Introducing SoftUpdate to FAT

20060411

Tanaka, Machida – Sony Corp.
Watanabe, Ohtan – AXE Inc.

Terminology

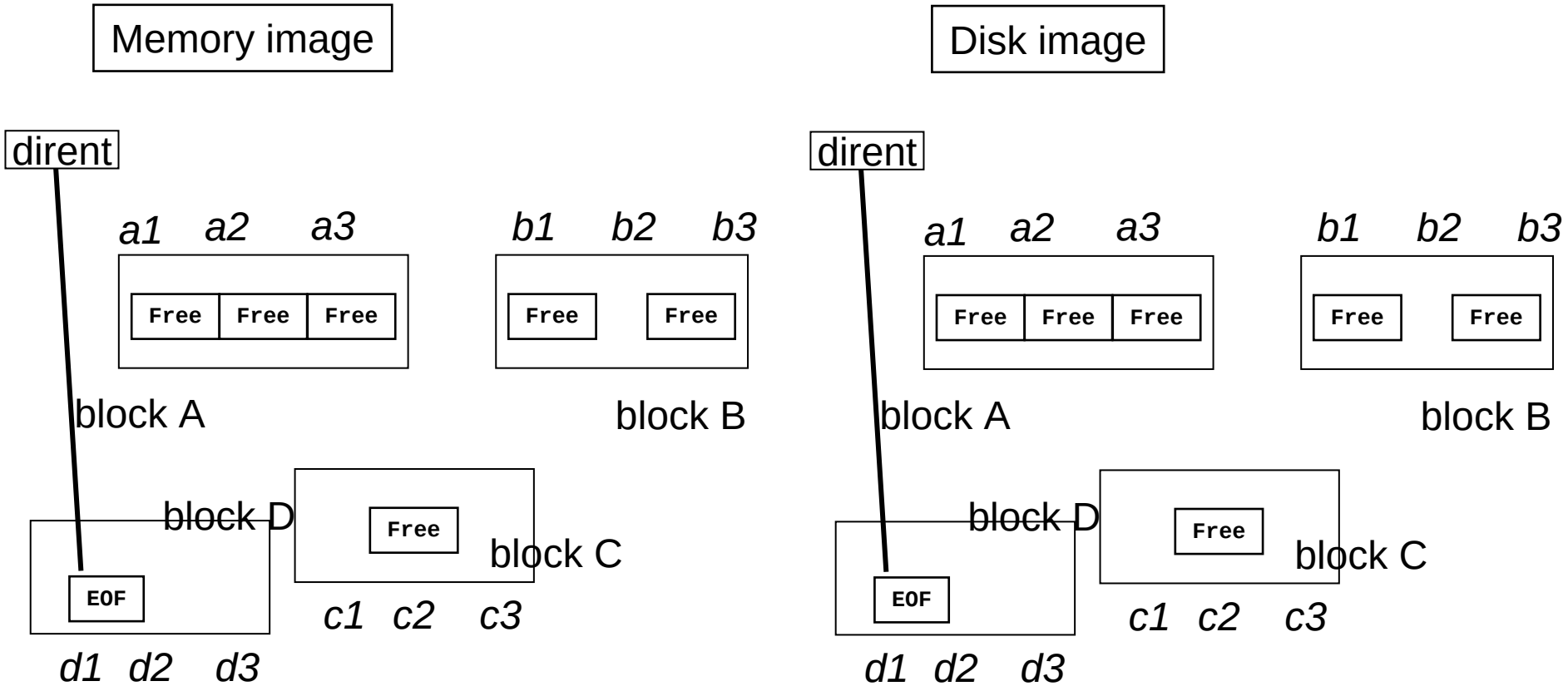
- block – buffer layer
  - Logical minimal I/O unit of File system
  - It consist of one or more sector

- sector – bio layer
  - Physical minimal I/O unit

Scope

- Purpose
  - Keep meta data on disk consistent, on suddenly power done
  - Offline fsck free
- What's SoftUpdate
  - Utilize write-back-cache, but keep meta data on disk consistent
- Address issues with following order
  - expansion and truncate cluster chain (allocation table operations)
  - Dirent operations
  - Online fsck – potential double link issue on move/rename
  - Other operations?
- *This presentation material is based on discussion with Ogawa-san, FAT maintainer*

