FAT improvment

Nov. 25th, 2005.

machida AT sm.sony.co.jp
mokuno AT sm.sony.co.jp
notanaka AT sm.sony.co.jp
sho AT axe-inc.co.jp
takawata AT axe-inc.co.jp

Remind: Issues around FAT with CE devices -1

- Hot unplug issues
  - File System corruption on unplug media/storage device
    - Almost same situation as power down without umount
  - Notification of the event to user space
    - Need to investigate, more
    - Application need to know what's happened precisely
      - How many un-plug and plug media/device evets occur
  - System stability after unplug
    - Almost same as I/O error recovery issues discussed on LKLM
      - http://developer.osdl.jp/projects/doubt/fs-consistency-and-coherency/index.html
      - http://groups.google.co.jp/group/linux.kernel/browse_thread/thread/b9c11bccd59e0513/4a4dd84b411c6d32?q=[RFD]+FS+behavior+(I%2FO+failure)+in+kernel+summit++lkml&rnum=1&hl=ja#4a4dd84b411c6d32
    - Need to select behavior of FS after unplug
      - All operations except umount() will report correctly error.
      - currently just makes FS read-only.
    - FS needs to survive even mounted block device disappeared
      - With some USB storage, block device is dismissed on un-plug

Remind: Issues around FAT with CE devices -2

- Other issues
  - Time stamp issues
    - local time, 2sec unit
  - Issues around mapping with UNICODE and local char code
    - N-1 mapping with SJIS(ShortNmae) <-> UNICODE (LFN)
    - Possible inconsistency between kernel and application side
    - interoperability with PC – OK with 2.6.x (at least 12)
  - Support file size over 2GB – OK with 2.6.x (at least 12)
  - FAT32 FS dirty flag

Discussions at previous meetings

- Why FAT? – It would be difficult to share back ground

Need continuous efforts to explain

- Journaling? – Use existing functions for robustness

Feedbacks from LKLM

Need to test FAT SYNC mount introduced at 2.6.12

Need to consider “Soft Update”

- Underlying layers – Elevator and Block device driver, like flash ROM, USB Mass and HDD

Feedbacks from LKLM

Need to consider BH_ordered is introduce.

Need to consider to isolate File system layer from underlying layers

FAT related works – Current state

- Began work at 8/M

- Misc Improvments
- Robustness with sync option
- Other FAT robustness
  - Avoid sector unaligned entry on FAT12 cluster allocation

- Not planed yet
  - Better handling underlying device, like Flash ROM
  - Notification of the event to user space
  - System stability and FS behavior after unplug
  - Possible char code problem
  - FAT32 dirty flag
- No plan to address
  - File Size > 2GB – Already Supported

Misc Improvements

- dirscan speedup
  - fat/fat_lookup-hint_1.patch
- fat: Handle broken free_clusters on FAT32 collectly
  - fat/fat32-brkn_frclstrs.patch
- fat: POSIX attribute mapping support for VFAT.
  - fat/vfat-posix-attr.patch

Robustness with sync option

- fs: generic_osync_inode() with OSYNC_INODE only passed
  - fs-osync-inode-only.patch
- fat: sync attr rework with generic_osync_inode() change
  - fs-osync-attr.patch
- Sync on write - Already included in 2.6.14
  - fat/fat-sync-write_1.patch
- fat: truncate write ordering issue
  - fat/fat-truncate-order-with-posix-attr.patch
- fat: rename write ordering issue
  - fat/vfat-avoid-double-link.patch

Soft Update

Soft Update

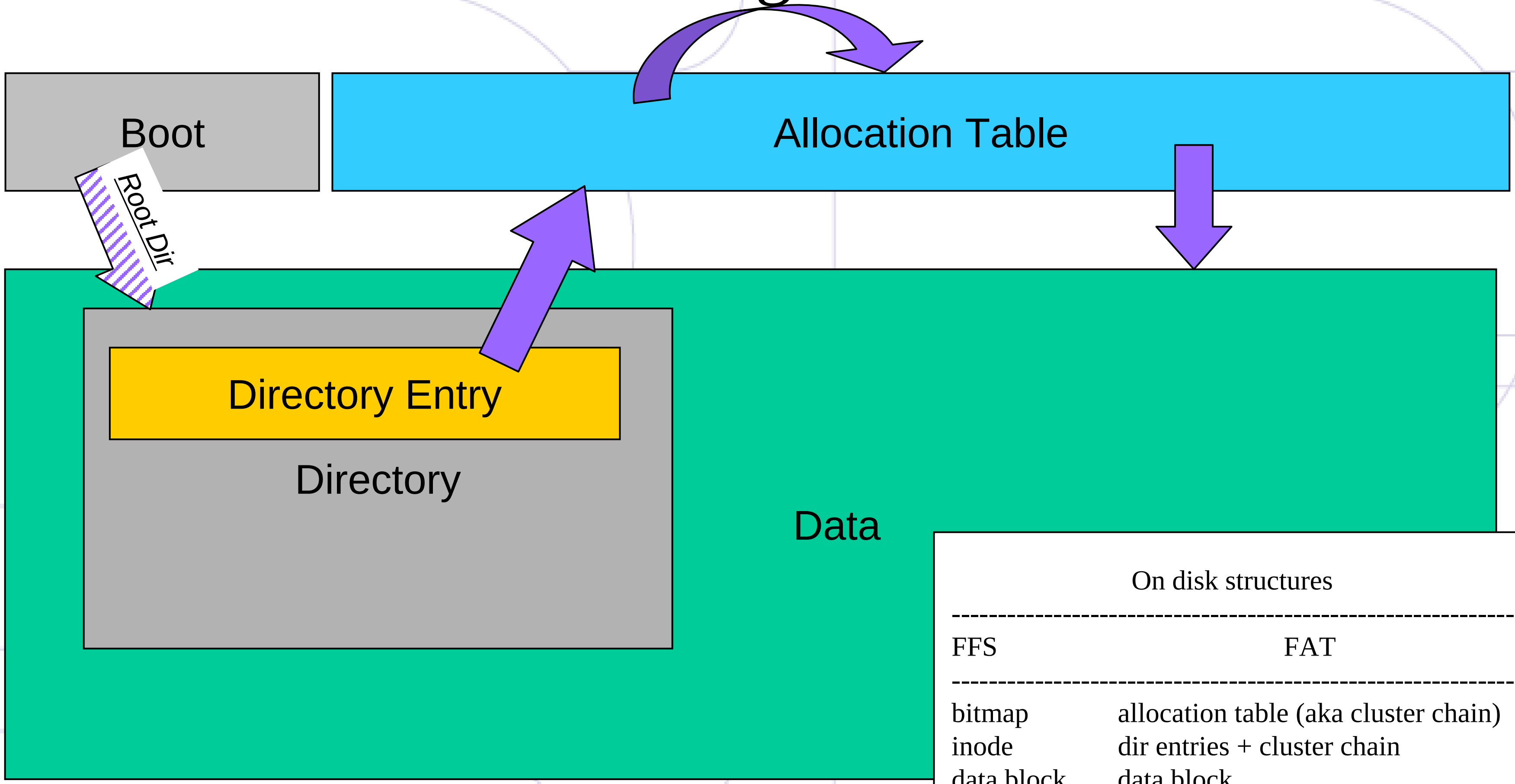
- Use write-back cache for metadata
  - async, not write-through
- Record updates with per structure relation basis, not block basis
  - avoid dependency circulation
  - Three flags introduced in * BSD

ATTACHED	metadata update started
DEP_COMPLETE	depdent metadata update complete
COMPLETE	complete data update complete

- On writing metadata, to keep metadata consistent
  1. roll back incomplete operations effect to the metadata
  2. write metadata to DISK
  3. roll forward incomplete operations effect to the metadata
  - This means both DISK and memory have consistent metadata, however on DISK we may have little older metadata.

