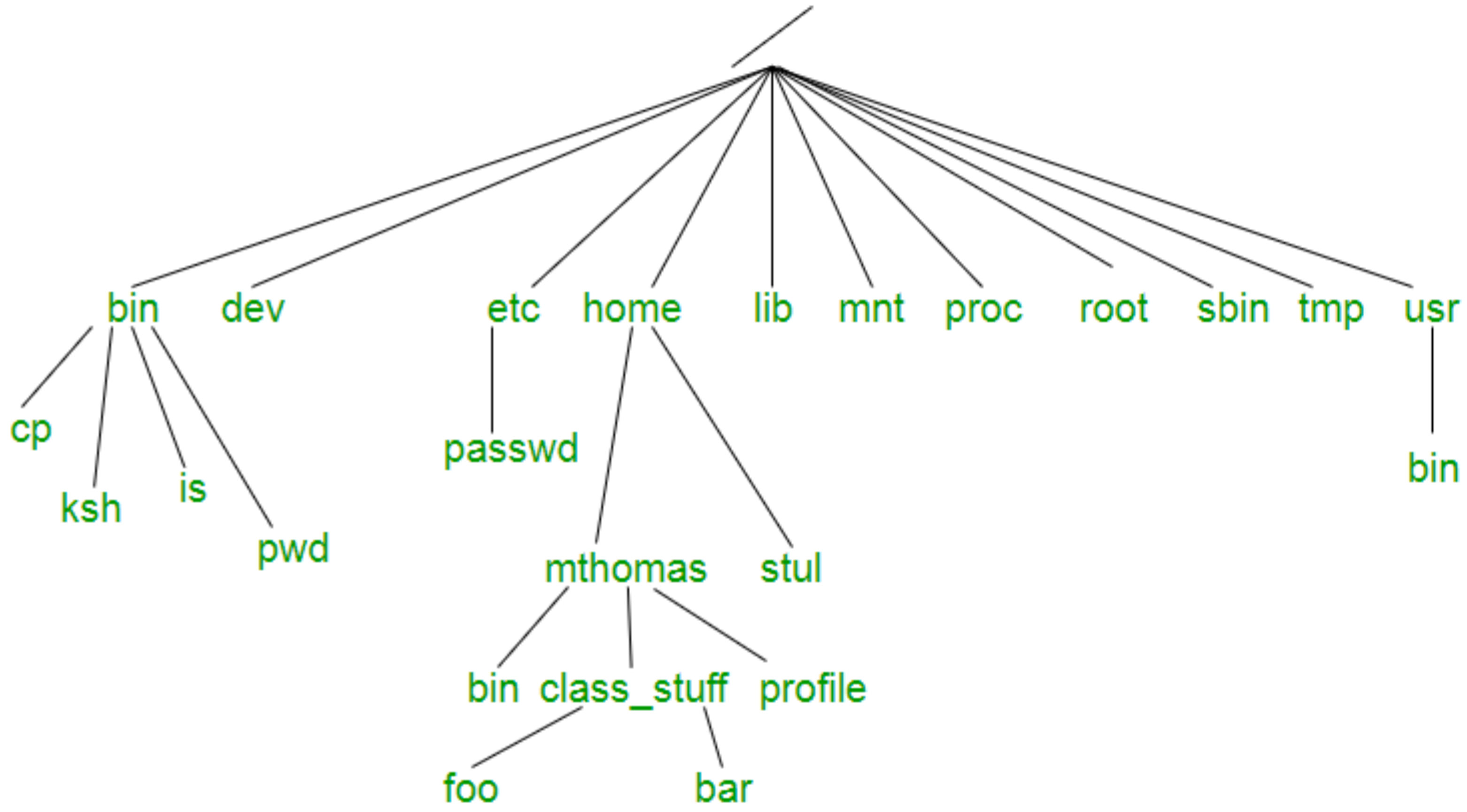
# Consistency Challenge

# Consistency in File Systems

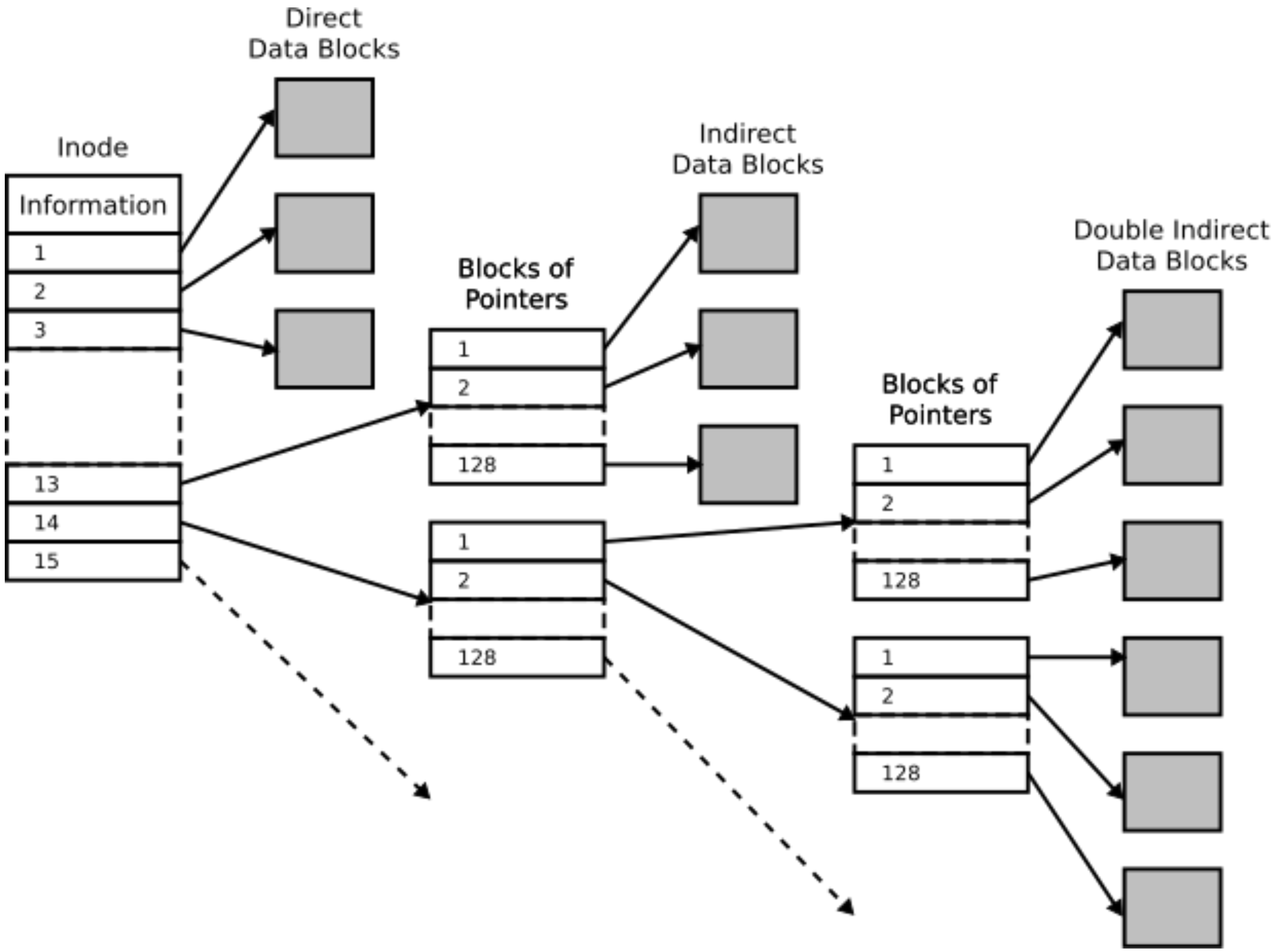
- File systems store metadata and user data
  - Present file in a hierarchical directory structure



