Systems Software &
Architecture Lab.
Seoul National University

2025.12.5

Project #5:
xFFS: Faster File System for xv6

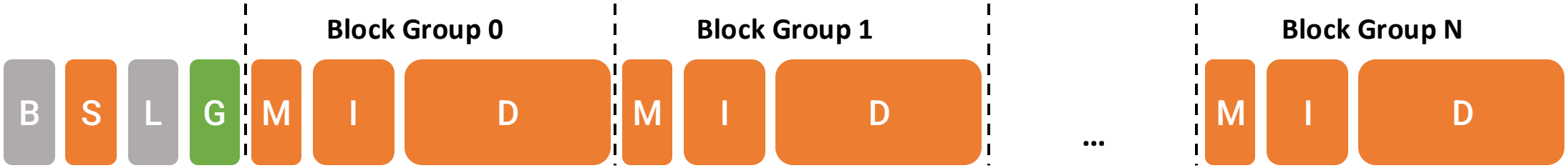
xv6 file system

- **Boot: not used (1 block)**
- **Super block: basic information of the file system (1 block)**
- **Log: write-ahead logging for crash consistency (LOGBLOCKS blocks)**
- **Inode: save Inodes** (NINODES blocks)
- **data bitMap: bitmap to track data block usage** (size derived from FSSIZE)
- **Data blocks**

Problems with xv6 file system

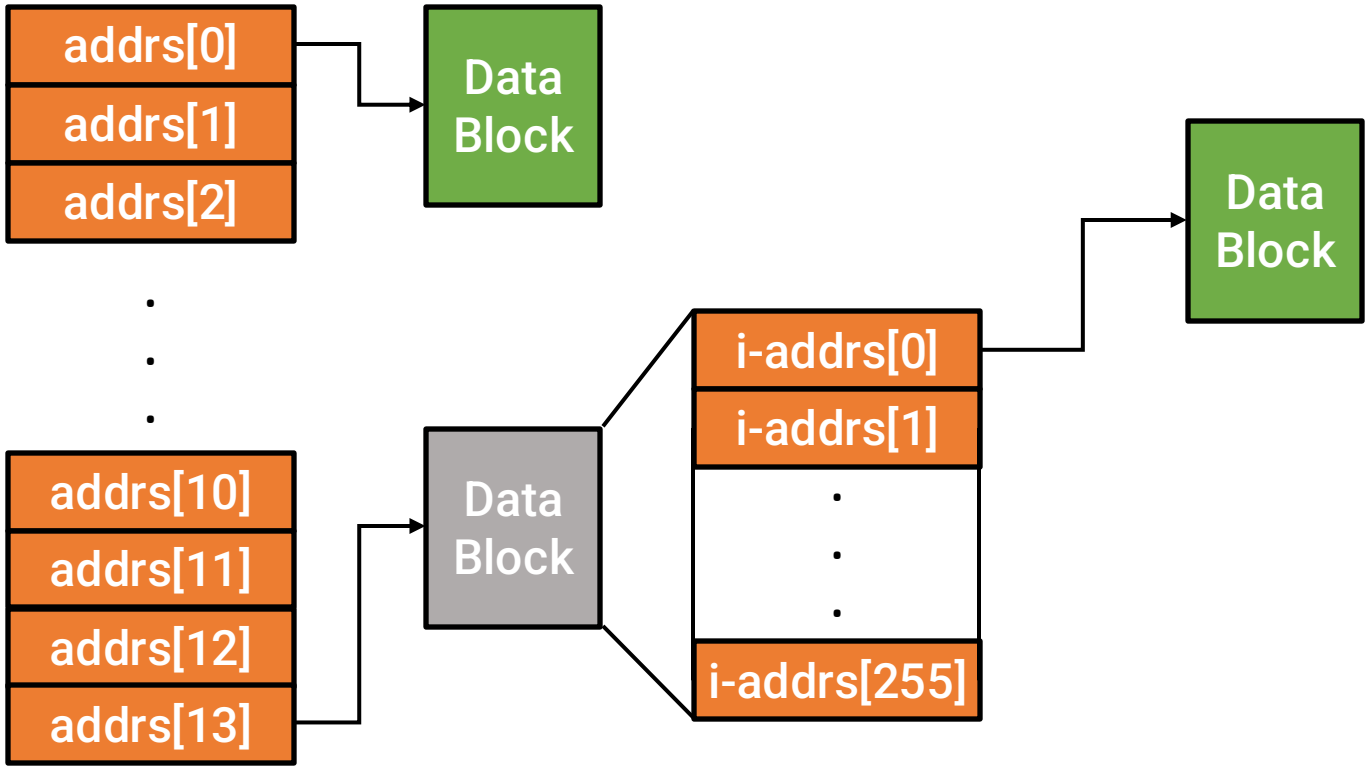
- As the filesystem ages, free blocks become scattered, causing newly written file data to be placed randomly across the disk.
- Inodes are stored far from the data blocks they reference, forcing long disk seeks whenever files or directories are accessed.
- Files inside the same directory do not receive adjacent inode numbers, leading to non-sequential inode accesses during directory traversal.

