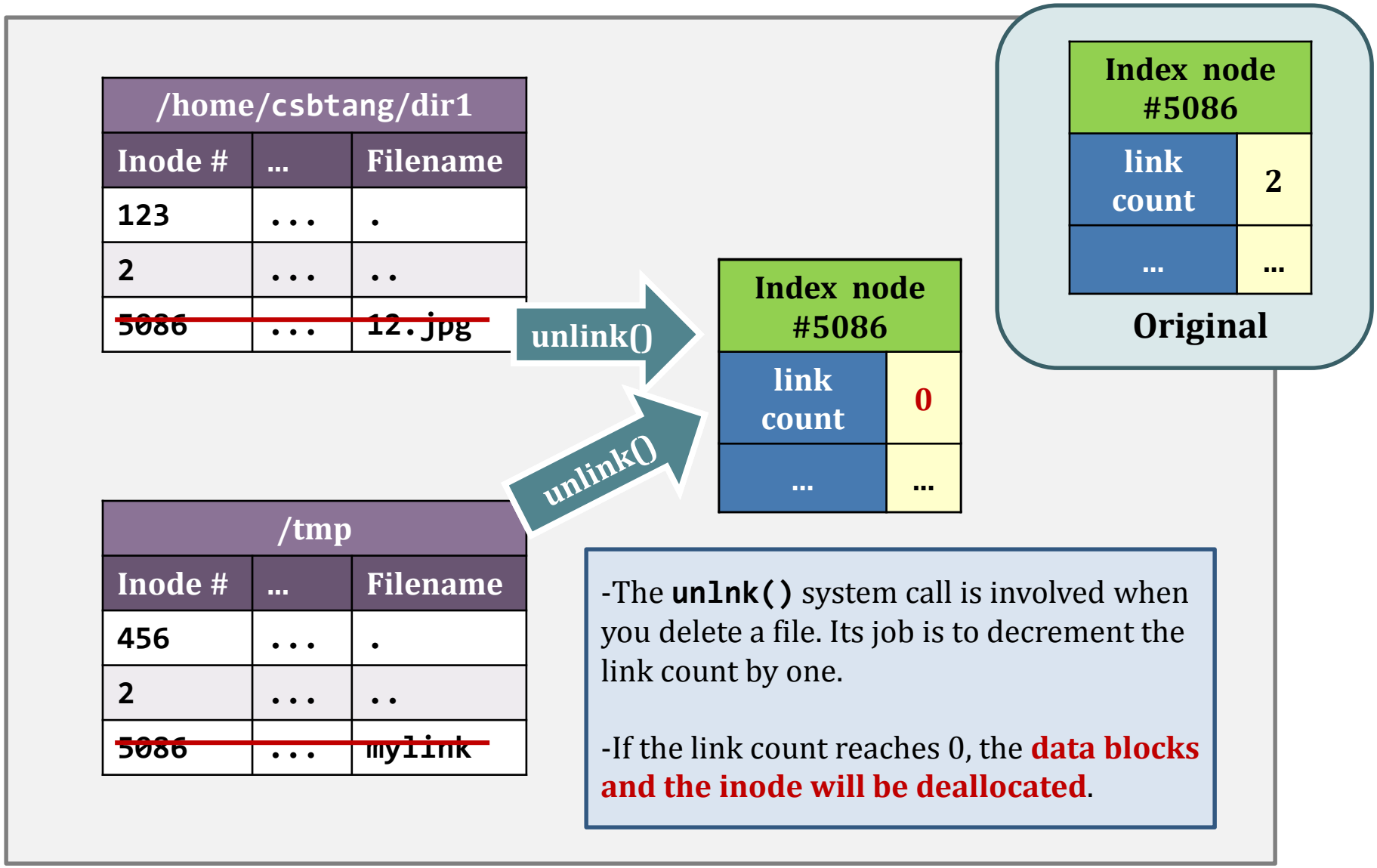


Ext2/3 – link file: examples on hard link

- Let's look at the link count of the root directory.
  - **20 sub-directories**: have a link "..";
  - **Root directory**: "." and ".." pointing to itself;
  - 20 + 2 = 22.

```
# ls -F /
bin/    home/            media/  rules.log  tmp/
boot/   initrd.img@      mnt/    sbin/      usr/
cdrom/  initrd.img.old@  opt/    selinux/   var/
dev/    lib/             proc/   srv/       vmlinuz@
etc/    lost+found/      root/   sys/       vmlinuz.old@
# stat /
  File: `/'
  Size: 4096          Blocks: 8          IO Block: 4096   directory
Device: 806h/2054d    Inode: 2           Links: 22
......
$ _
```

