STORAGE SYSTEMS PROJECT

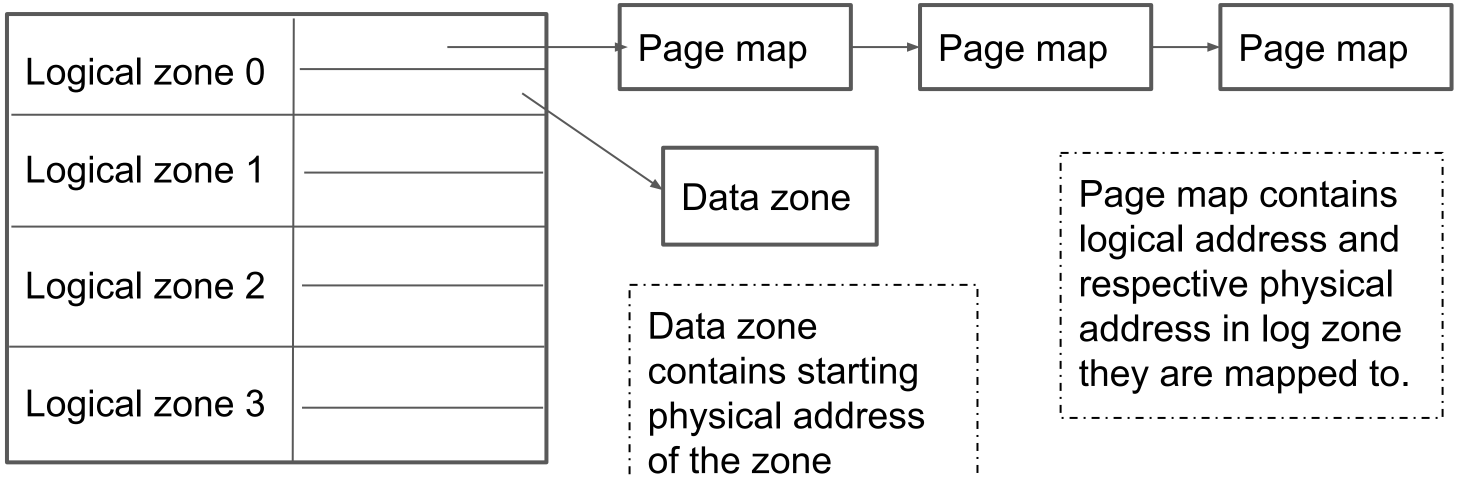
GROUP 19

*Stosys Team
Choosing us for presentation
*Us
VSFS
Partial working M5
imgflip.com

FTL DESIGN

Writes append to respective logical block page map

Reads looks up respective logical block page map initially. If not found, then retrieves data from mapped data zone

