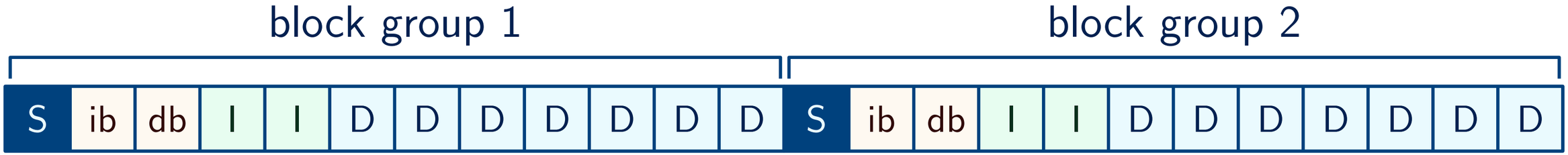
INF113: Journaling

Kirill Simonov

07.11.2025

Reminder

- We enhanced the very simple file system with block groups



- We discussed crash scenarios where the file system becomes **inconsistent**

- **Solution 1:** `fsck`, scan the whole file system after crash
  - Slow, does not detect errors that leave fs consistent
- **Today:** what if we leave a note to help us find which write failed?

Journaling

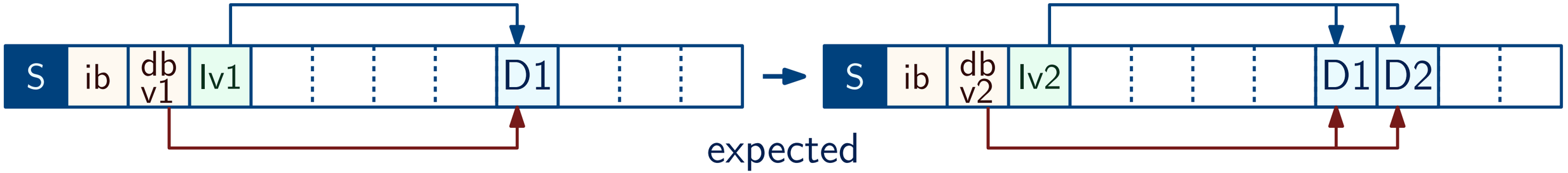
- Keep a portion of the disk as the **journal**
- Before any write operation, log the write into the journal first
  – Another term: "write-ahead logging"
- Little extra work for each write $\rightarrow$ Much easier recovery

Linux ext2 (no journaling)

Linux ext3 (with journaling)

Data journaling

Append a data block to a file: write data block (D2), new inode block (Iv2), new data bitmap (db v2)



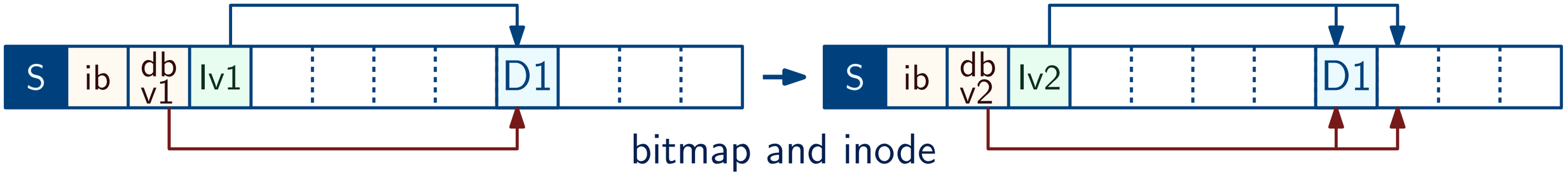
Writes to the journal:

1. Transaction begin block (TxB): addresses and transaction id
2. Copies of blocks to be written (Iv2, db v2, D2)
3. Transaction end block (TxE): transaction id

Checkpointing

1. **Journal write:** Write the transaction (begin+blocks+end), wait for it to complete

