Storage Systems

NPTEL Course

Jan 2012

(Lecture 37)

K. Gopinath

Indian Institute of Science

Orderings!

- Disk ordering (such as elevator alg)
- SCSI ordering (such as “ordered tag”)
- Msg Ordering (such as “virtual synchrony”)
- FS ordering (such as “synch, asynch, delayed write”, ordered write, soft updates, logging/ journalling, transactions)
- Appl ordering (such as “fsync, forder”, transactions)

Ordering Models for Storage

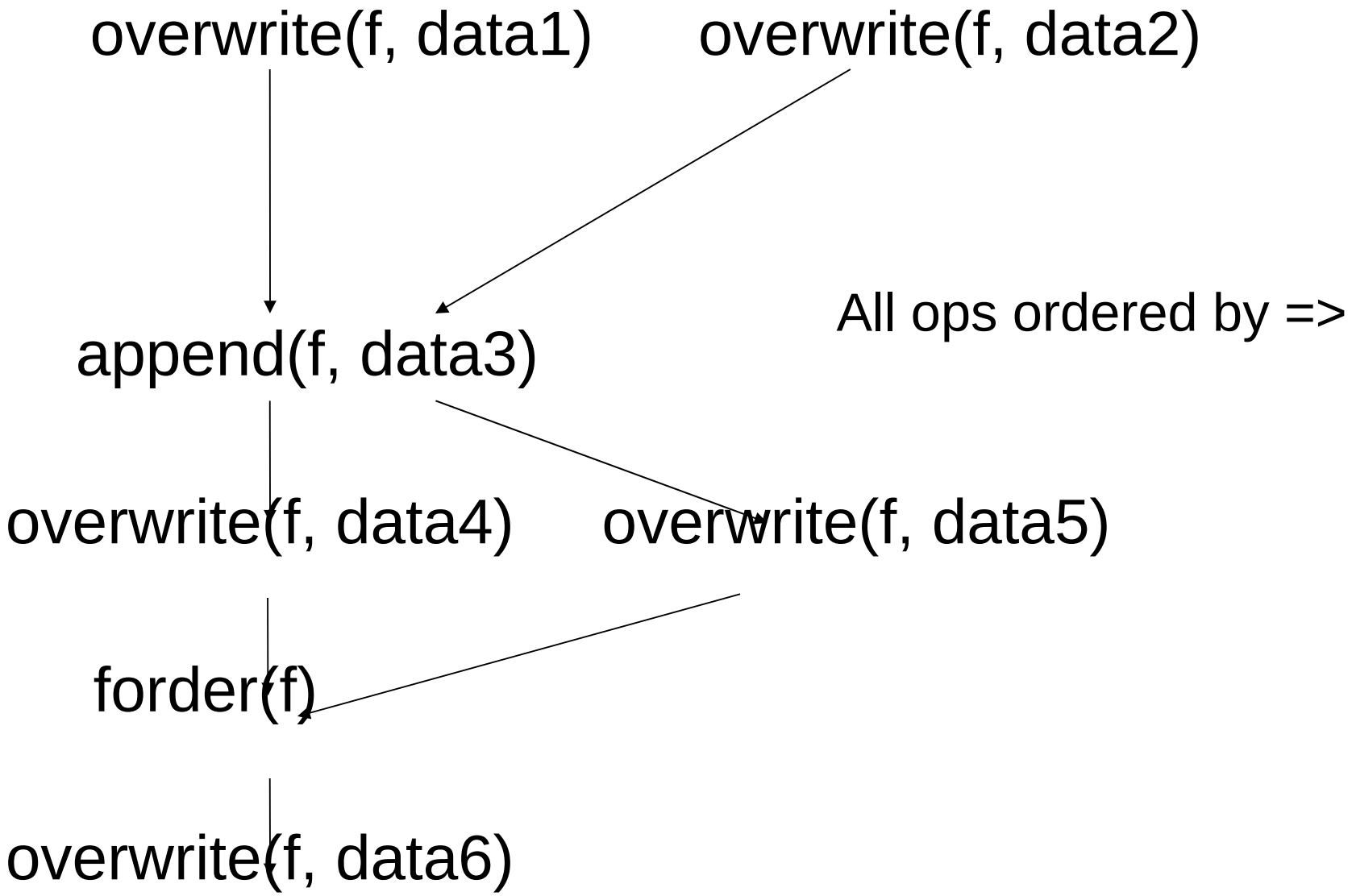
- Consider Echo distr FS model (ACM TOCS May'94 Mann et. al.)
  - replication (servers+disks), caching (==single-copy equiv), global naming (single-system view), distr security
  - Coherent write-back caches for both files+dirs thru ordered write-behind
    - write-behind: written back after a fixed time
    - write-back: written back after an unbounded time
  - Large caches that are transparent except on faults/crashes
- Avoid NFS drawbacks such as
  - incoherent caches (some NFS do close-on-sync but not dirs)
  - unlinking open file problem
  - applns can write even if no space avlbl on server