Unix File System Example

# Consistency in File Systems

- Unix stores location information in inode tables

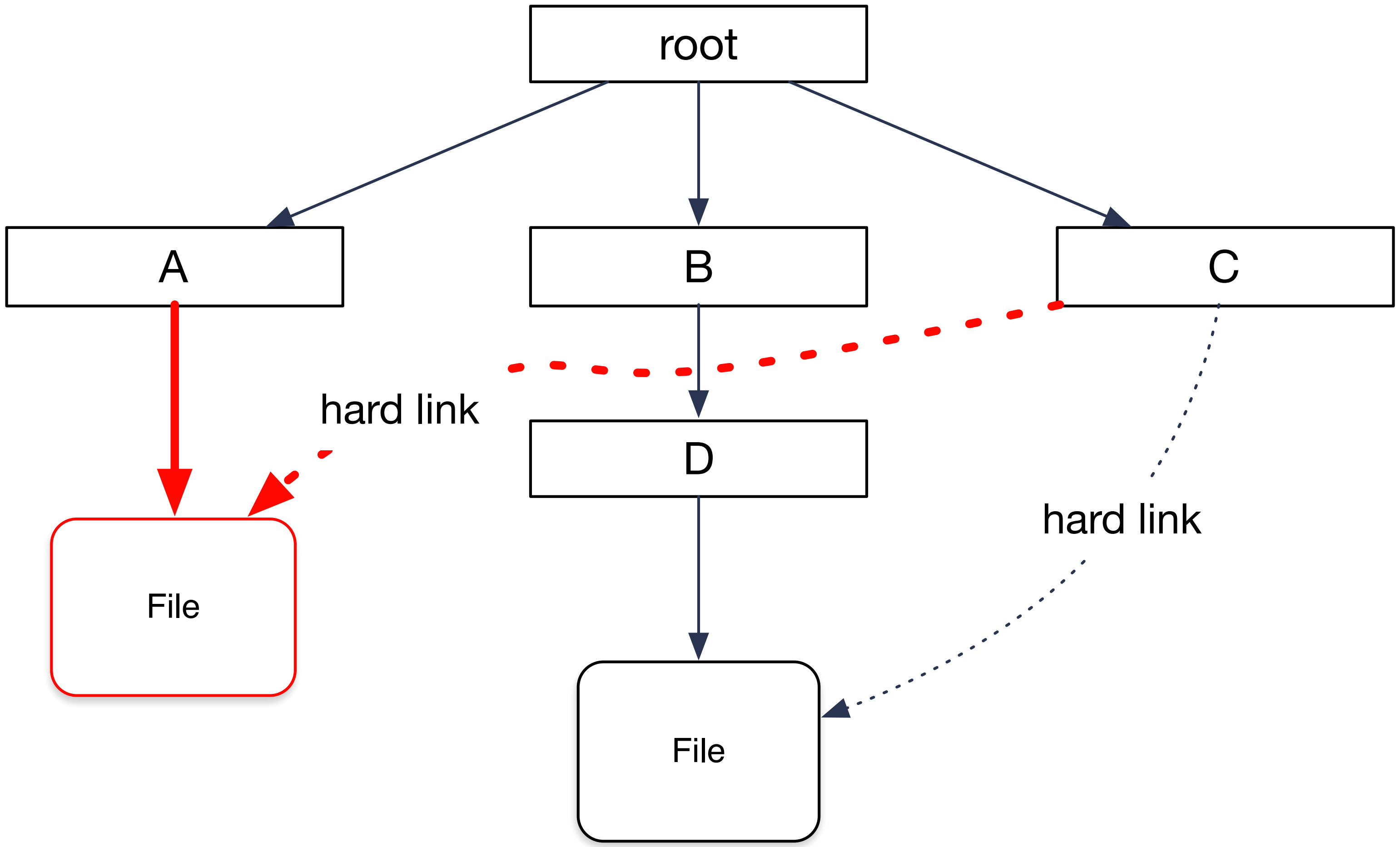


# Consistency in File Systems

- File systems need to survive crashes
