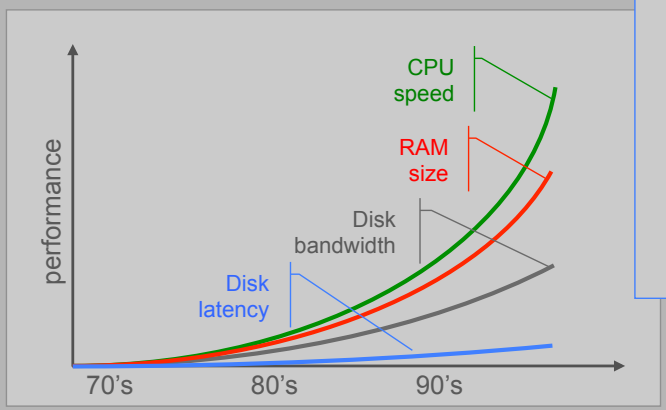
Log-Structured File Systems

- Memory has become cheap!
- This leads to large caches, and large write buffers.
- Problems with UNIX FFS
- Log-structured File Systems: General Ideas
- Write Buffering and Sequential Writing
- Some Practical Issues

Log-Structured File Systems

Observations (Early 90's):

Technology progress is uneven.



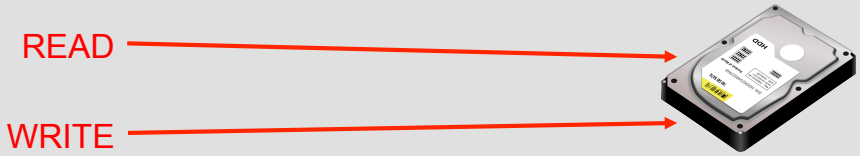
Processors:
- Speed increases exponentially.

RAM:
- Size increases exponentially.

Disk Technology:
- Transfer bandwidth: can significantly increase with RAID
- Latency: no major improvement

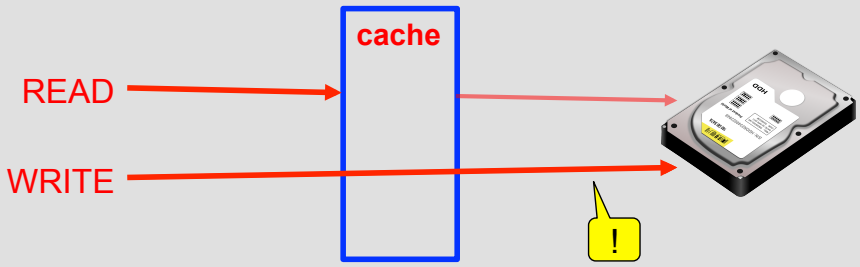
More RAM leads to …

Large File Caches:



More RAM leads to …

Large File Caches:



– **Caches** handle large portions of READ requests.

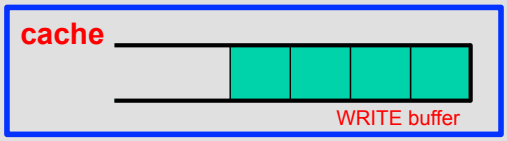
– Therefore, WRITE requests will dominate disk traffic.

More RAM leads to …

Large Write Buffers:

- Buffer large number of WRITE requests before writing to disk.
- This increases efficiency of individual WRITE operation (sequential transfer rather than random).

Disadvantage: Data loss during system crash.



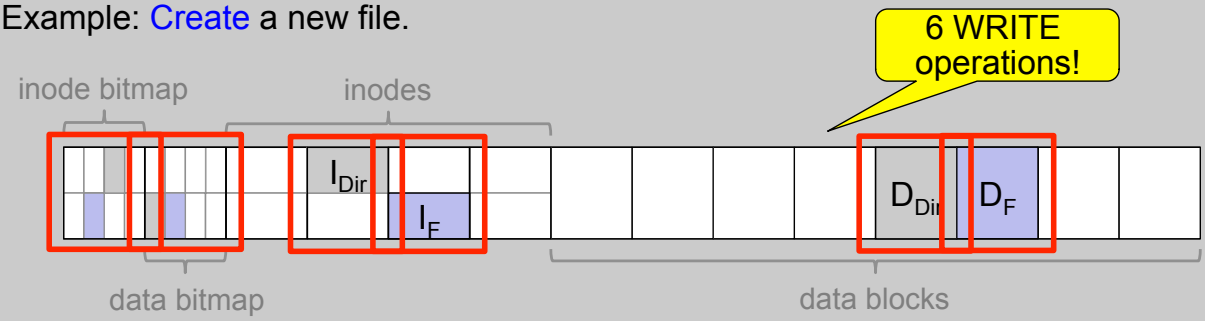
Problems with Berkeley Unix FFS …

PROBLEM 1:

FFS attempts to lay out file data sequentially, **but**

- Files are physically separated.
- inodes are separate from file content.
- Directory entries are separate from file content.

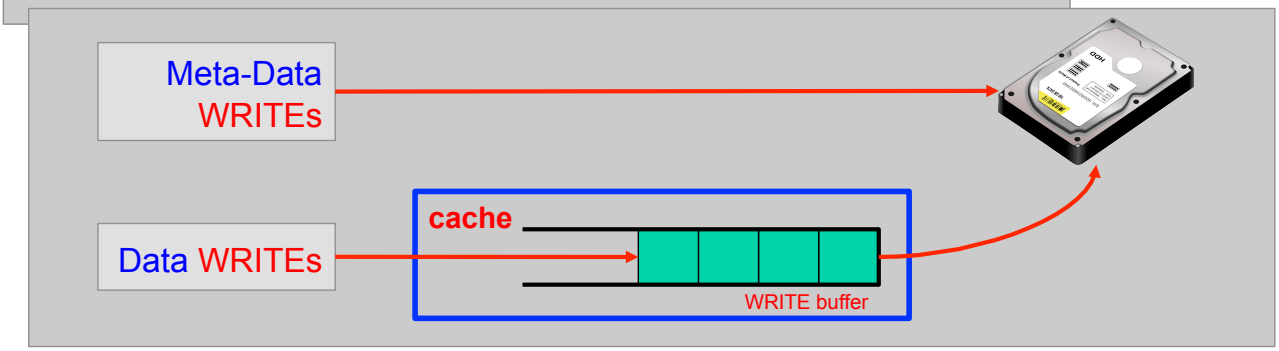
Example: Create a new file.



Problems with Berkeley Unix FFS …

PROBLEM 2:
FFS's WRITE operations are synchronized.

- File data is written asynchronously
- Meta-data (inodes, directories, bitmaps) written synchronously



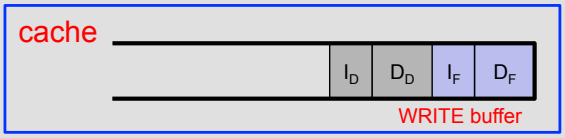
Log-Structured File Systems

Fundamental idea: Focus on Write performance!

- Buffer file system changes in file cache.
  - File data, directories, inodes, …
- Write changes to disk sequentially.
  - Aggregate small random writes into large asynchronous sequential writes.

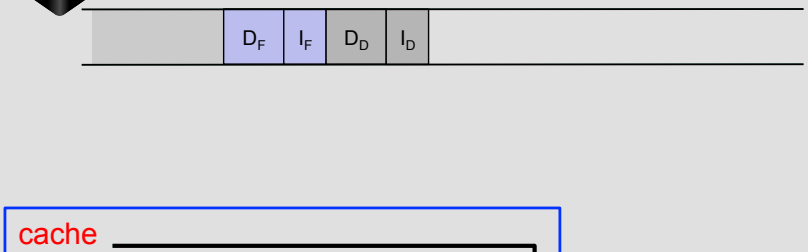
How to Write Sequentially

Write Buffering and Sequential Writes:



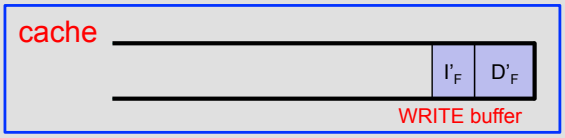
How to Write Sequentially

Write Buffering and Sequential Writes:



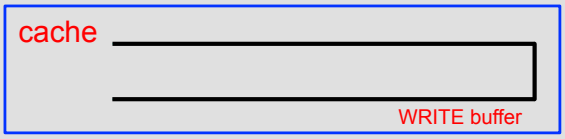
How to Write Sequentially

Write Buffering and Sequential Writes:



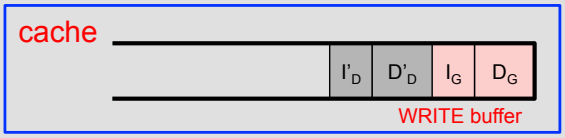
How to Write Sequentially

Write Buffering and Sequential Writes:



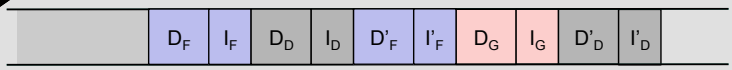
How to Write Sequentially

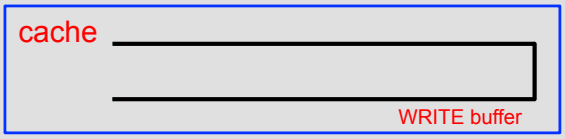
Write Buffering and Sequential Writes:



How to Write Sequentially

Write Buffering and Sequential Writes:





How to Write Sequentially

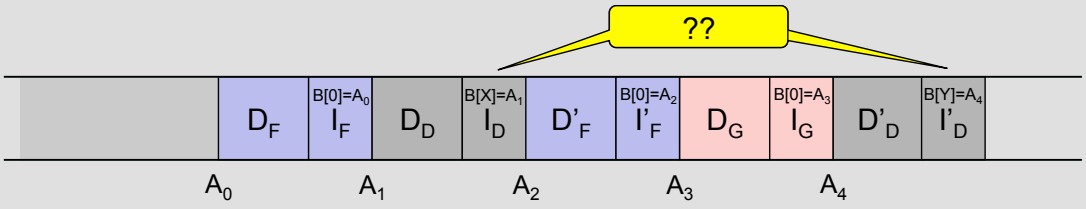
Write Buffering and Sequential Writes:

How to Write Sequentially: Issues

Write Buffering and Sequential Writes:

Issue 1: How to **read** data from the log

- *aka*, "how to find inodes?"

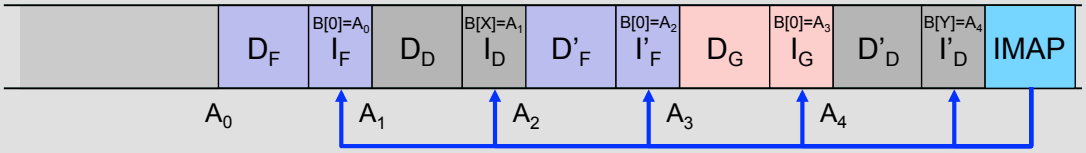


How to Write Sequentially: Issues

Write Buffering and Sequential Writes:

Issue 1: How to **read** data from the log

- *aka*, "how to find inodes?"



IOW: File Location and Reading

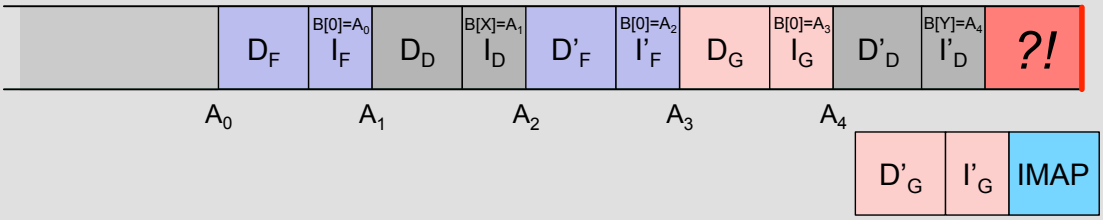
- Traditional "logs" require sequential scans to retrieve data.
- LFS adds index structures in log to allow for random access.
- inode is identical to FFS:
  - Once inode is read, number of disk I/Os to read file is same for LFS and FFS.
- inode position is not fixed.
  - Therefore, store mapping of files to inodes in **inode-maps**.
  - inode maps largely cached in memory.

How to Write Sequentially: Issues

Write Buffering and Sequential Writes:

Issue 2: How to **write** data to the log?

- *aka*, "how to find space for the blocks?"

