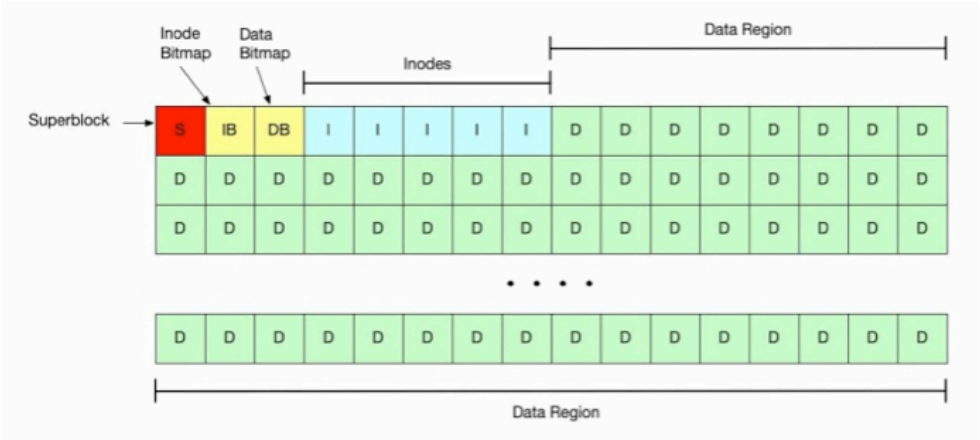
CSC369 A1a Proposal

I. A simple diagram of the layout of your disk image



ps. Since there's no limit on the number of inodes and files, the size of the disk, etc, it's **possible** to assign **more than** 1 blocks for the Inode Bitmap or Data Bitmap (instead of what's dictated by the diagram above), the number of blocks assigned for both bitmaps are determined by the parameters to mkfs

II. A description of how you are partitioning space

- Since the size of the disk image and the number of inodes are parameters to mkfs, number of blocks assigned for the whole data region is disk image size / block size(4096 B) (result +1 if disk image size % block size isn't 0).
- The 1st block is the Superblock which stores the metadata of the file system which includes information such as free inodes count, free blocks count, etc.
- Starting from the 2nd block is the block(s) for Inode Bitmap which is an array of bits — bit 0 stands for the inode at corresponding index is free while bit 1 means it is in use. The number of blocks allocated for inode bitmap is number of inodes / block size(4096 B) * 8 bits per byte (result +1 if number of inodes % block size(4096 B) * 8 isn't 0).
- Following the Inode Bitmap is the block(s) for Data Bitmap which is also an array of bits — bit 0 stands for the data block at corresponding index is free while bit 1 means it is in use. The number of blocks allocated for data bitmap can be calculated by the relation — 1 + number of blocks for inode bitmap + number of blocks for data bitmap (number of data blocks / block size(4096 B) * 8 bits per byte (result +1 if number of data blocks % block size(4096 B) * 8 isn't 0)) + number of blocks for inode table + x = number of blocks for disk (x denotes the number of data blocks).
- Following the Data Bitmap is the block(s) for Inode Table which is an array of inodes. Inode contains metadata of each entity (file or directory). The number of blocks allocated for Inode table is number of inodes * size of an inode / block size(4096 B) (result +1 if number of inodes * size of an inode % block size(4096 B) isn't 0).
- The rest of the blocks are allocated for data blocks.

III. A description of how you will store the information about the extents that belong to a file

- Inode consists of metadata and a pointer to the data block that contains extents that belong to a file, that means an inode has information such as the block number for data block that contains the extents.