  - After a crash, need to be able to recover to a consistent state

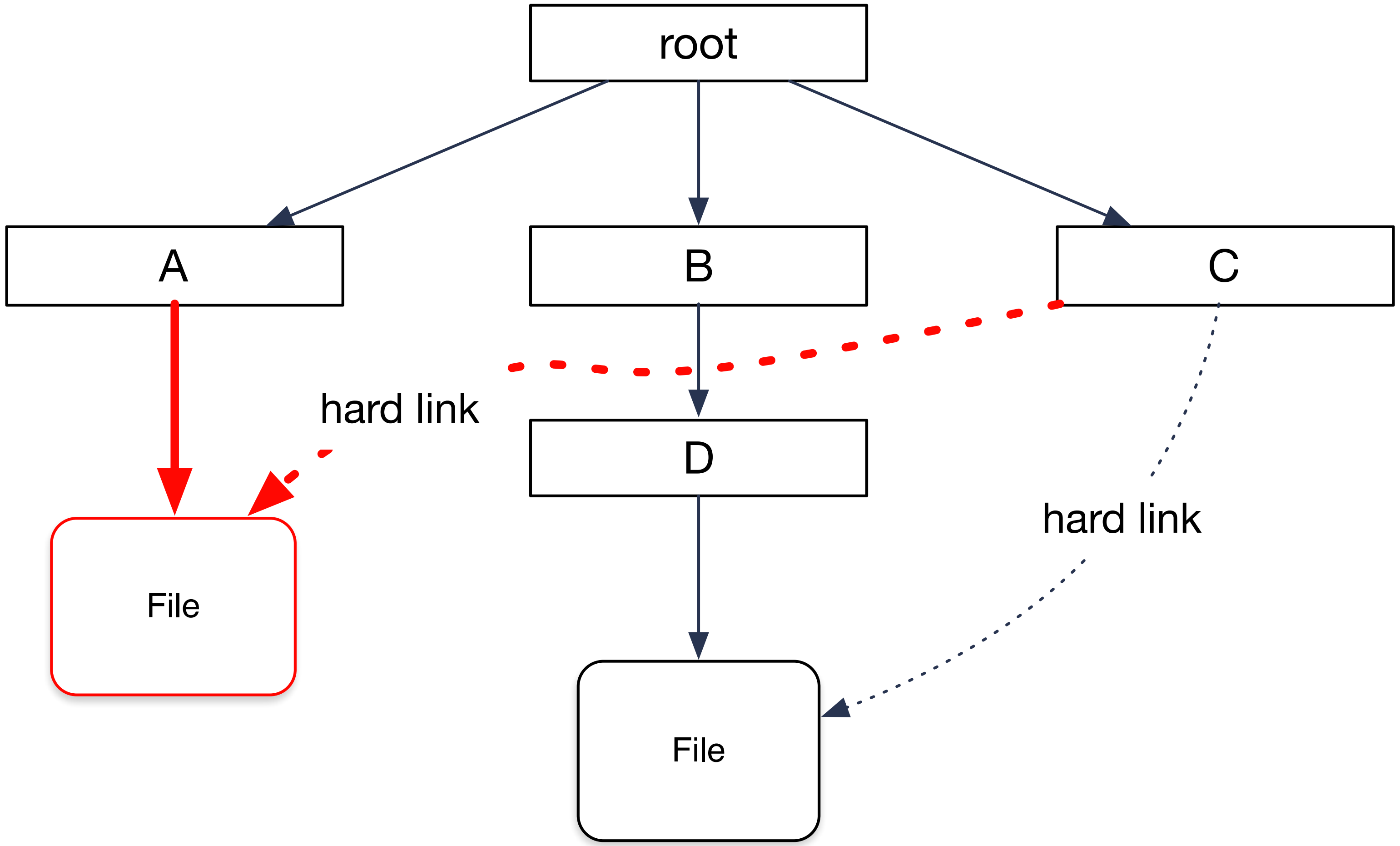
# Consistency in File System Example

- Move an existing file from one directory to another
  - We assume that each directory contains information about its files