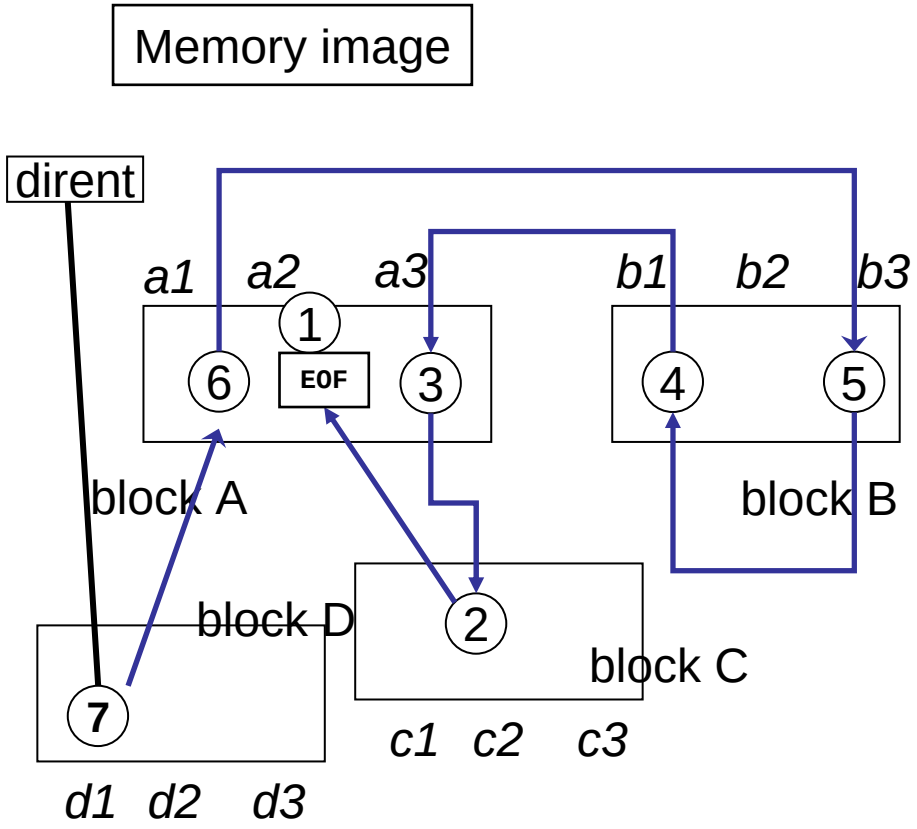
Alloc. table submission – Example expansion case #1 (before modification)



(n) denotes the order of modification

x ⟶ y x depends on y

Alloc. table submission – Example expansion case #1 (mem image)



(n) denotes the order of modification

x ⟶ y x depends on y

Extend cluster chain;

from:* dirent – d1

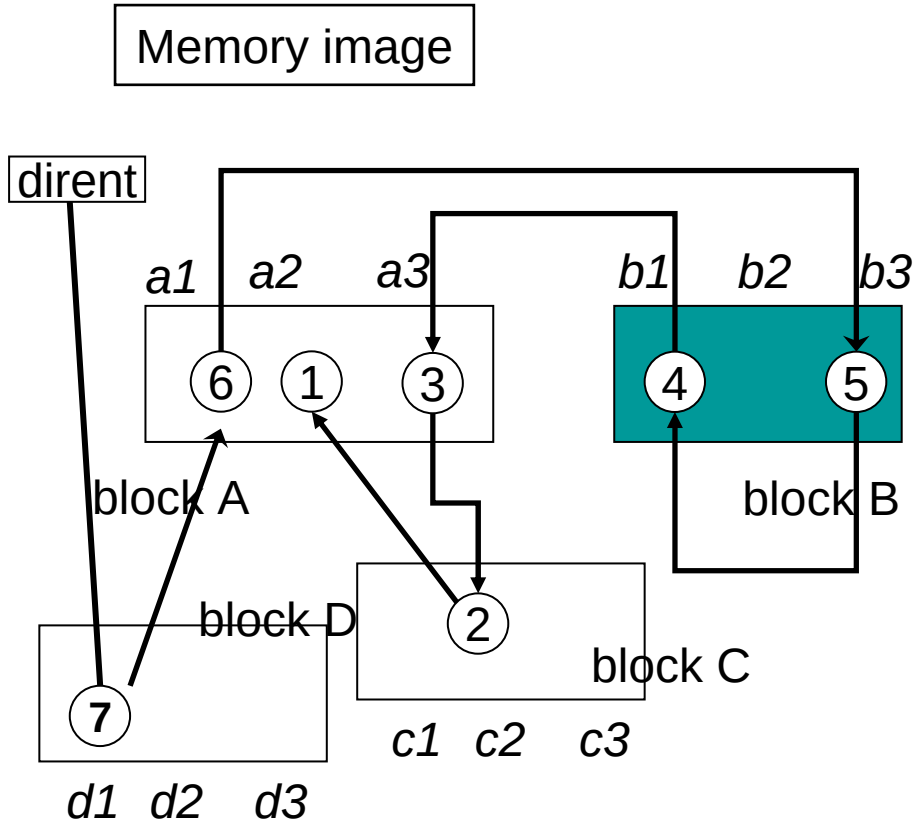
to: * dirent – d1 – a1 – b3 – b1 – a3 – c2 – a2

Order of modification on mem(old val)

- block A[FRE EOF(FRE) FRE]
- block C[??? a2(FRE) ???]
- block A[FRE EOF c2(FRE)]
- block B[a3(FRE) ??? b1(FRE)]
- block A[b3(FRE) EOF c2]
- block D[a1(EOF) ??? ???]

- Multiple modifications on same sector can be handled at one time
- On rollback, fat entry will be set to EOF. Not need to record whole modification history as transaction.

