CI583: Data Structures and Operating Systems
Journalled File systems

Journalled file systems

In *journalling*, the steps of a transaction are written to a *journal* before being *committed*, or written to disk.

If anything goes wrong whilst the changes are being written to disk, a recovery procedure repeats the steps from the journal when the system restarts.

If any change is made twice, that's not a problem because changes are idempotent – carrying them out n times is the same as carrying them out once.

Journalled file systems

There are two ways to do this:

- redo journalling (as described above), and
- undo journalling, in which the old contents of data blocks are stored in the journal, allowing the user to get back to the previous consistent state after a crash.

Journalled file systems

Making every change twice (once in a journal, once for real) risks doubling the amount of work, and journalling would not have been practical on the systems of the 70s.

However, aggressive caching and other techniques mean that we can minimise the work involved by batching up sets of changes and carrying them out in one go, rather than writing to the journal every time one byte is altered.

Journalled file systems

How big, then, should a transaction be?

Deleting a large file might require millions of operations, so that a single task is too big for the journal.

Carrying out each small change in its own transaction would be inefficient.

Journalled file systems

In practice, small changes are batched together into one transaction and big changes are broken up into several transactions.

Some systems (e.g. `ext3`) use time-based transactions.

The important thing is that we respect the ACID rules and take the system from one consistent state to another.

Shadow-paged file systems

Journalled file systems ensure consistency but involve rather a lot of work. There is a simpler solution: shadow-paged file systems.

Like journalling, shadow-paged systems are only viable in a context where memory is plentiful and processors are fast.

Examples include ZFS from Sun.

Shadow-paged file systems

The idea is simple – the whole file system is represented in memory as a data structure called the shadow-page tree, the root of which is called the überblock.

The tree contains pointers to all metadata and data blocks.

Shadow-paged file systems

Whenever a disk block is about to be changed a copy is made and it is that which is modified – the original is kept unchanged.

To link the modified copy into the tree, the parent node must be altered, but instead of changing it directly, that node is also copied, and so on, up to the root node.

This procedure is called copy-on-write.

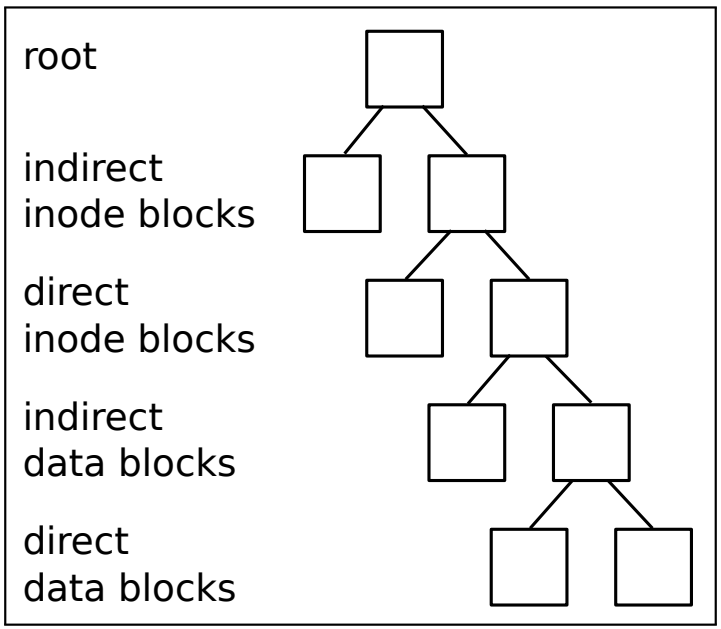
Shadow-paged file systems

The root node is modified directly in a single disk write.