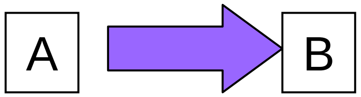
- See
  - M. K. McKusick & G. R. Ganger. “Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem.” *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, Jun 1999.

FAT FS Organization



On disk structures

FFS	FAT
bitmap	allocation table (aka cluster chain)
inode	dir entries + cluster chain
data block	data block



A effects to B(B is depend on A)

FFS operations

- Following have “Update Dependency”
  - file creation
  - file removal
  - directory creation
  - directory removal
  - file/directory rename
  - block allocation
  - indirect block manipulation
  - free map management

FAT FS operations

- Following operations possibly have “Update Dependency”, not considered yet
  - Append data to file (expand file)
  - Create file (expand dir)
  - Create dir (expand dir)
  - Remove file
  - Remove dir
  - Truncate file
  - Rename file/dir
  - Change attributes
  - Allocation table operations
    - Allocate
    - Release
    - Bind/ReBind/UnBind *)
  - writev *)

  *) I’m not sure we need to consider them separately

Apply Soft Update on FAT

- Considering one by one according with usage frequency, not whole at once.

- As 1st step, Moving “write operation” to Soft Update

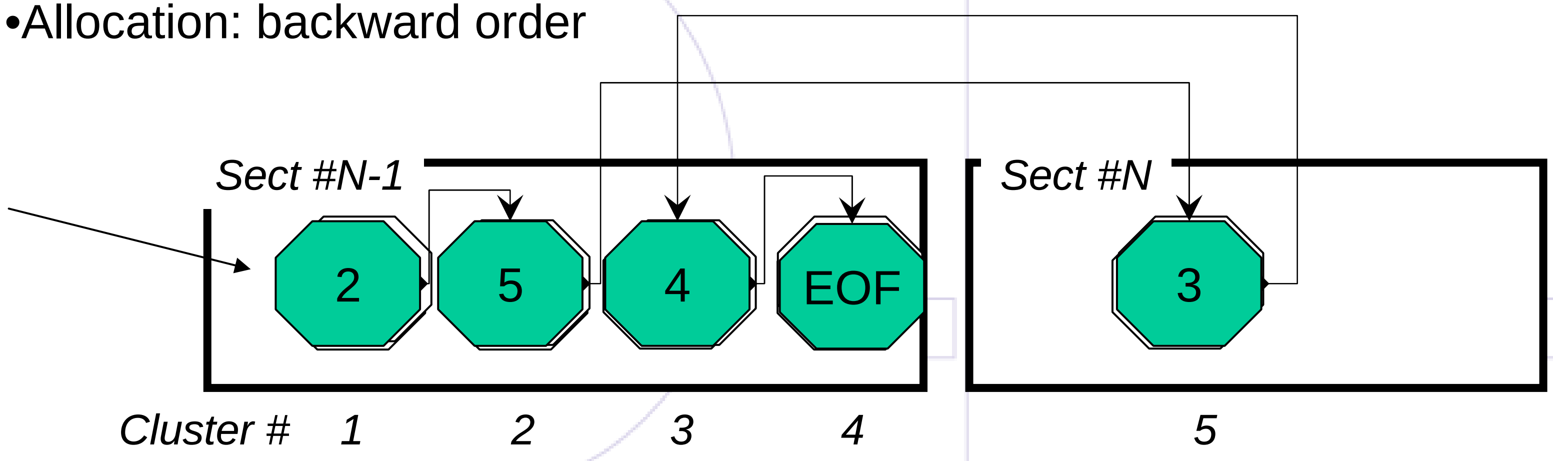
Allocation table operations

- allocate
  - Need backward order from tail to head, the entry which is pointed to, need to be updated, before the entry which is pointing to it.
    - need to update entries on allocation table, with backward order from tail to head.
    - need to update cluster chain field it the corresponding dirent, after cluster chain allocated, when the first data cluster about to be allocated.
  - Need to update size field in the corresponding dirent, after data written.

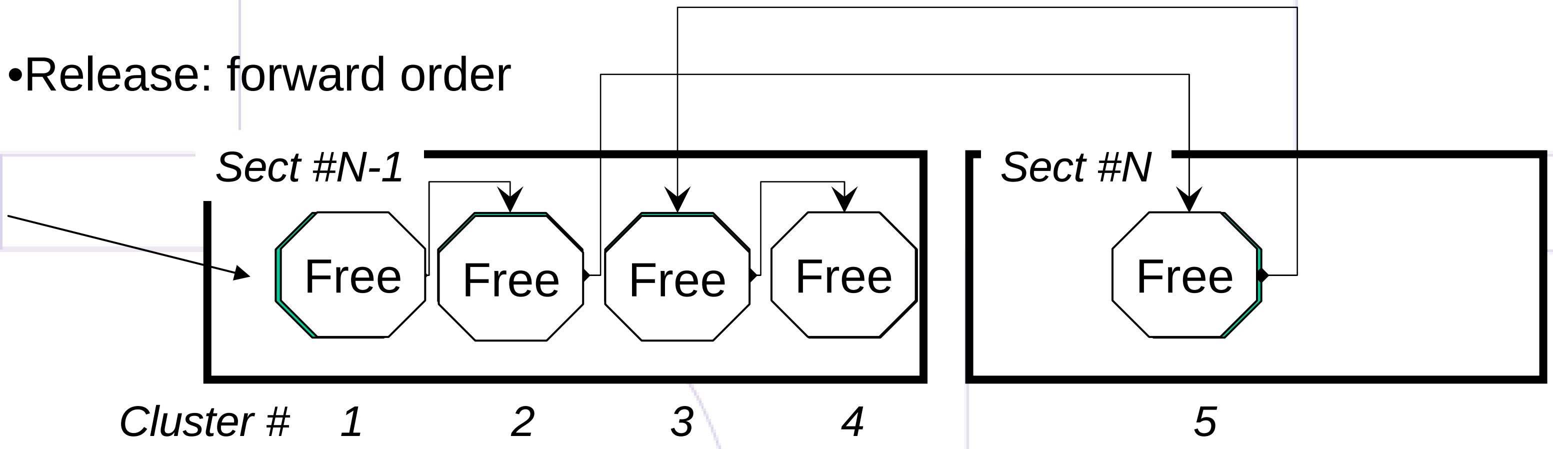
- release
  - Need to update size field in the corresponding dirent, first.
  - able to free and terminate entries on allocation table, with either forward or backward order, including cluster chain field it the corresponding dirent.
  - Prefer forward order from head to tail, the entry which is pointing to , need to be updated, before the entry which is pointed from it.
    - need to update cluster chain field it the corresponding dirent, before cluster chain freed, when the first data cluster about to be freed.
    - need to update entries on allocation table, with forward order from head to tail.
  - c.f. with backward order
    - can avoid cluster chain island while releasing
    - need to update twice than with forward order

Cluster Allocation and Release

•Allocation: backward order



•Release: forward order



Write Method (current)