Alloc. table submission – Example expansion case #1 (block B submission)



block B is being written;

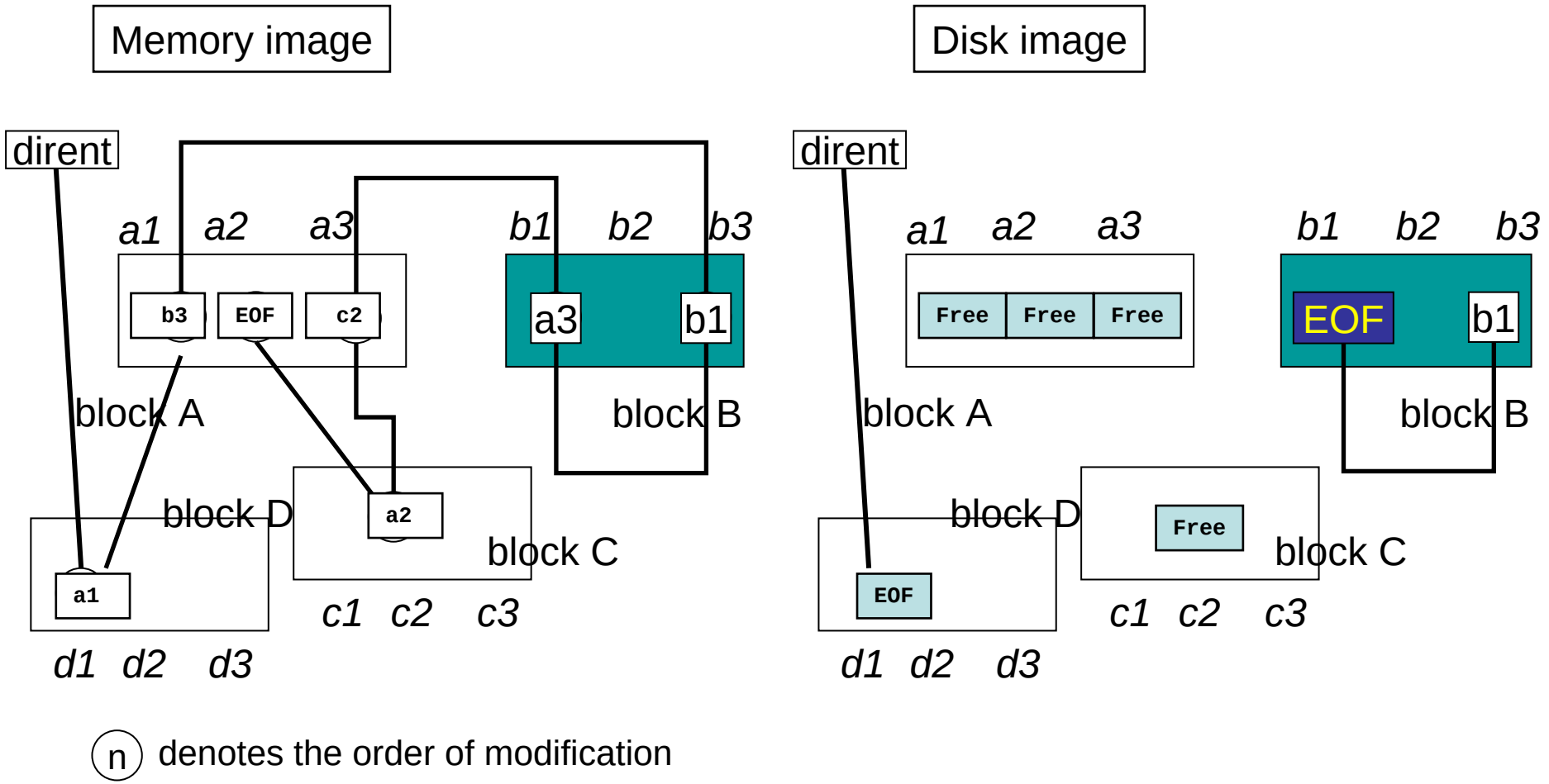
The entry ***b1***(content is a3) is depend on block A.

If block A is dirty, block B needs rollback.

Assume no sect is clean (not dirty)

