Tux2
Failsafe Filesystem

for Linux

Daniel Phillips, innominate AG
daniel.phillips@innominate.de

What is Tux2?

- Failsafe like a journalling filesystem
  - But doesn’t have a journal
  - No recovery procedure needed
  - No journal replay needed
  - Instant off/Instant on
- Based on Ext2 - convert on mount (.1 sec/gig)
- Full data and metadata protection
  - less overhead than metadata-only journalling
- Uses phase tree algorithm
  - Atomic update, single/multiple transactions
  - Checkpointing is a simple extension

Tux2's Phase Tree algorithm

- Started life in a database
  - 1989 - Nirvana db
  - persistent objects
  - crash proof
- Other failsafe approaches
  - Journalling
  - Soft Updates (needs fsck)
  - Logging file system
- Other atomic update algorithms
  - Shadow blocks
  - Auragen
  - WAFL (see: Evil Patents)

Phase Tree algorithm

- Filesystem must be structured as tree
  - metadata tree, not directory tree
  - Ext2 *not* structured as a tree
- Create branches of new tree in freespace of old
- Atomic update
  - single write of the filesystem metaroot
  - actually a set of locations
  - phase counter to find current
  - detect crash by metaroots not same
  - instant revert to last consistent state
- Allocation maps are part of tree

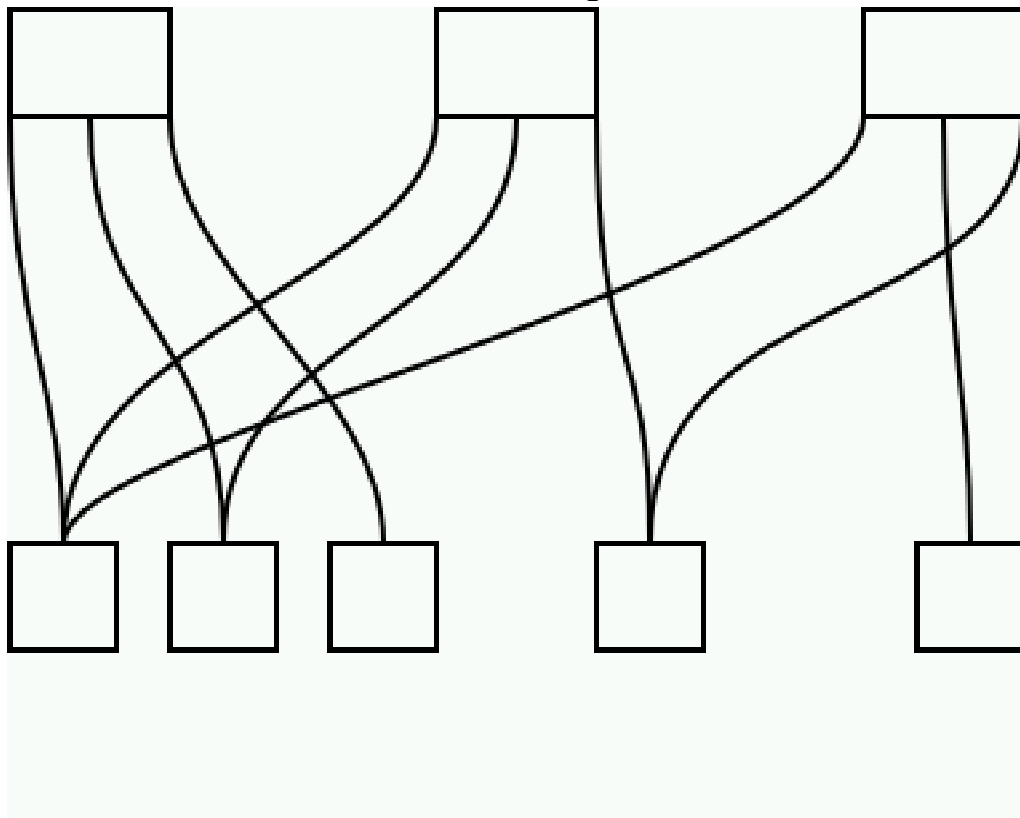
Phase Tree algorithm

- Phase start: copy the metaroot
- Update a block:
  - Make changes in copy of block
  - Update parent to point at copy
    - recurse up to metaroot
    - stop if already updated
  - Original block onto defree list
  - Applies to allocation bitmaps too
- Free a block:
  - Put it onto defree list
- Phase end:
  - Free all defreed blocks