Ext2/3 – removing file and link count

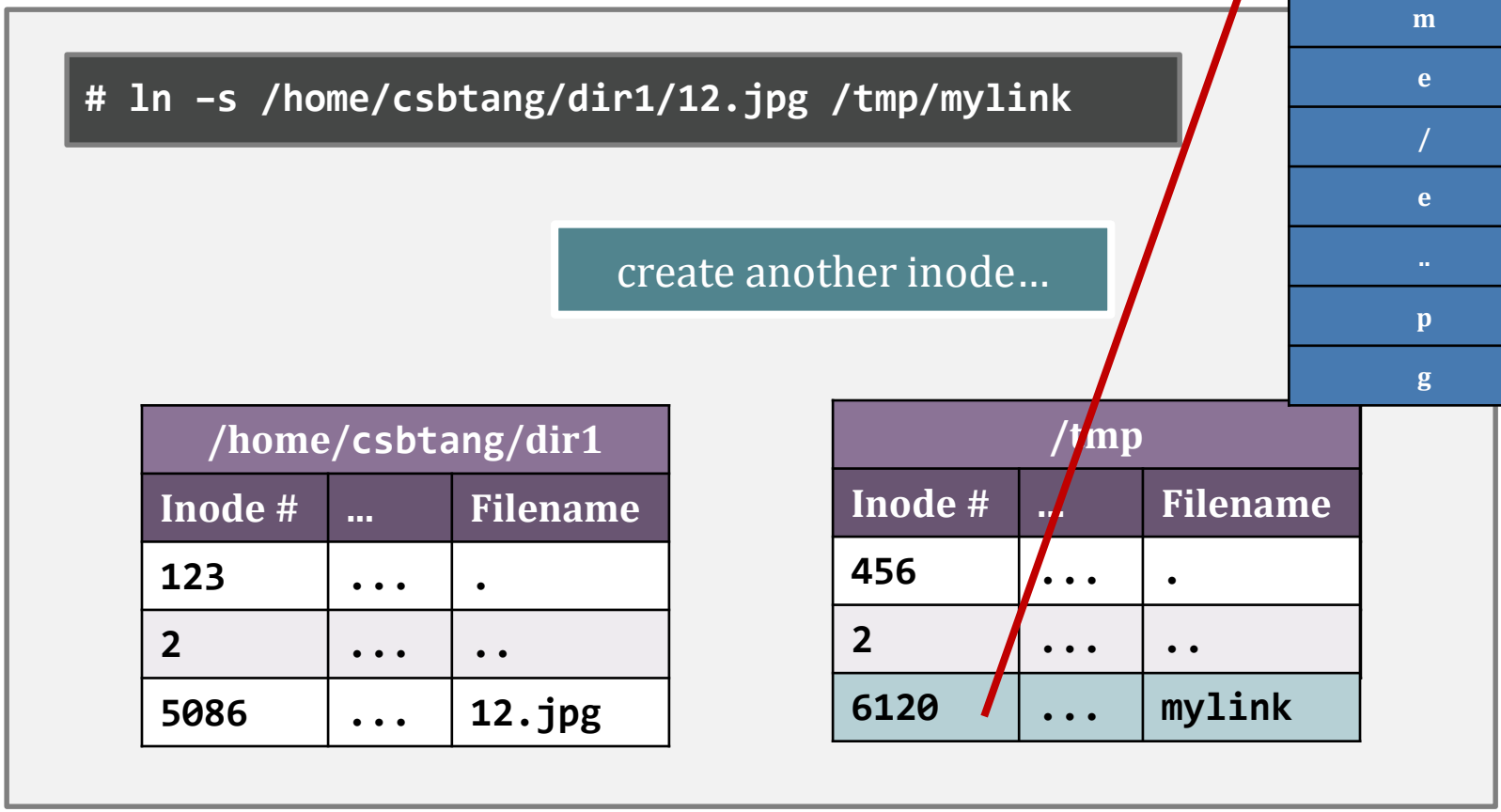


-The **`unlnk()`** system call is involved when you delete a file. Its job is to decrement the link count by one.

-If the link count reaches 0, the **data blocks and the inode will be deallocated**.