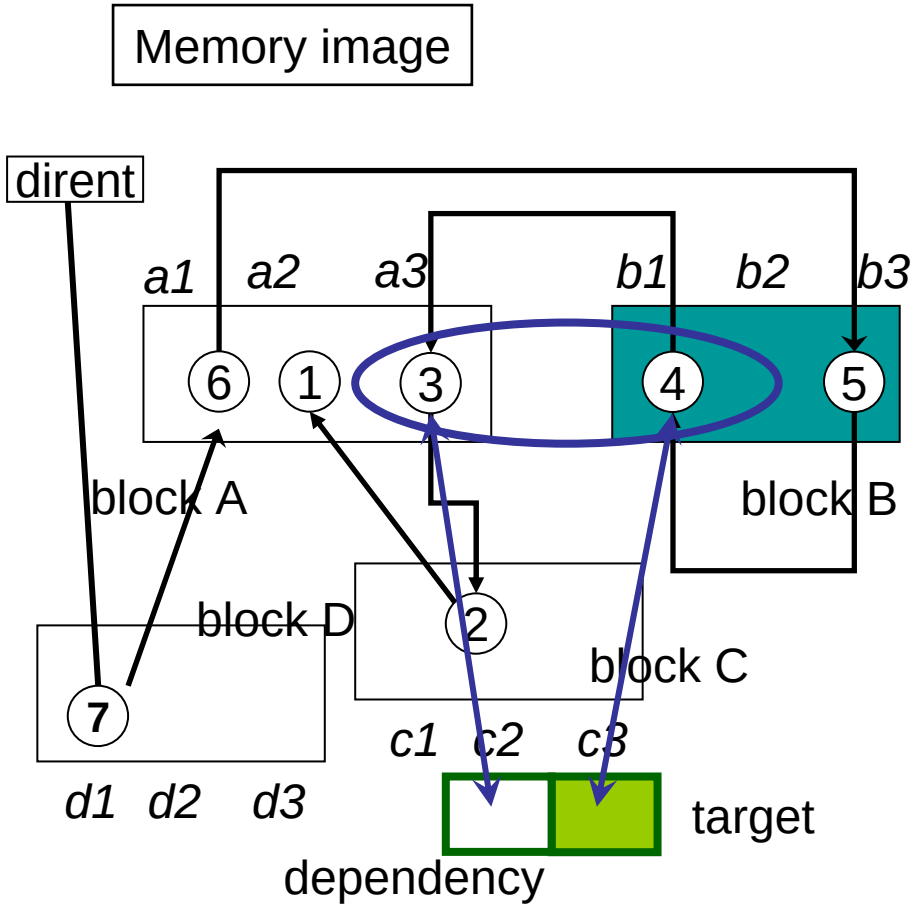
- Rollback before I/O submission
  - `block B[a3 ??? b1]`
    `->`
  - `block B[EOF ??? b1]`
- I/O submission;
  - `block B[EOF ??? b1]`
- Roll forward on callback of I/O completion
  - `block B[EOF ??? b1]`
    `->`
  - `block B[a3 ??? b1]`
  - mark block B dirty again

Alloc. table submission – Example expansion case #1 (Disc Image)



(n) denotes the order of modification

Basic design

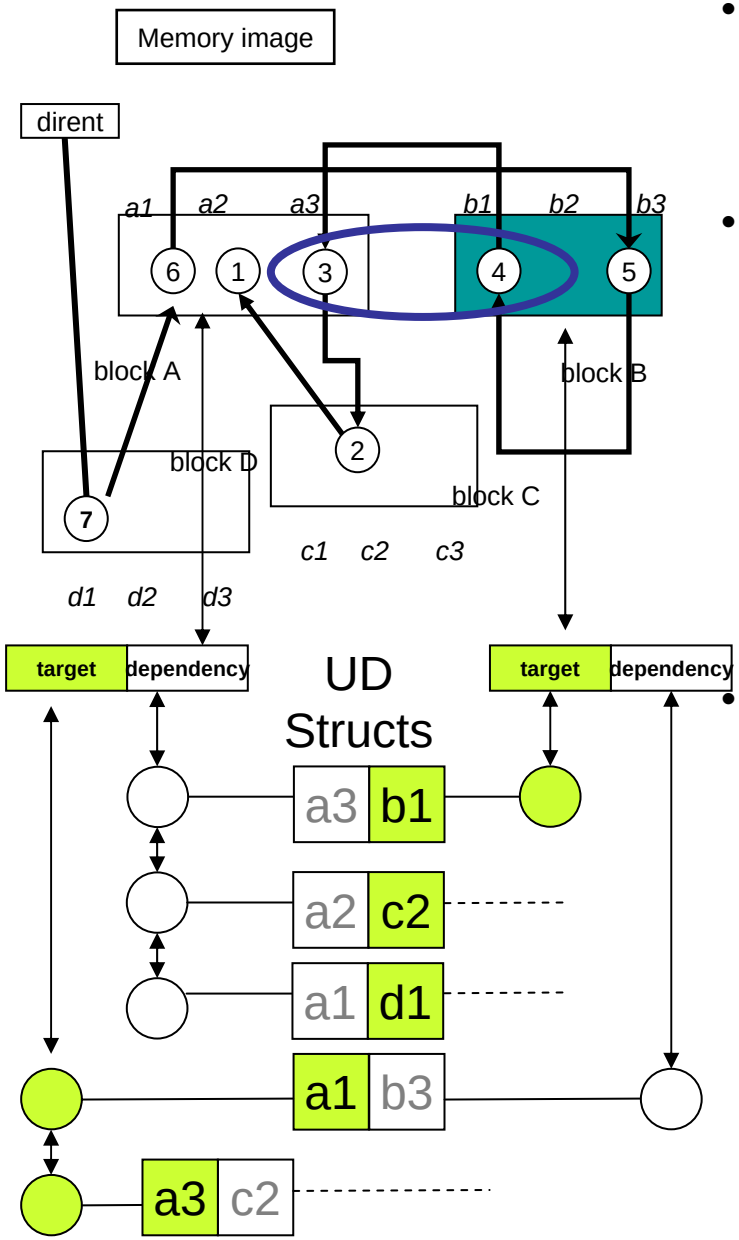


Unresolved dependency: b1 depends on a3

- Basic rule
  - On writing block B,
    - If *target*(a3) is dirty (not yet written to disk), then
      - Roll back entry of *dependency* in block B
      - Write block B
      - Roll forward it
    - Else (clean)
      - Write block B
- How express “unresolved dependency”(UD) and maintain it
  - Dirty
    - On modifying FAT entry on mem, allocate corresponding UD structure.
    - After writing FAT entry to disk. Release corresponding UD structure.
  - Clean
    - There is no data instance