- Current Implementation
  - Allocate new cluster and add cluster chain
    - link the cluster chain to the dirent, if needed.
  - write data
  - set new size
  - update mtime/ctime
  - set ATTR_ARCH flag

```
- write call tree
sys_write()
            do_sync_write ()
                        fat_file_aio_write ()
                                   generic_file_aio_write()
                                              __generic_file_aio_write_nolock()
                                                generic_file_buffered_write()

                                                  fat_prepare_write()
                                                    cont_prepare_write()
                                                      __block_prepare_write()
                                                        fat_get_block()
                                                          fat_add_cluster()
                                                            fat_alloc_clusters()
                                                            fat_chain_add()

                                                  fat_commit_write()
                                                    generic_commit_write()
                                                     __block_commit_write()
```

Write Method (SoftUpdate)

- Allocate an new cluster and add cluster chain
  - do updates on cluster chain and record them as ***(pos, old val)***
  - store and record pending link from the dirent to the cluster chain, if needed.
  - *mark buffer dirty*

Allocation Table I/O Submit

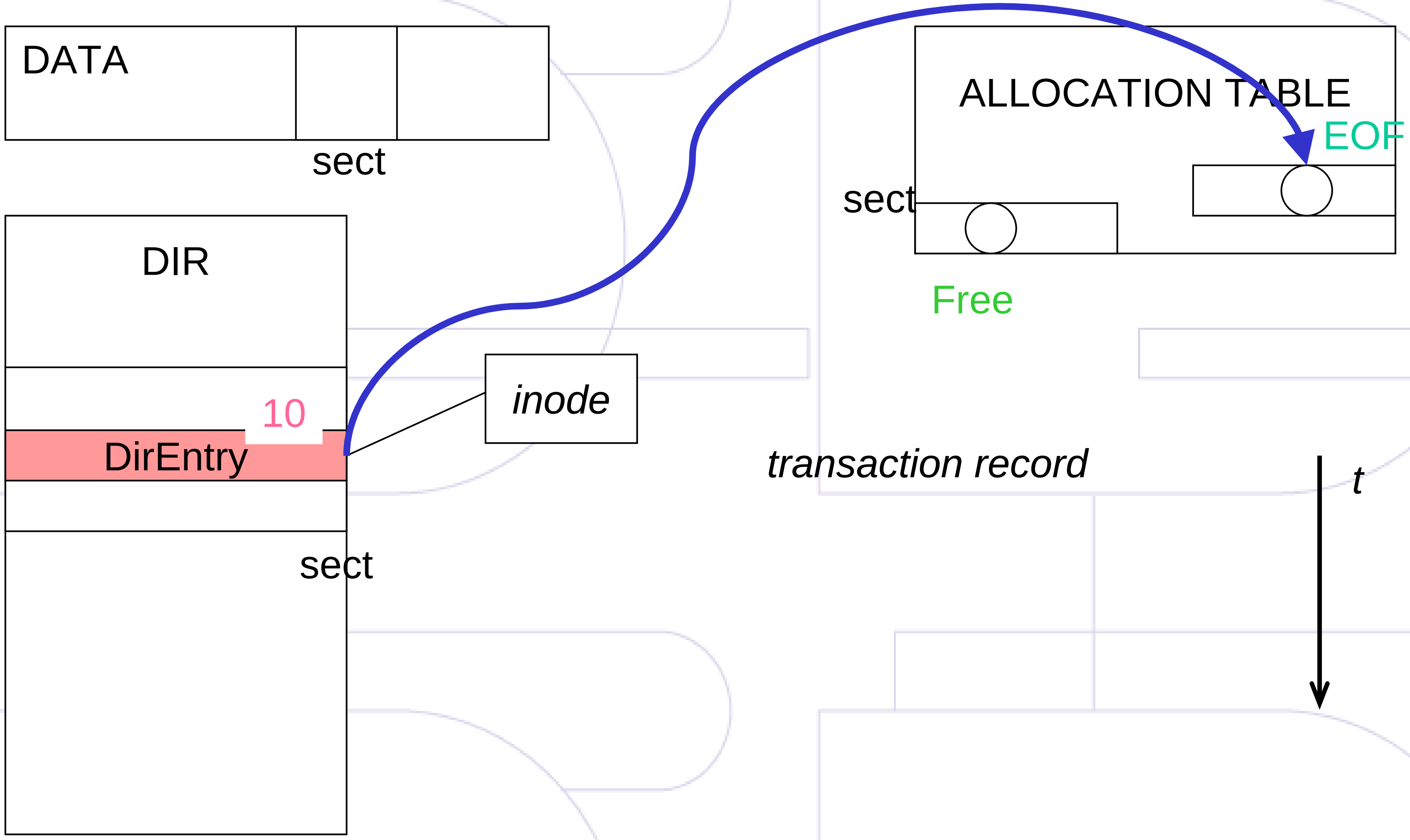
- write data
  - Write Data
  - *mark buffer dirty*

Data I/O Submit

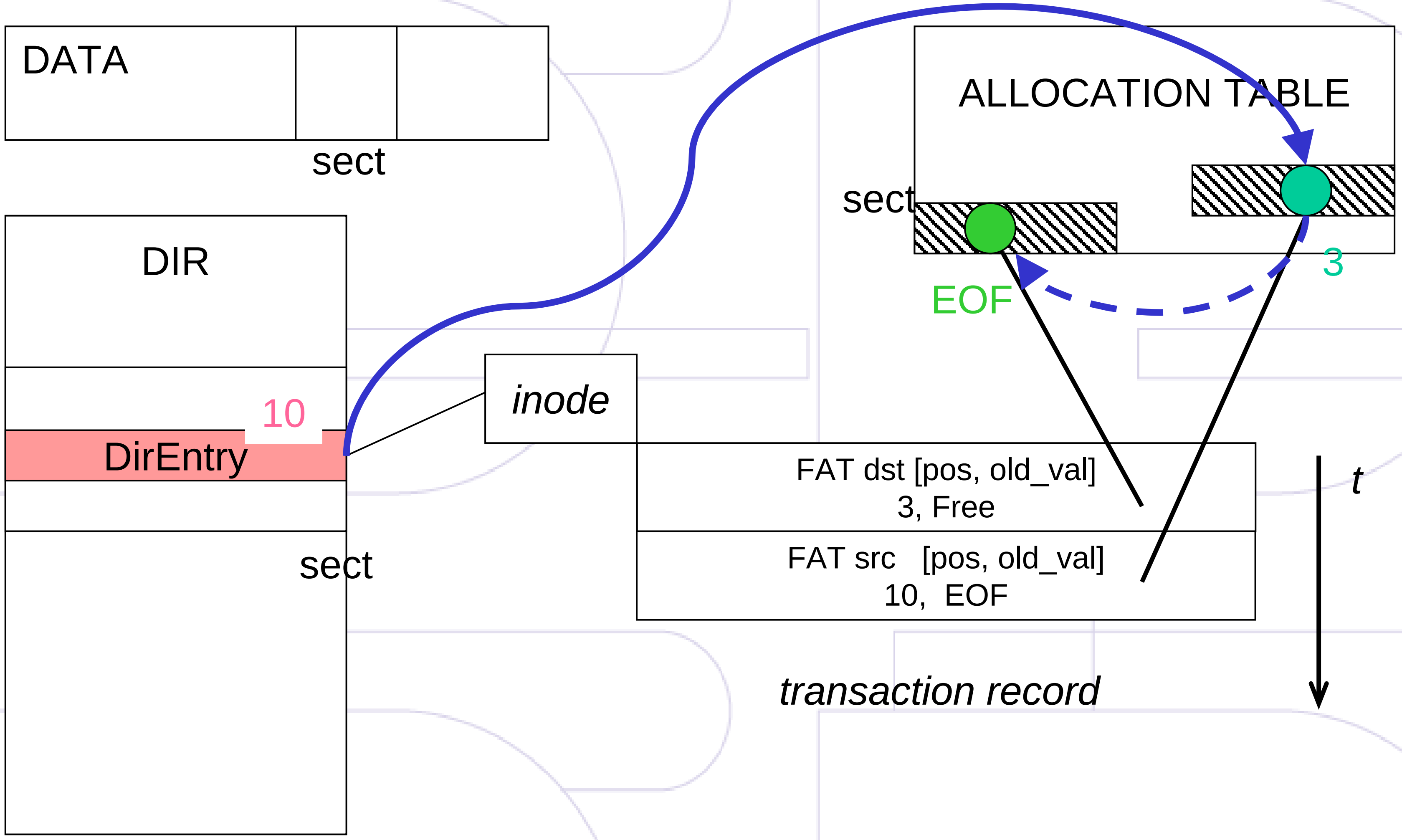
- set inode/dirent fields and write
  - link cluster chain to the dirent, if link is pending.
  - set new size
  - update mtime/ctime and set ATTR_ARCH, if needed
  - record those as ***(old link, old size, old attr, old time)***, because single dirent for short name holds them
  - *mark inode dirty*