Echo

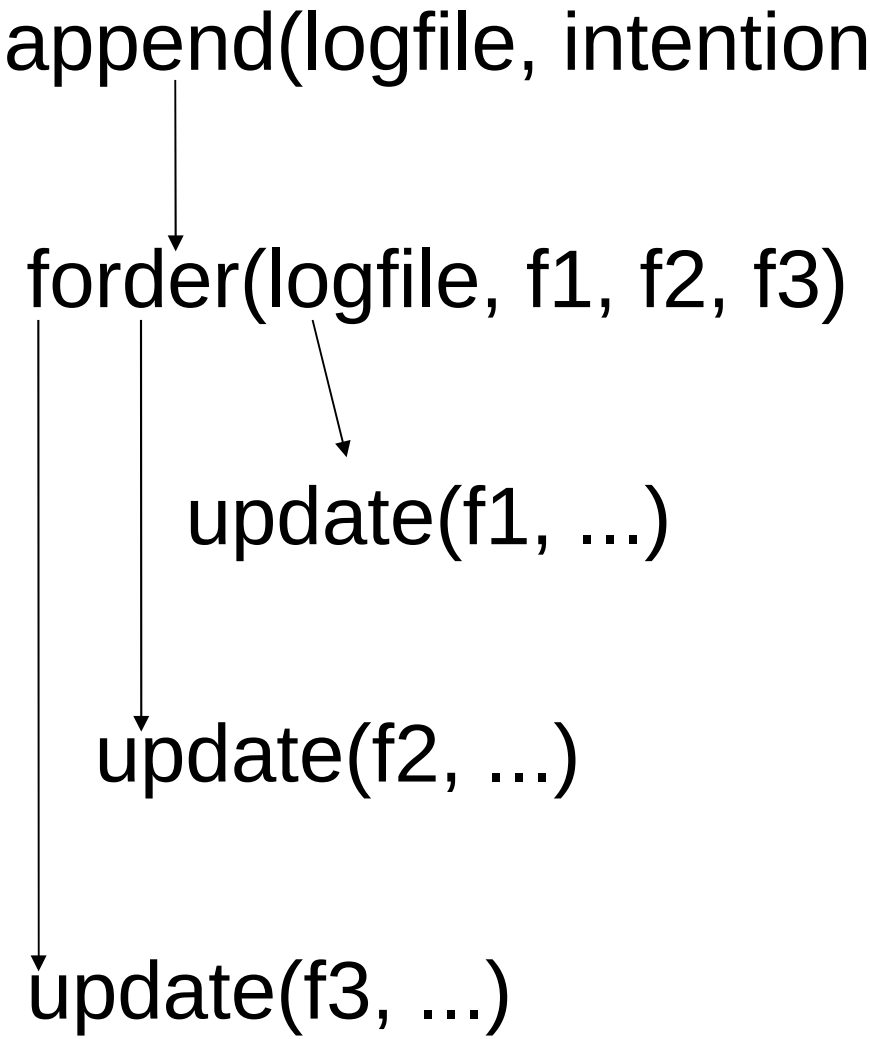
- Ordered & stable writes needed if writes can be discarded at any time
  - write requested by one client and observed by another: write should be stable
  - writes on same obj should be stable in logical order
    - overwrites (length preserving) by one client can be reordered as an "opt"
    - overwrite failure-atomic if only one block modified
  - fsync on dir and files should make them stable
  - forder: constrains ordering of write
    - forder(f1, f2,...): any pending ops on f1, f2,... logically performed before any ops ordered after forder
      - an “update” to each of its arguments: like “touch” in makefiles
    - returns immediately unlike fsync