X depends on Y;
call X as “*target*” Y as “*dependency*” just like in Makefile

Algorithm

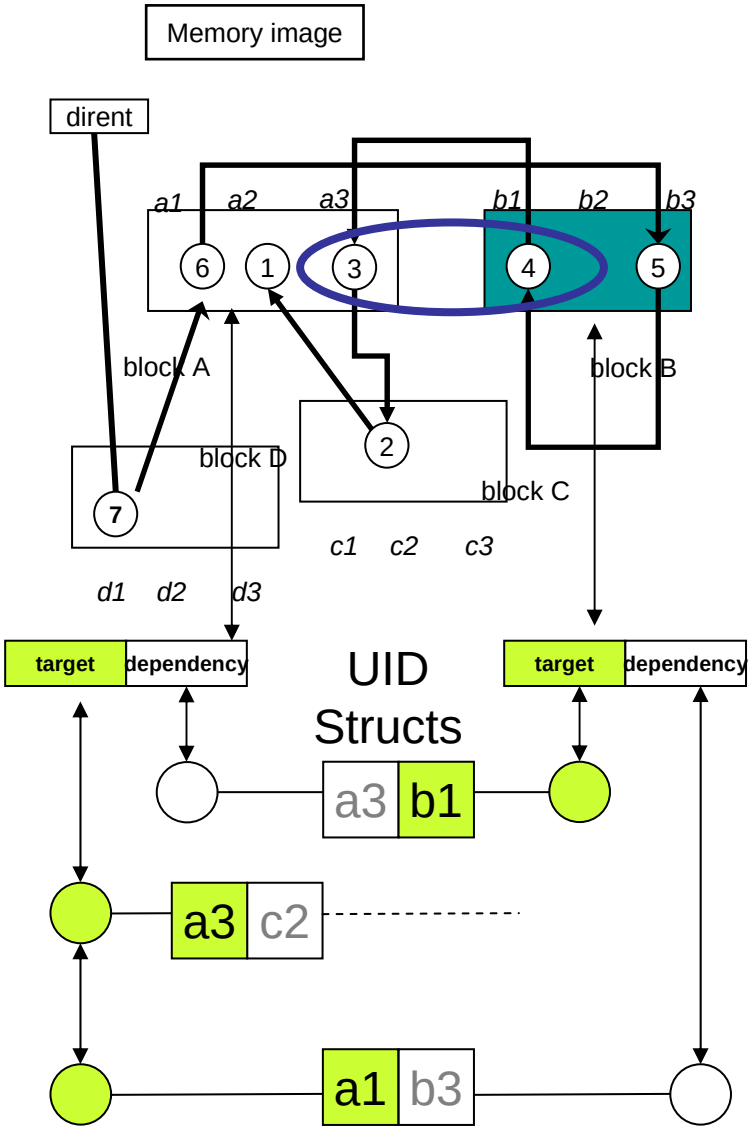


- On FAT manipulation between different blocks, add an UD “unresolved dependency” structure
  - Alloc “unresolved dependency” if different blocks
    - *target* = b1
    - *dependency* = a3
- On Submit BH
  - acquire mutex lock
  - For each UD struct where the target of submitting BH is *target*
    - // Assert (*target* block is dirty)
    - save = *(*target*)
      - Save = *(b1)
      - Save = a3
    - * (*dependency*) = FOF // roll back
      - *b1 = EOF
  - release mutex lock
  - continue conventional submitting BH

