B-TREE BASED FILE SYSTEM (BTRFS)

TEAM MEMBERS (ROLL NO):

- *RANAJIT SAHA (20172119)*
- *SANJOY CHOWDHURY (20172123)*
- *ADITYA NEEL (20172135)*
- *GOURAV PANWAR (20172124)*

CONTENT

- Features of BTRFS
- Data and metadata structure
- BTRFS data structure
- Directory Indexing
- BTRFS organisation
- Future Scope

BTRFS

- Journaling file system, that it keeps a "journal" of where files are located on the disk and of any other changes to the disk
- Extent based file system
- Transparent compression
- Well implemented snapshot feature
- Copy-on-Write (CoW)
- Dynamic inode allocation

THE FOREST VIEW (1/5)

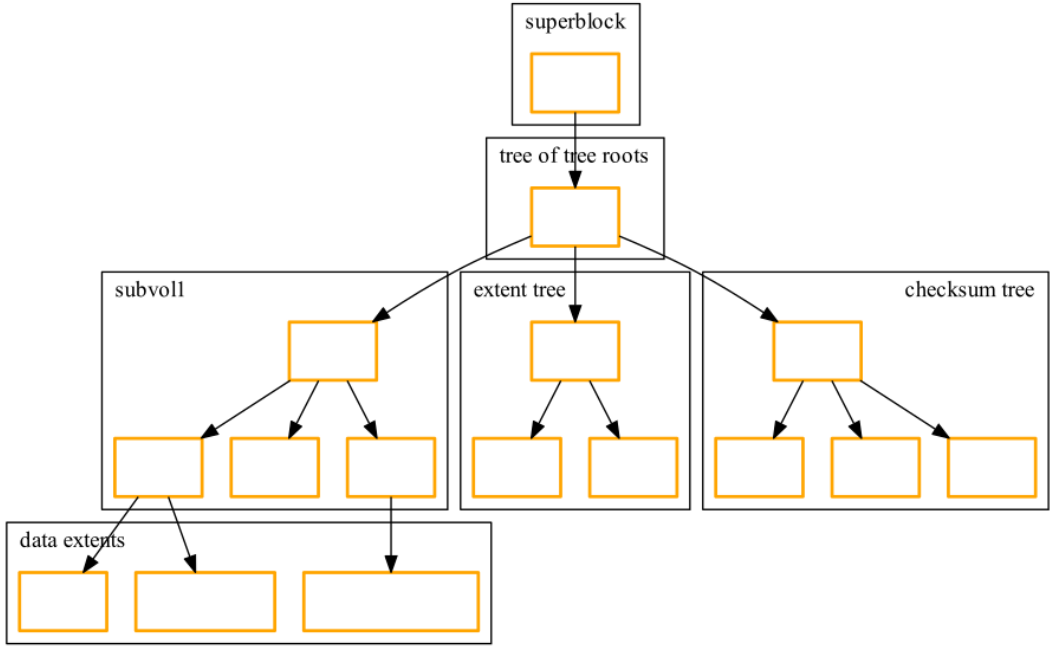


Fig: A forest file system

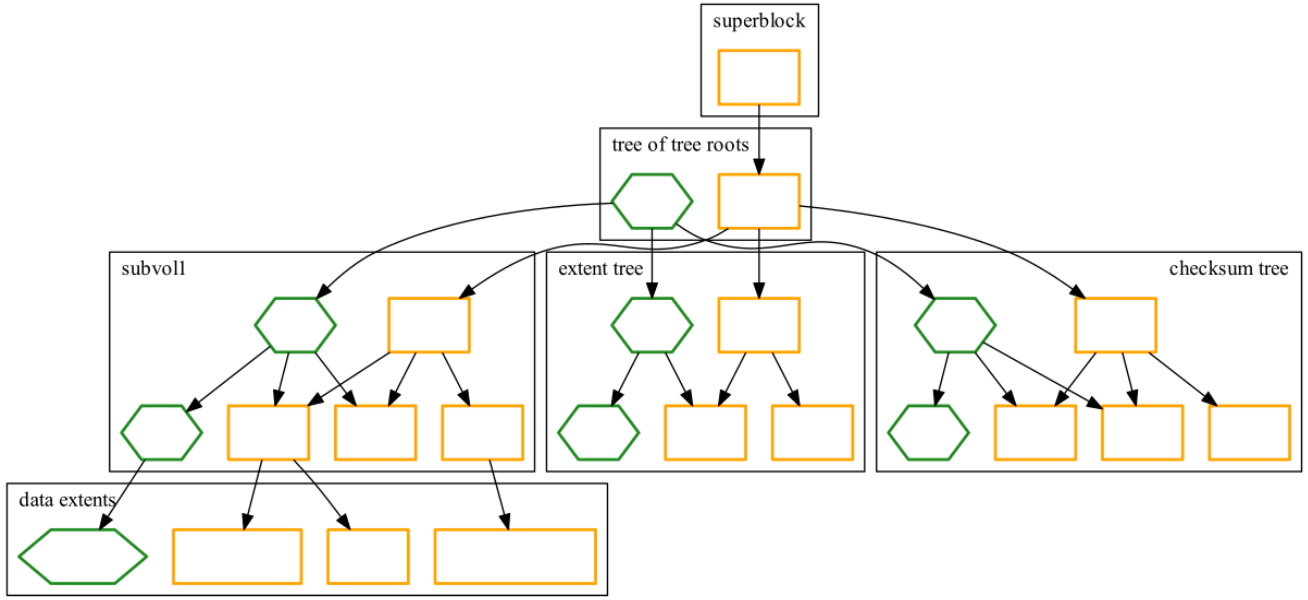


Fig: The changes that occur after modification; modified pages are colored green.

(2/5)..

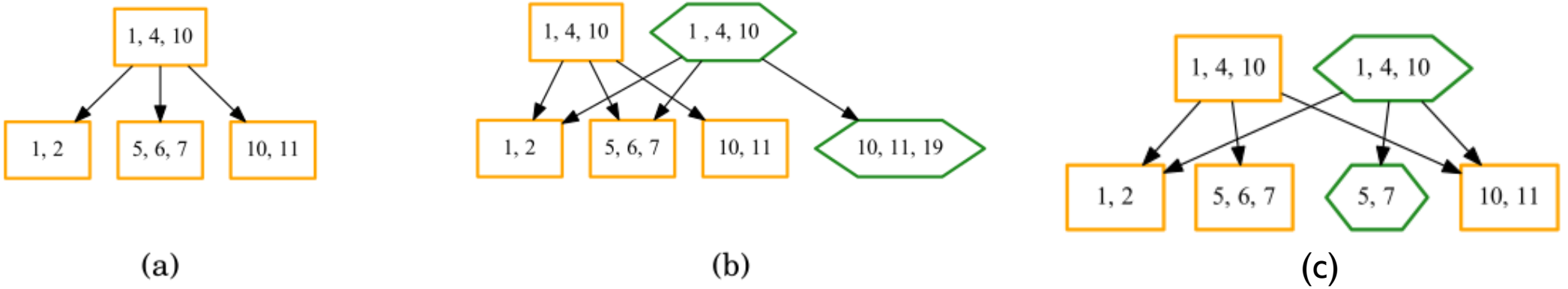


Fig : (a) A basic B-tree (b) Inserting key 19, and creating a path of modified pages (c) Deleting key 6 from original tree.

