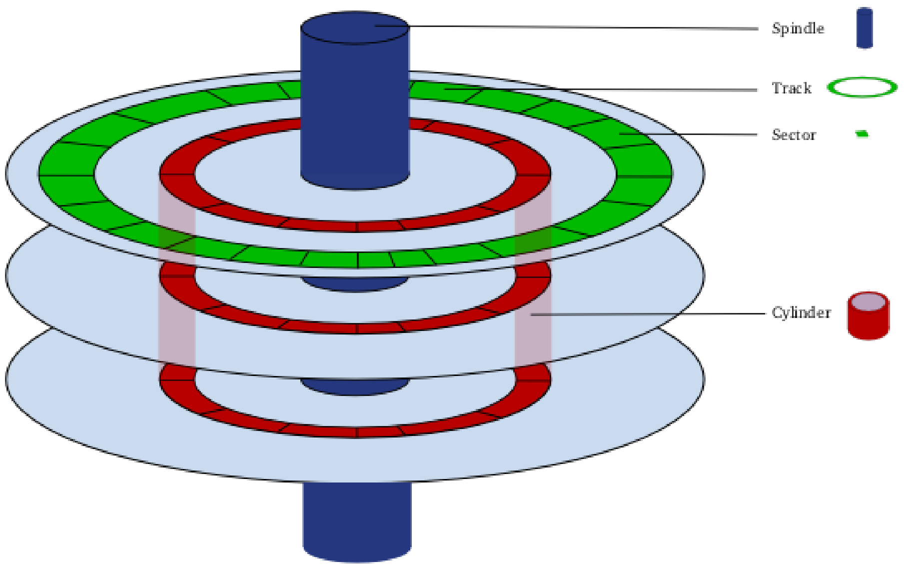
Storage & File Systems

3 February, 2004 William Josephson

Top view of a 36 GB, 10,000 RPM, IBM SCSI
server hard disk, with its top cover removed.
Note the height of the drive and the 10 stacked platters.
(The IBM Ultrastar 36ZX.)

A Typical Disk

- Logically organized as array of sectors
  - Each sector protected by ECC
- Platters divided in concentric tracks
  - CAV on older disks
  - Newer disks multi-zoned
- Tracks organized into cylinders
- Key performance characteristics:
  - Rotational delay
  - Seek time
  - Head/track and cylinder switch
  - Sustained transfer rate
  - Scheduling (zero-latency xfers)



Storage Technology Trends

- Disk trends in the last decade
  - Head switch time little changed
  - 2.5x improvement in seek time
  - 3x improvement in rotational speed
  - 10x improvement in bandwidth
  - $\approx 10^2$ denser
  - $\approx 10^3$ cheaper
  - Compare: the memory wall between processor & core
- Other mass storage technologies becoming popular, too
  - *e.g.* flash in small devices

Dealing with Disaster

- A typical modern disk has
  - MTBF of $\approx$ 1.2M hours
  - Unrecoverable ECC errors on order of 1 in 10^{15}
- Failure modes
  - Manufacturing defects (holes in the film, *etc.*)
  - Magnetic domains decay/flip (thermodynamics!)
  - Head crashes (physical/thermal shock, contamination)
  - ECC errors due to partial writes (esp. ATA disks)

The Unix Filesystem

- Filesystem consists of a (fixed) number of blocks
- Basic unit of organization is the `i-node`
- User data stored as a sequence of bytes in data blocks
- Directories are just special files containing index nodes
  - Directories map path components to `i-node` numbers
- Ken's filesystem was slow and vulnerable to failures
  - For instance, allocated blocks from a free list on disk

The BSD Fast File System

- CSRG addressed performance and reliability concerns
  - Increased the block size and introduced fragments
  - Improved allocation and layout policies
    - Allocate file blocks in "rotationally optimal" manner
    - Allocate file blocks in one cylinder group if possible to reduce fragmentation
  - Further work includes softupdates, clusters, traxtents, etc.
- Most operations still require multiple disk I/Os

Softupdates: Motivation

- Metadata updates are a headache:
  - Performance, integrity, security, & availability problems
- Traditional filesystems either:
  - Compromise on safety (*e.g.* Ext2, FAT)
  - Make extensive use of synchronous updates (*e.g.* FFS)
  - Use special-purpose hardware (*e.g.* WAFL)
  - Use shadow-paging or write-ahead logging
- Softupdates allow write back caches to delay writes *safely*
  - Low-cost sequencing of fine-grained updates

