File Versioning Systems

- It can be very useful to keep earlier versions of files.
- We can recover from mistakes.
- We can restore damaged files.
- We can compare versions to see the changes made.
  - Useful for security purposes (self-securing storage)
  - Also useful for other purposes – e.g. working on a project with others and you want to see the changes they made.
- Sometimes we want to use earlier versions and still hold on to the recent versions.
- Similar to code management services like CVS.
- Can be done in a variety of ways but all require extra disk space.

Methods of Versioning

- A new version could be created when the file is closed or saved.
- A new version could be created after every modification – **comprehensive** versioning. Obviously a lot more versions.
- Either way we can -
  - keep complete copies of all previous version
    - very space intensive
    - but fast to retrieve/recover
  - keep a journal (or log) of changes
    - the journal keeps a record of the changes between two versions
    - retrieving requires work to reproduce earlier versions
  - keep a tree with all data
    - finding any version takes the same amount of time
    - can be slow for current version if the tree is big

Example

A file with the contents:

```
“Dear Mum, I hope you are well.”
```

Gets modified and saved as:

```
“Dear Mum, I am doing really well in 2020. I hope
 you are well.”
```

Then as:

```
“Dear Mum, I thought I was doing really well in
 2020 until the lockdown came in. I hope you are
 well.”
```

Log Version

Original version was

```
Dear Mum, I hope you are well.
```

changes to get to version 2

`11i54` (54 chars inserted at position 11)

changes to get to version 3

`13d` (a deletion at position 13)

`am` (the deleted data, this must be kept)

`13i13`

`74i25`

The current version is always stored.

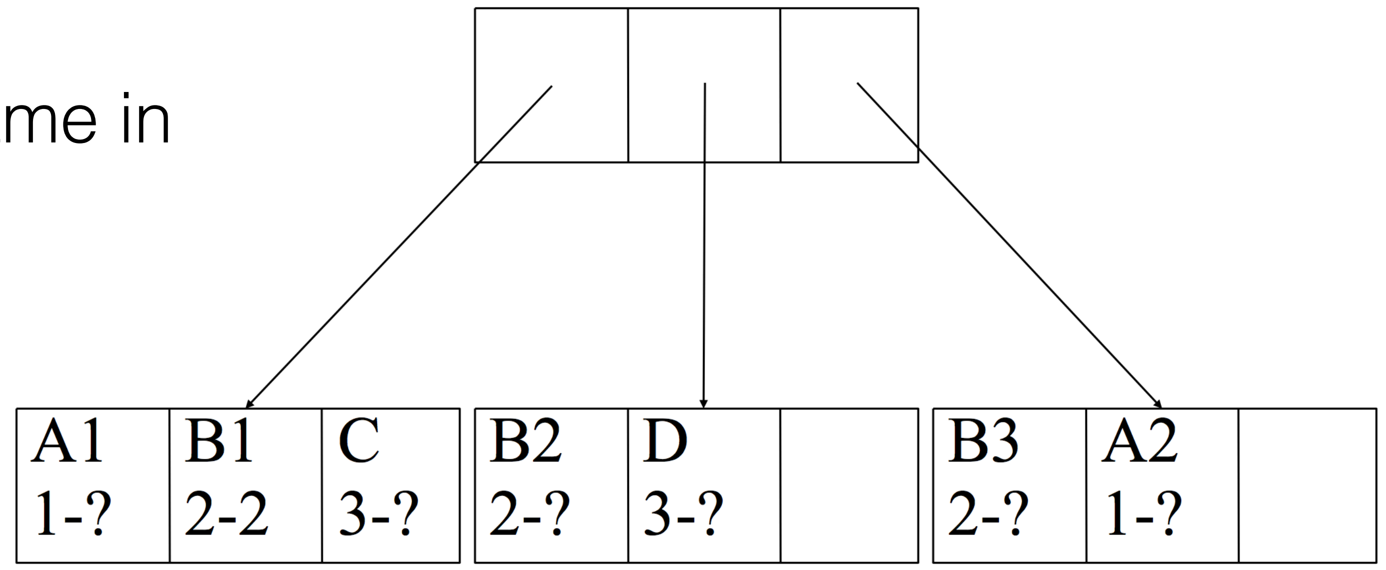
```
Dear Mum, I thought I was doing really well in 2020 until the
lockdown came in. I hope you are well.
```

To get previous versions have to go backwards through the log.

