Iowa State University
Department of Electrical and Computer Engineering

Lecture 34:
Data Integrity & Protection II

Agenda

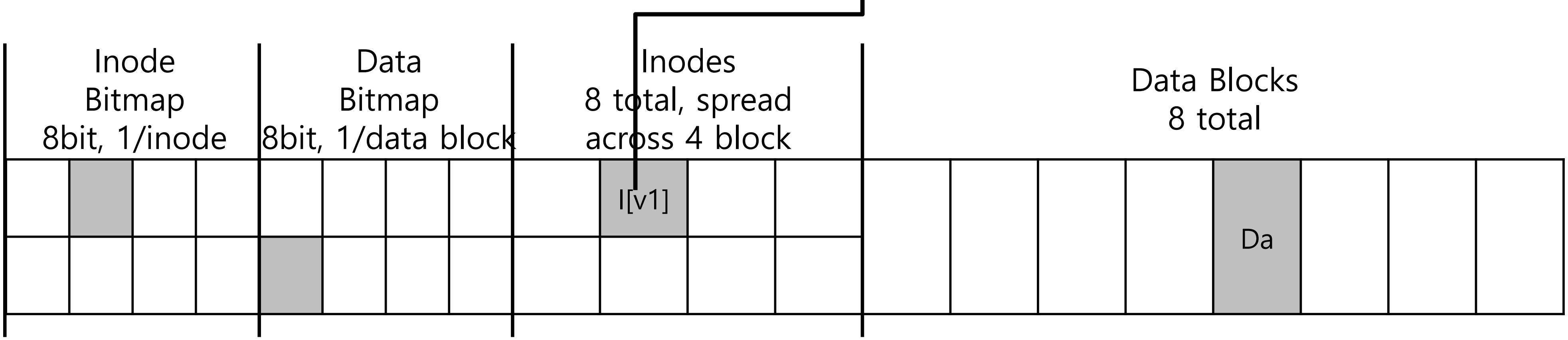
- Recap
- Data Integrity & Protection II
  - Journaling (cont')
  - FSCK

Recap

- Crash Consistency Problem
  - FS state before appending a new data block to a file

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

- Size of the file is 1
  - one block allocated
- First direct pointer points to block4 (Da)
- All 3 other direct pointers are set to `null`(unused)