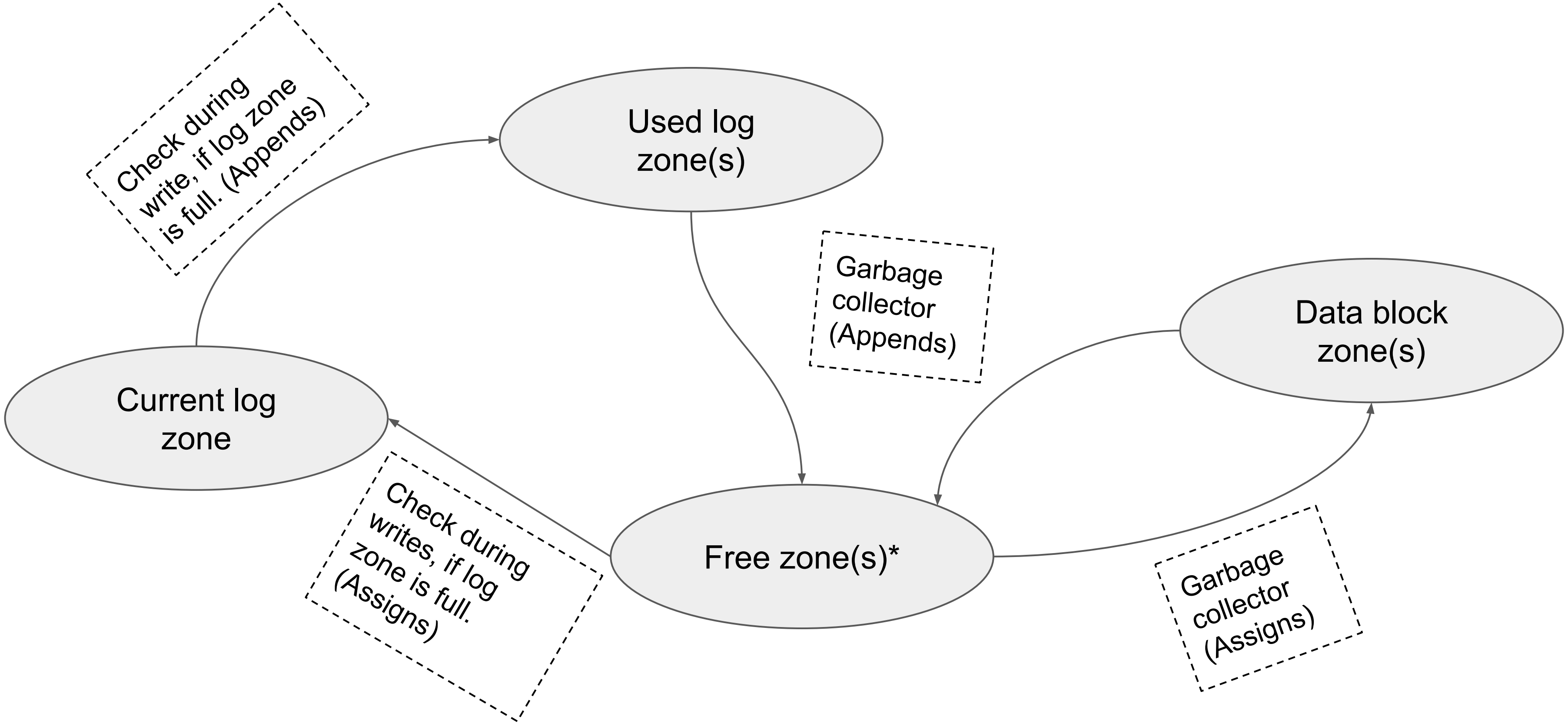


Data zone contains starting physical address of the zone

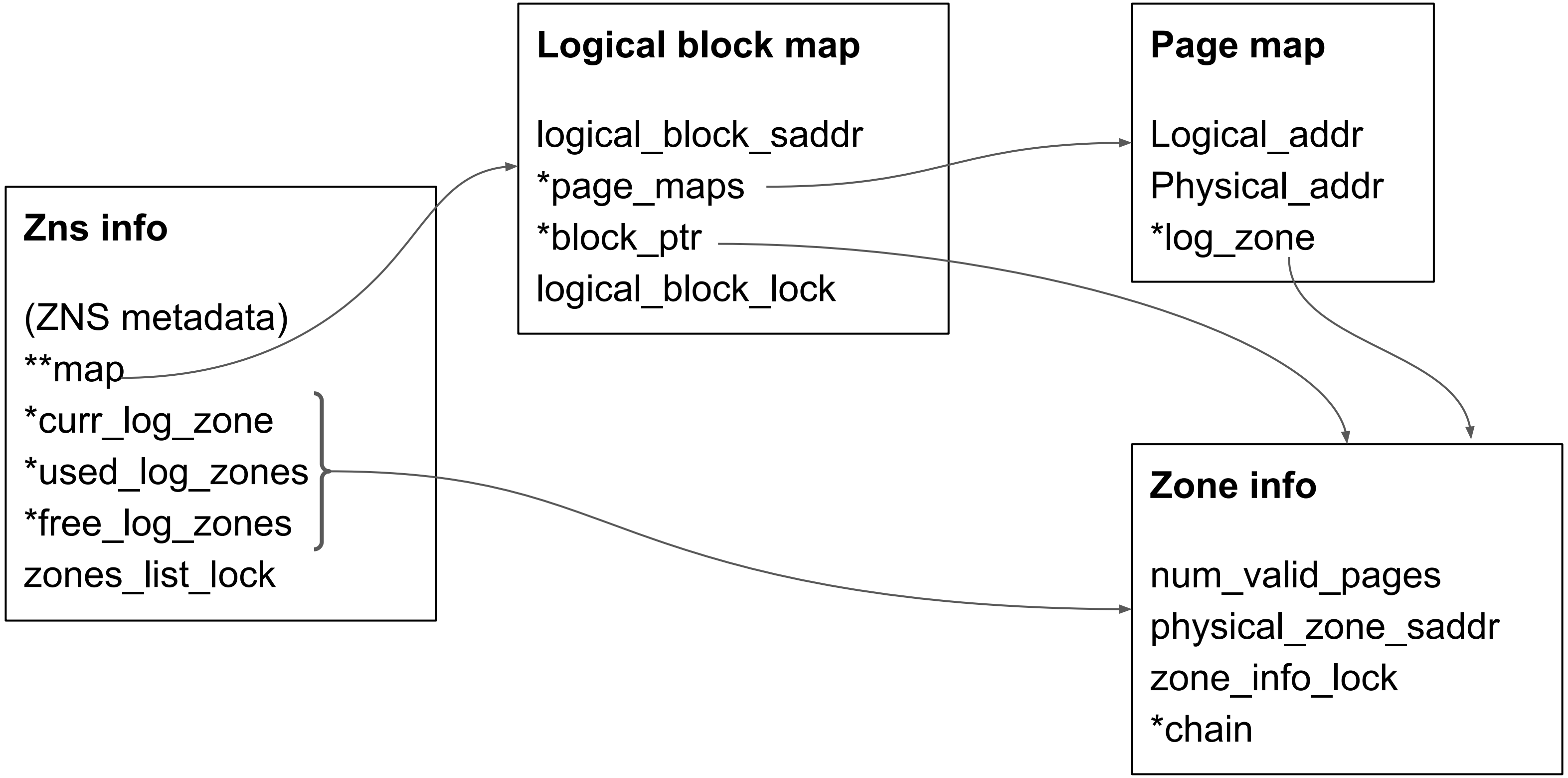
Page map contains logical address and respective physical address in log zone they are mapped to.

GC iterates over each logical block and merge their Page map and existing data zone to new data zone.