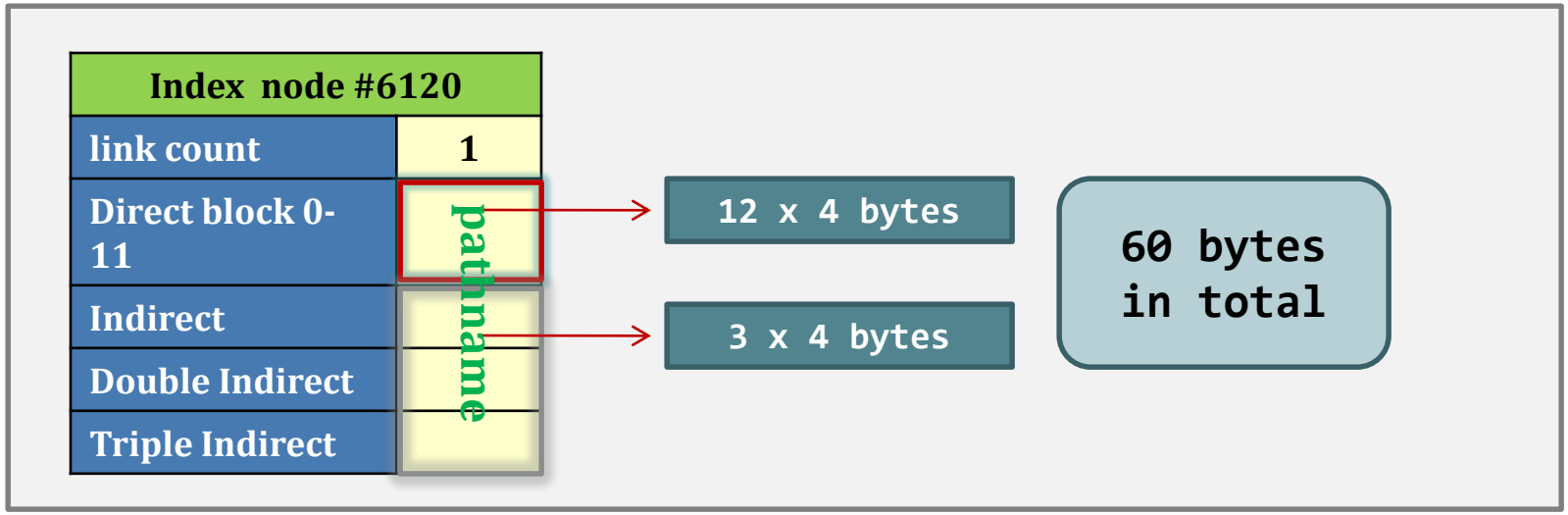
Ext2/3 – symbolic link

- A symbolic link **creates a new inode**
  - Vs hard link won't (but point to the same inod



Ext2/3 – symbolic link

- Symbolic link is pointing to a new iNode whose target's **pathname** are stored using the space originally designed for **12 direct block and the 3 indirect block pointers** if the pathname is shorter than 60 characters.
  - Use back a normal inode + **one direct data block** to hold the long pathname otherwise



Summary of Links

- Hard link
  - Sets another directory entry to contain the file number for the file
  - Creates another name (path) for the file
  - Each is “first class”
- Soft link or Symbolic Link
  - Directory entry contains the path and name of the file
  - Map one name to another name

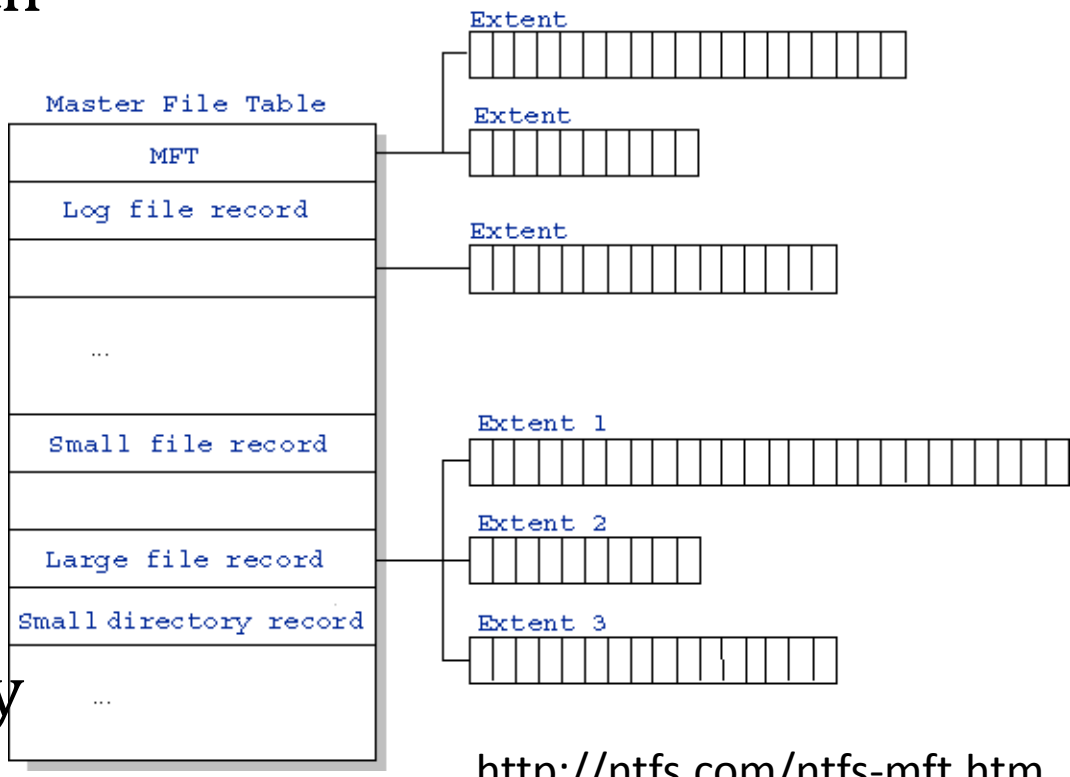
Property/Action		Symbolic link	Hard link
When the link is deleted		Target remains unchanged	Reference counter is decremented; when it reaches 0, the target is deleted
When target is moved		Symbolic link becomes invalid	Hard link remains valid
Relative path		Allowed	N/A
Crossing filesystem boundaries		Supported	Not supported (target must be on same filesystem)
Windows	For files	Windows Vista and later[20] (administrator rights required)	Yes
	For folders		No
Unix	For files	Yes	Yes
	For directories	Yes	Partial[21]

