OSTEP (Crash Consistency)

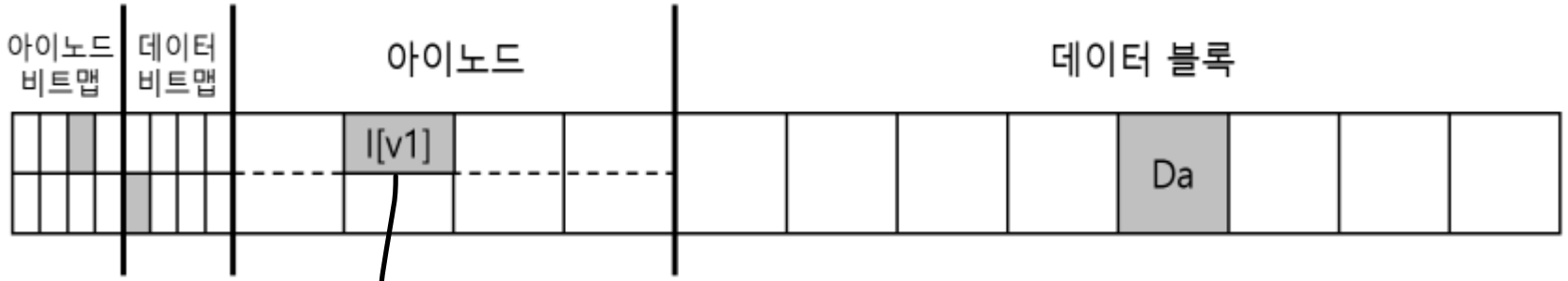
Main topic

How the file system safely updates the contests of the disk in the event of a system crash

Presenter: Kim Kyung Min

Ex) What if a crash occurs during an operation that appends a block to an existing file?

(disk structure in which information about existing files is stored)



Inode structure

```
owner          : remzi
permissions    : read-write
size           : 1
pointer        : 4
pointer        : null
pointer        : null
pointer        : null
```

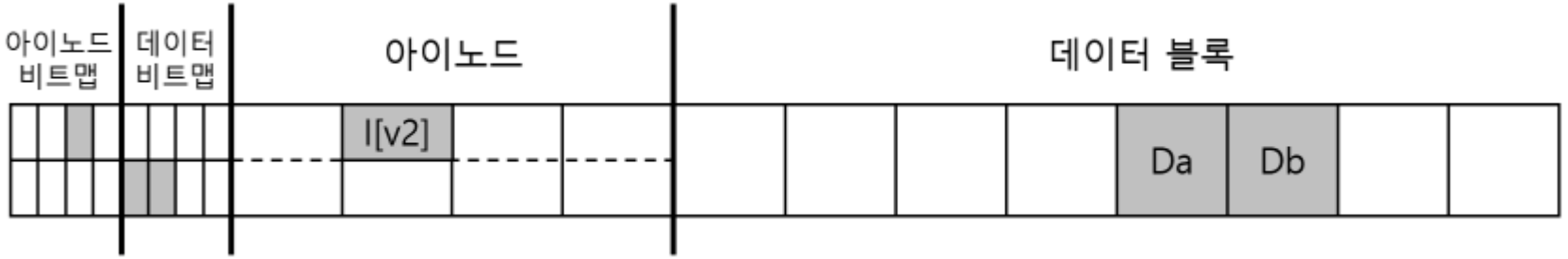
(before)

Add contents

```
owner          : remzi
permissions    : read-write
size           : 2
pointer        : 4
pointer        : 5
pointer        : null
pointer        : null
```

(after)

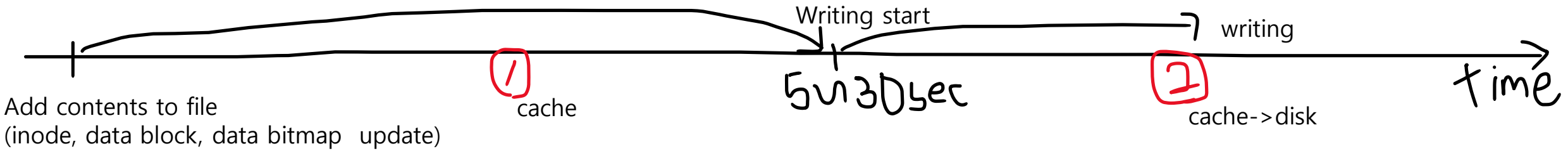
(disk structure after content is added)



Add content to file-> Add new data block -> Three disk data structures need to be updated

1. inode
2. Data bitmap
3. New data block

However, the result of write() is not immediately reflected on the disk!!



Crash scenario

1. Only data update: The data is on disk, but there is no inode pointing to it and no bitmap indicating whether it is allocated or not.