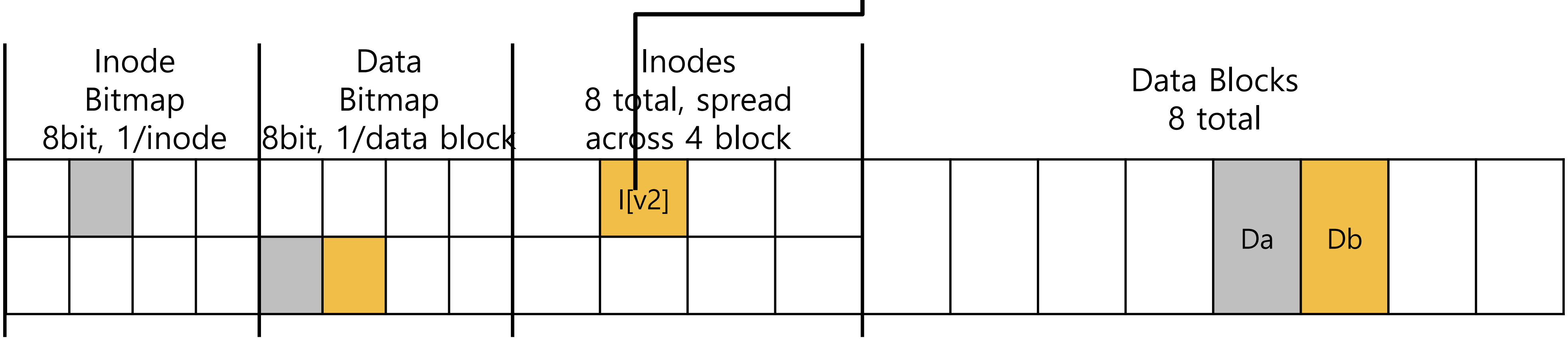


Recap

- Crash Consistency Problem
  - After update: all three writes are successful

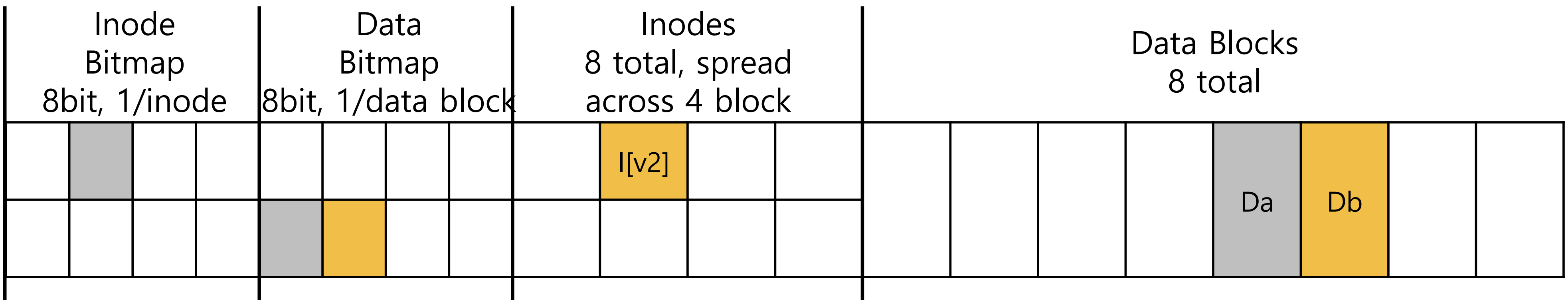
```
owner           : remzi
permissions     : read-only
size            : 2
pointer         : 4
pointer         : 5
pointer         : null
pointer         : null
```

- Data bitmap is updated
- Inode is updated (I[v2])
- New data block is allocated (Db)



Recap

- Crash Consistency Problem
  - Failure events may interrupt the writes
    - If only one single write is successful
      - Three possible outcomes
    - If only two writes are successful
      - Three possible outcomes
    - May lead to inconsistency & data loss



Recap

- Journaling
  - One way to address the crash consistency problem
    - Used in Ext3 & Ext4 (and many other FSes)
    - Also called Write-Ahead-Logging (WAL)
      - common in database community
  - Basic Idea
    - Do not write to the main FS data structures directly
    - Write to a “journal” data structure first
    - Update the main FS data structures only after all relevant writes are safely stored in the journal

Agenda

- ~~Recap~~
- Data Integrity & Protection II
  - Journaling (cont')
  - FSCK

Journaling

- A special data structure of the FS on disk
  - Introduced in Ext3
    - Occupy a small portion of the disk

Fig.1 Ext2 File system structure

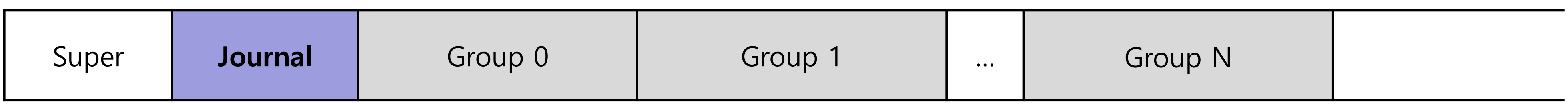
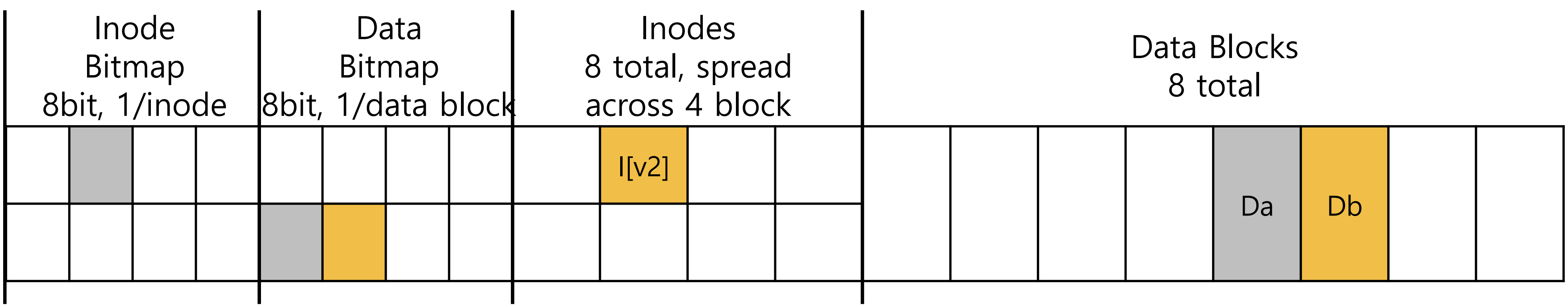