If we want to be able to roll forward from a checkpoint we need the new data in lines like `13i13`.

Multiversion B-tree

A1: Dear Mum, I_
A2: hope you are well.
B1: am
B2: _doing really well in 2020
B3: ._I_
C: thought I was
D: _until the lockdown came in



The version ranges (2-?) show which version the leaf is valid for.

? means up to the present.

Pros and Cons

- Log system
  - very compact
  - access to the current version is the same as without versioning
  - slow to revert to previous versions especially if there are many versions
  - can use checkpoints to improve this, but this adds considerably to the space requirement
- Tree system
  - very compact
  - quick to revert to any previous version
  - if there are lots of versions the tree can be big and then access to the current version will be a little slow
  - can keep a separate copy of the current version, this also adds to the space requirement
- No method works well if the data between versions is completely different. We are stuck with having to keep complete versions.

VMS versions

- When a file is closed VMS checks the number of versions. If the number is greater than the maximum number then the oldest version is discarded.

Windows XP onwards

It takes a checkpoint (restore points) of important system files on a regular basis.

- daily
- on installation of new drivers and applications

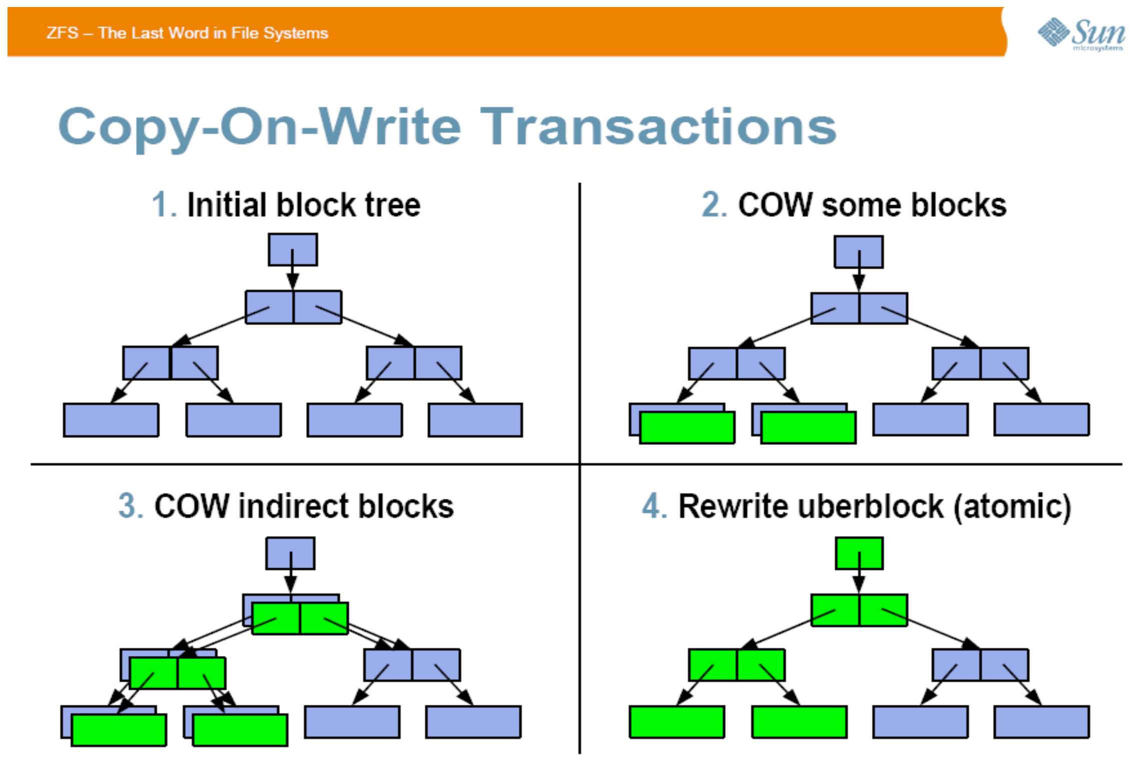
NTFS maintains a log of all changes to metadata, along with redo,undo information and whether the change was committed.

So it can recover all metadata to a consistent state after a crash (but not all data).