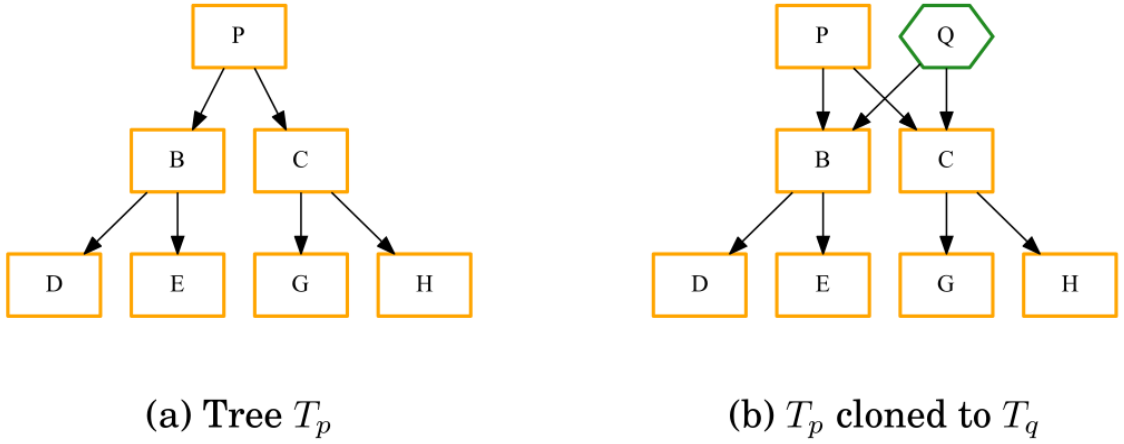
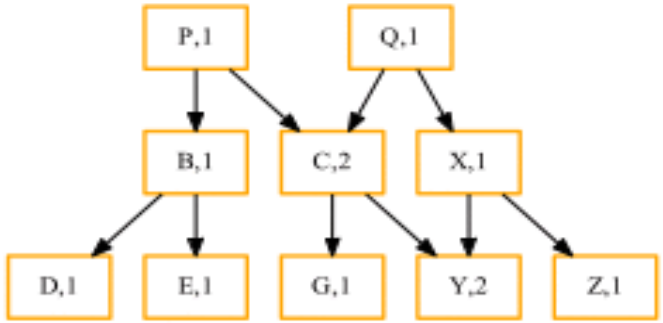
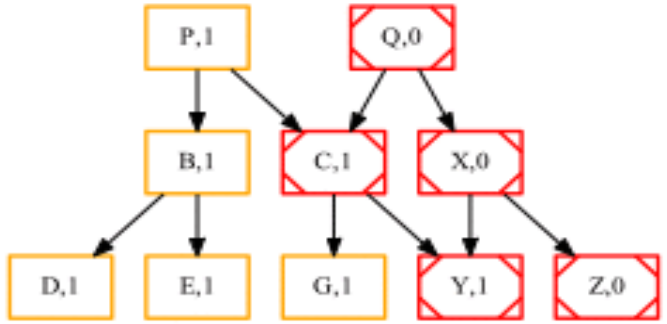


Fig : Cloning tree Tp . A new root Q is created, initially pointing to the same nodes as the original root P. As modifications are applied, the trees will diverge.

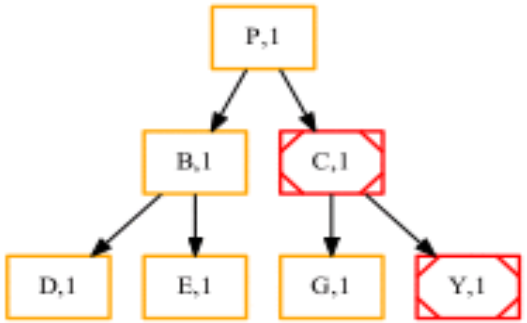
(3/5)..