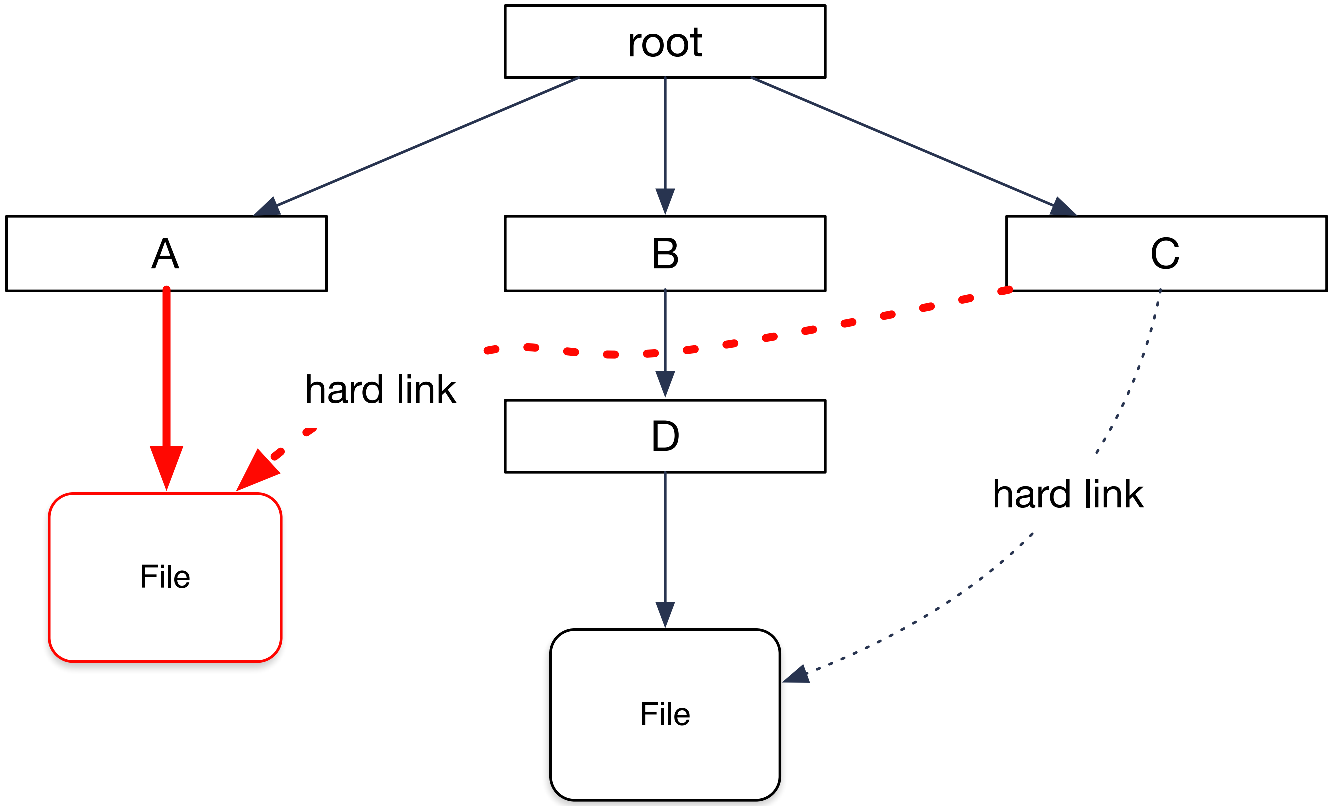
# Consistency in File System Example

- Directories are files with a list of names and inodes
    - In reality, there are other links such as to parent directory