Fig.2 Ext3 File system structure

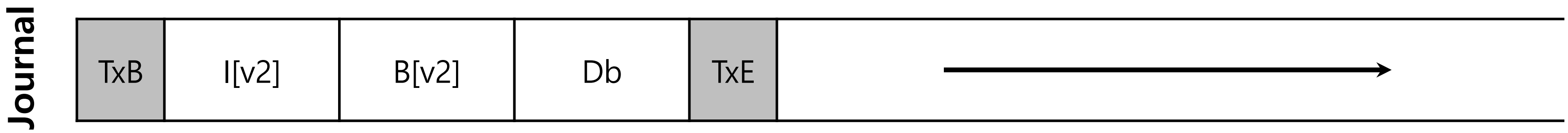
Journaling

- Data Journaling
  - One journaling mode in Ext3
  - Example
    - Update three blocks: inode (I[v2]), bitmap (B[v2]), and data block (Db)
    - Before writing them to their final locations, we first write them to the log (a.k.a journal)
      - How to know the three writes should be atomic?



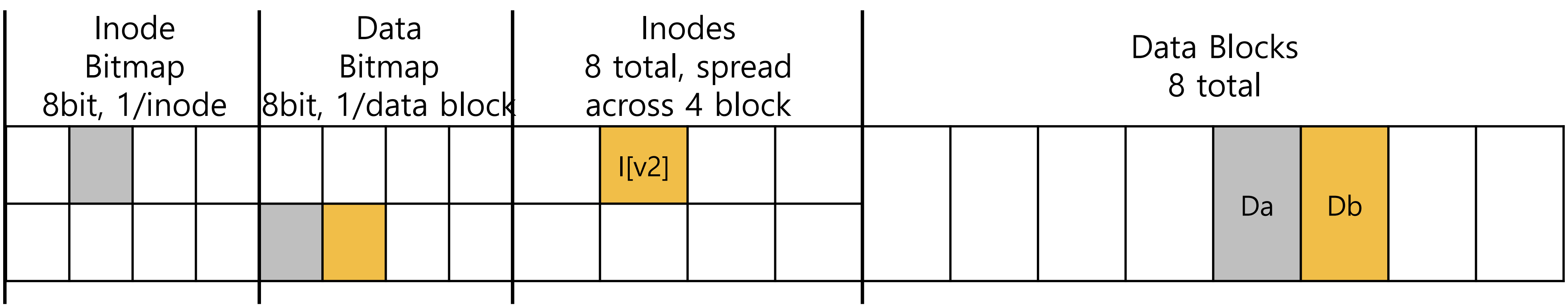
Journaling

- Data Journaling (Cont')
  - Record a transaction in journal
    - TxB: Transaction begin block
      - contain a transaction identifier(TID)
    - The middle three blocks contain the exact content of the blocks themselves
      - known as physical logging
    - TxE: Transaction end block
      - Marker of the end of this transaction
      - also contain the TID



Journaling

- Data Journaling (Cont')
  - Checkpointing
    - Once this transaction is safely on disk (in journal area), we are ready to overwrite the old structures in the file system
      - This process is called checkpointing
    - To checkpoint the file system, we issue the writes of I[v2], B[v2], and Db to their final disk locations