Echo Model

- Define 2 relations: -> (data dep) and => (partial order for stable writes)
  - => a subset of ->
  - both -> & => transitive
- o1->o2 if o1 is a write, o1 & o2 have an operand in common, o1 performed logically before o2, o1 not discarded when o2 performed
- o1=>o2 if o1->o2 and o1 & o2 writes but not overwrites
- if o1=>o2 and o1 discarded implies o2 discarded
- if o1->o2 and o1 & o2 on diff clients, o1 stable when o2 performed
- if o1=>o2 and o2 stable implies o1 stable
- if fsync(f) successful, f is stable


overwrite(f, data1)
overwrite(f, data2)
append(f, data3)
All ops ordered by =>
overwrite(f, data4)
overwrite(f, data5)
forder(f)
overwrite(f, data6)

Write-ahead logging



forder ensures that none of the updates will reach disk before the log record does.

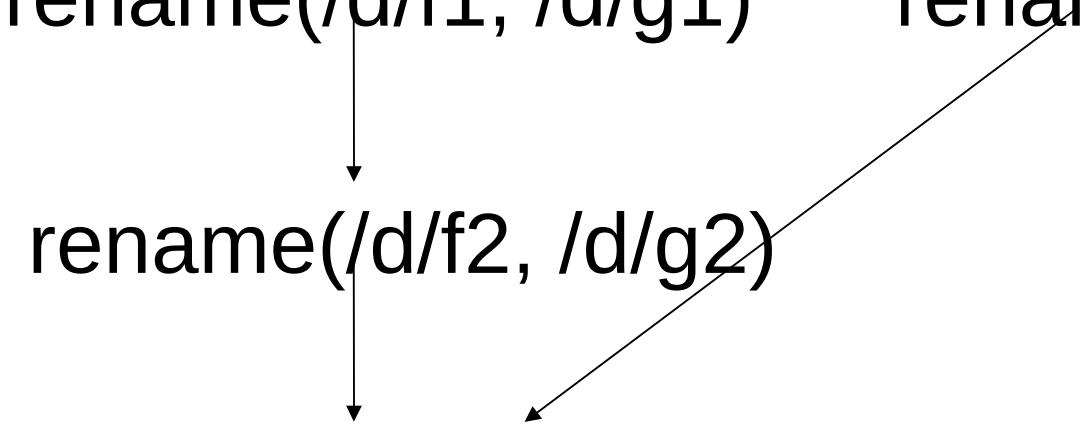
Soft updates in BSD fs tracks cached buffers through the -> relation but it does not have the => relation explicitly

Effected thru flush deamon

rename

rename(/d/f1, /d/g1)

rename(/e/f1, /e/g1)



rename(/d/f2, /d/g2)

rename(/d/f3, /e/f3)

rename(/a/f, /b/f) with preexisting /b/f : modifies a, b, /a/f, /b/f (4 ops)