DirEnt I/O Submit

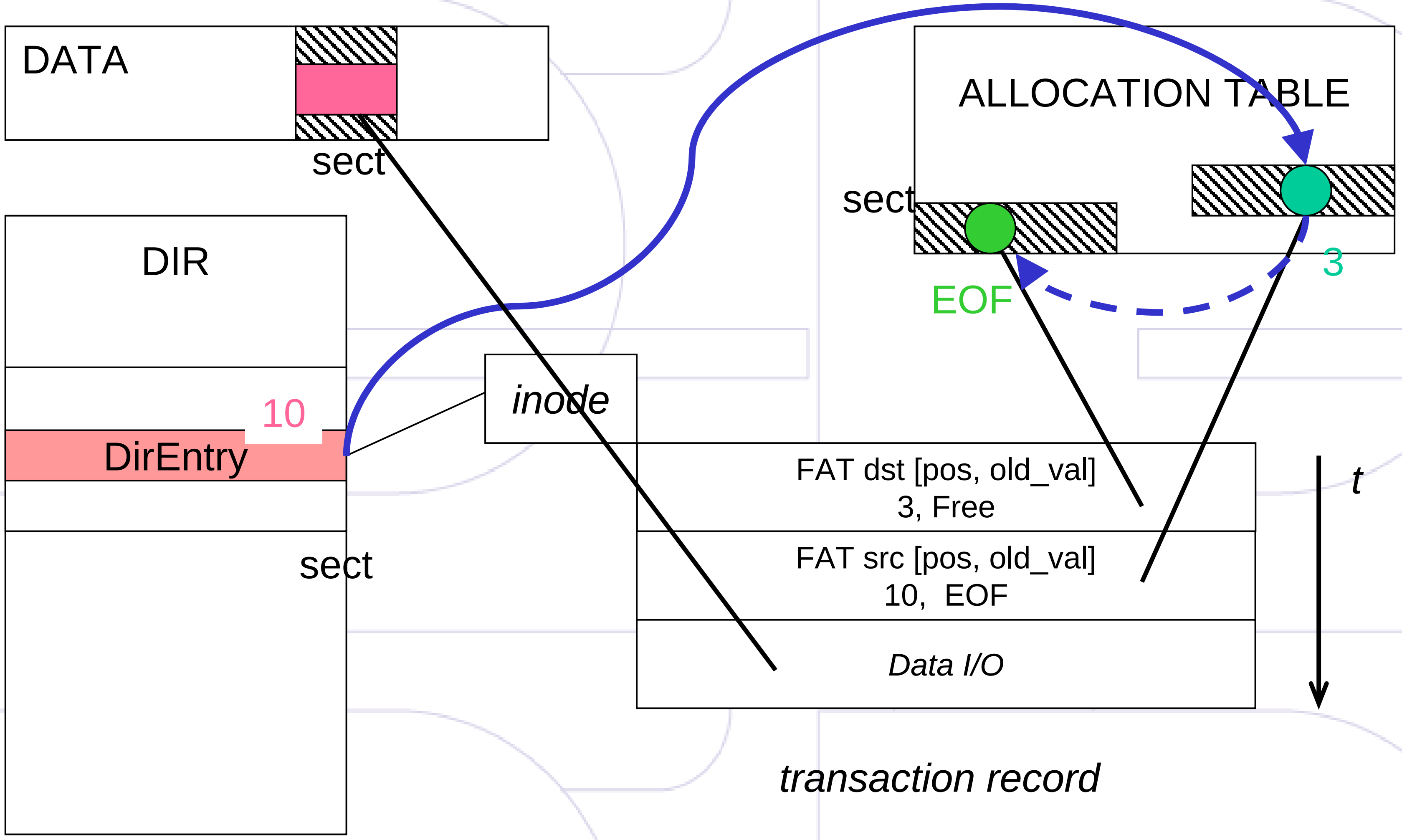
Transaction Record - 1



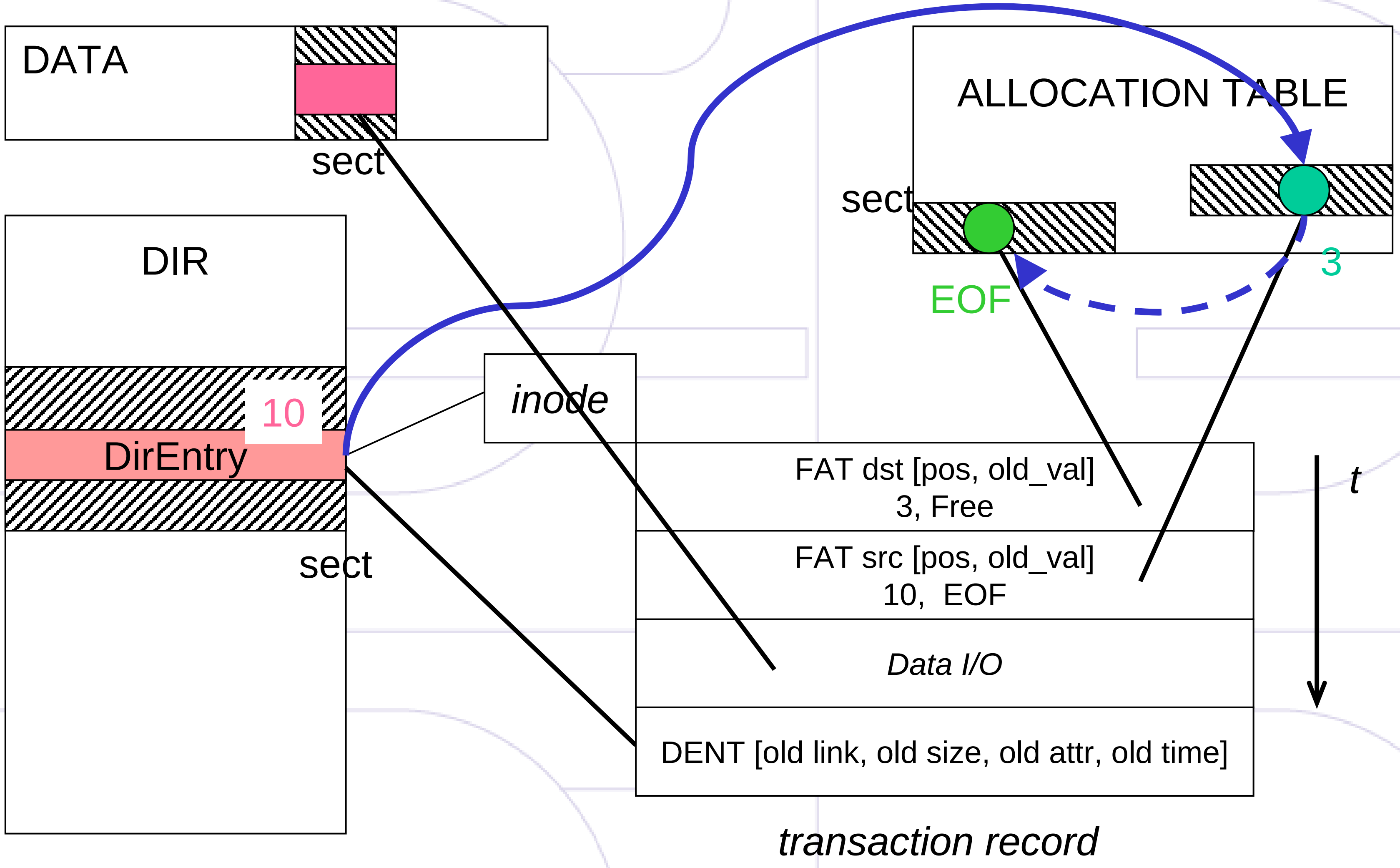
Transaction Record - 2



Transaction Record - 3



Transaction Record - 4

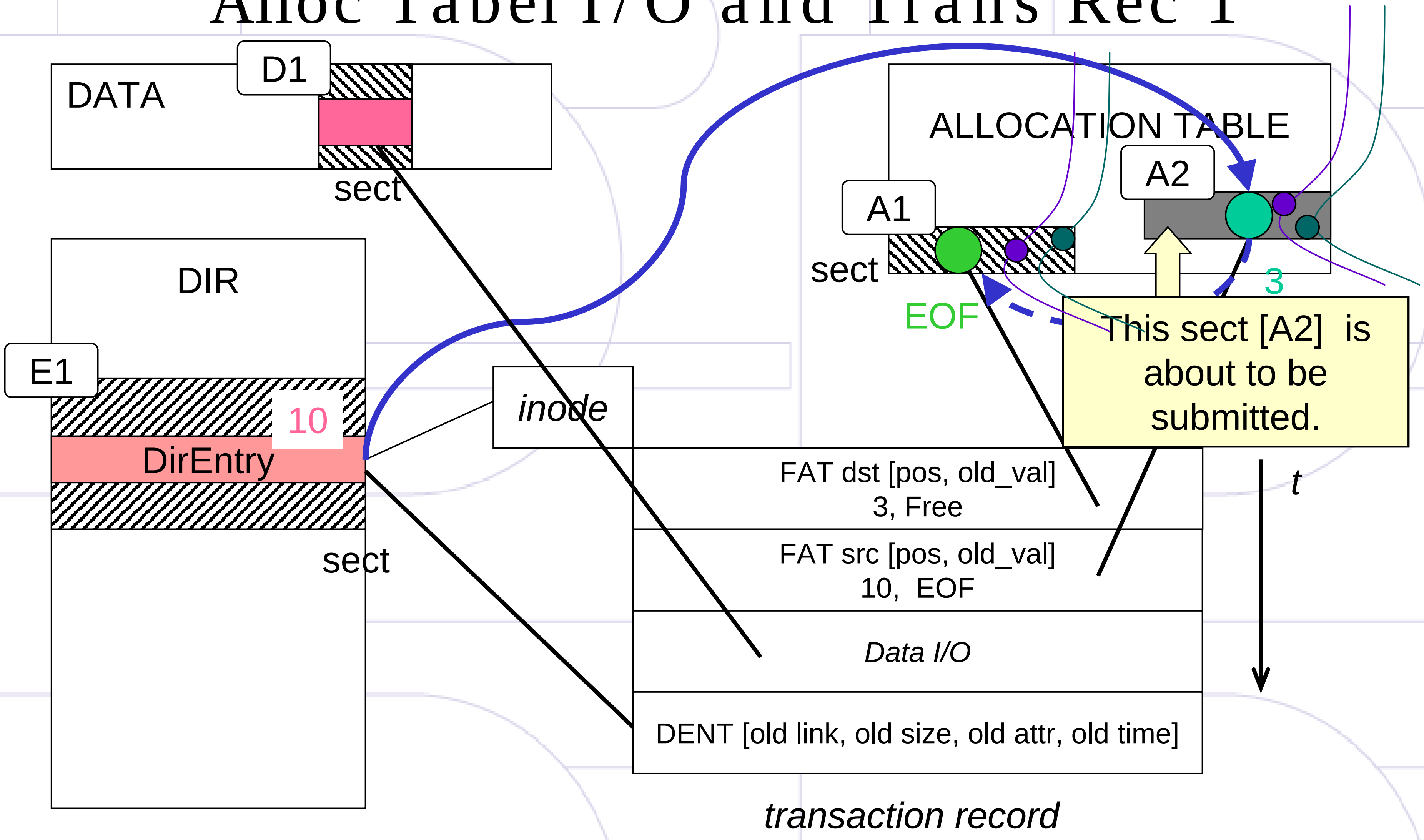


Submit Method (SoftUpdate)

- 基本原則：
  - すでに書ける状態?
    - submit BH
  - そうでないなら
    - 依存状態を解決