After BH I/O completion

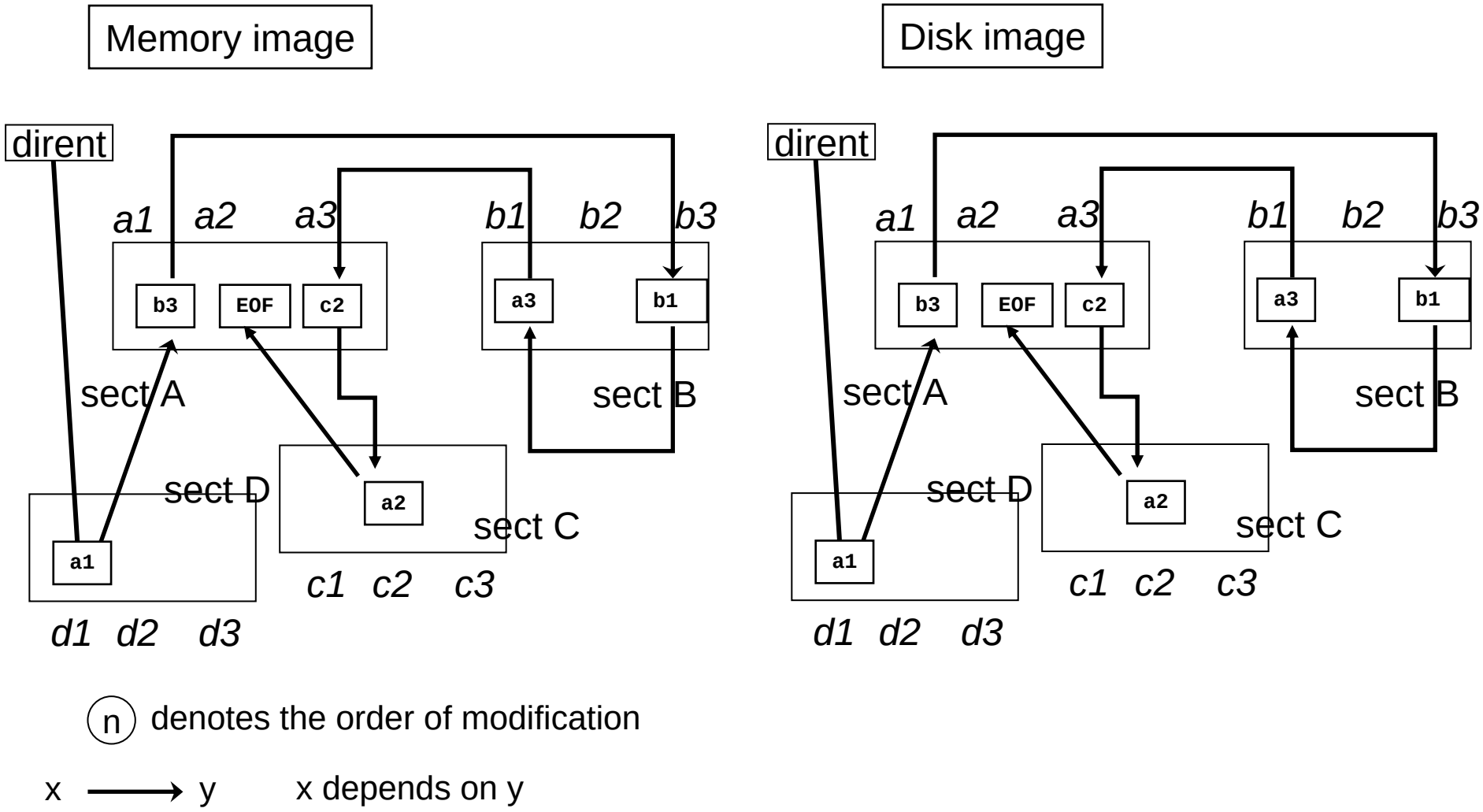
- acquire mutex lock
- For each UD struct where the target of BH I/O is *target*
  - * (*target*) = save // roll forward
    - *b1 = a3
  - Mark dirty again
    - Mark block B dirty
- For each UD struct where the target of BH I/O is *dependency*
  - Release the struct
    - Remove UID struct stands for [a1]->[b3] dependency
- release mutex lock
- continue conventional BH I/O completion

Impetration issues



- How implement to UD
  - Do not extend Buffer_header
  - b_private in BH could be used
    - BH_* Bit fields for private reserved also may help
    - own modified end_io would be used
    - E.g. jbd replace b_end_io with own method
  - Need to proper operation to bh refconut
  - Data structure for *target* of UD
    - offset inside in the block
      - block size can be looked up by super block
    - Save area for rollback (max 32bit)
  - Data structure for *dependency* of UID
    - *BH (pointer of Buffer Head)
- Efficiency
  - resolution of multe lock
  - Is a good enough that *dependency* and *target* liked to block through by bi-directional liner link
- Others
  - Can we control order of output with HW sector, not block with FS layer?
    - No maybe.
    - FS handles data with block size.