2. Only inode update: If you go to the address of the disk written in the inode and read it, you get meaningless data.

3. Only data bitmap update: Data block is marked as allocated, but no block pointed to by inode

4. Exclude data only: The data block is marked as allocated, and the inode also points to an address,
but the address actually contains meaningless data.

5. Exclude data bitmap only: The inode points to the address of the data correctly, but the bitmap is not allocated,
so there is a possibility that new data will be allocated to the same data block in the future

6. Exclude inode only: The block is written and the bitmap is said to be in use,
but it is not known which file the block belongs to because there is no address that the inode is pointing to.

→ **Problems such as inconsistencies between file system data structures, passing meaningless data to users, and space leaks**

Solution 1: file system checker (fsck)

- Resolves consistency issues on reboot by leaving the filesystem inconsistent

- Basic operation of Fsck
  1. super block: fsck first checks the contents of the superblock for errors.
  2. free block: Look at indirect and double indirect blocks to determine which data blocks are currently allocated
  3. inode state: Inspect each inode for damage or other problems
  4. inode link: Check the number of links of each allocated inode (number of links == number of directories with references to specific files)
  5. overlap: Check if there are different inodes pointing to the same block
  6. bad block: Check if there is an address outside the partition area
  7. directory test: Check that the contents of each directory are properly stored

- Just by looking at the basic behavior, we can predict that fsck is very slow.
  - Find all allocated blocks from the entire disk and read the entire directory tree
  - As disks grow in capacity and RAID becomes more popular, fsck is unusably slow

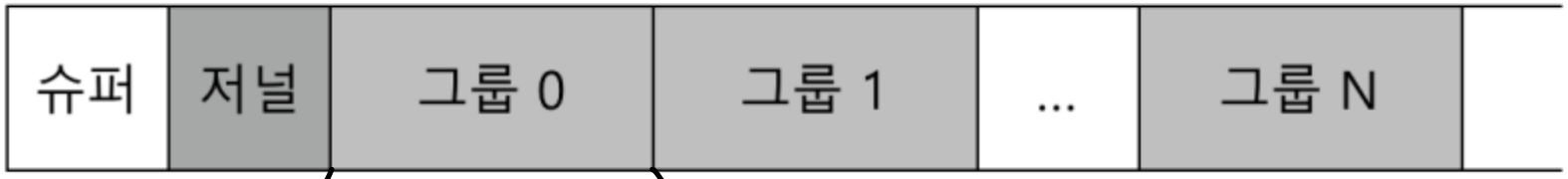
    ↳ Redundant array of independent disk

Solution 2: Journaling(or Write-ahead logging)

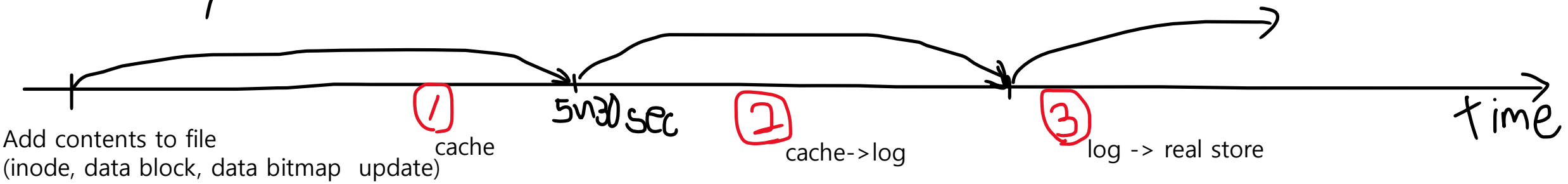
To log structure

- After updating, the information related to the information to be stored is recorded in the log, and then written to the desired data block.

(disk structure with journal data structure)



Inode, bitmap, data block



Add contents to file
(inode, data block, data bitmap update)

cache

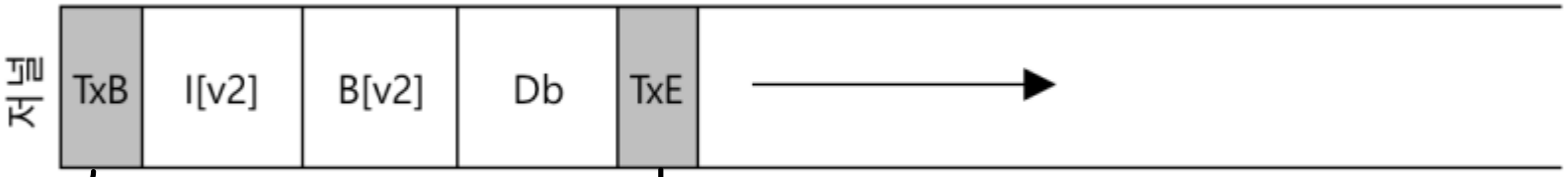
cache->log

log -> real store

- **Complexity due to two writes << improve system recovery performance**

Data journaling

(5 block in log structure)



Transaction end block: notifying the end of the transaction

Transaction begin block: information about the block to be updated

Can I request and use 5 blocks at will?