      - 必要なら関連するsectorを submit BH
    - submit BH

- 「すでに書ける状態？」
  - 依存する状態が既にDISC上にあれば書ける

Alloc Tabel I/O and Trans Rec 1



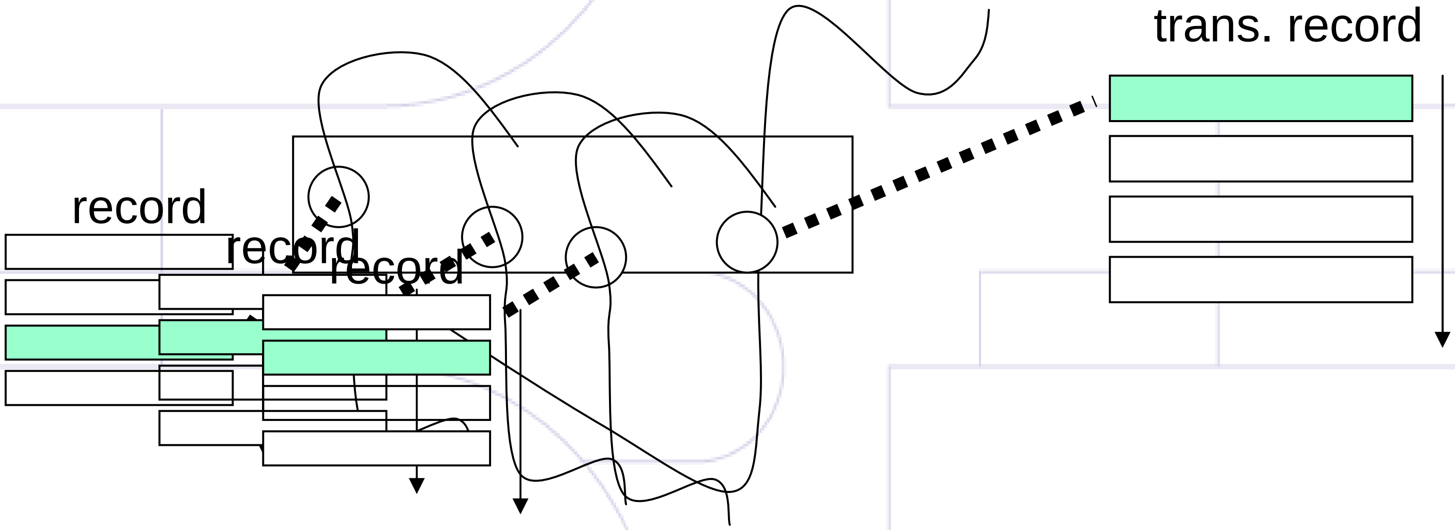
Alloc Tabel I/O and Trans Rec 2

- 状況
  - I/O 対象sectorがallocation table
  - The sect [A2] is about to be submitted.
  - No I/O submitted yet, regarding this inode.