Softupdates: Operational Overview

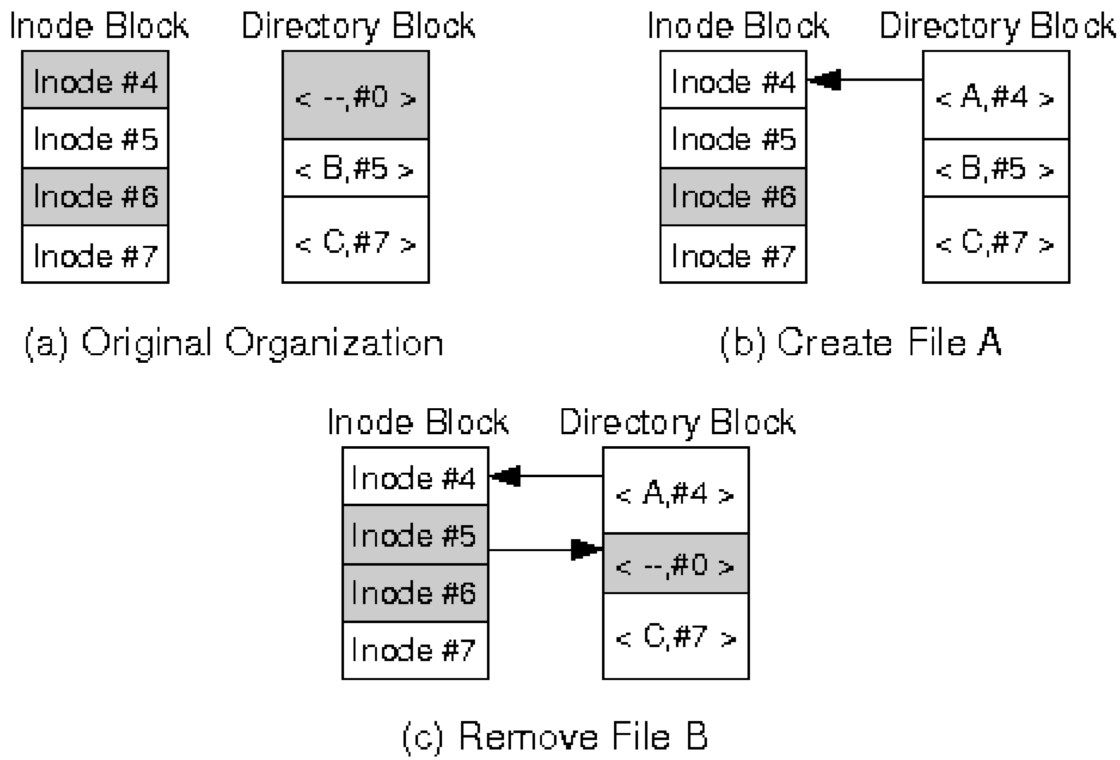
- Goal: better performance through fine-grain dependency tracking
- Softupdates allow for safe use of write-back caches for metadata:
  - Track depenedencies to present consistent view to client
  - Ensure state on stable storage is also *consistent*
    * May lose data not yet on disk, but disk image not corrupt
  - Dependency information consulted when flushing a dirty block
    * Aging problems avoided as new dependencies are never added to existing update sequences

Implementing Softupdates

- Maintain update list for each pointer in cache
  - File system operations add updates to each pointer affected
  - Updates can be rolled backwards or forwards as needed
  - Blocks are locked during a roll-back
- Simple block-based ordering is insufficient
  - Cycles must still be broken with synchronous writes
  - Some blocks may "starve" waiting for dependencies
  - Block granularity introduces false sharing