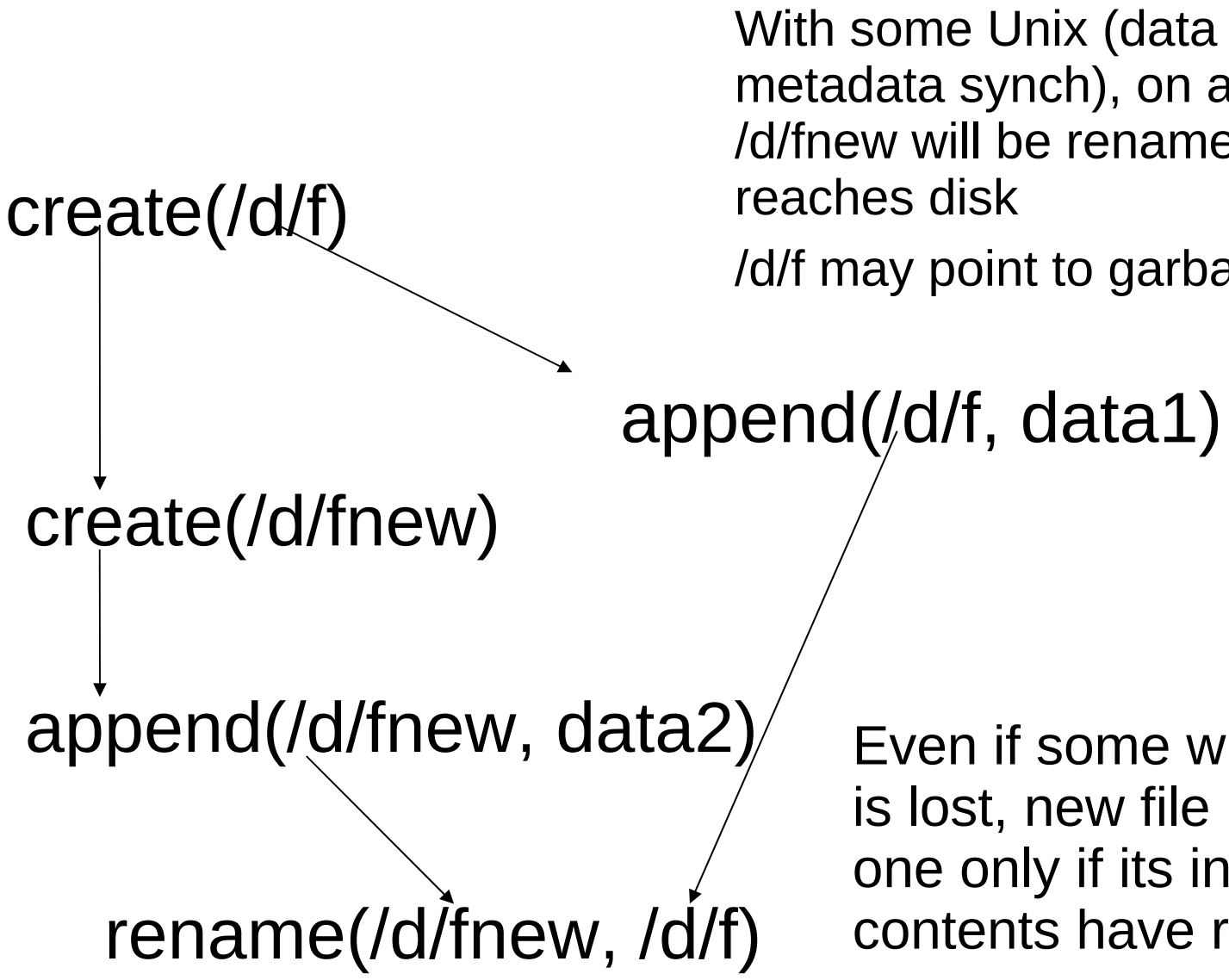
int rename(const char *old, const char *new);

DESCRIPTION

The rename() system call causes the link named old to be renamed as new. If new exists, it is first removed. Both old and new must be of the same type (that is, both must be either directories or non-directories) and must reside on the same file system.

The rename() system call guarantees that an instance of new will always exist, even if the system should crash in the middle of the operation.