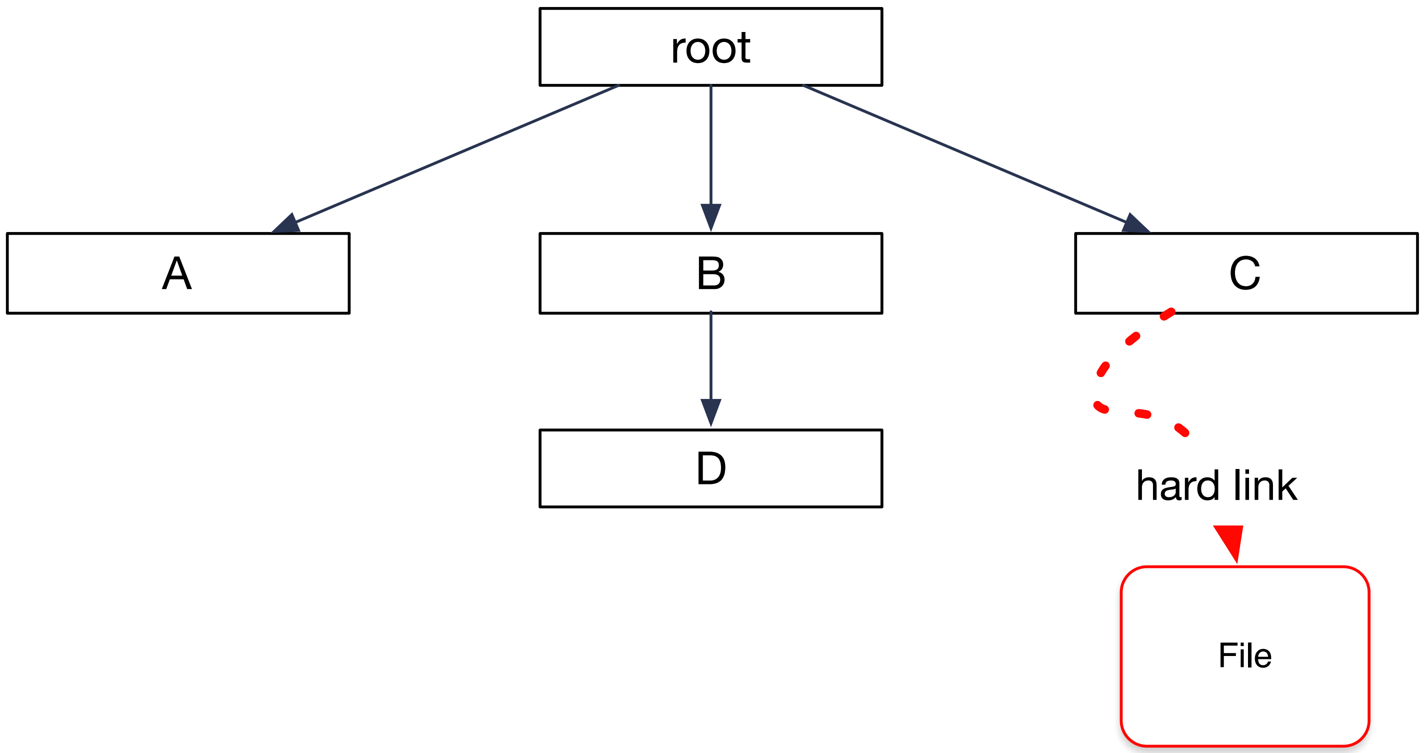
# Consistency in File System Example

- Need to change two directories at the same time



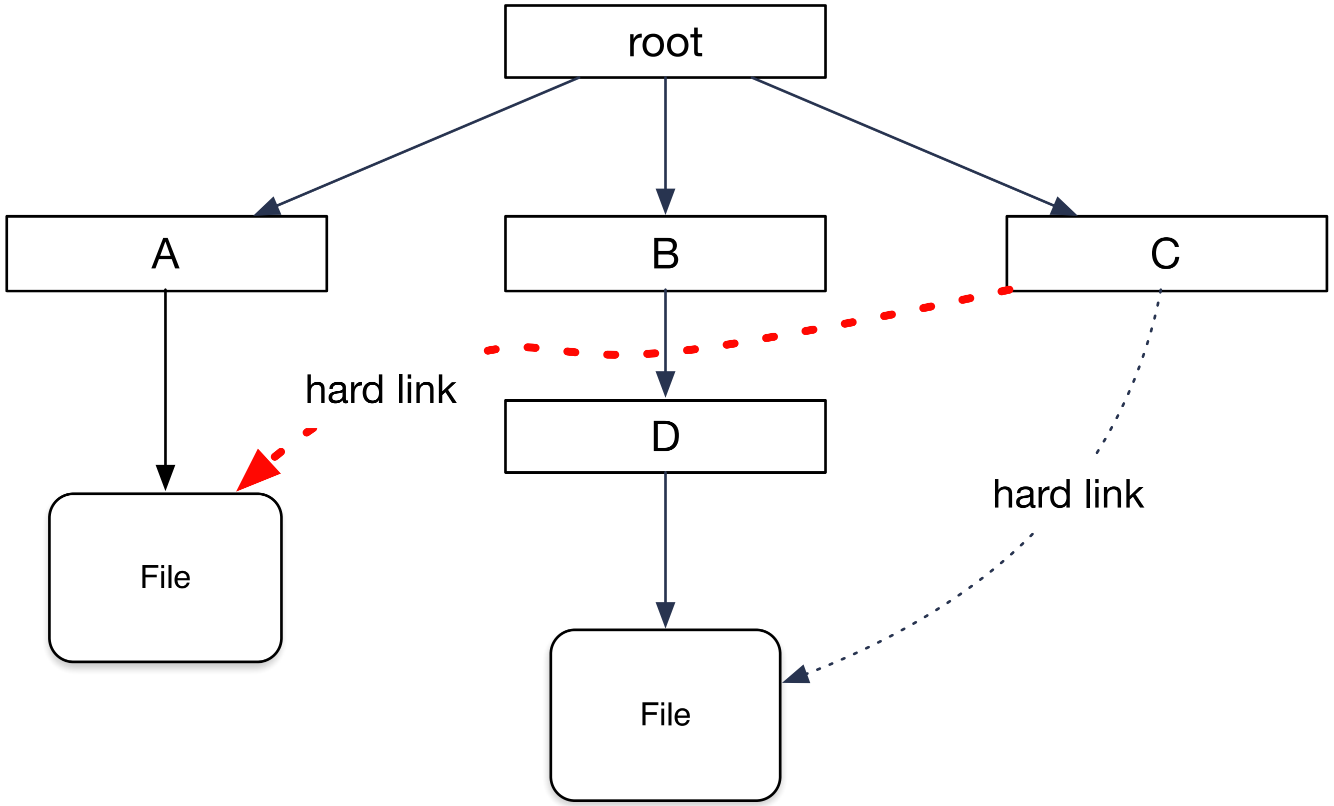
# Consistency in File System Example

- Changes to