Softupdates: Cyclic Dependencies

- Block level granularity of writes can introduce dependencies



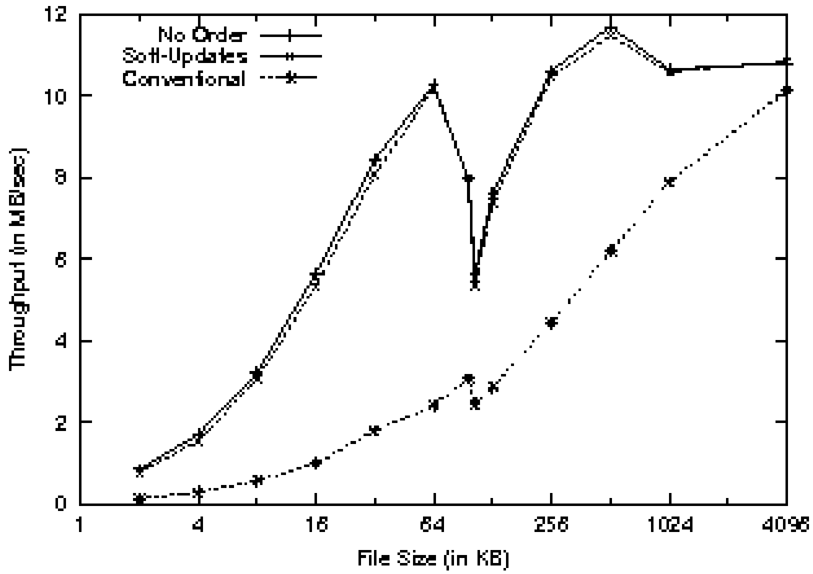
Shaded regions indicate free metadata structures

Softupdates for FFS

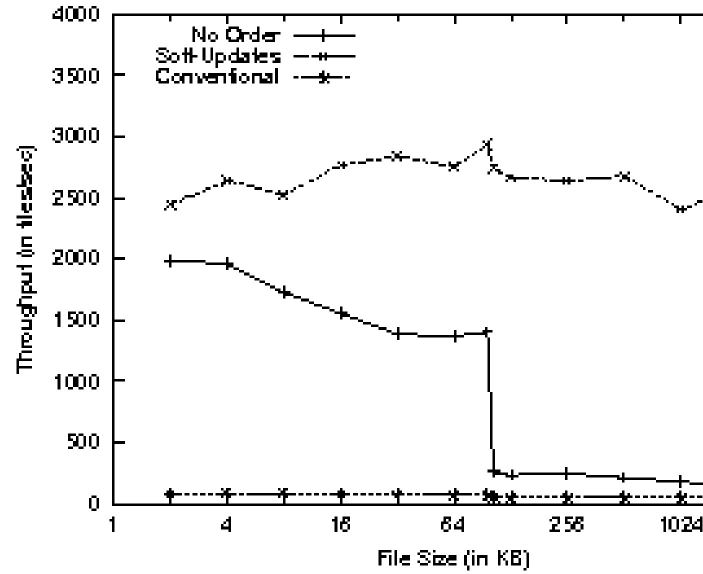
- Structural changes: block (de)allocation, link addition/removal
- File system semantics essentially unchanged
  - Synchronous metadata updates do not imply synchronous semantics (last write typically asynchronous)
  - Softupdates allow caching metadata with same write-back strategies as for file data
- With cheap update sequencing, can afford stronger guarantees
  - Can therefore safely mount filesystem immediately
  - Background `fsck` can reclaim leaked blocks

Softupdates: Performance, I

- Compare create and delete throughput as a function of file size for conventional, no order, and softupdates