Write a file and replace with another atomically



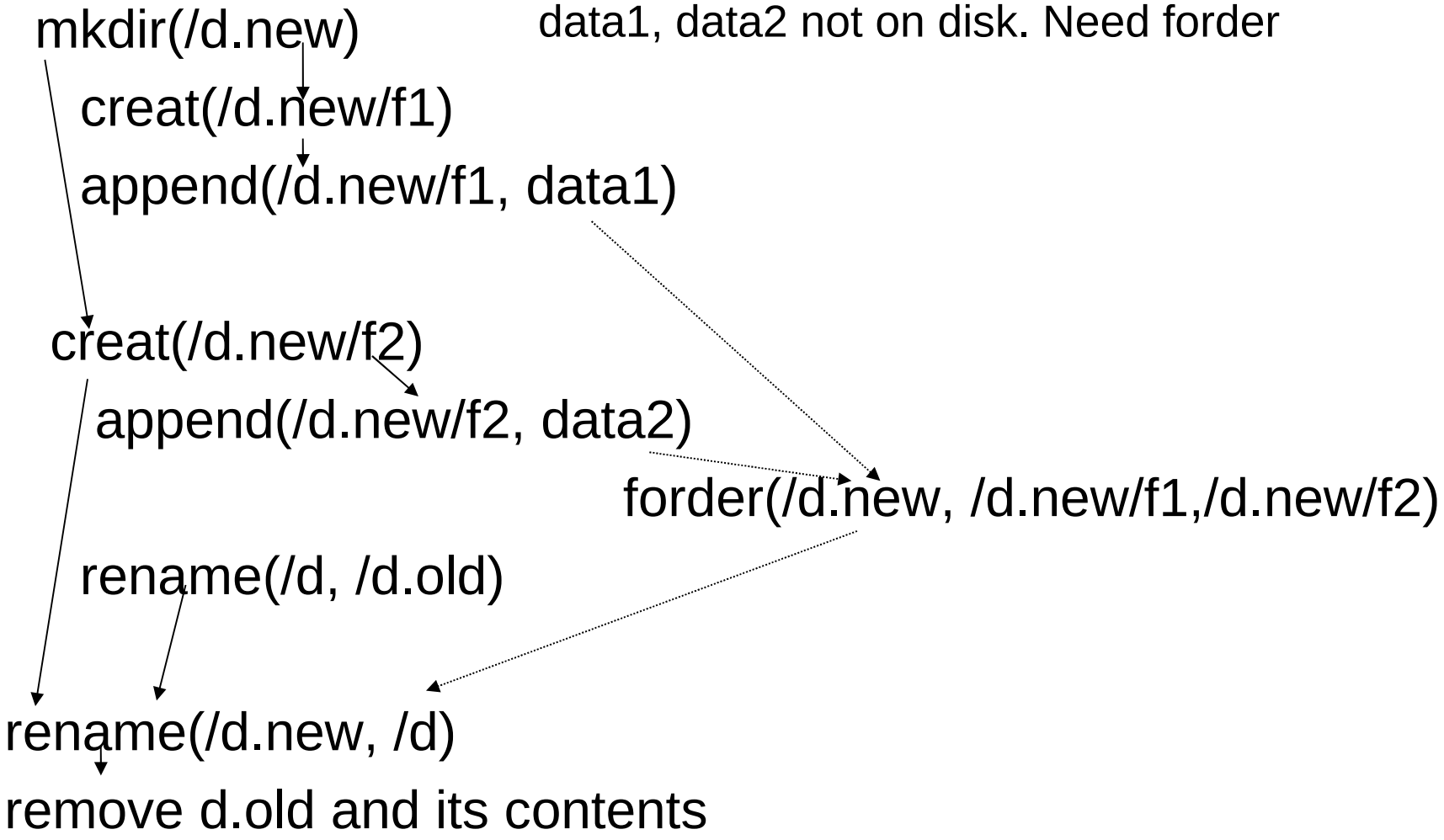
With some Unix (data not synch but metadata synch), on a crash:
/d/fnew will be renamed before data reaches disk

/d/f may point to garbage

Even if some write-behind is lost, new file replaces old one only if its intended contents have reached disk

Replace a dir with a new version

On a crash, possible that d has garbage files as data1, data2 not on disk. Need forder