NTFS

- New Technology File System (NTFS)
  - Default on Microsoft Windows systems
- Variable length extents
  - Rather than fixed blocks
- Everything (almost) is a sequence of <attribute: value> pairs
  - Meta-data and data
- Mix direct and indirect freely
- Directories organized in B-tree structure by default

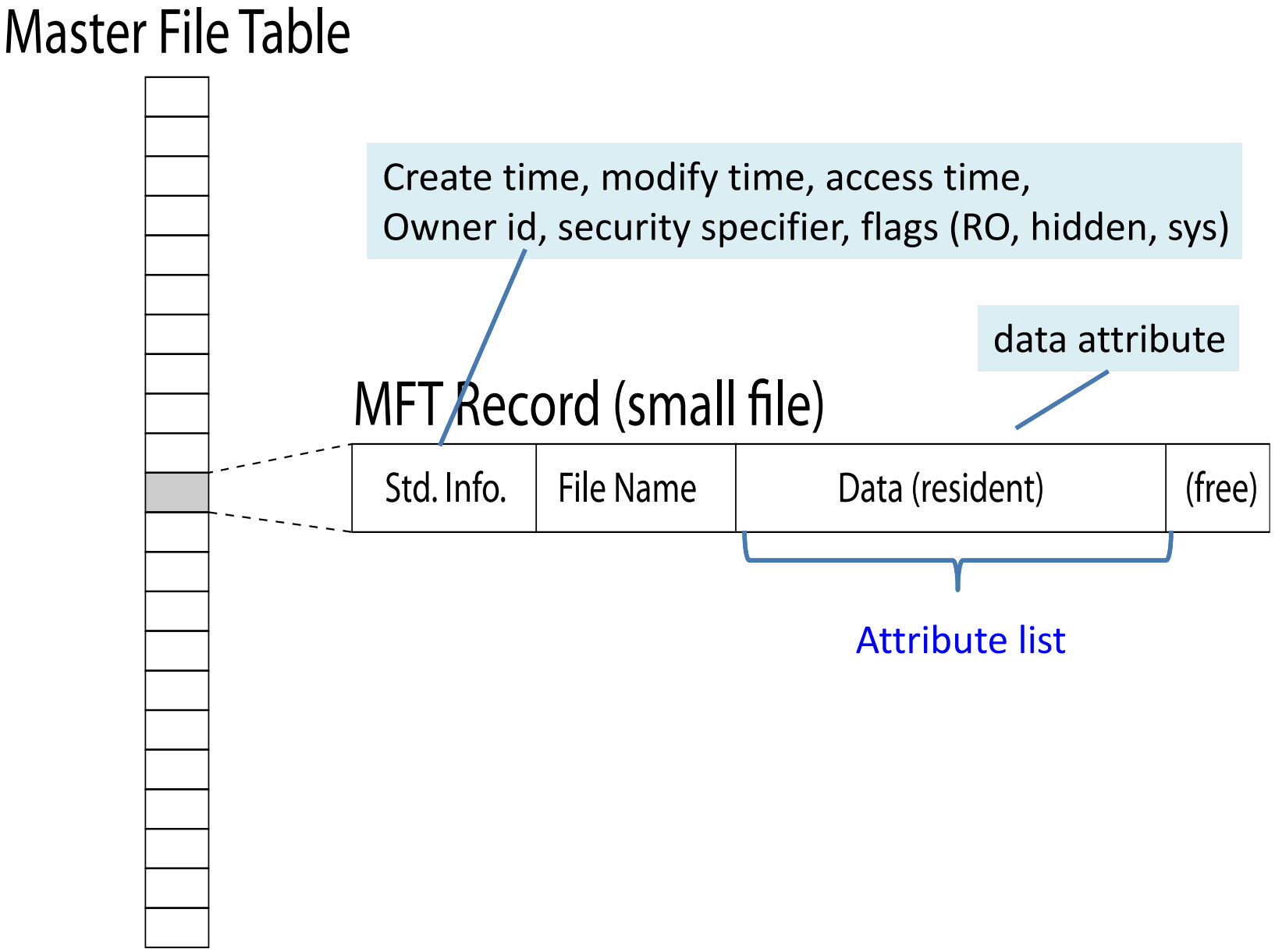
NTFS

- Master File Table
  - Database with Flexible 1KB entries for metadata/data
  - Variable-sized attribute records (data or metadata)
  - Extend with variable depth tree (non-resident)
- Extents – variable length contiguous regions
  - Block pointers cover runs of blocks
  - Similar approach in Linux (ext4)
  - File create can provide hint as to size of file
- Journaling for reliability