2. **Checkpoint:** Write the updates to their final locations in the file system



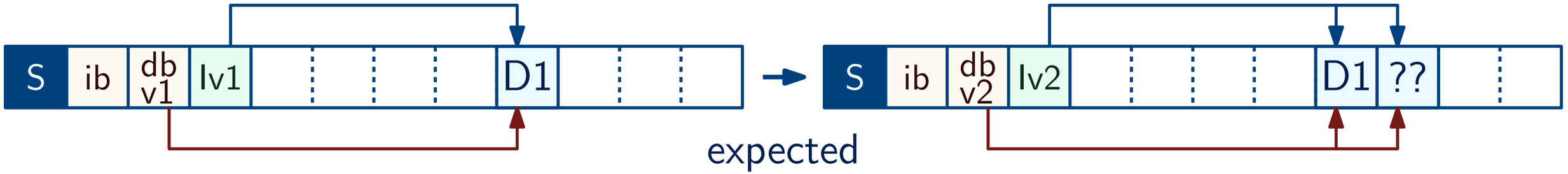
- If a crash happens during checkpointing, repeat from the journal
  - Need to know that a crash happened
  - And which transactions in the journal are not completed

even if consistent!

Crashes during logging

- Assume journal blocks are written in arbitrary order; crash before D2 is written

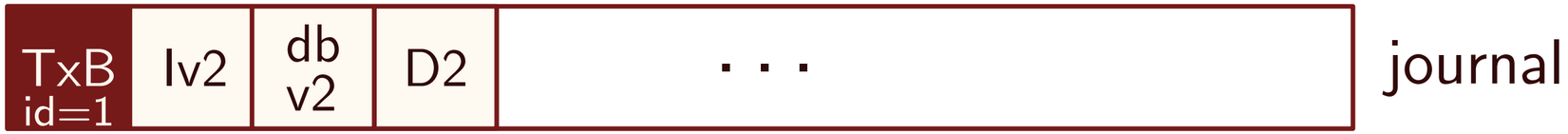
- System attempts to reapply the transaction, which looks valid



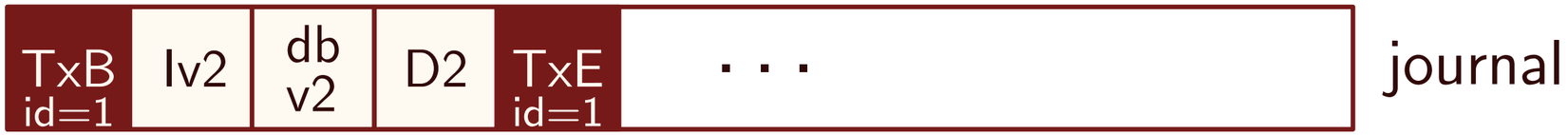
- The file is now corrupted!
- We need to make sure the transaction was fully written to the journal

Writing the transaction

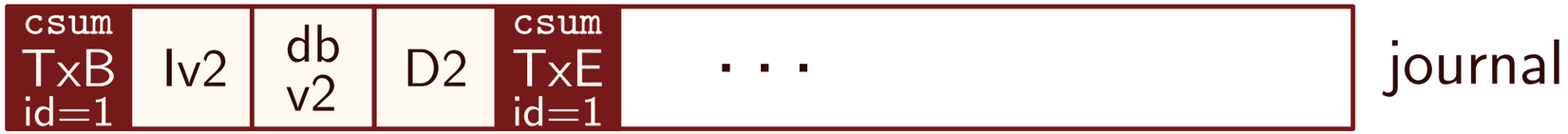
- Writing all blocks one after the other is too slow
- First write the start (TxB+blocks) in any order, wait to complete