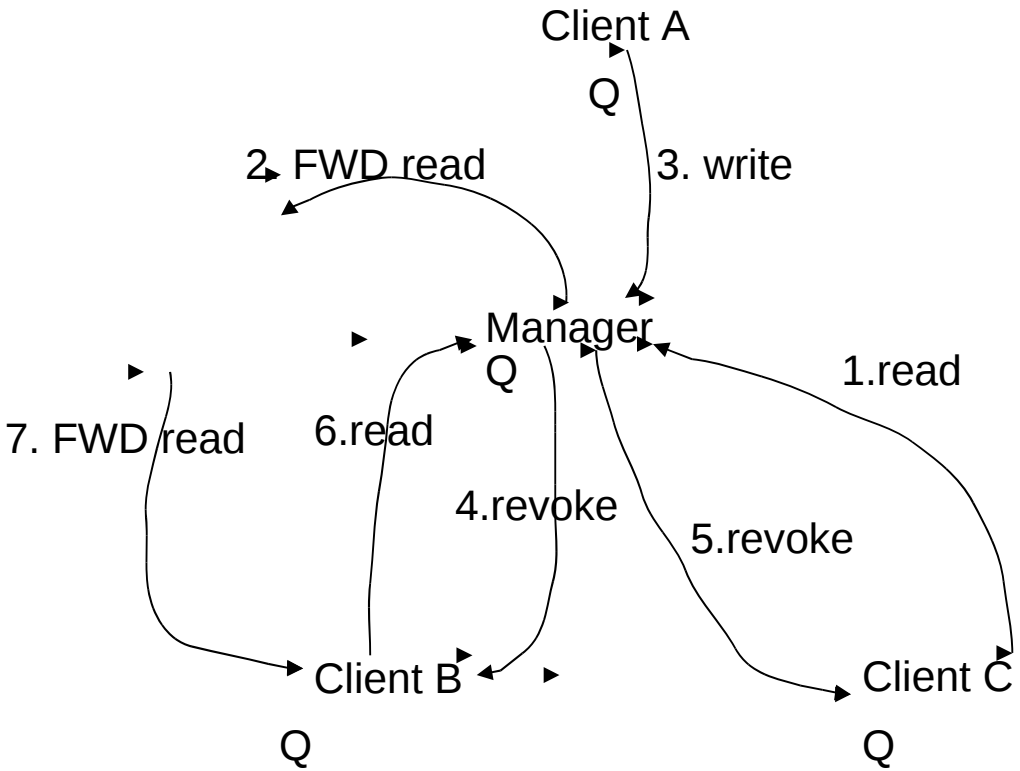


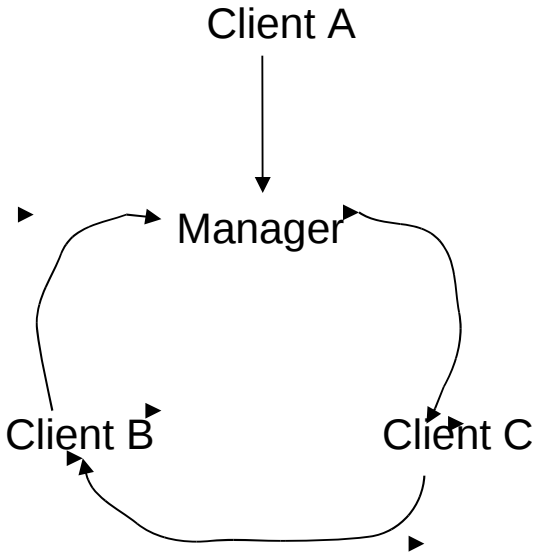
Failure Semantics

- An appln proc P depends on a write w if P issued w or if P issued r/w op o & w->o
  - If w discarded, then P has inconsistent view of FS
- If P depends on a discarded w,
  - std recovery mode: error on any further op on vol
    - Better, send asynch signal to all processes P affected
  - self-recovery mode: all open files on vol marked (incl P's cwd), any op on these error; file accesses thru absolute pathnames work still but not relative: useless in practice
    - Actually need failure handle (h) for each open along with fd
    - If w or o issued with h, w->o, then w->h
    - If w discarded, any op later issued with h in error
  - null-recovery mode: all open files of P on vol marked (but not P's cwd), any op on these error

Lack of orderings in dfs

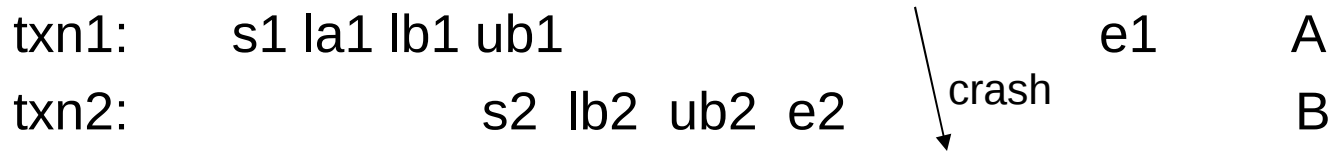
- Berkeley *xfs* '95: deadlock. Causal models
- Also, need forder type of ordering





Episode dfs redo-undo logging

- write-ahead logging: in each node, old and new value logging for every metadata upd
  - before bcache writes out any dirty data, it is also logged
  - at some time, metadata upd make it to disk.
    - txn aborts: undo upd to metadata
    - txn commits: redo upd
    - fsck still needed to handle hard I/O errors
- 2-phase locking?: locks acq without release until commit
  - Needed in redo-undo logging (no R of uncommitted data)
    - also, difficult with layered systems: lower layer locks have to be exported up
- cascading aborts?: B has changes of txn1+ txn2 but A none!



- Episode does not use 2-phase locking
  - do not have to keep locks till end of txn; can drop as soon as done
  - have to prevent uncommitted data from being read by others
- computes equivalence class (EC) of all active txns that modify same obj
  - all txns of this class commit or none
  - need to minimize size and duration of EC
  - delay use of "hot" data until close to txn commit
- adv over redo-logging: R of uncommitted data possible
  - no undo capability in redo-only: upd to home disk locs cannot occur until txn commits
    - concurrency reduced: constrains buf cache on when to write

Summary

- Considered orderings in a dfs
  - Broadly at device (disk/SCSI), msg (VS), appl (actually, dfs + appl)
  - Providing a proper ordering framework thru these layers not attempted so far
  - May be with storage class memories?