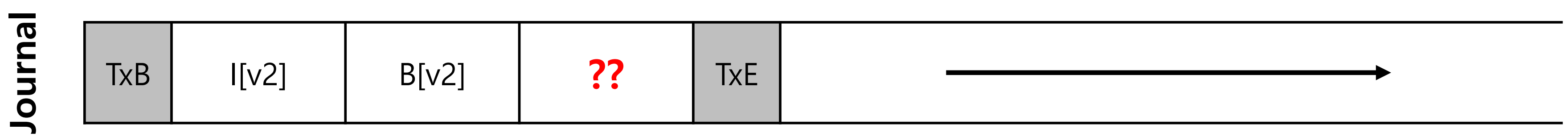


Journaling

- Sequence of operations with journaling (initial)
  - **(1) Journal write**
    - Write the transaction to the log and wait for these writes to complete
    - TxB, all pending data, metadata updates, TxE
  - **(2) Checkpoint**
    - Write the pending metadata and data updates to their final locations

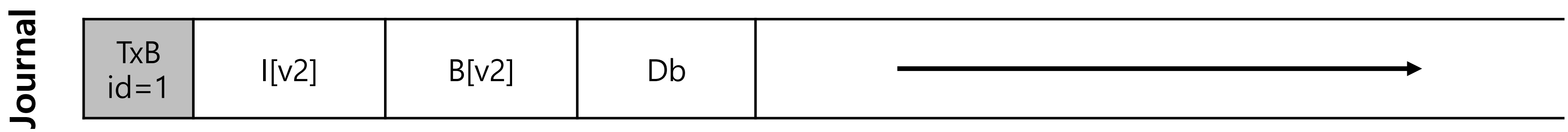
Journaling

- Two possible ways for journal write
  - Write a block at a time
    - 5 blocks (TxB, I[v2], B[v2], Dnb, TxE)
    - slow because of waiting for each to complete
  - Write all blocks at once
    - Five writes merged to a big single write
    - Faster, but unsafe
      - the disk internally may complete small pieces (e.g., 512B) of the big write out of order
        - TxE may be persisted first, but middle blocks may be lost due to power loss after TxE is persisted

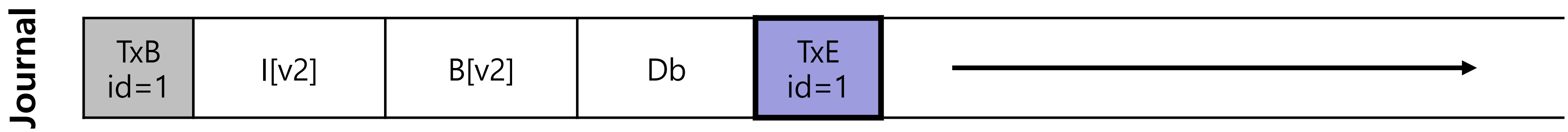


Journaling

- Enforce ordering
  - (1) write all blokes except the TxE block to journal



  - (2) write the TxE



  - An important aspect of this process is the atomicity guarantee provided by the disk.
    - The disk guarantees that a 512-byte write either happen or not
    - Thus, TxE should be a single 512-byte block

Journaling

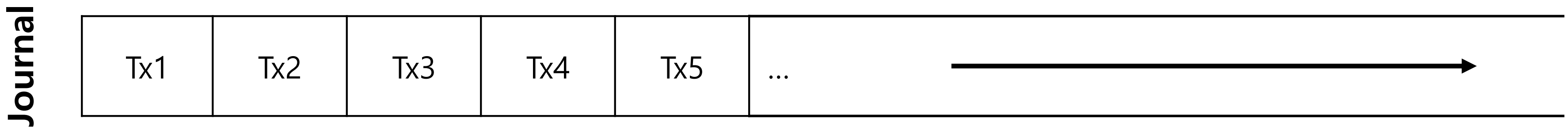
- Sequence of operations with journaling (refined)
  - (1) Journal write :
    - write the contents of the transaction to the log
  - **(2) Journal commit (added)**
    - **write the transaction commit block**
  - (3)Checkpoint
    - Write the pending metadata and data updates to their final locations

Journaling

- Recovery
  - If the crash happens **before the transactions** is committed to the log
    - The pending update is **skipped**
  - If the crash happens **after the transactions** is written to the log, but **before the checkpoint**
    - **Recover** the update as follow:
      - Scan the log and look for transactions that have committed to the disk
      - Transactions are replayed