- Then write the end block (TxE), wait to complete



- Alternative solution (ext4): any order, but write a checksum to TxB and TxE

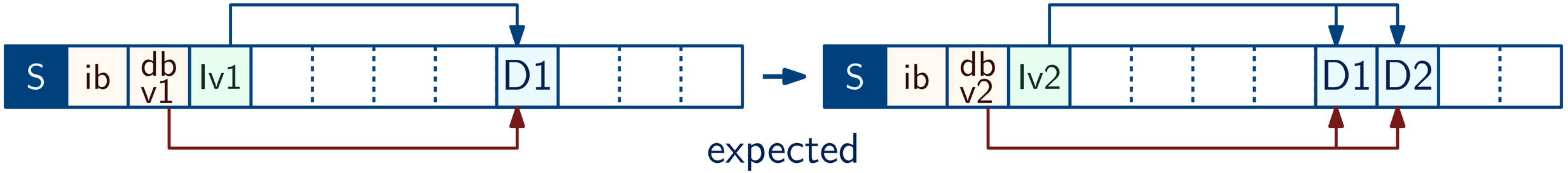


Final protocol

1. **Journal write:** Write the transaction (TxB+blocks), wait to complete

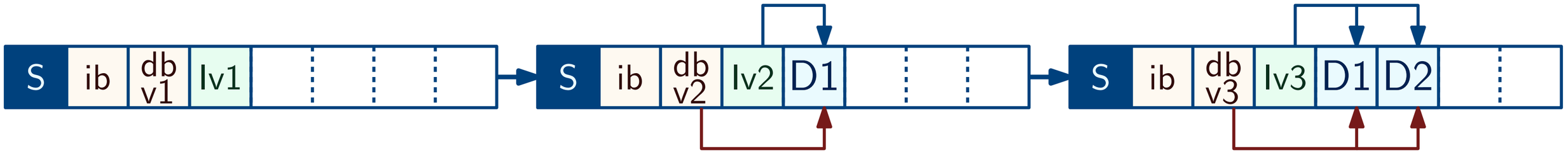
2. **Journal commit:** Write the transaction commit block (TxE), wait to complete

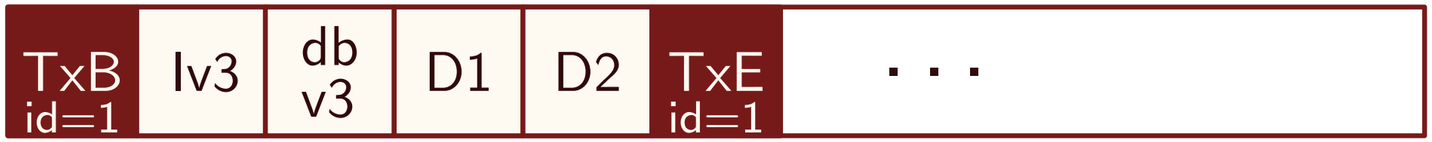
3. **Checkpoint:** Write the updates to their final locations in the file system



Batching updates

- Every write is now doubled: first in the log, then in the destination
- We can improve performance by **batching** write requests:
  - Gather writes for 5 seconds, merge and send to the disk
  - Journal records a single transaction
- Rewriting same data block multiple times → one write sent to disk, one log entry
- Appending multiple blocks to the same file → one inode rewrite, one bitmap rewrite