IV. Describe how to allocate disk blocks to a file when it is extended

Since in a1fs, a file can only be extended or truncated from the end, the steps involved to allocate disk blocks to extend a file are:

1. Compute the number of blocks required by: buffer size / block size (result+1 if buffer size % block size isn't 0)
2. Compute the starting data block by: data block number of the last data block for the file + 1
3. Look through Data Bitmap to see if the extent(calculated from above steps) is free or not
4. If not, scan through data bitmap again (from last data block of the file to the end of bitmap, then from the beginning of the bitmap to the last data block of the file) and find the first available extent with such number of contiguous blocks
5. If there isn't such extent available, extent with the closest number of continuous blocks than the required number of blocks is preferred. Break up large extent if all smaller ones are made use of and still need more extents for the file extension — look for extents based on this principle until enough extents are found
6. Get the information about which data block contain the extents for the file from the inode of the file. Add found extents to the exiting array of extents in the data block and check if the inode has reached the max number of extents allowed
7. Write to the newly found data blocks and mark them as in use in data bitmap
8. Update inode fields

V. Explain how your allocation algorithm will help keep fragmentation low

The above algorithm tries to extend the last extent of a file (Step 3) or allocate a single extent (Step 4) if possible, which keeps file fragmentation low.
It also tries to use up available extents that have less contiguous blocks and try to avoid breaking up large extents (Step 5) to keep external fragmentation low.

VI. Describe how to free disk blocks when a file is truncated or deleted

- The new file size in bytes is parameter to the a1fs truncate function. Compute the new block size of the file by : new file size in bytes / block size (result +1 if new file size in bytes % block size isn't 0)
- Scan through the array of extents in the extents data block pointed to by the inode, blocks that will be kept are the ones made up of the new block size of the file
- Free the memory space for blocks that are no longer needed and mark them as free in data bitmap. Update the number of contiguous blocks in the extent.

VII. Describe the algorithm to seek to a specific byte in a file

1. Compute block number and offset in block for the target byte offset.
    1. blocknum = offset / blocksize
    2. blockoff = offset % blocksize
2. Scan through the array of extents in the extents data block pointed to by the inode and find the extent that contain i_block[blocknum-1]. Read all the data blocks referred to by the extents up and including i_block[blocknum-1]
3. Find the next extent and its starting block if blockoff isn't 0. Use i_block[blocknum] to read the data block and use blockoff to seek to the byte.
4. If offset is larger than the maximum file size return an error.

VIII. Describe how to allocate and free inodes

- To allocate an inode, assign its memory space on the stack using the malloc() function call. Populate the fields for its metadata. Mark the inode as in use in inode bitmap.
- To free an inode, free its memory space by the free() function call. Mark the inode as free in inode bitmap.

IX. Describe how to allocate and free directory entries within the data blocks that represent the directory

- To allocate a directory entry:
  - Find the inode for its parent directory by traversing the path.
  - Find the last data block in the last extent from the array of extents in the extents data block pointed to by the inode. Allocate an inode for the entry. Append the entry to the list of directory entries in the data block.
  - If the block is full before appending the list, find an available data block using the algorithm described in IV., initialize a list in the new block and append the entry to list.
  - Update the inode for the parent directory if new data block is used and its metadata.
- To free a directory entry:
  - Find the inode for the entry and the data block for its parent directory that contains the entry by traveling the path.
  - Free the inode and data blocks for the entry as described in VI. and VIII.
  - In the data block for the parent directory, remove the entry from the list of entries and put the last entry in the list onto that position, which makes listing all entries in a directory easier because there is no gap in the list.
  - Update the inode for the parent directory to reflect the data blocks changes and update its metadata.
- Reading all entries in all data blocks pointed to by the inode for the parent directory indicates all valid entires in the directory have been read.

X. Describe how to lookup a file given a full path to it, starting from the root directory

1. Use C string operations to slit a path into its components
2. The 1st node corresponds to the root directory. Look through its data blocks referred to by the array of extents in the extents data block pointed to by the inode and find the next entry by identifying its name in the lists of entries of data blocks for root directory.
3. For each entry in the path:
   - Its inode number is indicated by the directory entry. File type is one of the metadata information included in inode. If the entry is a directory, use the algorithm in Step2 to find the next entry in the path.
   - If the entry is a file, we've reached the desired file. Look through and read its data blocks referred to by the array of extents in the extents data block pointed to by the inode.