Journaling

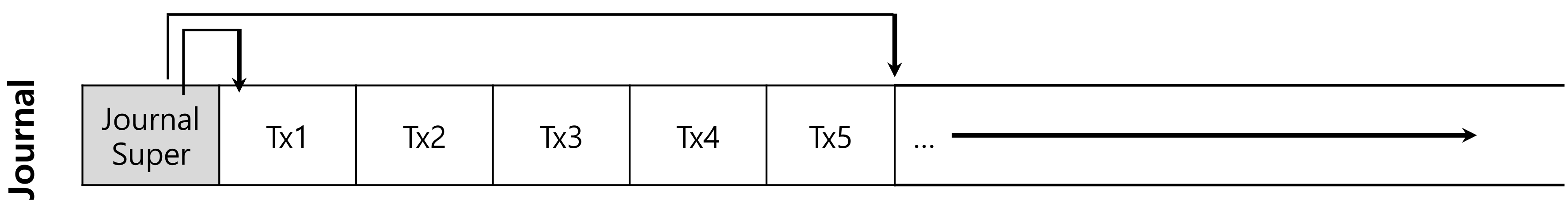
- Making the log “infinite”
  - The log is of a finite size
    - Two problems arise when the log becomes full



- The larger the log, the longer recovery will take
- No further transactions can be committed to the disk

Journaling

- Making the log "infinite"
  - Solution: treat the log as a circular data structure, re-using it over and over
    - the journal is referred to as a circular log
  - Once a transaction has been checkpointed, the file system should free the corresponding log space
  - journal super block
    - mark the oldest and newest transactions in the log.
    - record which transactions have not been check pointed



Journaling

- Sequence of operations with journaling (refined again)
  - (1) Journal write :
    - write the contents of the transaction to the log
  - **(2) Journal commit (added)**
    - **write the transaction commit block**
  - (3) Checkpoint
    - Write the pending metadata and data updates to their final locations
  - (4) Free
    - mark the transaction free in the journal by updating the journal Superblock

Journaling

- Problem of data journaling mode
  - Every piece of user data is written twice
    - One to the journal, the other to the actual FS data region
- In practice, multiple options with tradeoffs
  - E.g., Ext3/4 support three journaling modes

Journaling

- **Linux manual of Ext4**
  - **data=**{**journal**|**ordered**|**writeback**}
    - Specifies the journaling mode for file data. Metadata is always journaled. To use modes other than **ordered** on the root filesystem, pass the mode to the kernel as boot parameter, e.g. *rootflags=data=journal*.
    - **journal** All data is committed into the journal prior to being written into the main filesystem.
    - **ordered** This is the default mode. All data is forced directly out to the main file system prior to its metadata being committed to the journal.
    - **writeback** Data ordering is not preserved – data may be written into the main filesystem after its metadata has been committed to the journal. This is rumoured to be the highest-throughput option. It guarantees internal filesystem integrity, however it can allow old data to appear in files after a crash and journal recovery.

FSCK

- The File System Checker (**`fsck`**)
  - finding FS inconsistencies and repairing them
    - E.g., e2fsck for Ext2/3/4
  - `fsck` checks FS metadata including super block, bitmaps, Inode state, Inode links, etc. to make sure the FS metadata is internally consistent
  - can't fix all problems
    - E.g., The file system looks consistent but the inode points to garbage data.

FSCK

- The File System Checker (**`fsck`**)
  - finding FS inconsistencies and repairing them
    - E.g., e2fsck for Ext2/3/4
  - `fsck` checks FS metadata including super block, bitmaps, Inode state, Inode links, etc. to make sure the FS metadata is internally consistent
  - can't fix all problems
    - E.g., The file system looks consistent but the inode points to garbage data
  - The checker itself may cause problems!
    - **"Towards Robust File System Checkers"** published at USENIX File and Storage Technologies [**FAST'18**]

Agenda

- **Recap**
- **Data Integrity & Protection II**
  - **Journaling (cont')**
  - **FSCK**

Questions?

*acknowledgement: slides include content from "Modern Operating Systems" by A. Tanenbaum, "Operating Systems Concepts" by A. Silberschatz etc., "Operating Systems: Three Easy Pieces" by R. Arpaci-Dusseau etc., and anonymous pictures from internet.