(a) Initial trees T_p and T_q



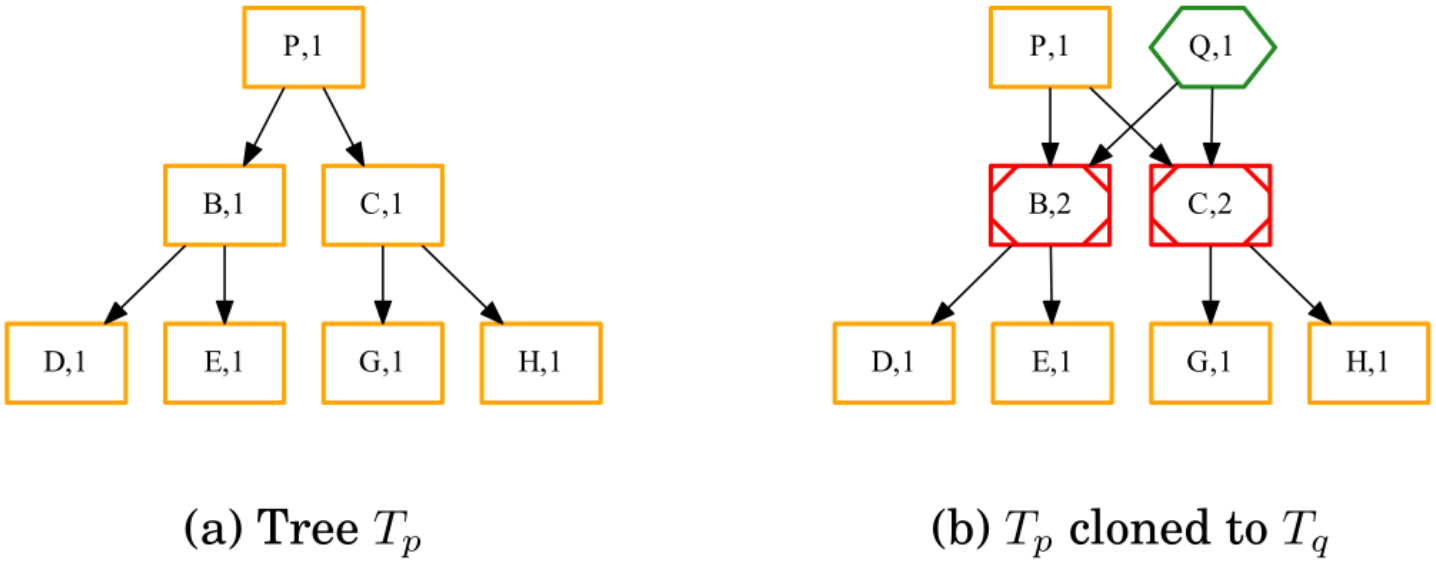
(b) Deleting T_q



(c) Removing unallocated nodes

Fig : Deleting a tree rooted at node Q. Nodes {X, Z}, reachable solely from Q, are deallocated. Nodes {C, Y}, reachable also through P, have their ref-count reduced from 2 to 1.

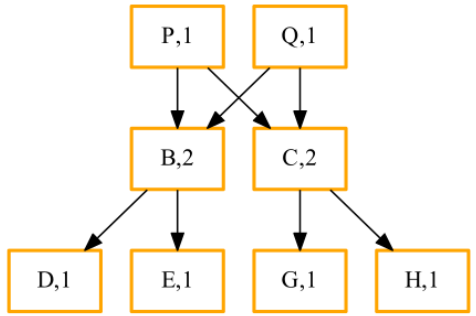
(4/5)..



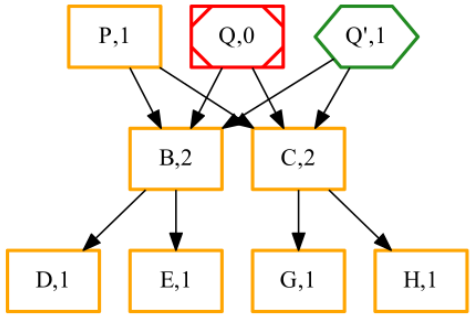
(a) Tree T_p (b) T_p cloned to T_q

Fig : Cloning tree Tp . A new root Q is created, initially pointing to the same nodes as the original root P. The ref-counts for the immediate children are incremented. The grandchildren remain untouched.

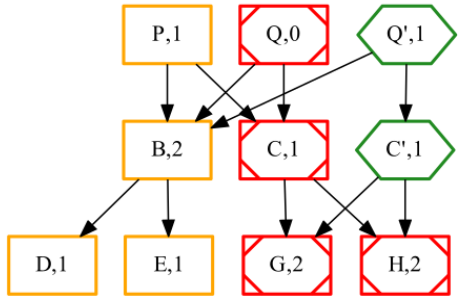
(5/5)..



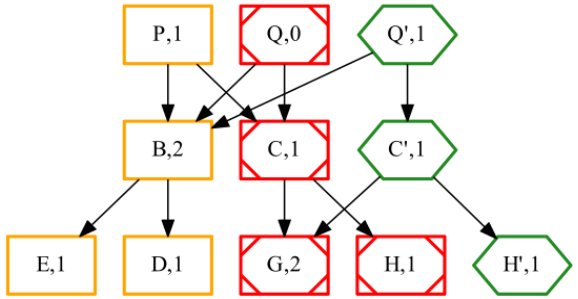
(a) Initial trees, T_p and T_q



(b) Shadow Q



(c) shadow C



(d) shadow H

Fig : Inserting a key into node H of tree T q .The path from Q to H includes nodes {Q, C, H}, these are all CoW-ed. Sharing is broken for nodes C and H; the ref-count for C is decremented.

COW CONTD...

- Deletion Algorithm:

— ref-count(N) > 1. Decrement the ref-count and stop downward traversal. The node is shared with other trees.
— ref-count(N) == 1. It belongs only to q. Continue downward traversal and deallocate N.

COMPRESSION

- Compression is implemented at the extent level.
- Two compression algorithms are supported: ZLIB and LZO 2 .
- Trade-off between compression ratio and speed.
- ZLIB is slower, but provides higher compression ratios. LZO is faster, but provides a worse compression ratio.