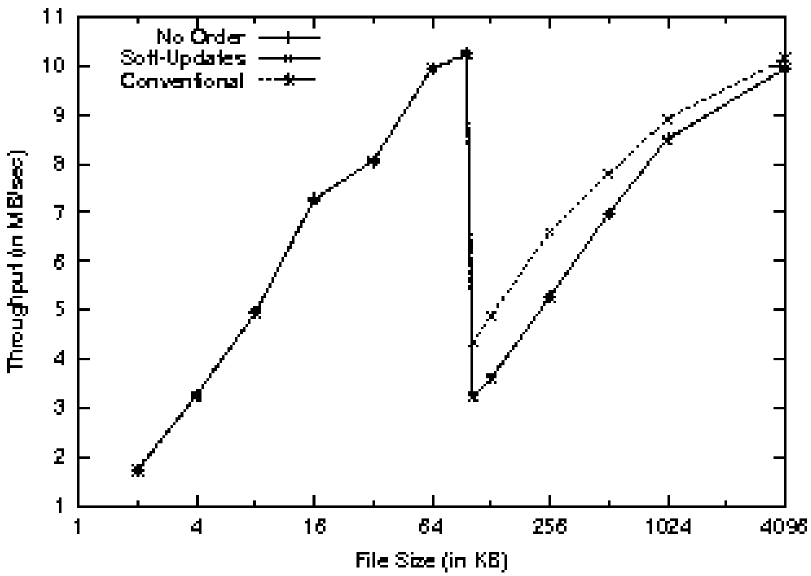
Create Microbenchmark



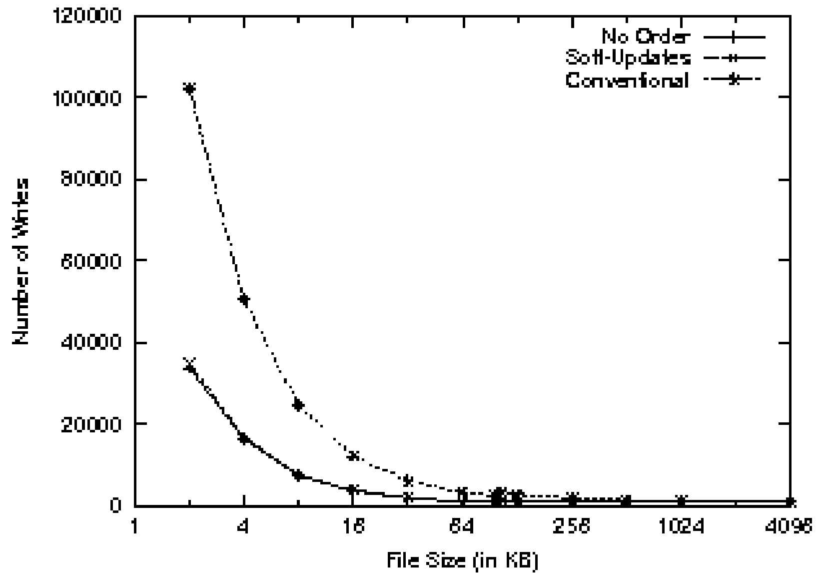
Delete Microbenchmark

Softupdates: Performance, II

- Read performance improved by delayed writes
  - Better indirect block placement
- Softupdates coalesces metadata updates in the create benchmark



Read throughput (MB/s)



Total Disk Writes for Create Benchmark

Softupdates: Performance, III

- In macrobenchmarks, softupdates also performs well
- Postmark – small, ephemeral file workload (mail server):
  - No order: 165 tps; Softupdates: 170tps; Conventional 45 tps
- On a real mailserver: softupdates offers 70% fewer writes
- `fsck`: virtually instantaneous on 4.5G file system *vs.* conventional `fsck` time of almost three minutes

The Log-structured File System: Motivation

- CPU speed increases faster than disk speed – I/O bottleneck
- Aggressive caching can improve read performance
- Relative write performance suffers
  - Can't naively cache writes and still maintain safety
  - Many filesystems use synchronous writes for metadata
  - So metadata dominates for small files typical for Unix

LFS: Operational Overview

- Goal: improve throughput through better write scheduling
- Write performance drives filesystem design:
  - Treat the disk as a circular log
  - Write all data and metadata blocks together in the log
  - Attempt to keep large free extents
    * Batch writes in “segments”
    * Segment size chosen on the basis of disk geometry

LFS: On-Disk Data Structures & Crash Recovery

- Data, index nodes, and other metadata all written to the log
  - Inodes are not written at fixed locations
- Fixed check-point regions record inode map locations
  - Inode maps are aggresively cached to avoid disk seeks
- Log is regularly checkpointed
  - Flush active segment and inode maps
  - Serialize dependent operations with additional log records
- On restart, either truncate or roll the log forward; no `fsck`

LFS: The Cleaner

- Logically, the log is infinite, but the disk is finite
  - Garbage collect (“clean”) old segments
- Cleaner can either “thread” or copy and compact the log
  - LFS uses a hybrid approach, copying entire segments
- Compare cleaning policies on the basis of *write cost*
  - Average time disk is busy per byte of *new* data:

$$\text{wc} = \frac{\text{read} + \text{write live} + \text{write new}}{\text{write new}} = \frac{2}{1-u}$$