D — Crash — Changes to A: File is lost but for hard link



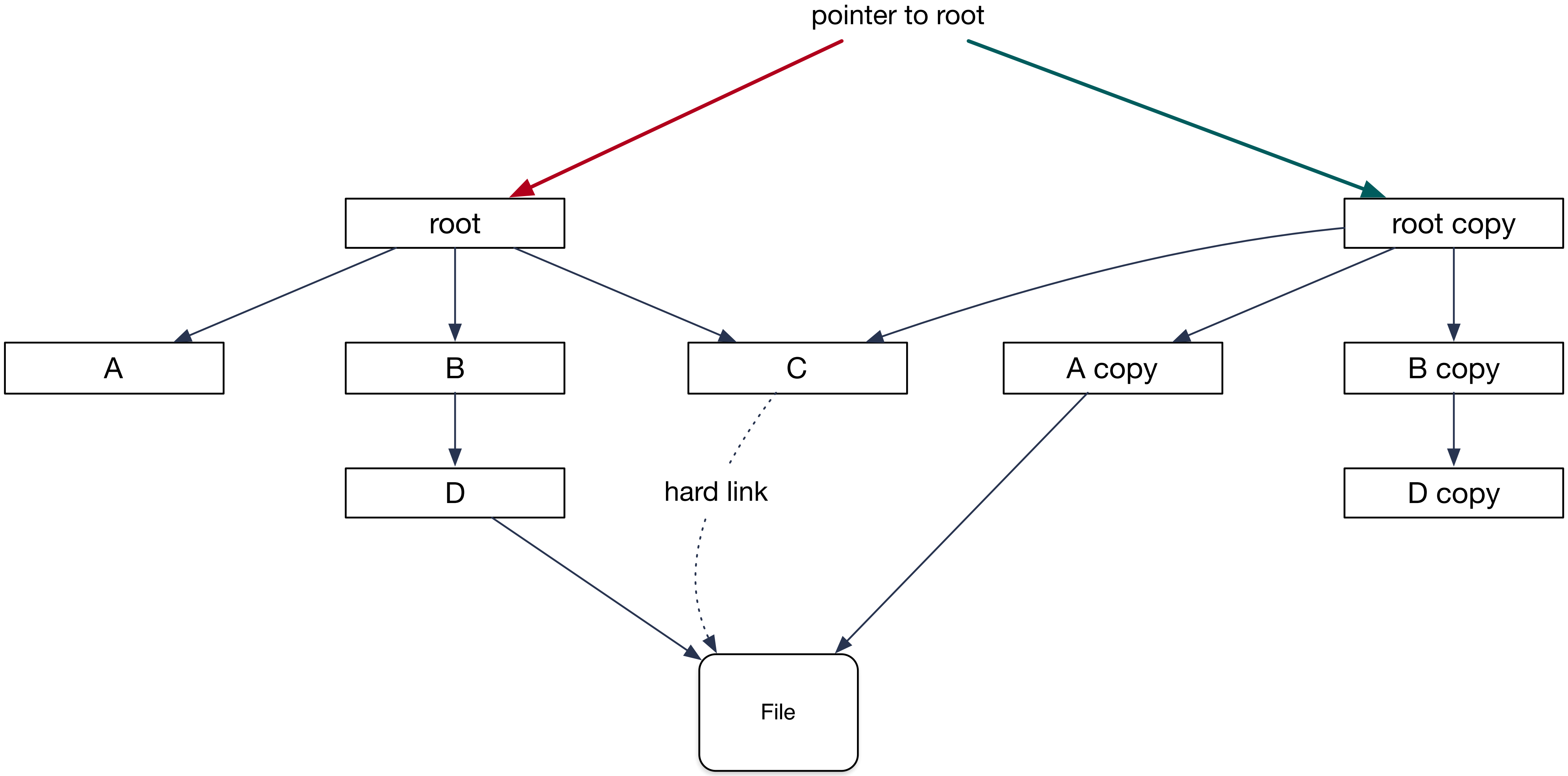
# Consistency in File System Example

- Changes to A made — Crash — Changes to D not made:
  - File is in two directories