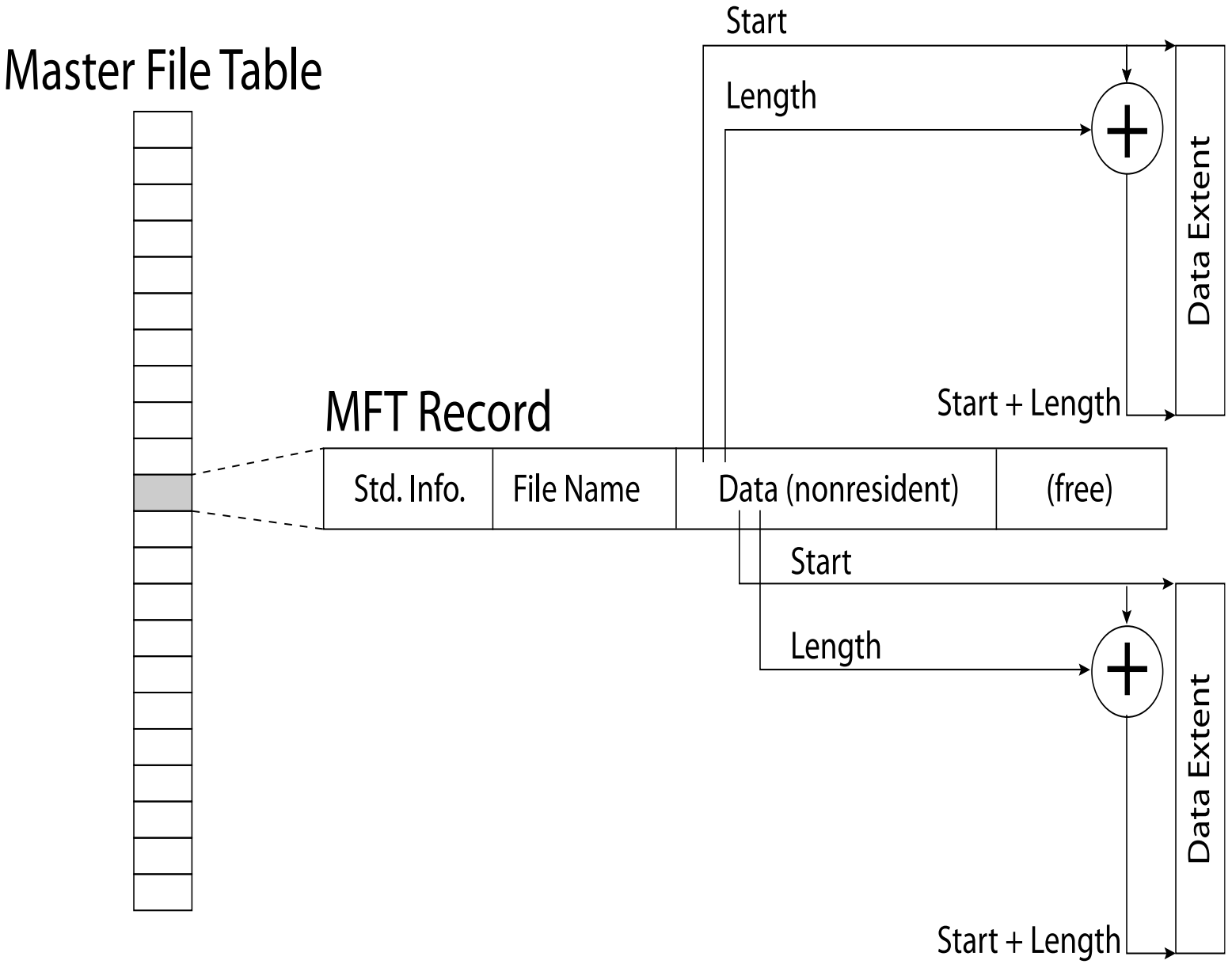


http://ntfs.com/ntfs-mft.htm

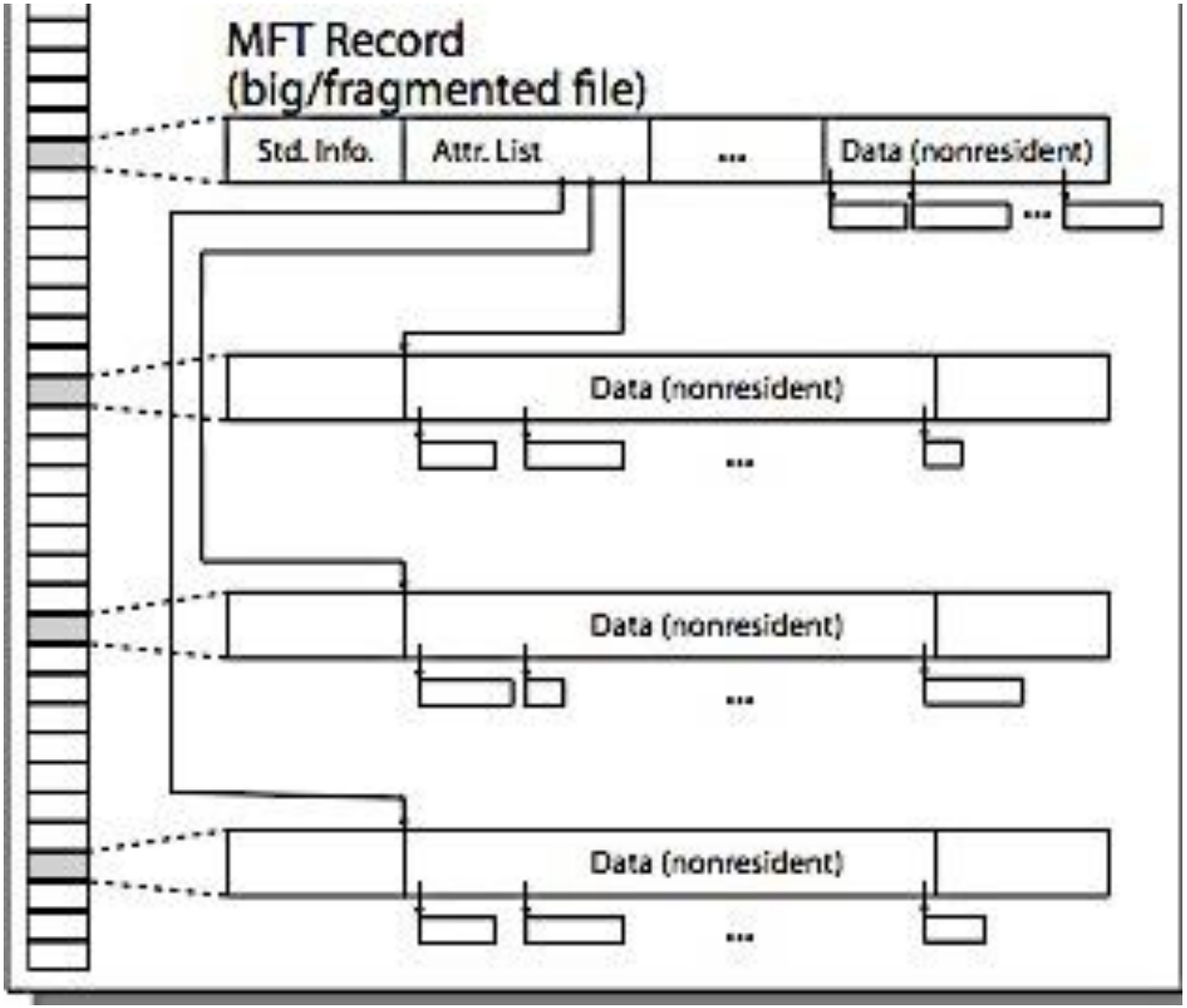
NTFS Small File