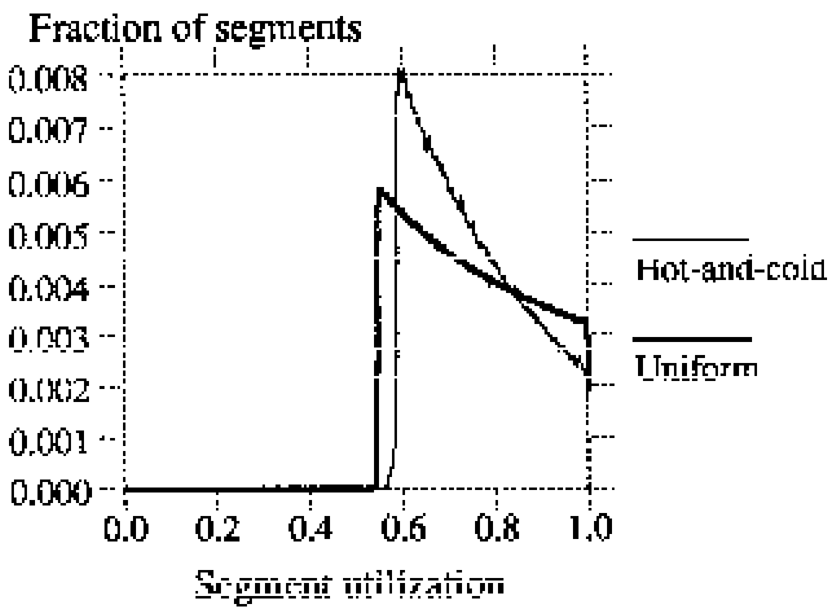
LFS: Cleaner Policies

- When to clean?
  - At night, in the background, during idle periods, on demand
- How much to clean?
  - Fixed number of segments or bytes
- Which segments to clean?
  - Poorly utilized segments, well-chosen cost/benefit metric
- How to reorganize cleaned segments?
  - Group for locality, group by age, expected write cost

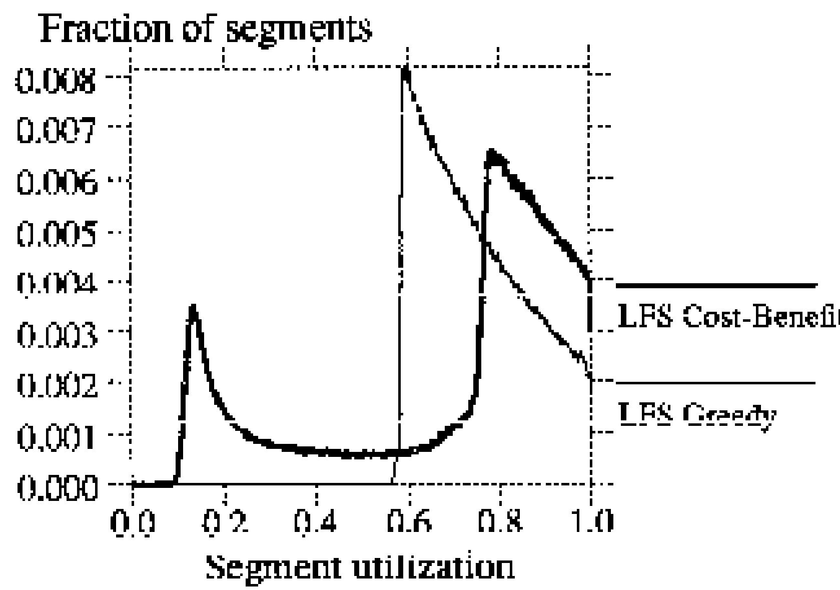
LFS: Cleaner Policies, II

- Hot and cold segments *not* equal ⇒ bimodal behavior
  - free space in cold segment more valuable

$$\frac{\text{benefit}}{\text{cost}} = \frac{\text{free space generated} * \text{age of data}}{\text{cost}} = \frac{(1-u) * \text{age}}{1+u}$$