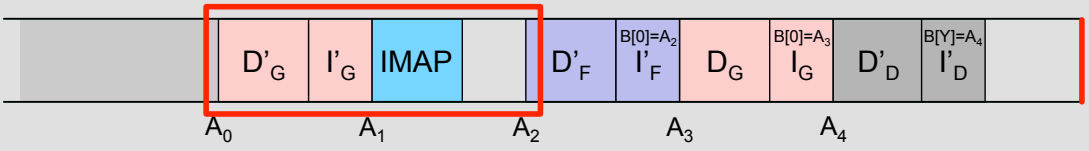


How to Write Sequentially: Issues

Write Buffering and Sequential Writes:

Issue 2: How to **write** data to the log?

- *aka*, "how to find space for the blocks?"



Free-Space Management

Issue: How to maintain sufficiently-long segments to allow for sequential writes of logs?

Solution 1: Thread log through available "holes".

– Problem: Fragmentation

Solution 2: De-Fragment disk space (compact live data)

– Problem: cost of copying live data.

LFS Solution: Eliminate fragmentation through fixed-sized "holes" (**segments**)

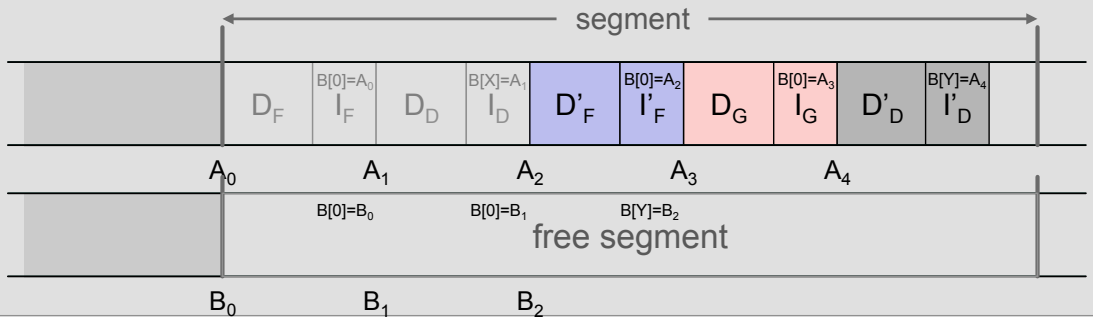
– Reclaim segments by copying **segment cleaning**.

Garbage Collection: Segment Cleaning

Write Buffering and Sequential Writes:

Issue 3: How to **maintain** free segments?

- Garbage Collection

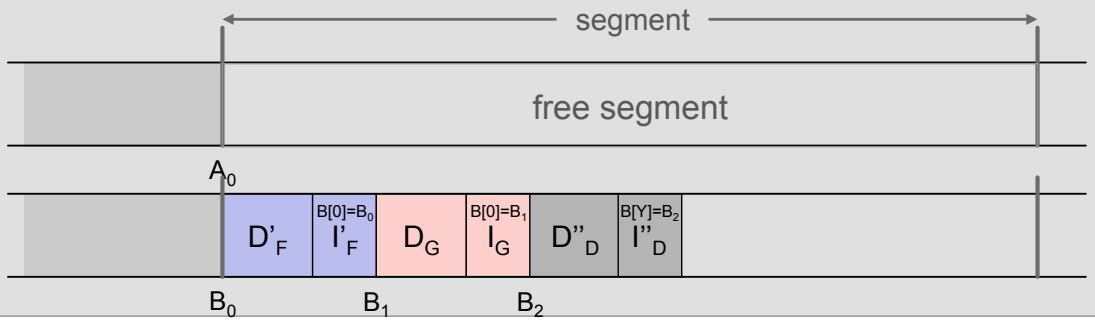


Garbage Collection: Segment Cleaning

Write Buffering and Sequential Writes:

Issue 3: How to **maintain** free segments?

- Garbage Collection



Segment Cleaning: Mechanism

Compact live data in segments by

1. Read number of segments into memory.
2. Identify live data in these segments.
3. Write live data back into smaller number of segments.

Issue: How to identify live data blocks?

– Maintain **segment summary block** in segment.

Note: There is no need to maintain free-block list!

Log-Structured File Systems

- Memory has become cheap!
- This leads to large caches, and large write buffers.
- Problems with UNIX FFS
- Log-structured File Systems: General Ideas
- Write Buffering and Sequential Writing
- Some Practical Issues