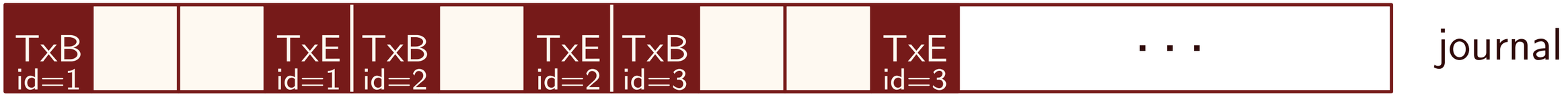


one transaction in the journal

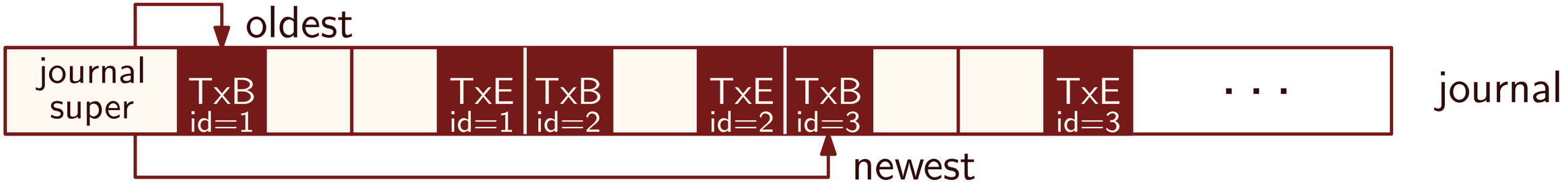
- Saves rewrites in other cases too, e.g., for creating two files within the same directory

Clearing the log

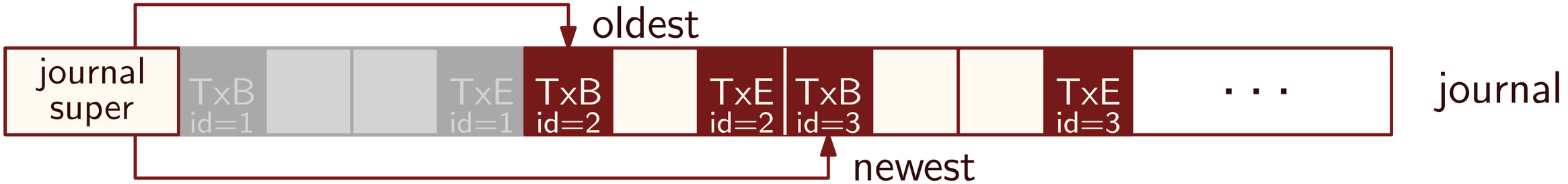
- Eventually the journal runs out of space



- **Circular log**: keep the oldest and the newest unapplied transaction

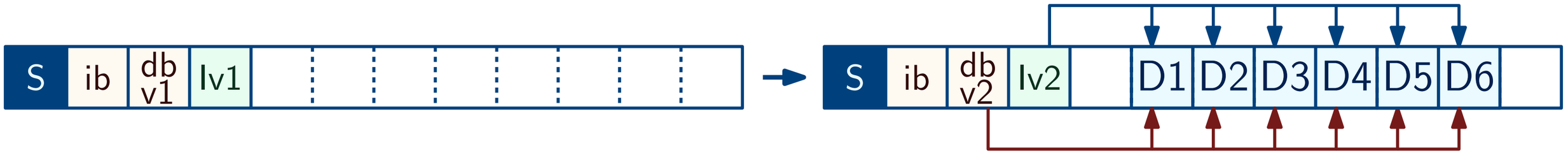


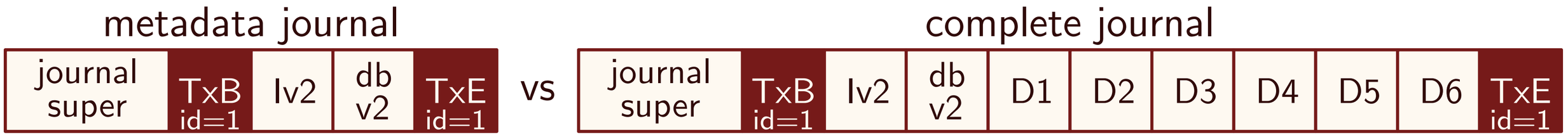
- After the transaction is checkpointed, mark it free—shift the "oldest" pointer



Metadata journaling

- Every block to be written is also saved in the journal → twice the number of writes
- **Metadata journaling:** Only save file system metadata to the log