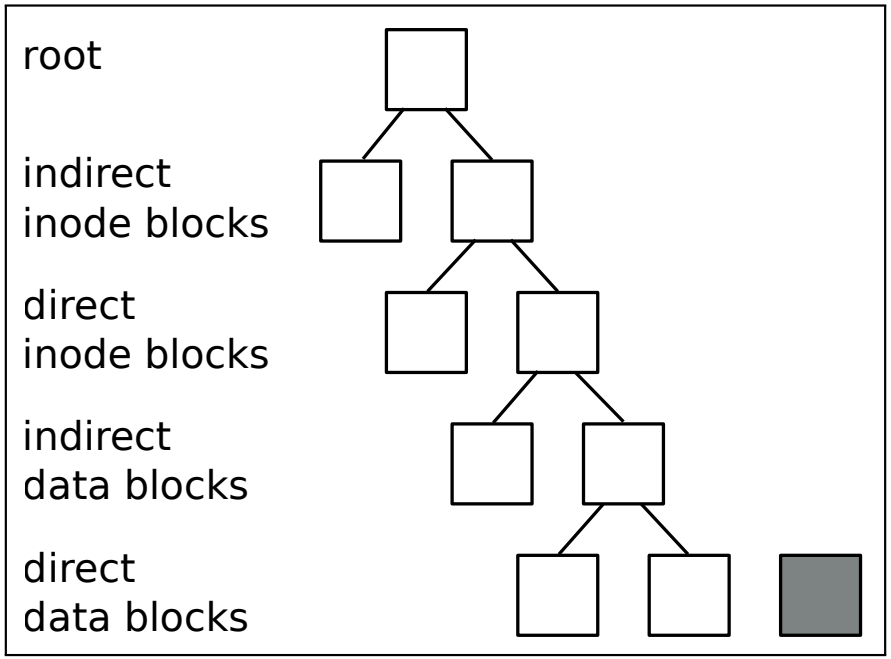
If the system crashes while an update is in progress but before the root node has been altered, the system comes back up with the old copy of the tree.

By keeping the original root node, we have a snapshot of the system as a whole.