DYNAMIC INODE ALLOCATION

- Inode are created only when they are needed.
- Ext series uses static inode tables – only advantage is corruption detection.
- Static inode table wastes space, slows down fsck, it needs inode blocks to be at fixed location.
- In dynamic allocation inode can be kept close to the file.

EXTENT ALLOCATION TREE

- The extent allocation tree holds extent-items, each describing a particular contiguous on-disk area.
- An important operation on the extent-tree is finding all references into a disk area. The extent items are sorted by start address, and there are no overlaps between items.
- The back references also serve as the ref-count; when it reaches zero, the extent can be freed.

EXTENT BLOCK GROUPS

- The disk up into chunks of 256MB or more.
- For each chunk, information about the number of blocks available is recorded.
- Flags indicate whether the extent is for data or metadata.
- At creation time disk id broken up into 33% for metadata and 66% for data alternatively.

DATA & METADATA STRUCTURE

- *Advantages of B-Tree based structure :*
  - Increase in overall increase of tree height
  - Less rearrangement
  - Block based access – Entire tree need not to be in memory
  - Efficient search in O(nlogn)
  - CoW speeds up operation

BTRFS DATA STRUCTURE

- Composed of three basic structures
  - Block Header – btrfs_header
  - Key – btrfs_key
  - Item – btrfs_item

BTRFS_HEADER

- Checksum for the block contents
- uuid of the file system
- Level of the block in the tree
- Block number where the block resides

```
struct btrfs_header
{
   u8 csum[32 bytes];

   u8 fsid[16];

   __le64 blocknr;

   __le64 owner;

   __le16 nritems;

   __le16 flags;

   u8 level;
}
```

BTRFS_KEY

- Object ID – Analogous to the inode. It defines the unique ID for each node.
  Object id is Most Significant Bits of key – results in grouping together all info associated with particular object ID.
- Type – Specifies the type of the item e.g. inode, file data etc.
- Offset – Indicates the byte offset for a particular item. Inode has offset value 0.

```
struct btrfs_key
{

   u64 objectid;

   u32 type;

   u64 offset ;

   8 level;

}
```

BTRFS_ITEM

- Key – describes the item
- Offset – it is the offset of item
- Size – size of item

```
struct btrfs_item
{

   struct btrfs_key key ;

   __le32 offset ;

   __le16 size ;

}
```

BTRFS NODES

Non Leaf Nodes

- Contains a pair of key and block headers
- Key specifies the location of the item being searched
- Header contains the information about where next node or leaf in the B-Tree is located on disk - given by "blocknr" field of btrfs_header structure

Leaf Nodes

- Leaf nodes are broken up into two sections, item header and data, that grow towards each other.
- Data is variably sized.
- Offset and size filed gives the location of data associated with an item

BTRFS LEAF NODE STRUCTURE

Fig: A leaf node with three items. The items are fixed size, but the data elements are variably sized.