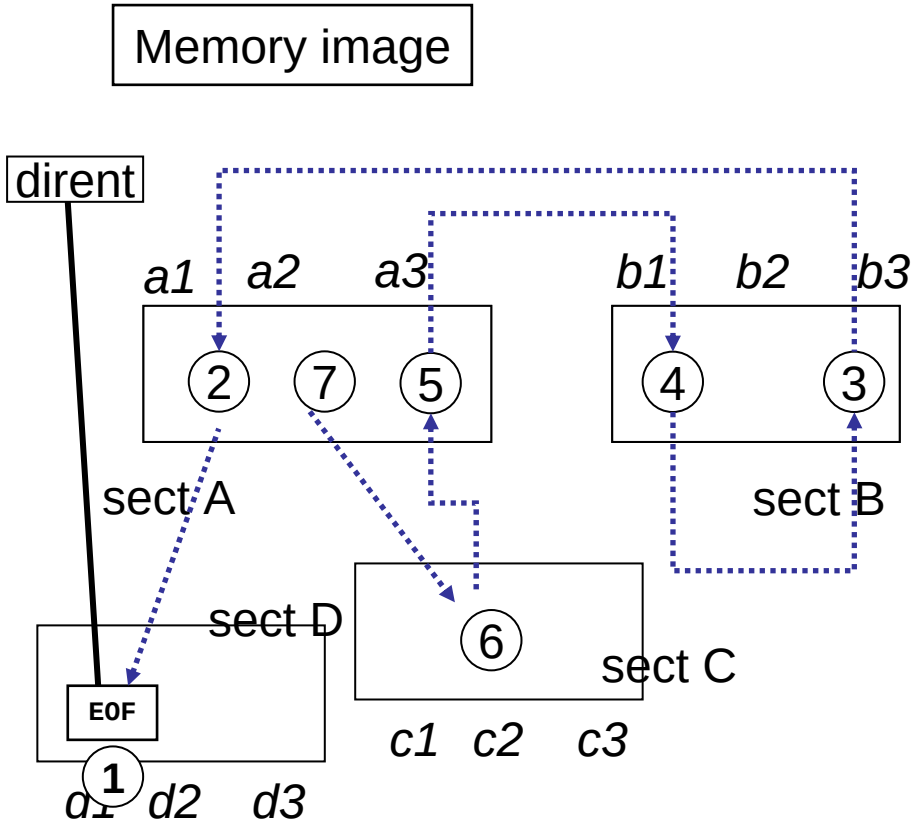
Alloc. table submission – Example truncate case #1 (before modification)



(n) denotes the order of modification

x ⟶ y x depends on y

Alloc. table submission – Example truncate case #1 (mem image)



n denotes the order of modification

x ⟶ y x depends on y

shurink cluster chain;

from:* dirent – d1 – a1 – b3 – b1 – a3 – c2 – a2

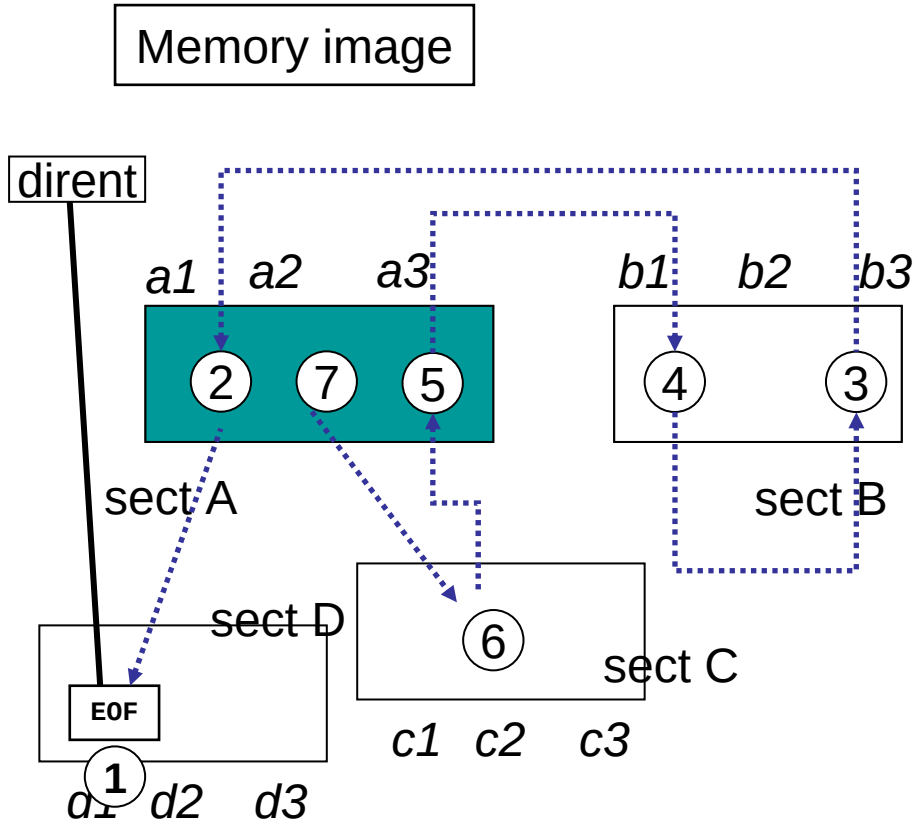
to: * dirent – d1

On mem modification `(old val)`

- `sect D[EOF(a1) ??? ???]`
- `sect A[FRE(b3) EOF c2]`
- `sect B[FRE(a3) ??? FRE(b1)]`
- `sect A[FRE EOF FRE(c2)]`
- `sect C[??? FRE(a2) ???]`
- `sect A[FRE FRE(EOF) FRE]`

- Multiple modifications on same sector can be handled at one time
- On rollback, fat entry will be set to EOF. Not need to record whole modification history as transaction.

Alloc. table submission – Example truncate case #1 (block A submission)



block A is being written;

The entry a1, a2 and a3 are depend on block D, C and B respectively.