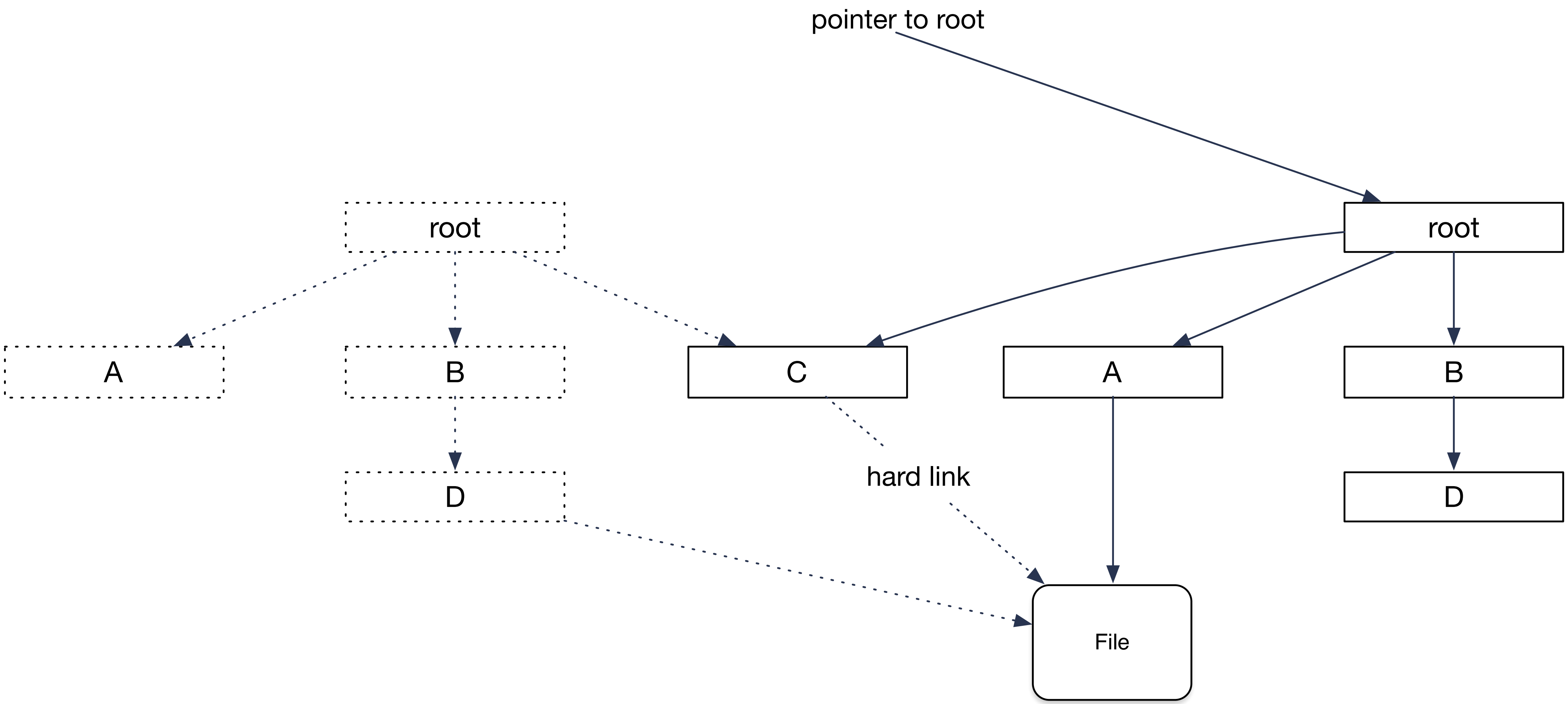
# Consistency in File System Example

- To maintain consistency, we can use extensive copying
  - The red link is valid, all changes are made in the copy



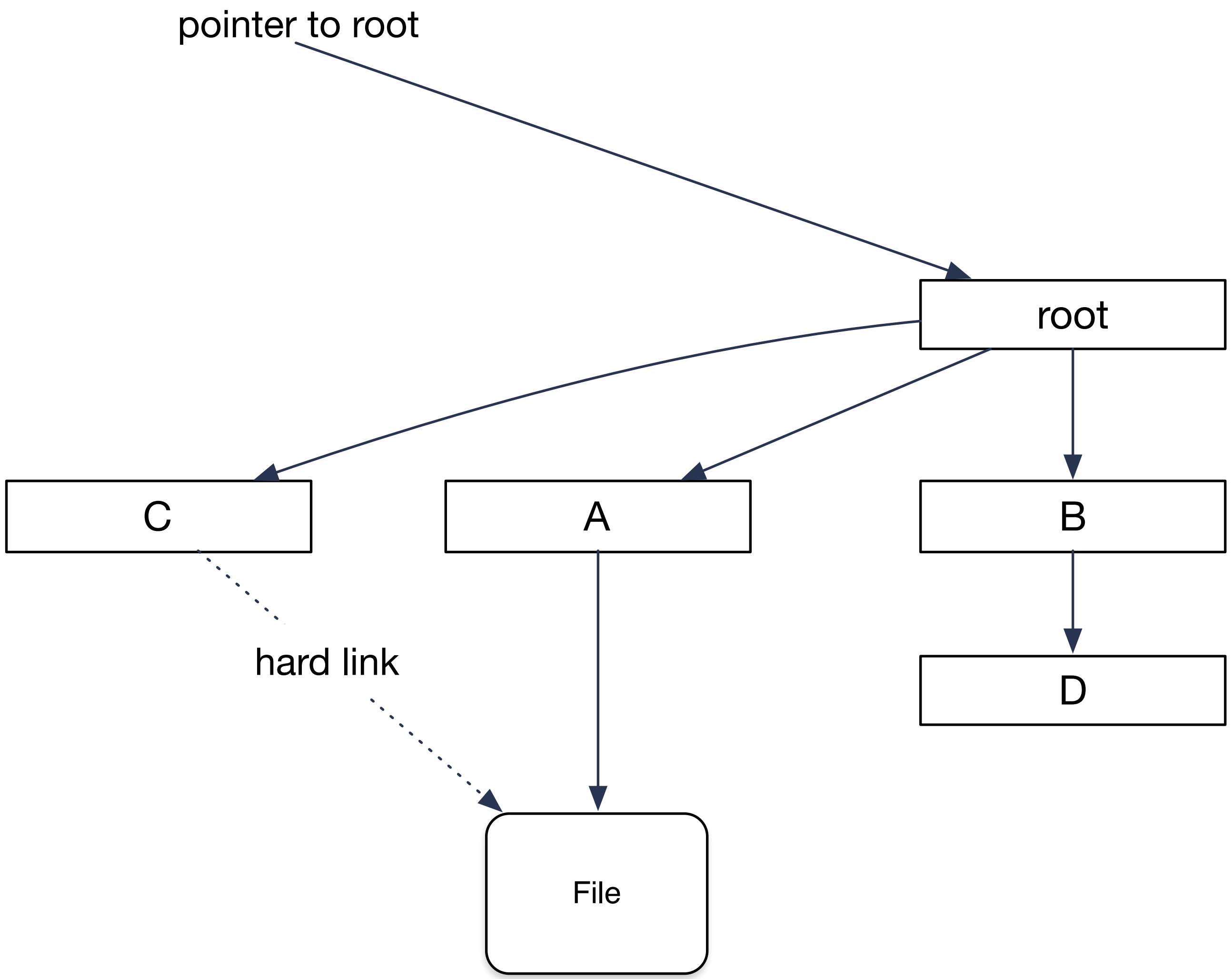
# Consistency in File System Example

- When the new copy is built, can switch to the other link
  - Garbage-collect all data



# Consistency in File System Example

- File system is always in a good state because the switch, updating the pointer to root, is atomic