- 依存する変更
  - [A2]の変更は[A1]の変更に依存
  - [A1]の変更を先に行い依存状態を解決する必要がある

Alloc. table submit method - 1

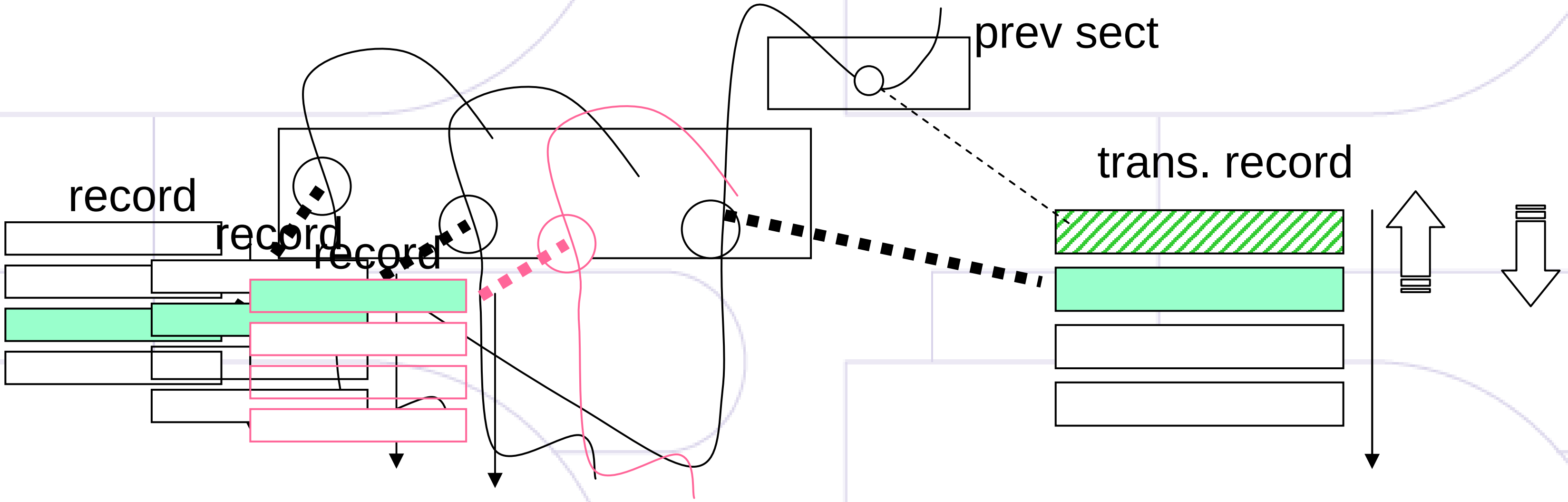
- 対象セクタから更新の古い順に1つのレコードに注目
  - 未解決な依存がない（このinode中で最古の未処理レコードに相当)なら、次に古いinodeに関するレコードへ



CE Linux Forum

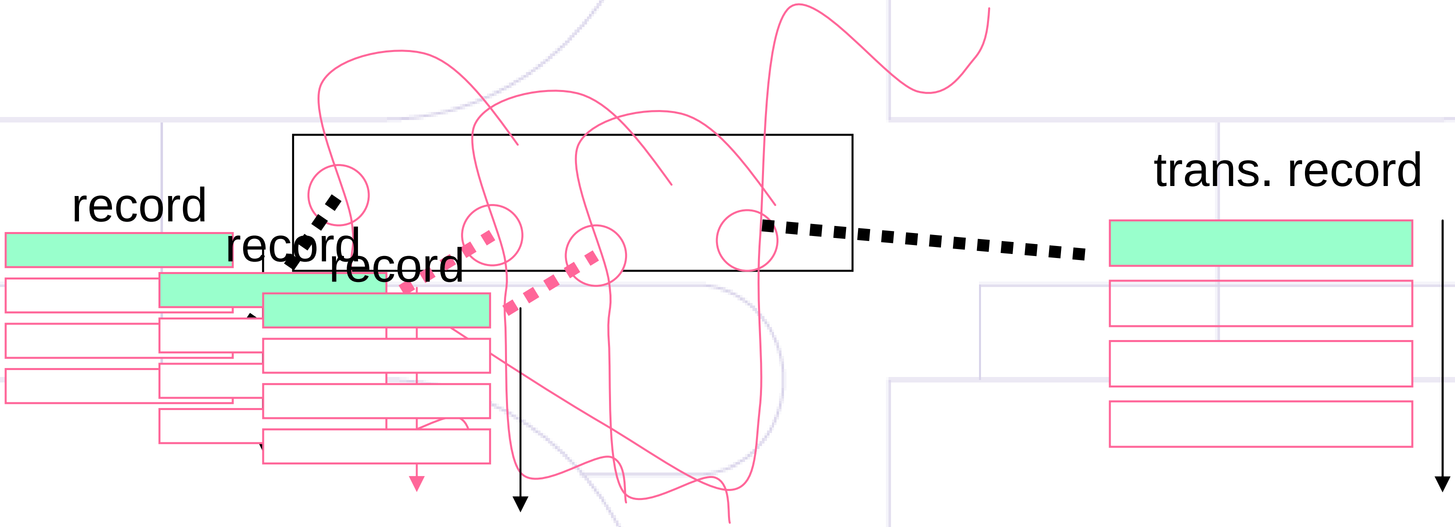
Alloc. table submit method - 2

- 対象セクタから古い順に1つのchainに注目
  - 未解決な依存がある（このchain中で最も古いレコードに相当しない)なら、
    - 該当レコードをroll back
    - 1つ前のレコードに関連するセクタの依存を解決
    - roll forward