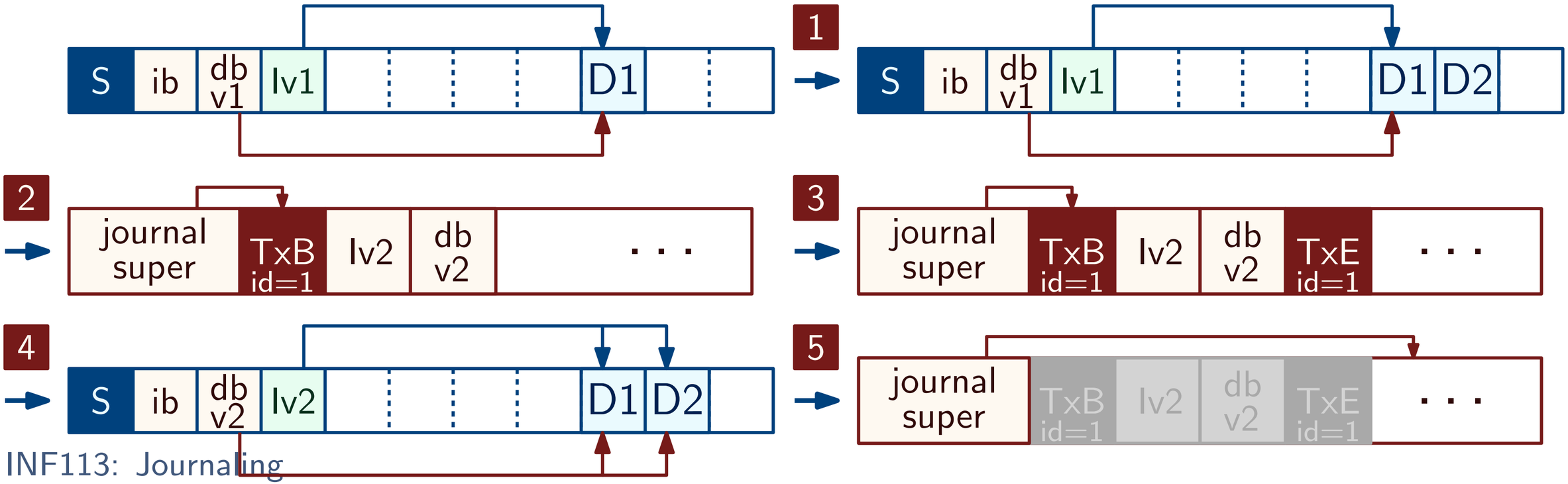




- User data is saved directly to the destination
- Most data is user data → much smaller overhead

Metadata journaling protocol

1. **Data write:** Write the data to the final location, wait to complete
2. **Journal metadata write:** Write the begin block and metadata to the log, wait
3. **Journal commit:** Write the transaction commit block to the log, wait to complete
4. **Checkpoint metadata:** Write the metadata updates to their final locations
5. **Free:** Later, mark the transaction free in journal superblock

