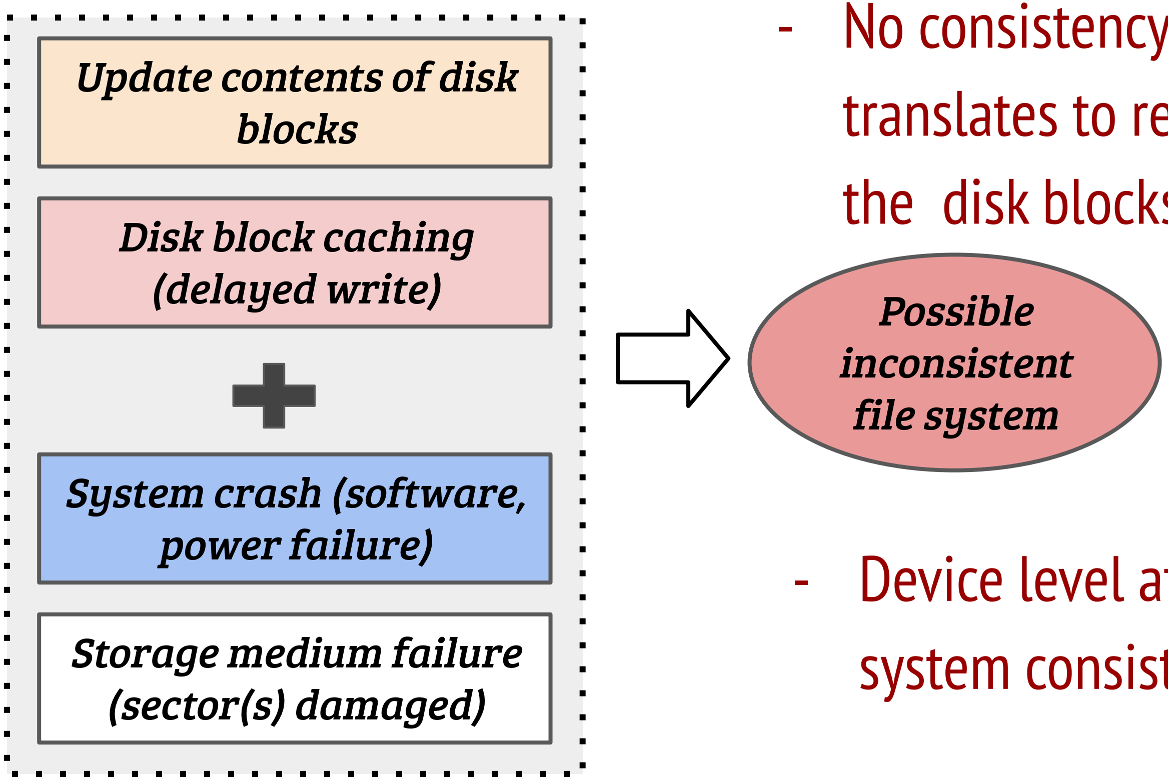
CS330: Operating Systems

Filesystem: consistency

Recap: File system inconsistency

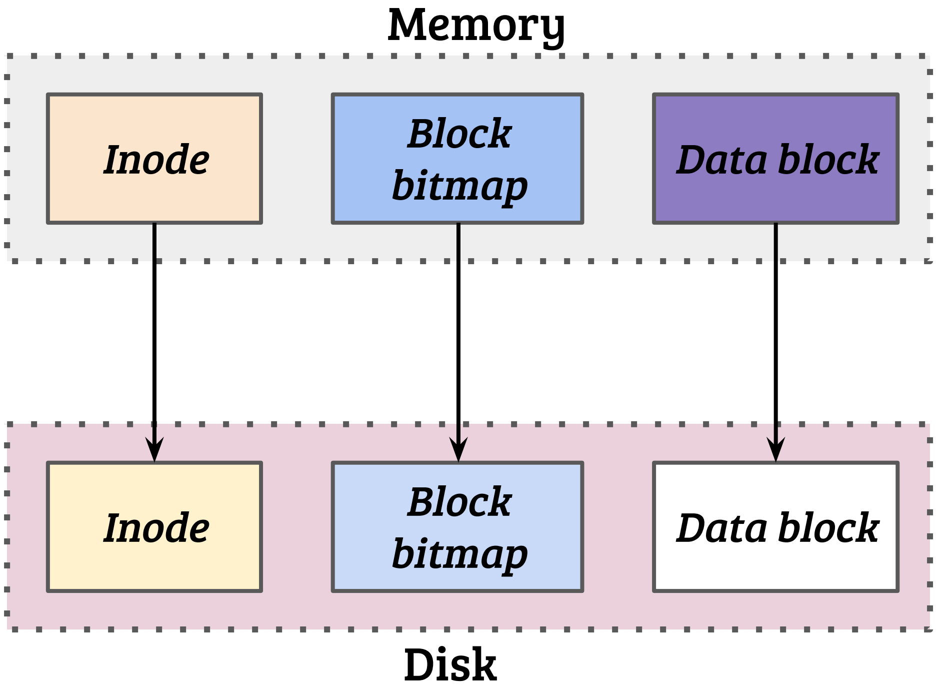
- No consistency issues if user operation translates to read-only operations on the disk blocks


Update contents of disk blocks
Disk block caching (delayed write)
System crash (software, power failure)
Storage medium failure (sector(s) damaged)
Possible inconsistent file system

- Device level atomicity may impact file system consistency

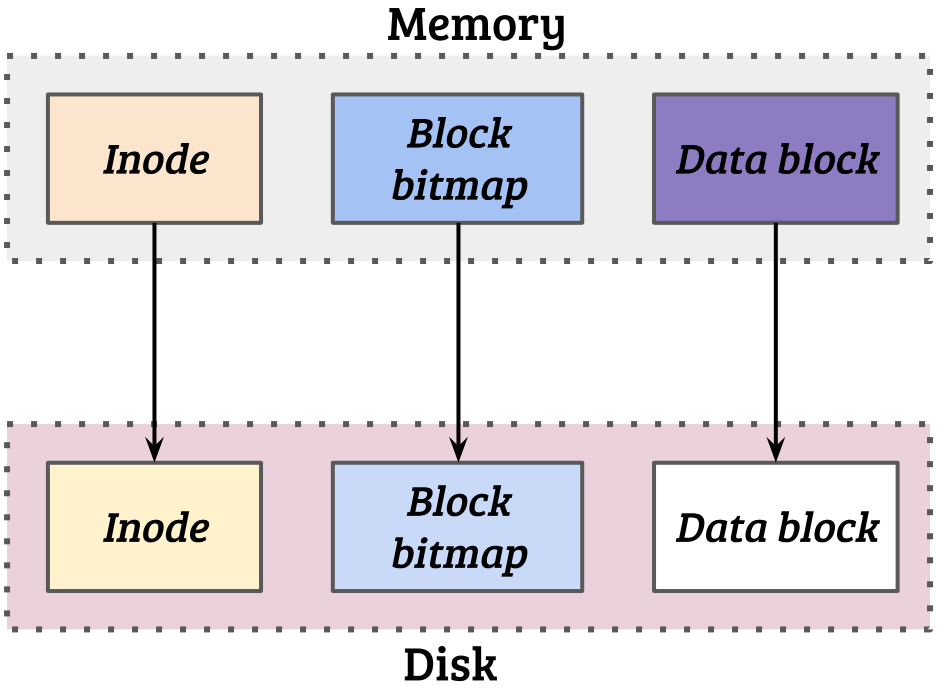
Example: Append to a file

- Steps: (i) seek to the end of file, (ii) allocate a new block, (iii) write user data
- Inode modifications: size and block pointers
- Block bitmap update: set used block bit for the newly allocated block(s)
- Data update: data block content is updated



Example: Append to a file

- Steps: (i) seek to the end of file, (ii) allocate a new block, (iii) write user data
- Inode modifications: size and block pointers
- Block bitmap update: set used block bit for the newly allocated block(s)
- Data update: data block content is updated



Three write operations reqd. to complete the operation, what if some of them are incomplete?