Alloc. table submit method - 3

- 対象セクタ内のレコードを全て処理
- 依存が解決されているので、該当セクタをsubmit BH
- 各レコードに submit済みのマーク



Alloc. table submit method - 4

- 完了時
  - すべてのalloc table I/O完了かチェックしmark
  - syncerがいないなら、次の書き込みをスケジュール

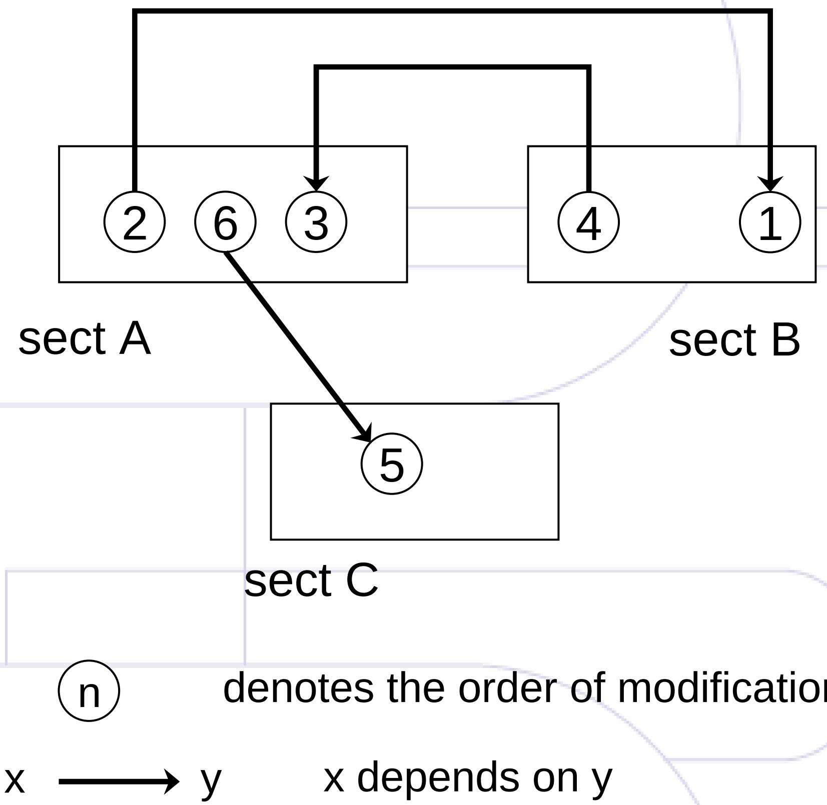
Alloc. table submit method Algorithm

```
proc(csect) {
  foreach crec in 古い順 allrec(csectを汚している) {
    if (is_marked(crec)) continue;
    mark(crec);
    if crecは最も古い continue;
    depend_sect = sector(prev(crec));
    if (depend_sect == csect) continue;
    if (csectは、既にsubmit済み) wait done;
    rollback(crec);       <--- 印だけつけて、本当に必要なときまで rollback
                          を遅らせる手もある。
    proc(depend_sect);
    if (csectは、既にsubmit済み) wait done;
    rollforward(crec);
  }
  if (csectは、既にsubmit済み) wait done;  <---- 必要ないかも
  submit_bh(csect);
}
```

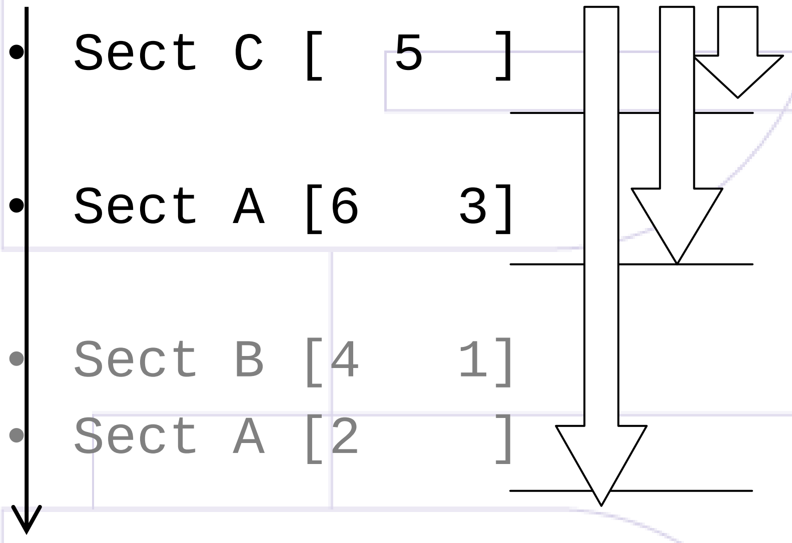
Alloc. table submit - Example

memory image

sect A is about to be written



- Disk I/O would be done as following order;



Q: When stop writing?

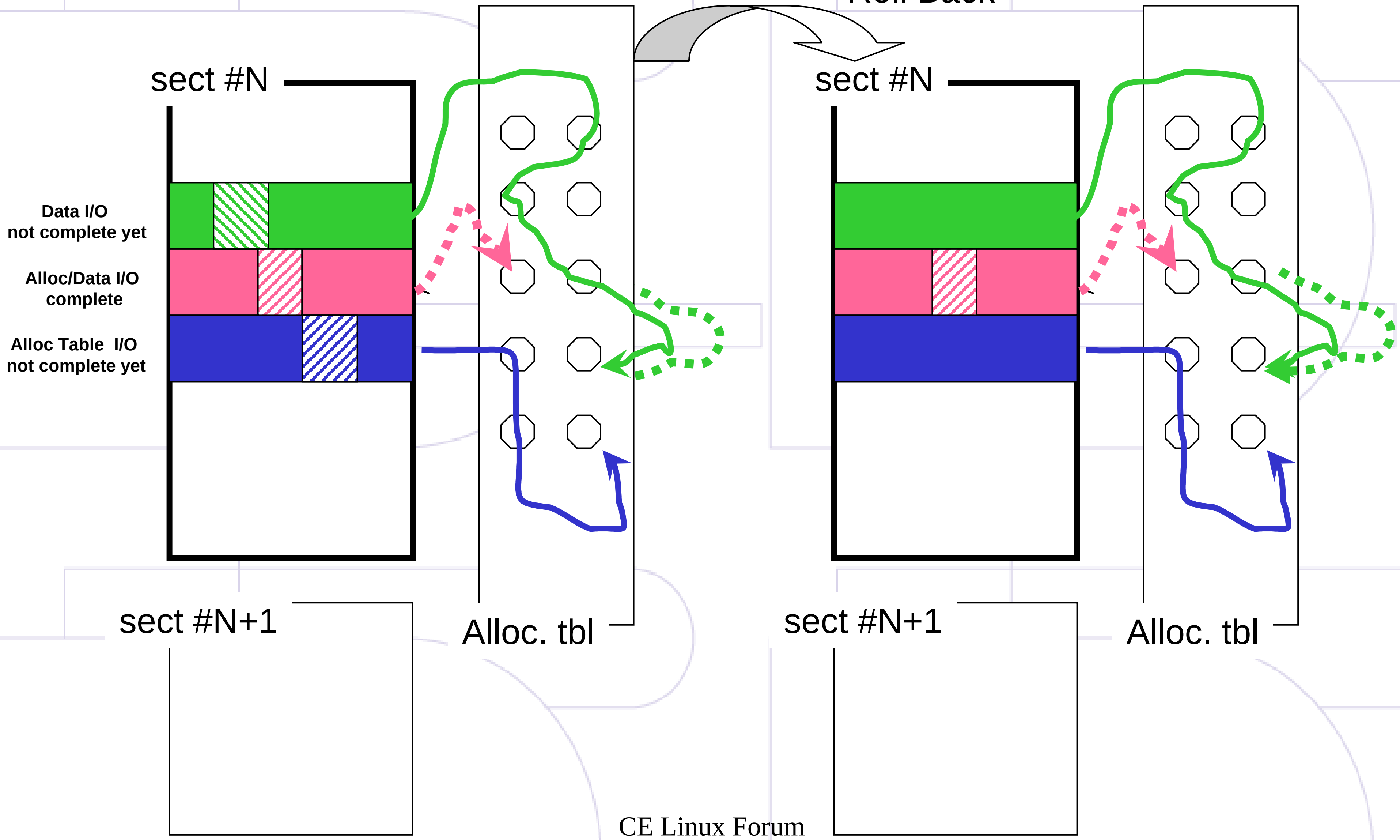
Data Submit Method

- Submit
  - 既に書ける状態（依存するAlloc Table I/Oが完了 or submit済み）なら
    - submit & submit 済のマーク
  - まだなら、
    - dirtyのまま放置
    - syncerがいないなら、被依存側処理をスケジュール
- 完了時
  - data I/O完了mark
  - syncerがいないなら、次の書き込みをスケジュール