NO!!

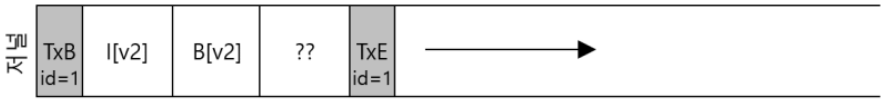
One block at a time

Too slow!!!

Multiple blocks at once

Disks randomly determine the completion order through scheduling

(data block update x)

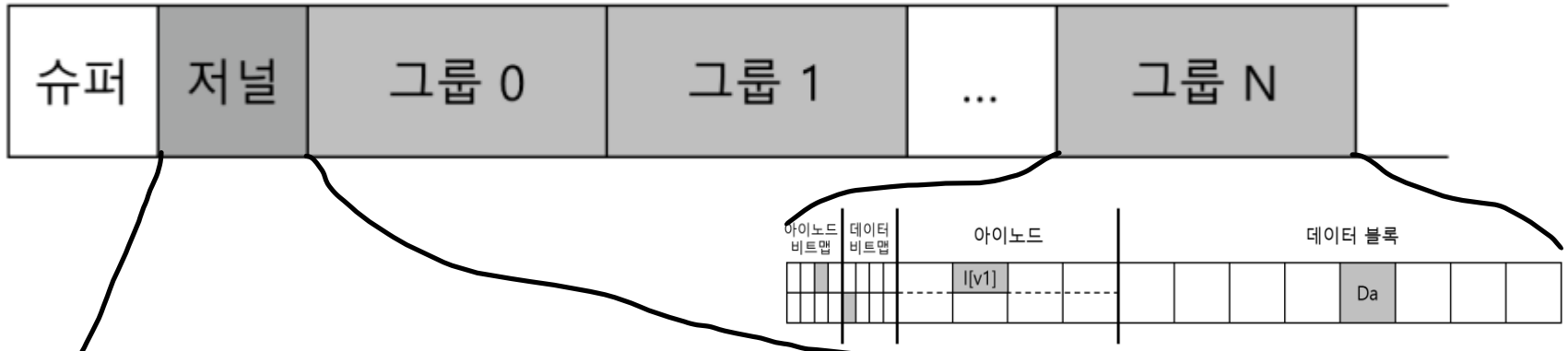


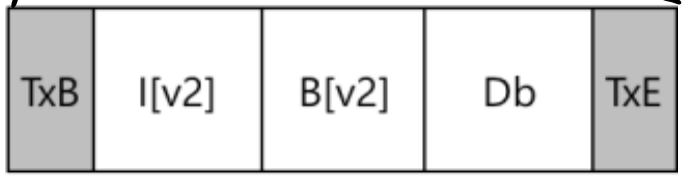
TxE block update == success end of transaction

Even though no data block is recorded,
the transaction appears to have ended successfully

① **TxE block should be written last to log**

(disk structure)





Log size is limited

Log space is exhausted
as transactions are continuously added

No more file system updates can be performed

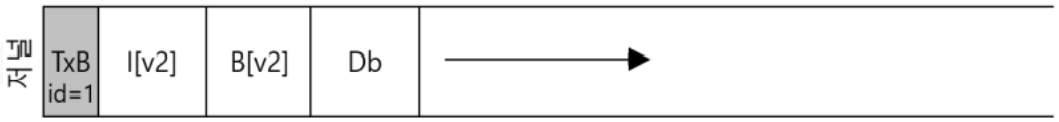
Journal super block: Contains information
that identifies transactions that have not been checkedpointed.

①

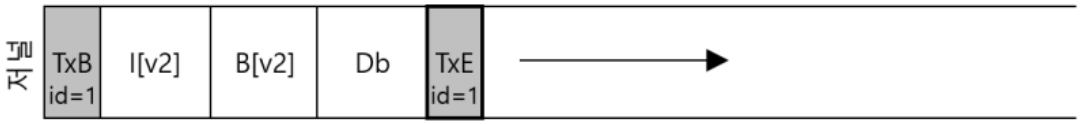
Contains information that identifies transactions that have not been checkedpointed.