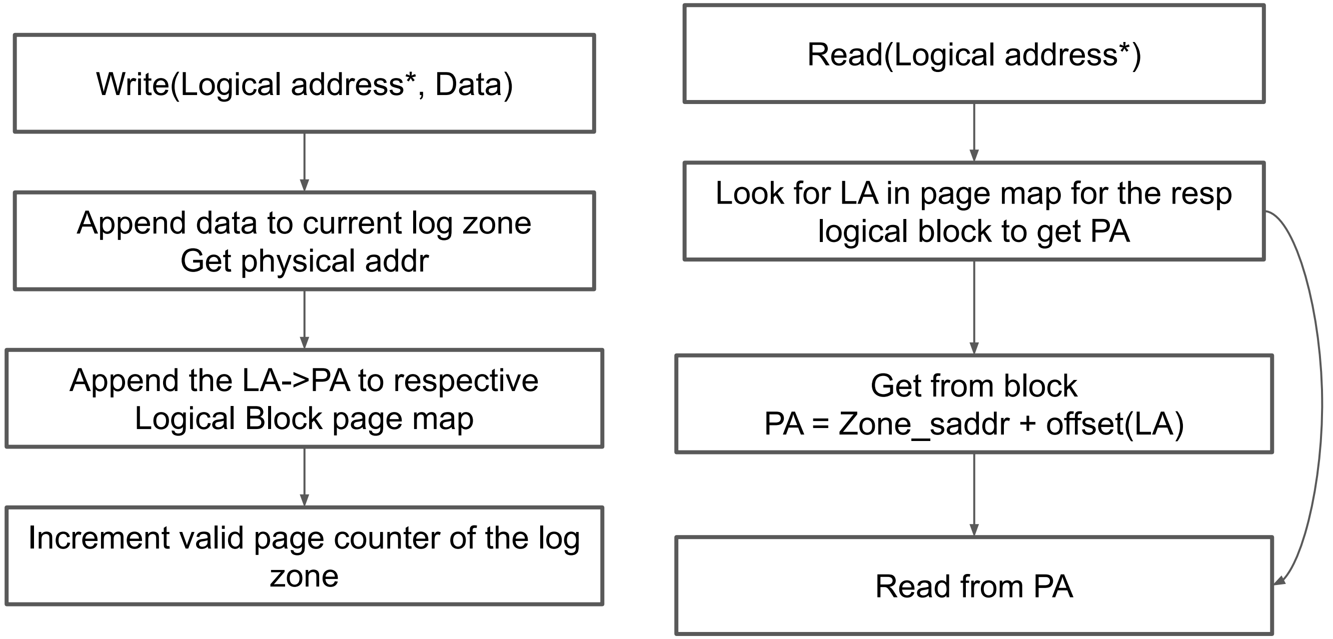
Zone State


Check during write, if log zone is full. (Appends)
Used log zone(s)
Garbage collector (Appends)
Data block zone(s)
Current log zone
Check during writes, if log zone is full. (Assigns)
Free zone(s)*
Garbage collector (Assigns)

Data structures



Functionalities

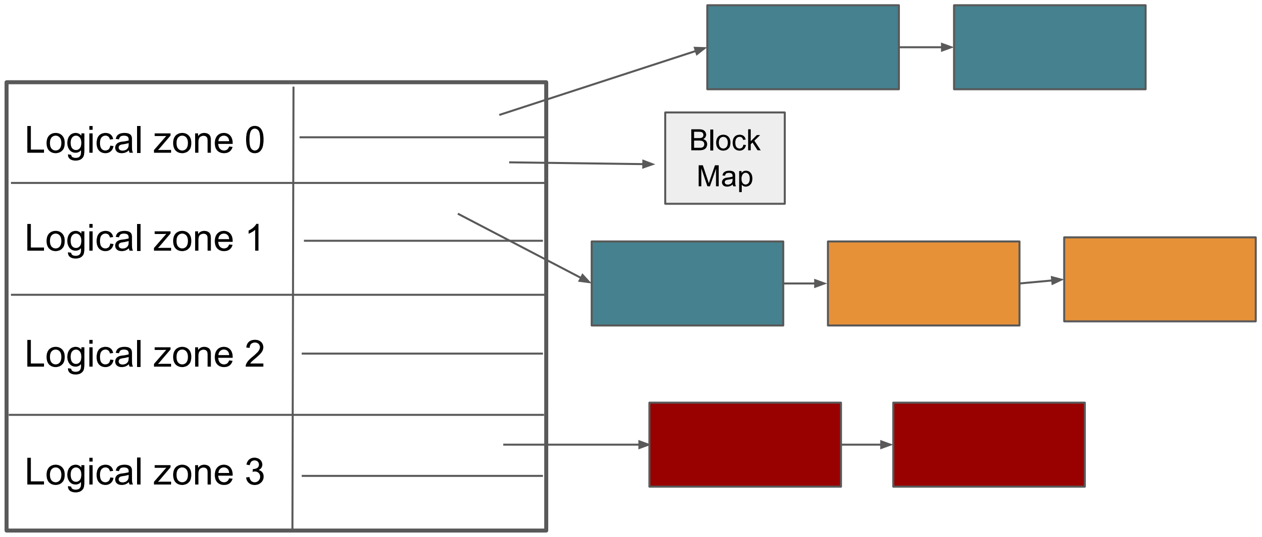

Write(Logical address*, Data)
Append data to current log zone
Get physical addr
Append the LA->PA to respective Logical Block page map
Increment valid page counter of the log zone
Read(Logical address*)
Look for LA in page map for the resp logical block to get PA
Get from block
PA = Zone_saddr + offset(LA)
Read from PA