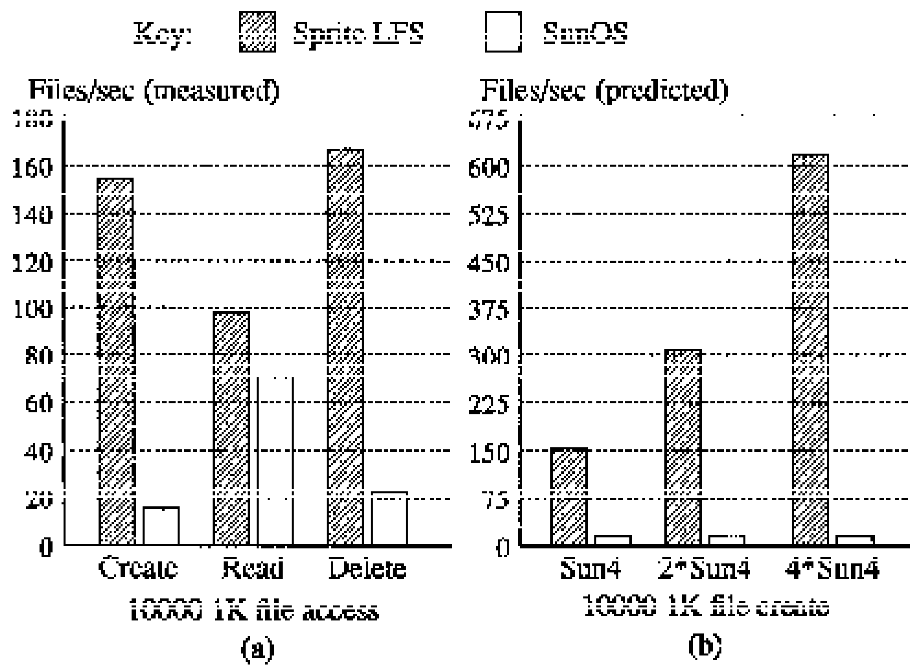
Greedy



Cost-Benefit

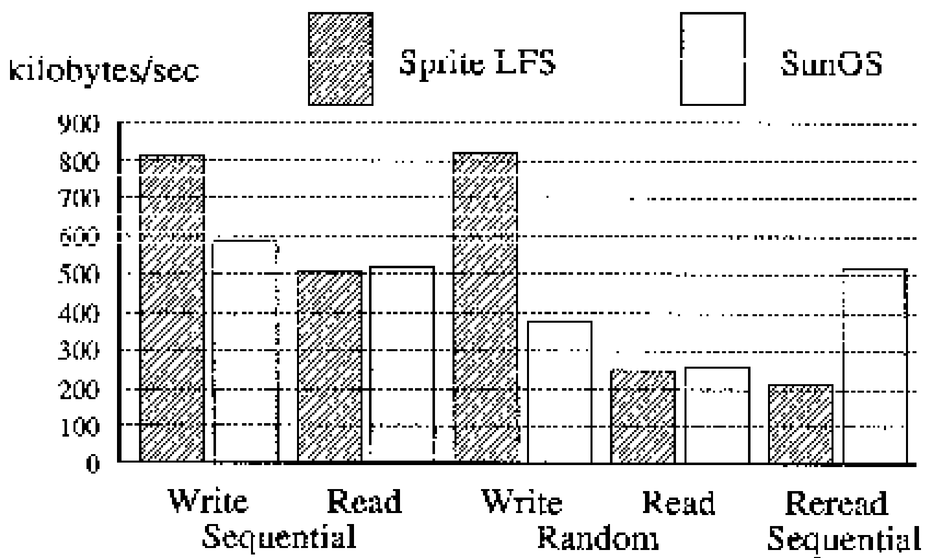
LFS: Performance, I

- Small file comparison of LFS and SunOS UFS in simulation
  - LFS offers better throughput with lower overhead
  - Implies that LFS will scale better given technology trends
  - Somewhat artificial (*e.g.* re-read files in order written)



LFS: Performance, II

- Large file comparison of LFS and SunOS UFS in simulation
- LFS performs well in simulation for common workloads
- LFS does not handle some corner cases as well
  - High disk utilization and/or little idle time
  - Read-heavy workloads; sequential read after random write



Some Perspective on Filesystems

- We read about LFS and Softupdates
  - Both seek to improve performance while maintaining safety
  - Both are targeted at general purpose workloads
- Recent research has focused on:
  - Application-specific workloads (*e.g.* hummingbird)
  - Backup (*e.g.* WAFL, Venti+Fossil, several start-ups)
    - Tape is obsolete – small, slow, sequential access, expensive
  - Finding data in secondary and tertiary storage:
    - Regulatory and policy changes mandate long-term archival
    - Massive amounts of raw data (*e.g.* MGH's NMR unit)