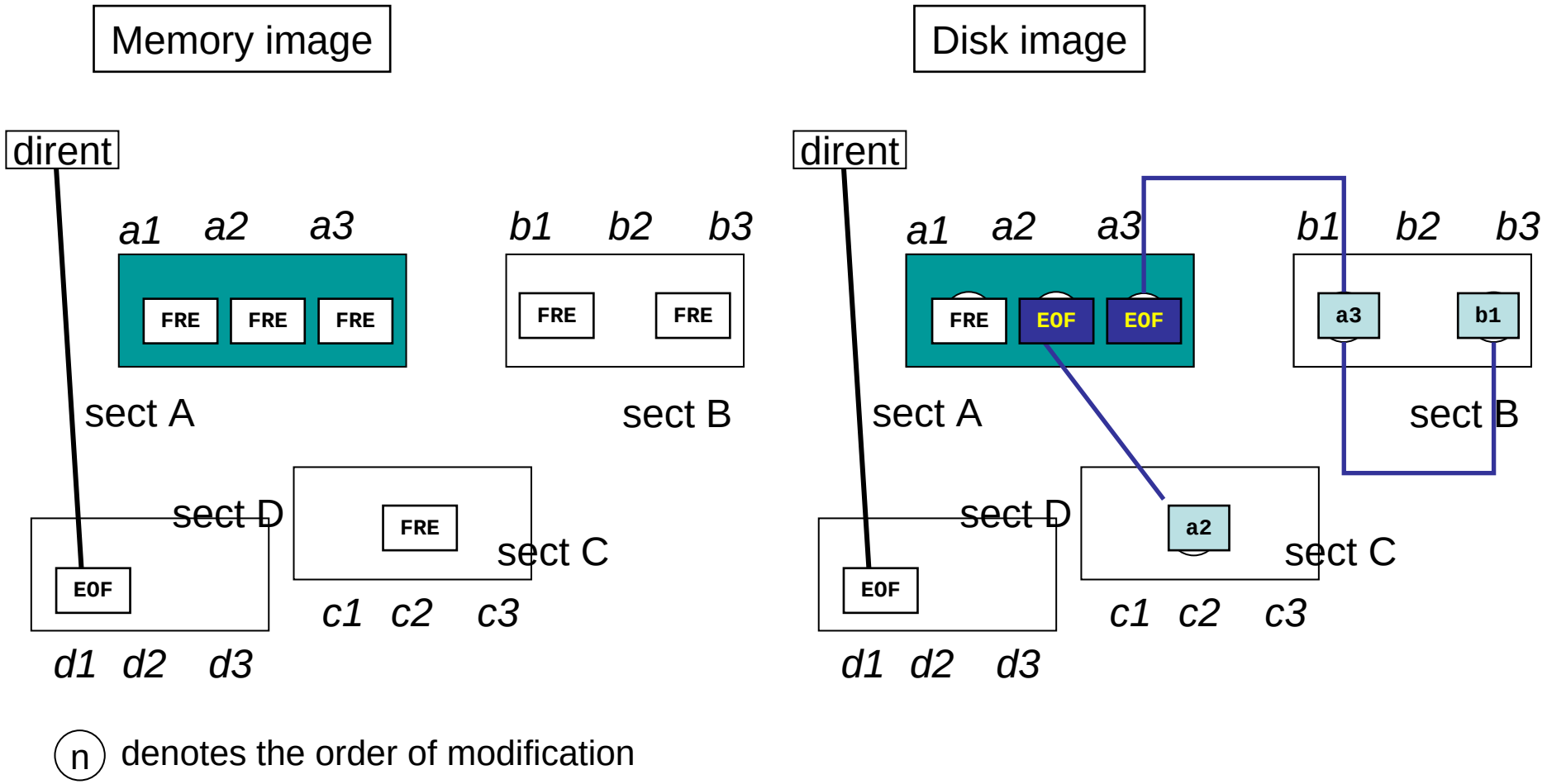
If block D is dirty, rollback a1 is needed.

Same for (C, a2) and (B, a3).

Assumed block D is clean (not dirty)

- Rollback before I/O submission
  - `blockA[FRE FRE FRE]`
    `->`
  - `BlockA[FRE EOF EOF]`
- I/O submission;
  - `blockA[FRE EOF EOF]`
- Roll forward on callback of I/O completion
  - `blockA[FRE EOF EOF]`
    `->`
  - `blockA[FRE FRE FRE]`
  - mark block A dirty again

Alloc. table submission – Example truncate case #1 (Disk image)



(n) denotes the order of modification

Dirent operations – size (1)

- Add new data on a file
  - Alloc and add cluster chain with the above manner
  - Write data body(s) on memory as async way (same as nomarl write())
  - Update size on Dirent
    - On update Dirent
      - allocate size-dependency structure
        - » original size value
        - » * inode
    - On submitting Dirent,
      - // If all cluster chain entries and data body is not yet written out
      - For each size-dependency struct, traverse cluster chain,
      - for each cluster entry
        - » check there’re no UID where *dependency* is same as cluster entry
        - » and
        - » check dirty flag of corresponding data blocks are clean
        - » Roll back size field
        - » Write
        - » Roll forward size field
      - Else
        - » write
        - » release size-dependency structure

Dirent operations – size (2)

- Truncate file to shrink down
  - Update size on Dirent
    - Issue dirent I/O, just after after updating size field
  - Shrink cluster chain