Logical addr = LA_passed/LBA size

Logical block = Logical addr / NPZ

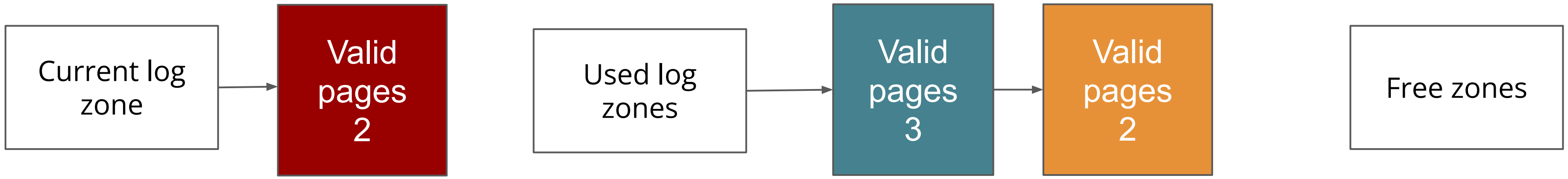
Offset = Logical addr % NPZ

Garbage collector working



Garbage Collector

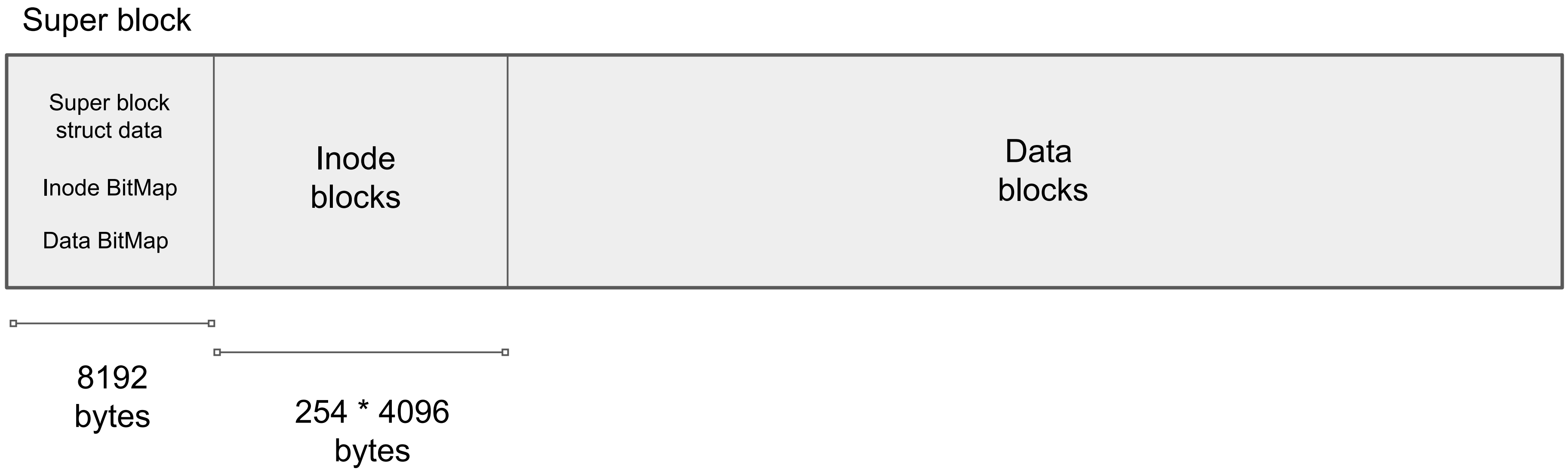
```
For {
 if(lzone->log != NULL)
  lock(logical_zone#);
    gc_log = lzone->log;
    lzone->log = NULL;
  unlock(logical_zone#);
  nz = get_free_zone();
  Merge(gc_log, lzone->block, nz);
  Check_used_log_zones_to_free();

  #Change lzone to next logical block
}
```



FTL highlights

- Support to do any multiples of page size write
- Direct data zone append
- BitMaps to validate read is done after write

FILE SYSTEM DESIGN



*We followed very simple file system method with inplace updates.