# Consistency in File Systems

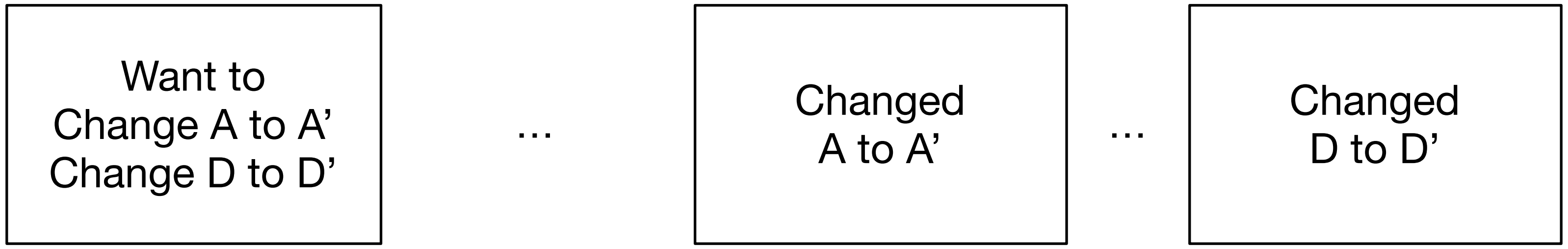
- Keeping the file system always in a consistent state is too expensive.
  - Alternative: Fix after a crash
    - Simple model: just try to make sense of what is to be found
    - Use a journal

# Consistency in File Systems

- Journaling:
  - Before making changes, commit them to the journal
  - Make the changes
  - Write completion into the journal
- In case of crash, can redo the unfinished operations in the journal

# Consistency in File Systems

- State of the journal without crash



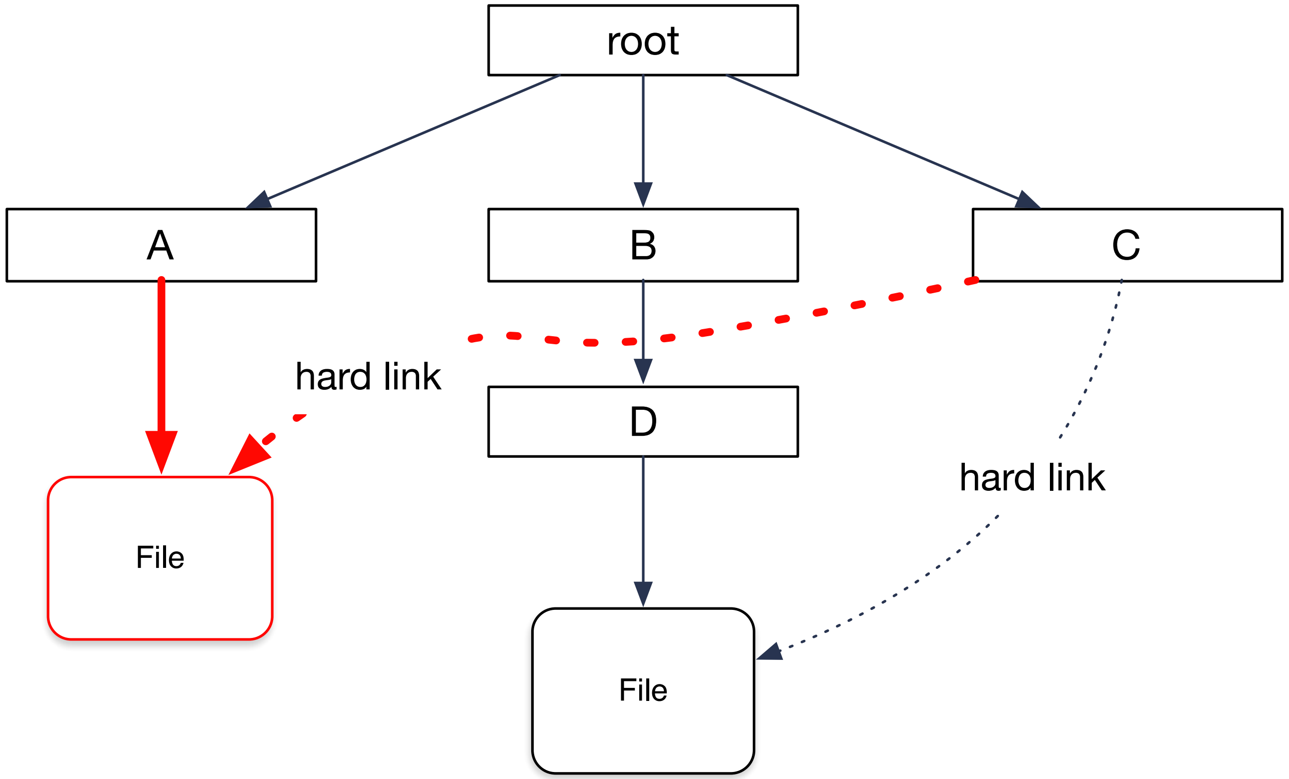
Want to
Change A to A’
Change D to D’

…

Changed
A to A’

…

Changed
D to D’



root

A

B

C

D

File

File

hard link

hard link

# Consistency in File Systems

- Crash before writing the first block to the journal
    - No problem, system remains in the old state
- Crash before changing either A or D
    - Unfinished journal entry means that both changes are initiated
- Crash after changing one but not the other
    - No problem, we change both of them
        - Changes from A to A’ and D to D’ are idempotent:
            - Doing the same operation twice is no error

# Consistency in File Systems

- Crash after changes but before entry is made into journal
  - No problem, we just redo the operations again
- Crash after change entries is made:
  - System stays in new state
- We can mark journal entries as old or remove them to ensure that we do not spend too much time redoing operations in the journal

# Consistency in Distributed Systems