Windows and Mac

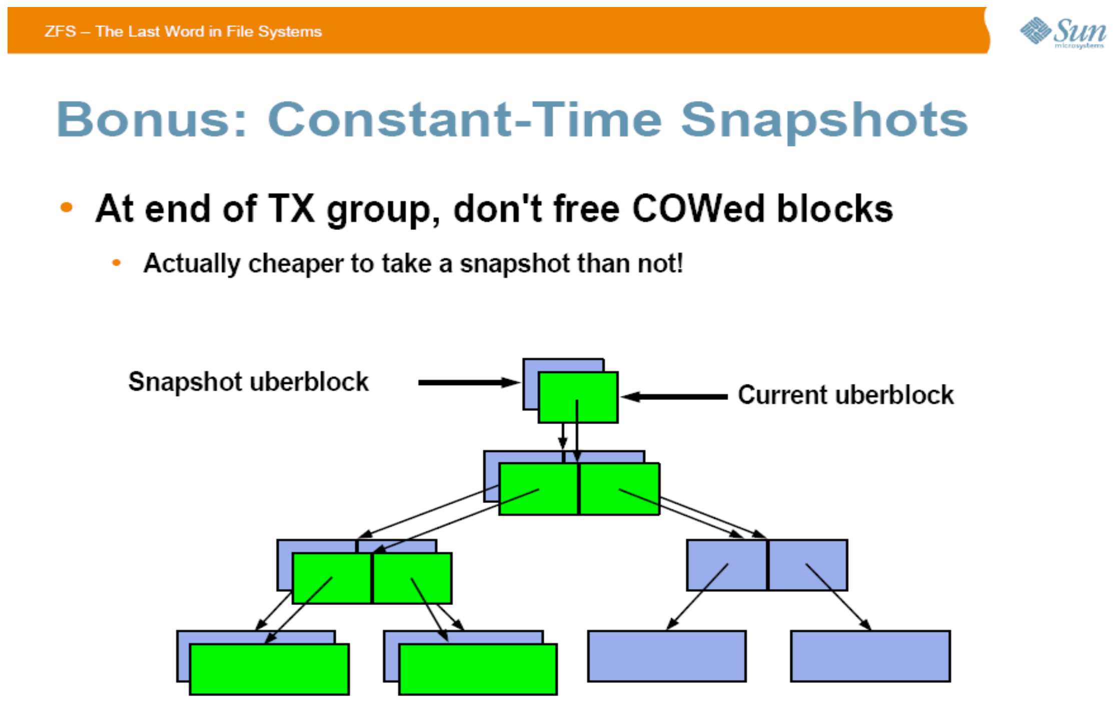
- Windows - Volume Shadow Copy
  - Keeps copies of files on volumes which have the service turned on
  - Also used to create restore points
  - Works at the block level - only modified blocks are copied
  - Typically made once a day
  - Users cannot trigger new versions of individual files
  - Users can access versions from Previous Versions tab of file properties
  - Not available at GUI level in Windows 8 but still in Windows Server and came back in Windows 10 (but based on File History)
- OS X Lion onwards - Versions
  - Auto saves individual files (if enabled by the application)
  - Works on chunks (intelligently determined by content) - only modified chunks are copied
  - Typically made every hour (autosave works every 5 minutes or during pauses)
  - Users can trigger new versions for individual files

ZFS Snapshots



ZFS – The Last Word in File Systems

Copy-On-Write Transactions



ZFS – The Last Word in File Systems

Bonus: Constant-Time Snapshots

- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!

Pruning

- All conventional versioning systems use pruning to keep the amount of data stored under control.
- Different heuristics
  - a fixed number of versions
  - treat some changes as more important than others
  - “observe” user behaviour, e.g., most often accessed
  - the user has to explicitly request a version be held
- snapshot systems – keep versions of files at particular times
- only keep versions for a small number of files

Self-Securing Storage

- All metadata, directories and critical files (OS files) are kept on a versioning system.
- Any intrusion (that uses files) can be tracked because the intruder cannot erase changes they have made to the system.
- We need to maintain all versions between checks for intrusion.
- This is referred to as the detection window.
- If the system is unable to keep enough versions we signal an alarm.
  - Either something has gone wrong, in the sense of not enough space allocated for a normal amount of usage.
  - Or someone is trying to force a pruning to hide their tracks.