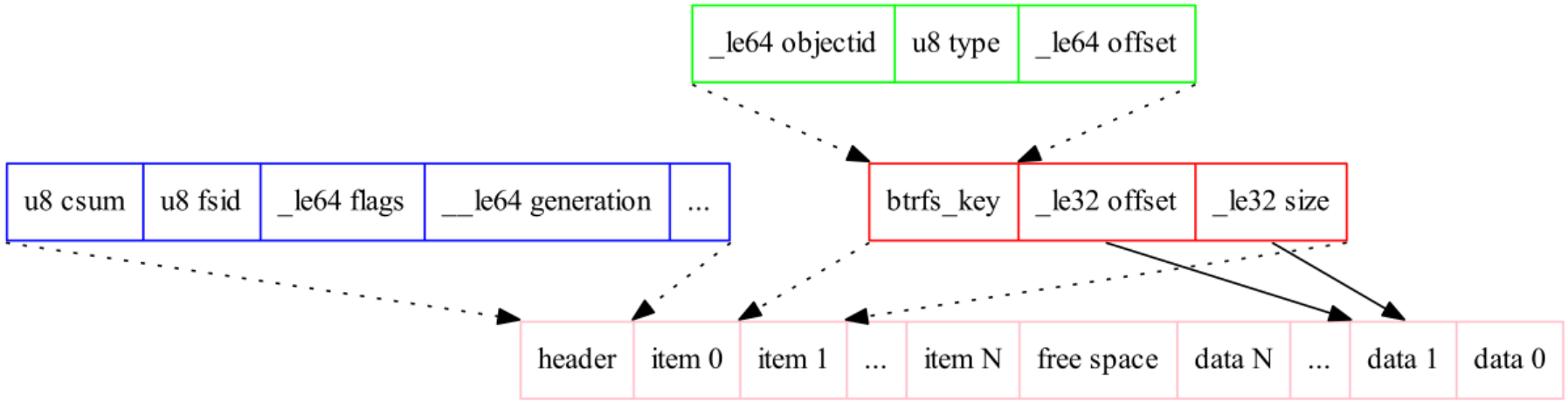


Fig: A detailed look at a generic leaf node holding keys and items

DIRECTORY INDEXING

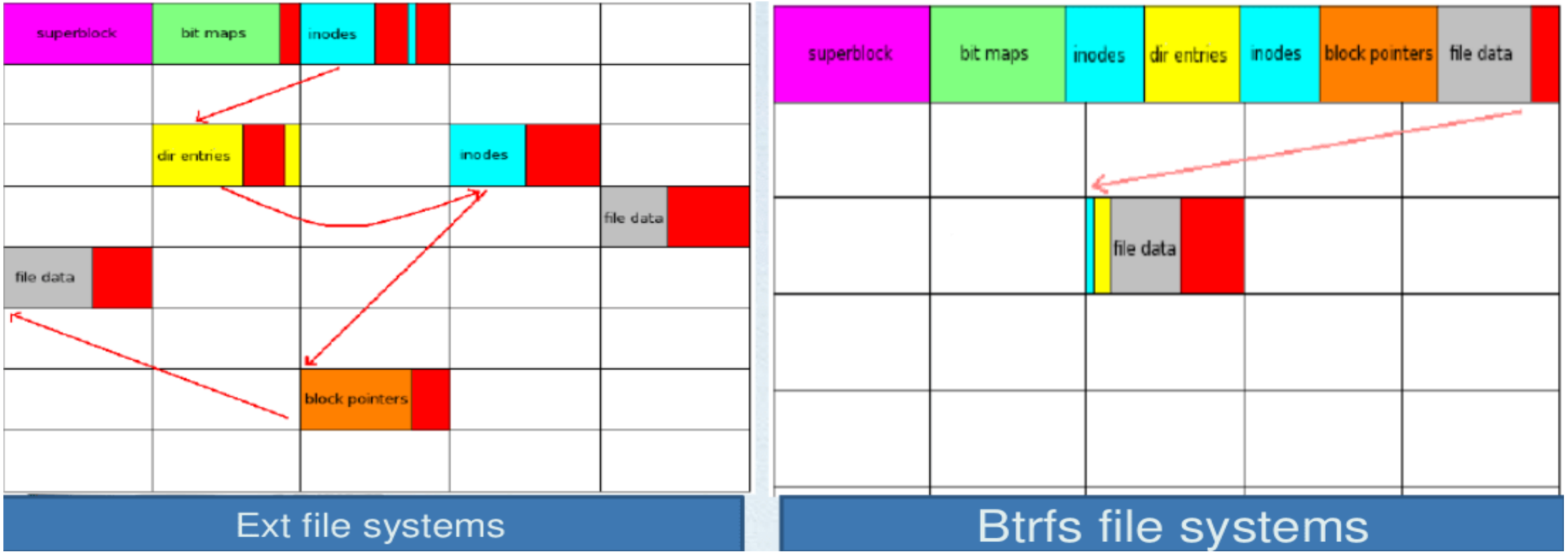
- Directories are indexed in the following way:

Using hash keys for filename lookup and directory item's key is composed as follows

FILE STORAGE

- Inline extents are used for small files.
- Large files are stored in extents and are (disk block, disk num blocks) pairs which records the area of disk for the file.
- Extents store logical offset- allows write in the middle of extent.

BTRFS ORGANIZATION OF FILES, INODES, DIRECTORIES, POINTERS, ETC.



FUTURE SCOPE

- There is a possibility of fragmentation problem in this project as we are considering contiguous memory allocation.
- This issue can be circumvented by implementing paging or segmentation in the future.
- Data structures for page table, segment table can be implemented for efficient memory allocation.

Thank You

https://github.com/schowdhury671/BTRFS-project