xFFS: Faster File System for xv6



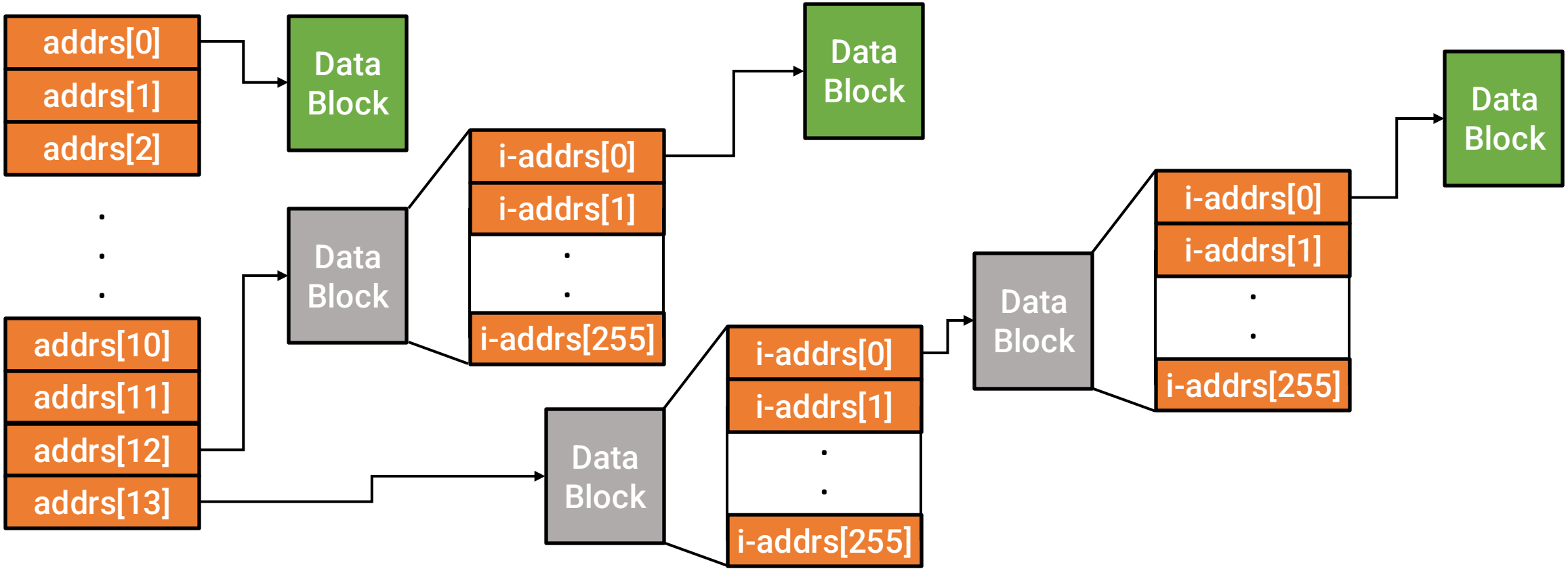
- **Global Description Table:** Saves per-group statistics **(1 block)**
  - Fast allocation decisions (read one block to see all groups)
- **Each block group has its own data bitmap block and inode blocks**
- **Parameters per block group**
  - Data bitmap block: 1
  - Inode blocks: 8
  - Data blocks: 2,039

Original inode



12 + 256 data blocks

Extended inode



11 + 256 + 256 * 256 data blocks

Part 1. Extend mkfs for xFFS (30 points)

- **Build xFFS on-disk layout with block groups**
  - Modify mkfs to write the xFFS header (B,S,L,G), divide the disk into fixed-size block groups, and initialize all per-group metadata (bitmaps, inode areas).