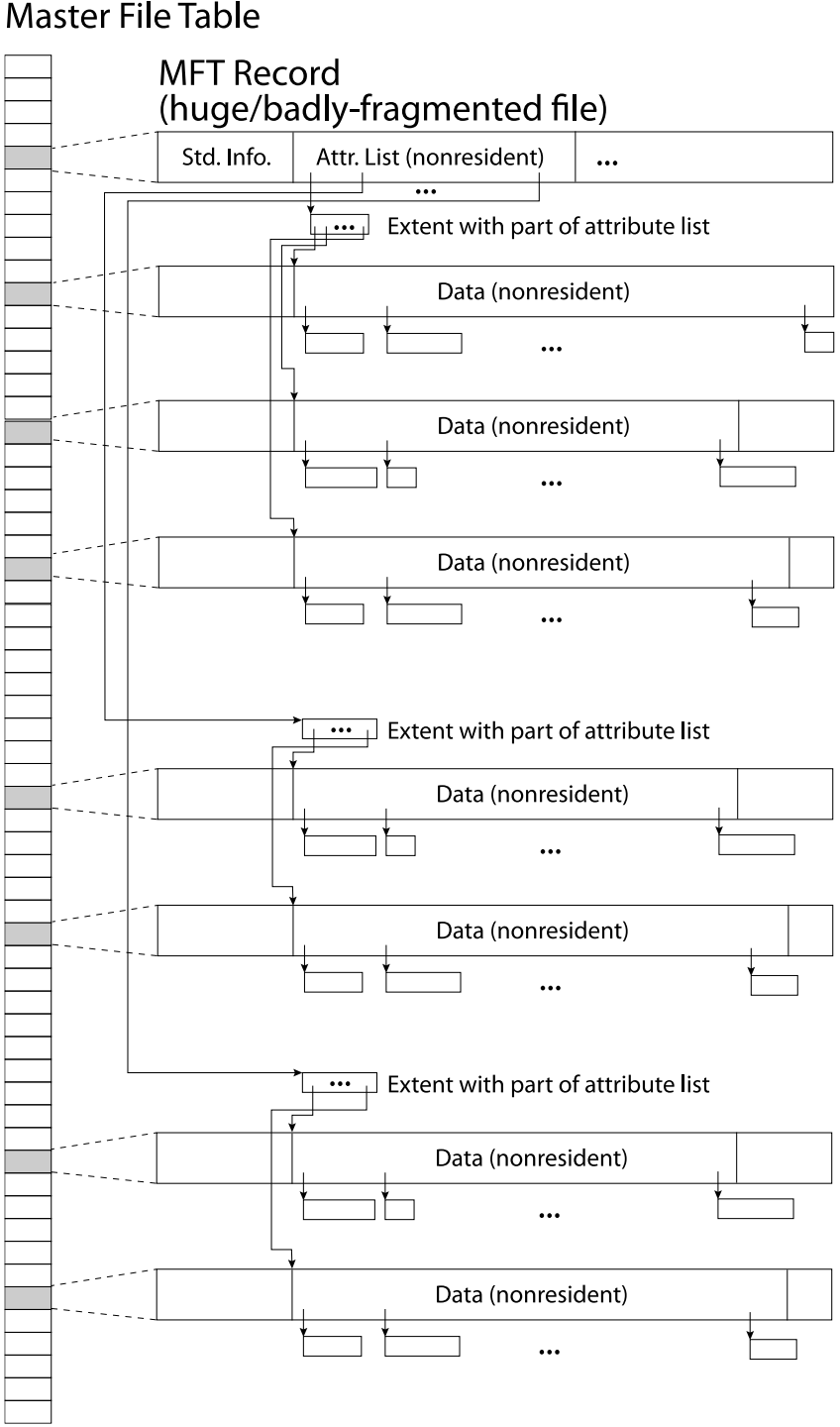


NTFS Medium File



NTFS Multiple Indirect Blocks

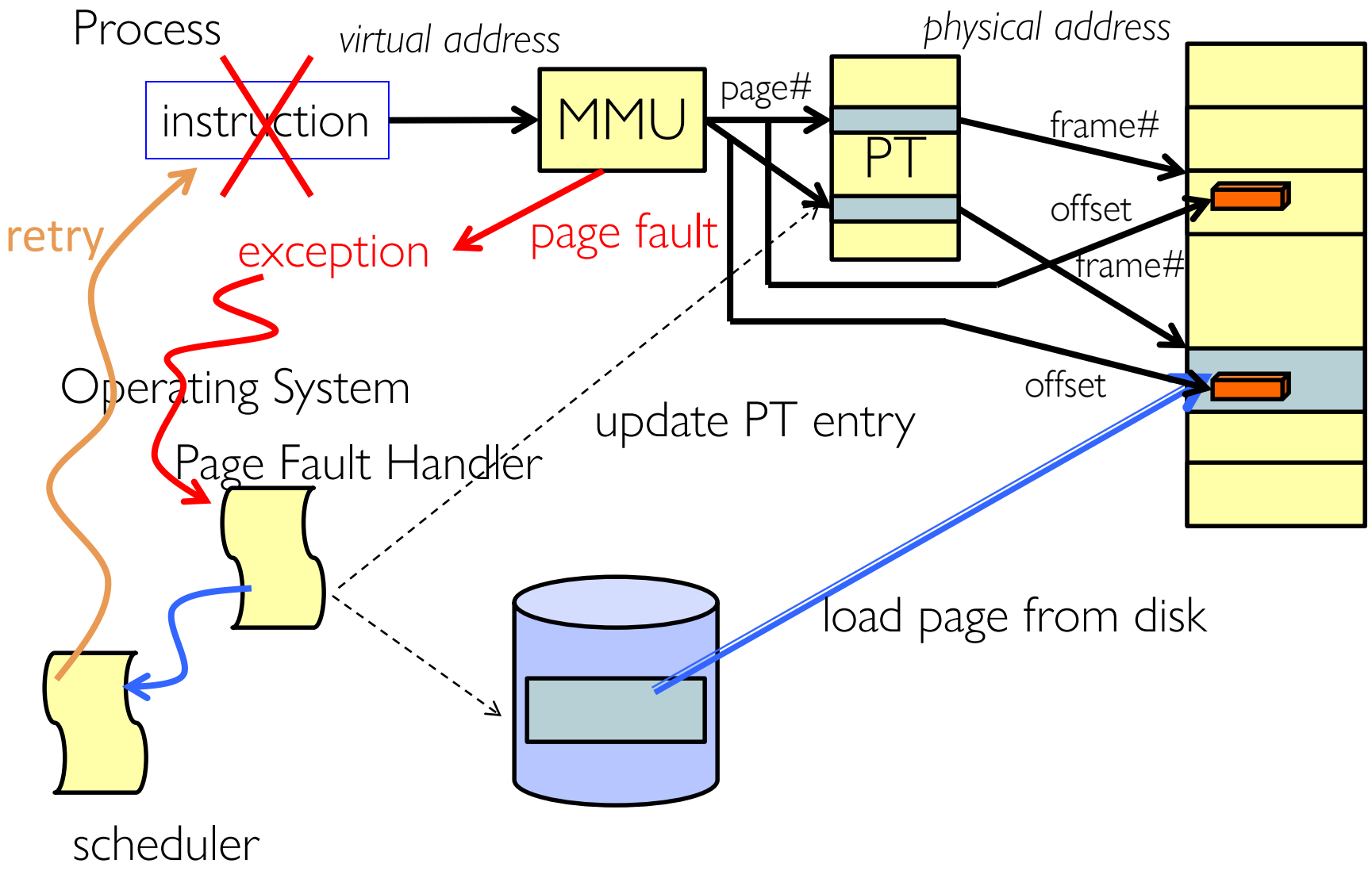



Master File Table
MFT Record
(huge/badly-fragmented file)
Std. Info.
Attr. List (nonresident)
...
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)