Failure scenarios and implications

Written	Yet to be written	Implications
Data block	Inode, Block bitmap	File system is consistent (Lost data)
Inode	Block bitmap, Data block	File system is inconsistent (correctness issues)
Block bitmap	Inode, Data block	File system is inconsistent (space leakage)

- All failure scenarios may not result in consistency issues!

Failure scenarios and implications

Written	Yet to be written	Implications
Data block, Block bitmap	Inode	File system is inconsistent (space leakage)
Inode, Data block	Block bitmap	File system is inconsistent (correctness issues)
Inode, Block bitmap	Data block	File system is consistent (Incorrect data)

- Careful ordering of operations may reduce the risk of inconsistency
- But, how to ensure correctness?

File system consistency with *fsck*

- Strategy: Do not worry about consistency, recover after abrupt failures
- During FS mount, check if it had been cleanly unmounted when it was last used, How to know?

File system consistency with *fsck*

- Strategy: Do not worry about consistency, recover after abrupt failures
- During FS mount, check if it had been cleanly unmounted when it was last used, How to know?
  - Maintain the last unmount information on superblock

File system consistency with *fsck*

- Strategy: Do not worry about consistency, recover after abrupt failures
- During FS mount, check if it had been cleanly unmounted when it was last used, How to know?
  - Maintain the last unmount information on superblock
- If the FS was not cleanly unmounted, perform sanity checks at different levels: *superblock*, *block bitmap*, *inode*, *directory content*

File system consistency with *fsck*

- Strategy: Do not worry about consistency, recover after abrupt failures
- During FS mount, check if it had been cleanly unmounted when it was last used, How to know?
  - Maintain the last unmount information on superblock
- If the FS was not cleanly unmounted, perform sanity checks at different levels: *superblock*, *block bitmap*, *inode*, *directory content*
- Sanity checks and verifying invariants across metadata. Examples,
  - Block bitmap vs. Inode block pointers
  - Used inodes vs. directory content

File system consistency with *journaling*

- Idea: Before the actual operation, note down the operations in some special blocks (known as journal)
- Journal entry for append operation: *[Start] [Inode block] [Block bitmap] [Data block] [End]*
- When the FS is updated (in a delayed manner) and mark the journal entry as completed (also known as *checkpoint)*

File system consistency with *journaling*

- Idea: Before the actual operation, note down the operations in some special blocks (known as journal)
- Journal entry for append operation: *[Start] [Inode block] [Block bitmap] [Data block] [End]*
- When the FS is updated (in a delayed manner) and mark the journal entry as completed (also known as *checkpoint)*
- Recovery mechanism: journal entries are inspected during the next mount and operations of non-checkpointed entries are re-performed

File system consistency with *journaling*

- Idea: Before the actual operation, note down the operations in some special

- Journal write should not only be synchronous but also performed in the specified order (especially the end marker)
- Failure after updating some blocks and rewritten during recovery is not an issue as the data is consistent at the end
- Failures during journal write is not a problem w.r.t. file system consistency

mount and operations are re-performed

Metadata journaling: performance-reliability tradeoff

- Journaling comes with a performance penalty, especially for maintaining the data in the journal
- Metadata journaling: data block is not part of the journal entry
- Practical with tolerable performance overheads
- Example journal entry for append: *[Start] [Inode block] [Block bitmap] [End]*
- Strategy: First write the data block (to disk) followed by the journal write and metadata commit afterwards, Why?

Metadata journaling: performance-reliability tradeoff

- Journaling comes with a performance penalty, especially for maintaining the data in the journal
- Metadata journaling: data block is not part of the journal entry
- Practical with tolerable performance overheads
- Example journal entry for append: *[Start] [Inode block] [Block bitmap] [End]*
- Strategy: First write the data block (to disk) followed by the journal write and metadata commit afterwards, Why?
    - If the metadata blocks are not written, FS can be recovered
    - If journal write fails, a write is lost (syscall semantic broken)