- **Implement large-file support using double-indirect blocks**
  - Support files larger than what direct + single-indirect pointers can address by allocating and linking double-indirect structures and their referenced data blocks.

- **Validate filesystem consistency and correctness**
  - Ensure all regions are non-overlapping, metadata counts match the declared geometry, and the final disk image boots and mounts correctly with xFFS enabled.

Part 2. Extend the xv6 to mount and operate on xFFS (30 points)

- **Enable xv6 to mount and interpret xFFS structures**
  - Modify the kernel to read the xFFS superblock, recognize the magic number, parse the GDT, and use block-group geometry when allocating inodes/blocks and updating bitmaps.

- **Support large-file access with double-indirect addressing**
  - Implement double-indirect block handling so the kernel can read and write files larger than the direct + single-indirect limit.

- **Retain xv6 semantics while using xFFS allocation rules**
  - Keep the original system calls and directory behaviors, but apply xFFS's on-disk layout and use a simple first-fit allocator over the entire file system.

Part 3. Implement a block-group-aware allocation policy (30 points)

- **Design a locality-aware allocator using block-group statistics**
  - Keep directories and their files within the same block group by using GDT information (free blocks, free inodes, etc). Choose groups that minimize seek distance and preserve locality.

- **Allocation heuristics inspired by FFS**
  - New directories are placed in groups with few existing directories and many free inodes; allocate file inodes and data blocks in the same block group as their parent directory.

- **Implement sync() to persist GDT updates**
  - Maintain per-group summaries in memory and write them back to disk via a new sync() syscall for durability.

Part 3. Grading the implemented policy

- **Seek distance is measured by block “tracks”**
  - Blocks are grouped into tracks of **32** blocks. Every disk I/O adds the absolute difference between the current block’s track and the previously accessed track.

- **Your allocator must meaningfully reduce head movement**
  - Workloads include sequential, random, and metadata-heavy access patterns. A good policy should cut total seek distance by ≥ 20% compared to global first-fit.

- **Caching is disabled before measurement**
  - The provided bdrop() syscall clears cached blocks (refcnt == 0), ensuring evaluations reflect true on-disk access patterns, not buffer-cache hits.

Part 3 & bonus workload characteristics

- **Inode–data block locality is tested**
  - The allocator is evaluated on how tightly it places a file's data blocks near its inode, reducing seek distance during file accesses.

- **Directory inode–file inode locality impacts performance**
  - Workloads measure whether files/directories created within the same directory are closely allocated, minimizing cross-group traversal.

- **Workload includes high metadata activity and mixed access patterns**
  - The test suite heavily exercises mkdir, unlink, writes, and a combination of random and sequential reads—favoring allocators that maintain strong locality under intensive directory operations.

Notifications

- **Do not modify the order of fields in the superblock**
  - If additional metadata is required, append new fields rather than reordering existing ones.
- **You may optimize mkfs for better initial placement**
  - Improvements to directory/file placement are allowed, but you must not create any files or directories beyond those explicitly provided as input to mkfs.
- **Modify only the files permitted by the assignment**
  - Changes should be limited to: fs.c, fs.h, defs.h, mkfs.c, sysfile.c
- **Grading runs take time (~10 minutes each)**
  - Because the queue gets heavily congested near the deadline, begin your assignment early to avoid long delays.

Due date

- Due
  - 11:59 PM, December 21 (Sunday)

- Submission
  - Run the make submit command to generate a tarball named xv6-pa5-{STUDENTID}.tar.gz in the xv6-riscv-snu directory
  - Upload the compressed file to the submission server
  - The total number of submissions for this project will be limited to 30
  - Only the version marked FINAL will be considered for the project score

Thank you!