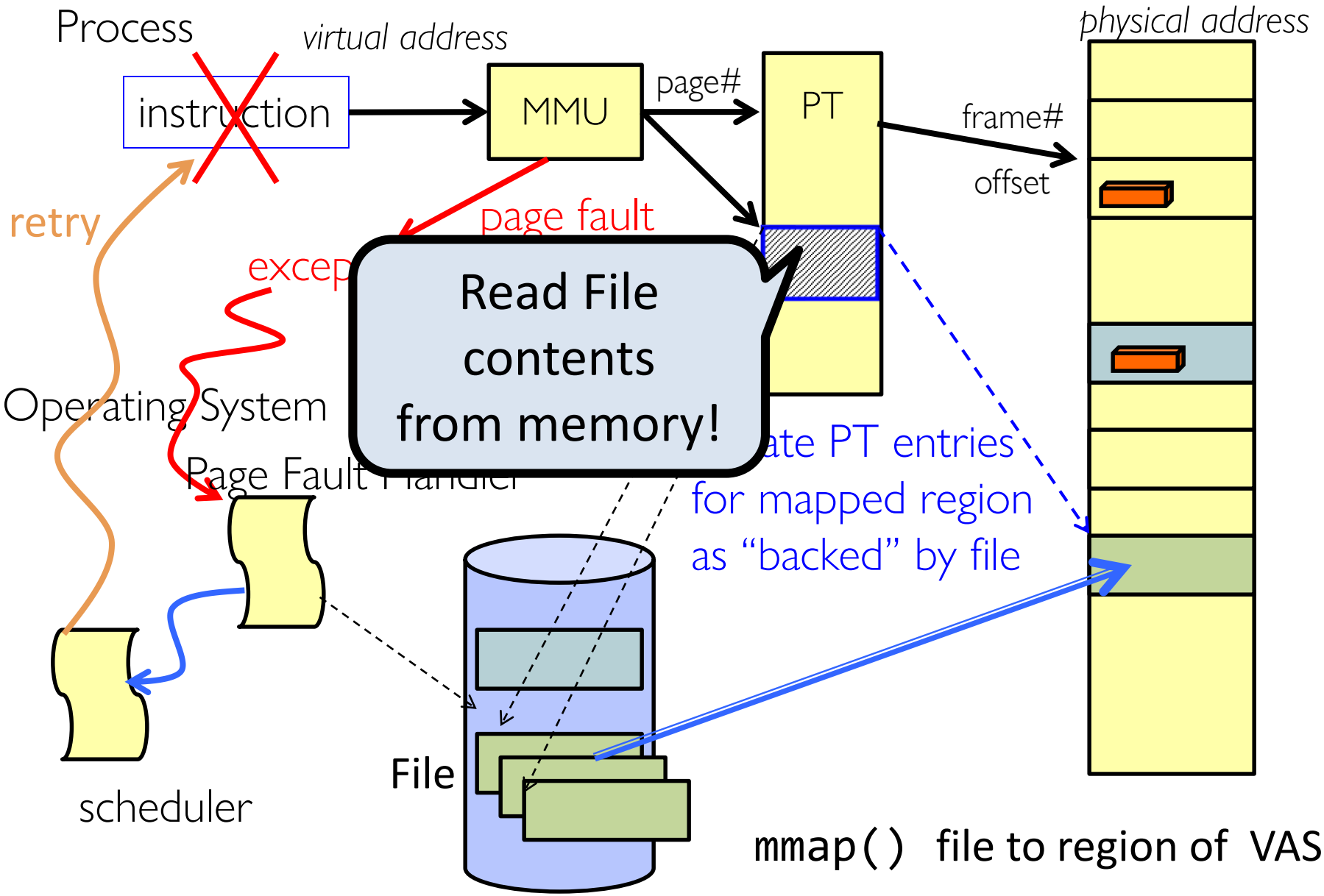
Memory Mapped Files

- Traditional I/O involves explicit transfers between buffers in process address space to/from regions of a file
  - This involves multiple copies into caches in memory, plus system calls
- What if we could "map" the file directly into an empty region of our address space
  - Implicitly "page it in" when we read it
  - Write it and "eventually" page it out
- Executable files are treated this way when we `exec` the process!!

Recall: Who Does What, When?



Using Paging to `mmap()` Files



File System Summary (1/2)

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for size, access and usage patterns
  - Maximize sequential access, allow efficient random access
- File defined by header, called "iNode"
- Naming: translating from user-visible names to actual sys resources
  - Directories used for naming for local file systems
  - Linked or tree structure stored in files
- Multilevel Indexed Scheme
  - iNode contains file info, direct pointers to blocks, indirect blocks, doubly indirect, etc..
  - NTFS: variable extents not fixed blocks, tiny files data is in header

File System Summary (2/2)

- 4.2 BSD Multilevel index files
  - iNode contains pointers to actual blocks, indirect blocks, double indirect blocks, etc.
  - Optimizations for sequential access: start new files in open ranges of free blocks, rotational optimization
- File layout driven by freespace management
  - Integrate freespace, iNode table, file blocks and dirs into block group
- Deep interactions between memory management, file system, sharing
  - `mmap()`: map file or anonymous segment to memory

Thank You!