Data journaling process

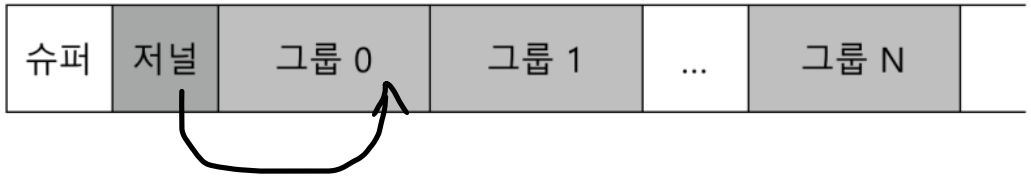
1. Log write: Write the contents of the transaction except TxE to the log



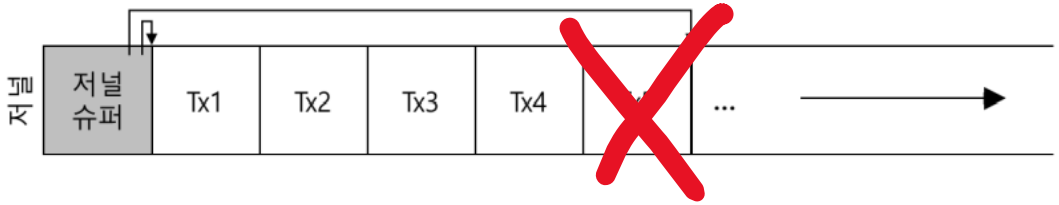
2. Log commit: write TxE to log(transaction is commited)



3. checkpoint: write data to real store



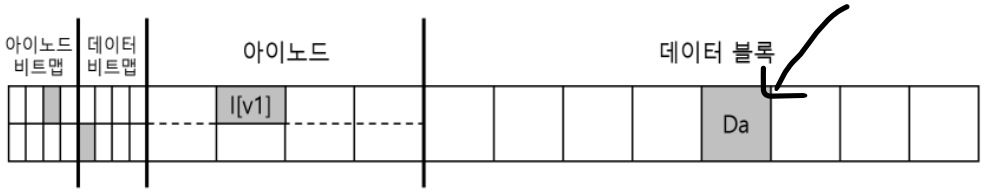
4. free: After a certain period of time, the journal superblock is updated to release the journal transaction.



Crashes don't actually happen very often, but every data block is written to disk twice??...

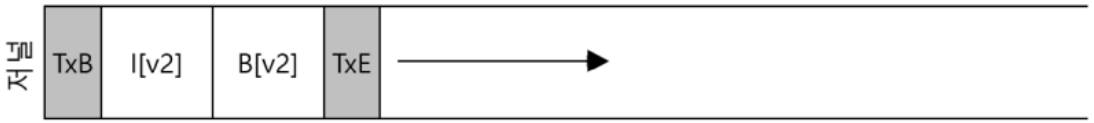
MetaData journaling

1. Data write: write data to real store.

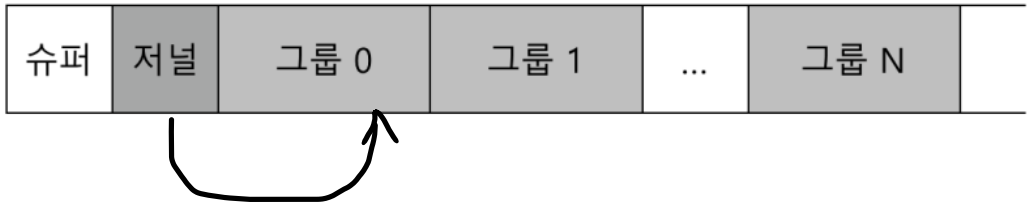


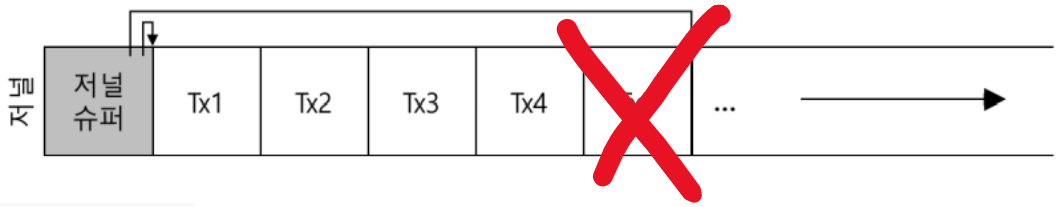
2. Write log metadata: Write the contents of the transaction except TxE to the log

3. Log commit: TxE block write to log(transaction is commited)



4. Checkpoint metadata: Updates the updated metadata to the last location on the file system.





5. free: Indicate that the transaction has been canceled in the journal super block

The key to crash consistency: write the object to which the pointer is pointed before the object to which it points

(metadata journaling timeline)