DirEnt Submit Method

- Submit
  - すべてのdirentが、すでに書ける状態（Alloct TableとData I/Oが完了済み or submit済み）なら、
    - submit & submit 済のマーク
  - そうでないなら
    - まだかけないdirentに関する変更をrollback
    - まだ dirty なら submit 、 submit 済のマーク
    - rollbackした分をrollforward
- 完了時
  - TableとData I/Oが完了済みを確認 // debug
  - transaction record クリア

Rollbacks on dirent



疑問点1

- rollbackで各inode間の時間関係がずれるのでは？
  - そもそも依存関係がなければ、問題ないはず。
    - 書き込みー書き込みなら、依存は出ないはず。
    - 消去ー書き込みだと同じFATエントリを書きあうので依存ができる。
- rollbackでdirtyでなくなるのはどう判定できるか？
  - 未submitのレコードが残っているか否かで判定
- rollbackで本来書こうとしていたsector以外のsectorが書かれても問題ないか？
  - 余分にかかれるのは問題ない。
  - 本来の sectorもsubmitされないと、caller側の暗黙の仮定と会わなくなる可能性がある
    - sync しようとしているときには問題
- 本来書こうとしていたsectorとメモリとsyncする必要があるかは？
  - sync mount時には必要
  - そうでないなら必要ない

疑問点2

- rollbackでdiscとの齟齬が生じる可能性があるか？
  - ---[DISC]---------[MEM] -> t
  - * この中間点をsubmitしている
  - 依存が解決されているならOKなはず

- General: rollback で*mark_inode_dirty() and mark_buffer_dirty()*の作業を戻す必要があるか？
  - *Not yet considered*

- Actual cluster table I/O, data I/O and DirEnt I/O and other FS I/O submission would be done as page or buffer I/O like submit_bh() through page daemon. How can we identify the target FS, inode, type of I/O and it’s related data?

修正箇所

How page daemons write back files wo sync - 1

- pdflush (without sync)

```
– wb_kupdate()
–   writeback_inodes()
–     sync_sb_inodes()
–       __writeback_single_inode()
–         __sync_single_inode()
–           do_writepages()
–             generic_writepaes() // fat has writpage(), not writepages()
–               mpage_writepages() :fs/mpage.c
–                                  // getblk passed as NULL
–                 blkdev_writepage() for dir, fat_writepage() for file
–                   block_write_full_page()
–                     __ block_write_full_page()
–                       submit_bh()
–           write_inode()
–             fat_write_inode()
–               mark_buffer_dirty()
```

How page daemons write back files wo sync - 2

- wbc

sync_mode	WB_SYNC_NONE
nr_to_write	MAXWRITEBACK_PAGES 以下
nonblocking	1
for_kupdate	1

- __ block_write_full_page()
  - submit I/O by submit_bh(), if mapped and locked
  - do redirty if already locked by others
- fat_write_inode()
  - just do mark_buffer_dirty()
    - because sync_mode == WB_SYNC_NONE

How page daemons write back files wo sync - 3

```
* pdflush periodic writeout
rm /a/foo
EXIT:[pid:     267](rm)

/ #
:submit_bh:trace:[8:pdflush]
Call trace:
 [c0061a90] submit_bh+0x1e8/0x1ec
 [c0062f08] __block_write_full_page+0x208/0x43c
 [c0067d84] blkdev_writepage+0x1c/0x2c
 [c008b710] mpage_writepages+0x278/0x460
 [c0067934] generic_writepages+0x14/0x24
 [c0042fe8] do_writepages+0x38/0x58
 [c008964c] __writeback_single_inode+0x88/0x3d0
 [c0089f70] sync_sb_inodes+0x1b8/0x2d4
 [c008a4c8] writeback_inodes+0x180/0x1b4
 [c0042d38] wb_kupdate+0xd4/0x168
 [c0043c34] pdflush+0x154/0x260
 [c0032338] kthread+0xec/0x128
 [c0004554] kernel_thread+0x44/0x60
```

How page daemons write back files with sync - 1

- pdflush (sync)
  - background_writeout()
  - writeback_inodes()
  - sync_sb_inodes()
  - __writeback_single_inode()
  - __sync_single_inode()
  - do_writepages()
  - generic_writepaes() // fat has writpage(), not writepages()
  - mpage_writepages() :fs/mpage.c
  - // getblk passed as NULL
  - blkdev_writepage() for dir, fat_writepage() for file
  - block_write_full_page()
  - __ block_write_full_page()
  - submit_bh()
  - write_inode()
  - fat_write_inode()
  - mark_buffer_dirty()

How page daemons write back files with sync - 2

- wbc
  - sync_mode WB_SYNC_NONE
  - nr_to_write MAX_WRITEBACK_PAGES以下
  - nonblocking 1

- __ block_write_full_page()
  - submit I/O by submit_bh(), if mapped and locked
  - do redirty if already locked by others

- fat_write_inode()
  - just do mark_buffer_dirty()
    - because sync_mode == WB_SYNC_NONE

How page daemons write back files with sync - 3

```
•    * forced sync through pdflush
•    / #  touch /a/foo
•    EXIT:[pid:     268](touch)
•    / # sync
•    :submit_bh:trace:[8:pdflush]

•    Call trace:
•     [c0061a90] submit_bh+0x1e8/0x1ec
•     [c0062f08] __block_write_full_page+0x208/0x43c
•     [c0067d84] blkdev_writepage+0x1c/0x2c
•     [c008b710] mpage_writepages+0x278/0x460
•     [c0067934] generic_writepages+0x14/0x24
•     [c0042fe8] do_writepages+0x38/0x58
•     [c008964c] __writeback_single_inode+0x88/0x3d0
•     [c0089f70] sync_sb_inodes+0x1b8/0x2d4
•     [c008a4c8] writeback_inodes+0x180/0x1b4
•     [c0042bb8] background_writeout+0xc8/0x114
•     [c0043c34] pdflush+0x154/0x260
•     [c0032338] kthread+0xec/0x128
•     [c0004554] kernel_thread+0x44/0x60
•    WRITE Start 142
•    EXIT:[pid:     269](sync)
```

How page daemons write back files (kswapd)

- kswapd
  - balance_pgdat()
  - shirnk_zone()
  - shrink_cache()
  - shrink_list()
  - pageout()
  - fat_writepage()
  - block_write_full_page()
  - __block_write_full_page()
  - submit_bh()

Block device issues

Underlying block device

- BH_ordered flag
  - Purpose: Ensure write ordering (including media/device side)
    - E.g. Support code is inside in IDE driver.
    - It works as following, if HDD support cache flush operation
      - submit data I/O to HDD
      - flush HDD cache
  - Issues
    - If device driver doesn’t support this feature, block I/O request would be failed.
    - FS layer need to handle explicitly
  - Alternatives
    - For General
      - wait every I/O, if BH_ordered is set
        - » submit I/O
        - » wait I/O completion
    - For devices without cache or with write through cache
      - use noop elevator
  - Solution
    - Block I/O layer needs to provide transparency to FS.
      - wait I/O on submit if BH_oreded is set and device driver dosen’t support it.

Better Flash ROM support

- Issues
  - Current block device driver
    - “sector” – minimal data transfer unit with device hardware.
  - Flash ROM
    - two transfer unit, one for read/write ops and another for erase op.
    - erase unit > read/write unit, in general
    - if one read/write unit is broken, need to abandon entire erase unit.
    - translation layer may hide some of or most of them
    - write op may have strong relation with erase op

- File system layer
  - If unit of read and write may have different size, it may be good for robustness and performance… (need to be considered)
    - Cluster chain of FAT12
    - size of unit to be written as atomic operation

- Elevator
  - Write ops for the same erase unit could be done at once