Structures

```
Super Block {
    bool Persistency;
    uint32 InodePtr;
    uint32 DataPtr;
}
```

```
Inode {
    uint32 Inode_no;
    char entityName[235];
    bool IsDir;
    uint64 FileSize;
    uint64 Indirect_lba;
    uint64 Direct_lbas[480];
}
```

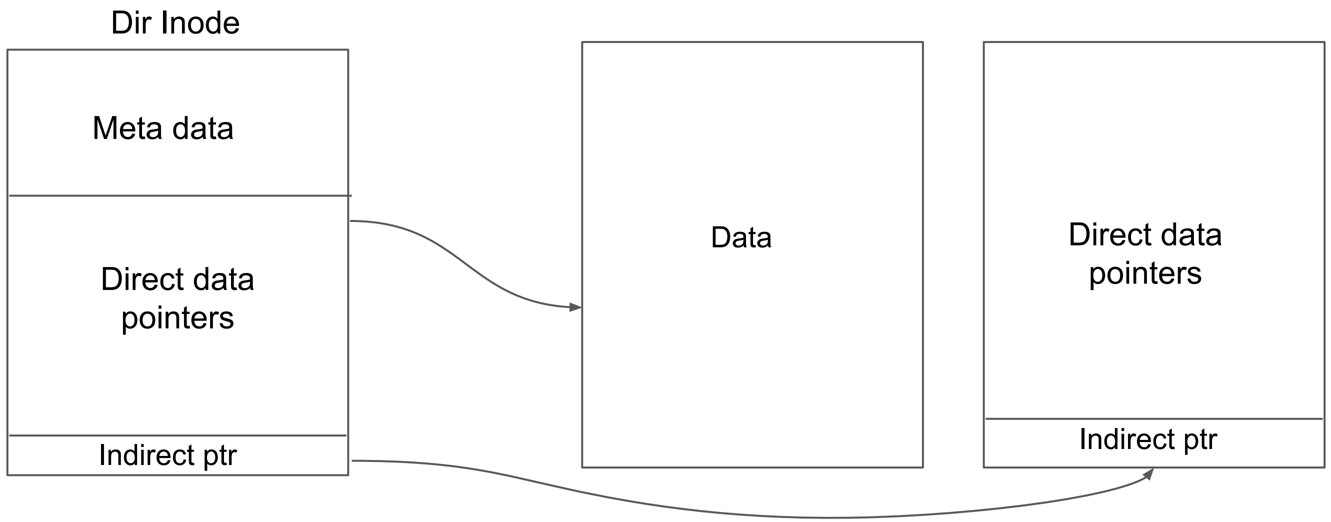
```
Indirect_ptr {
    uint64 addr;
    uint64 Indirect_lba;
    uint64 Direct_lbas[510];
}
```

```
MYFS_Dir {
    MYFS_DirData Entities[16];
}
```

```
MYFS_DirData{
    char EntityName[252];
    uint32 inode;
}
```

*Data structures are defined to be aligned with page size

In device


Dir Inode
Meta data
Direct data pointers
Indirect ptr
Data
Direct data pointers
Indirect ptr

Main functions

Get_Path_Inode(FilePath, inode)

```
//If in mem
Look_Mem(FilePath, inode);

//If not
Get_Path_Inode(ParentPath, P_Inode)
Child_Inode# = Load_Children(P_Inode, Child_Name)
Addr = Child_Inode#*4096+SBlockOff
Read_From_NVM(Addr, inode);
```

Update Parent

CreateFile

CreateDirectory

RenameFile

DeleteFile

Persistency

FTL

Clear all data from log zone to data zone

Retrieve zone 0 to store block mapping

At start, check if mapping data is present in zone 0. If so, retrieve.

FILE SYSTEM

Flush all Inode data to NVM

Store Super Block with persistency byte set

to true along with Bit maps

At start, Read Logical addr 0 retrieve super block and check if persistency set to true. If so, load bitmaps and root Inode.

If Time

- Make fully functional M5
- Log based File System
- More than one thread GC
- Observe how various parameters and intricate design changes affect performance.

Takeaways

- Never take Storage system and Software security together
- Storage system terminologies are brain teasers
- Extensive use of GDB for debugging
- Confidence to implement complex system
- Jump start to masters degree