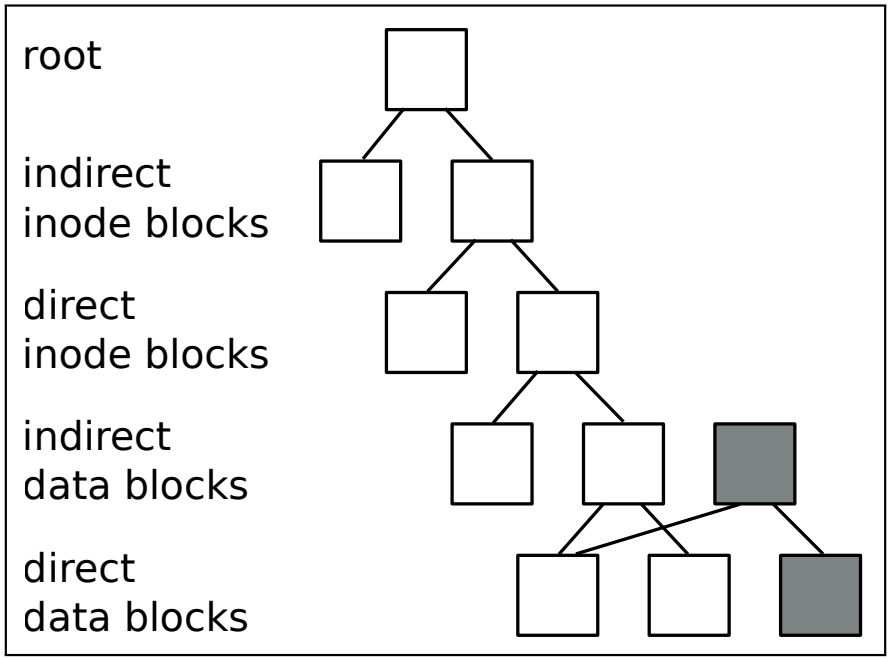
Shadow-paged file systems



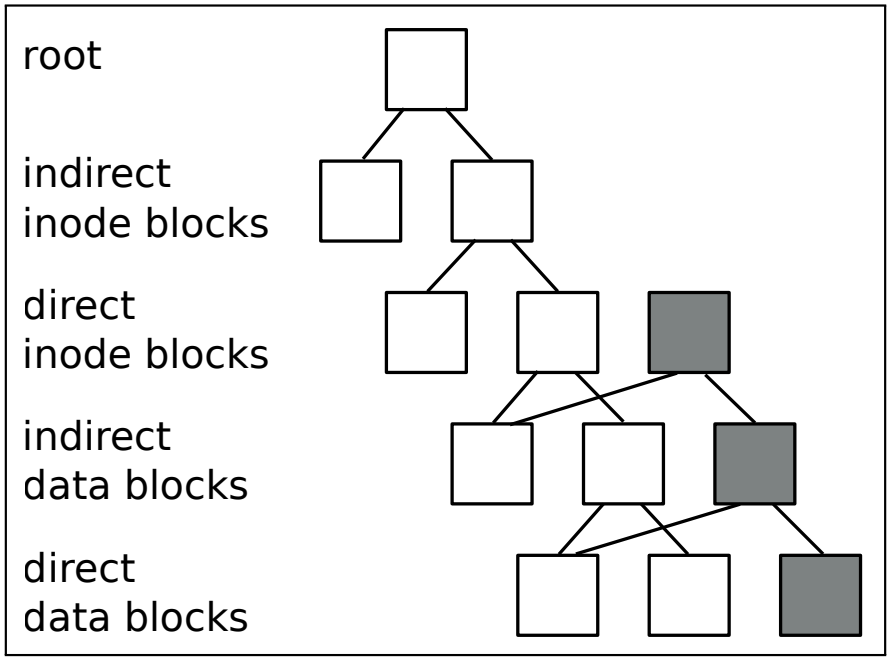
Shadow-paged file systems



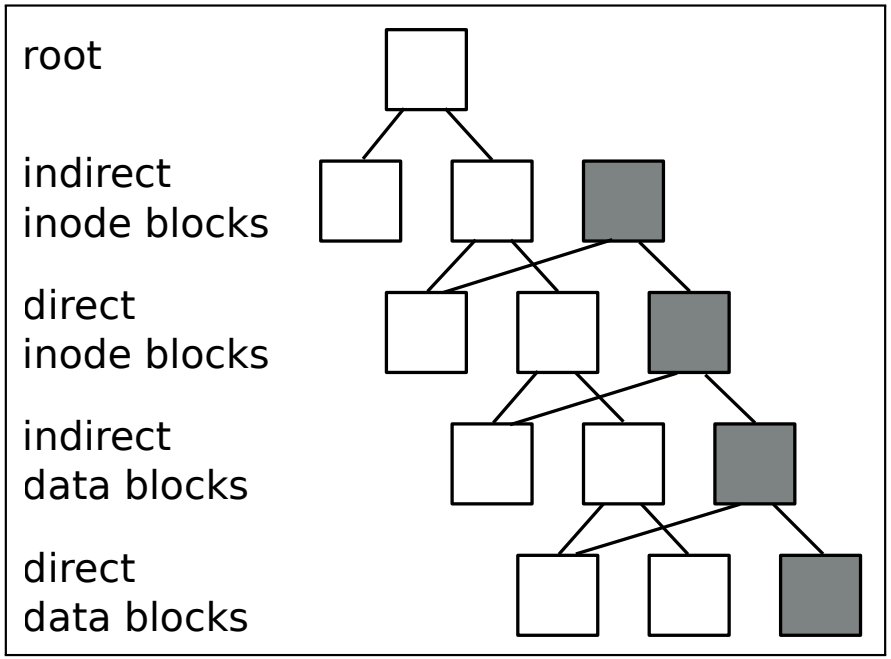
Shadow-paged file systems



Shadow-paged file systems



Shadow-paged file systems



Shadow-paged file systems