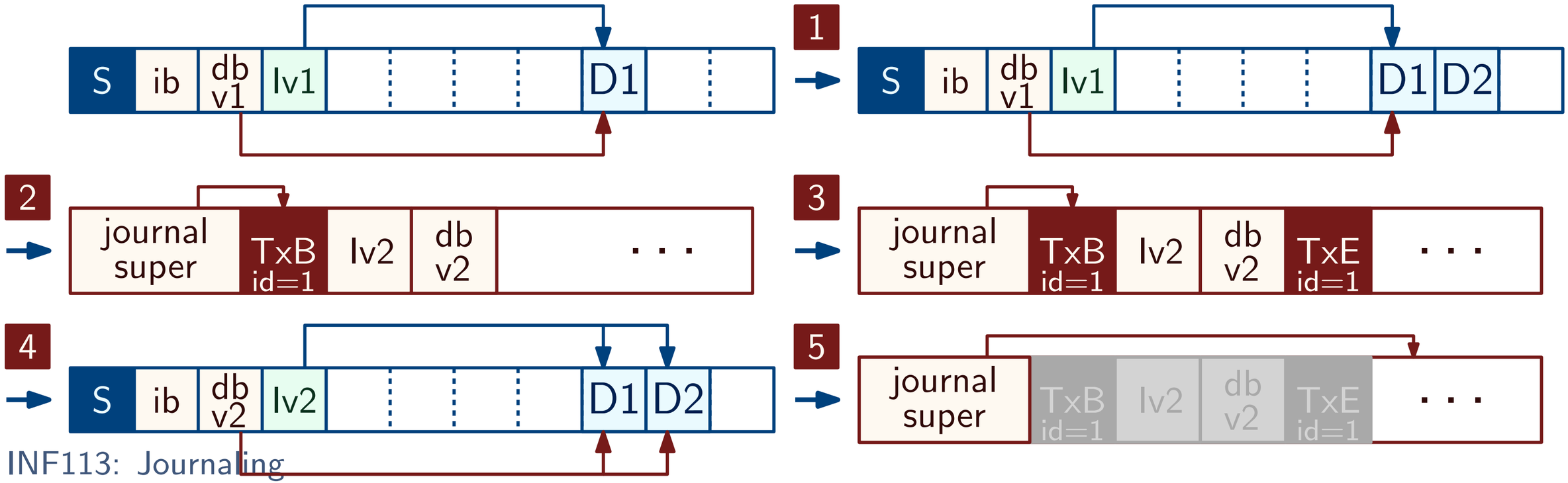


Crash scenarios

1. Crash after data write: no metadata/journal changed
2. Crash after journal write: incomplete transaction, discarded on restore
3. Crash after journal commit: steps 4 and 5 reattempted upon restore
4. Crash after checkpoint: 4 and 5 reattempted upon restore
5. Crash after free: All changes are saved to disk

write lost

write saved

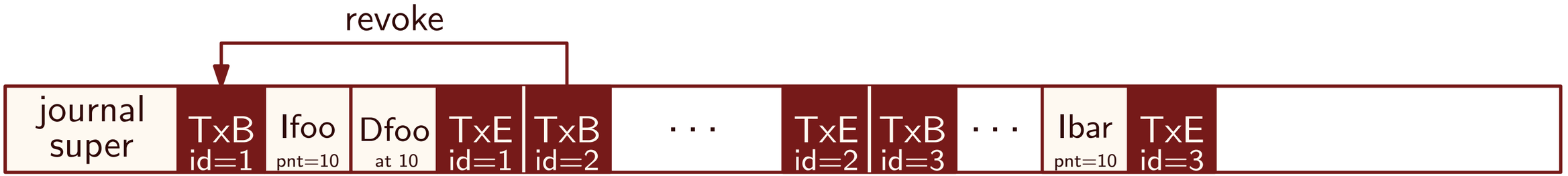


Block reuse

Metadata journaling may have an issue with reusing data blocks:

1. Add a file to directory `foo`, data in block 10
2. Remove directory `foo`
3. Create file `bar`, its data gets written to block 10
4. System crashes, and the journal is replayed
   - The outdated data block of `foo` is written to block 10
   - The data of `bar` is not in the journal, and is lost

Solution: Do not reuse same data blocks until checkpoint is completed
Or **revoke** outdated journal entries (ext3)—special entry in the journal



Summary

- Journaling allows to quickly and reliably repair the file system in case of a crash
- Only adding metadata to the journal greatly reduces the overhead
- (Some) file systems that use journaling:
  - ext3/ext4: user can choose between data/metadata journaling
  - ntfs: some form of metadata journaling
- Alternative solutions:
  - **Soft updates:** writes are ordered so that inconsistency is impossible, or is a leak
  - **Log-structured fs:** journal is the whole file system, data is read from the journal
  - **Copy-on-write:** no in-place writes, always to a new block (e.g., APFS)

Next week & exam

- Today is the final topic of the course
- Wednesday next week—Q&A session for course topics and Assignment 3
- Friday next week—no lecture
- Group sessions next week as usual
- Assignment 3 deadline next Friday (Nov 14)
- Exam preparation:
  - Try past exams—"Exam preparation" module on Mitt
  - Revisit lecture slides/recordings
  - Revisit book chapters—see the list on course homepage