Phases

- Two priority relationships
  - Write phase blocks before metaroot
  - Write metaroot before next phase blocks
- Gives partial ordering of writes
  - phase, root, phase, root...
- Could overlap
  - all phase blocks before metaroot
  - metaroot before next metaroot
- Gives partial ordering of writes
  - root+phase, root+phase...
  - forces later block reuse, more bookkeeping
  - probably not worth it

Three metaroots



A B C C’ B’

Phase Change

- Enabled after metaroot committed
- Branching phase becomes recording phase
- New branching metaroot is copy of old

- Phase change triggered on any of:
  - Total blocks written
  - Elapsed time
  - Memory pressure
  - Free blocks low

- Change on transaction boundary

- Tuning the phase length
  - total blocks

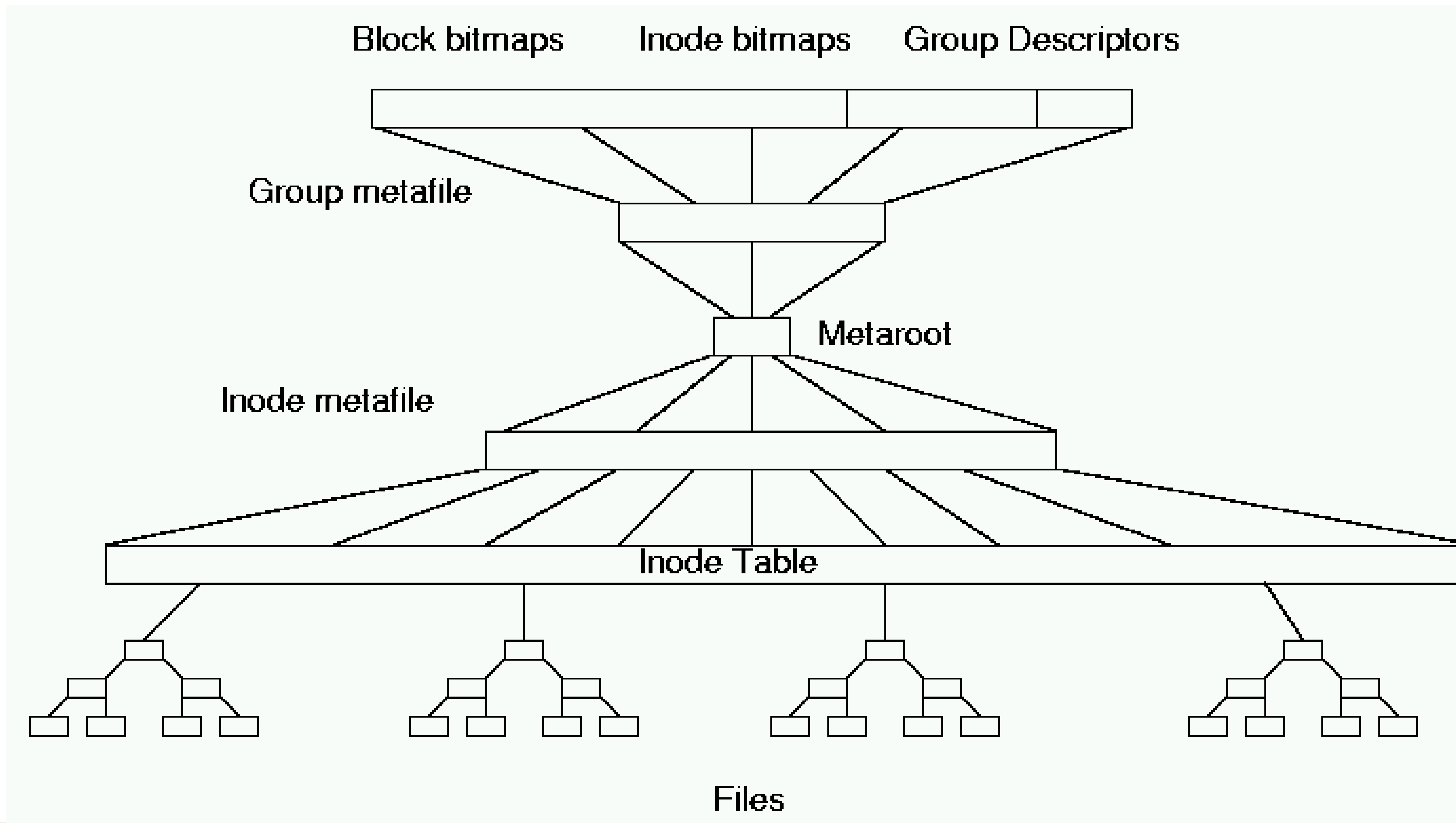
How Ext2 became a tree

- Super block
  - Constant data only
  - Set of Metaroots
- Metaroot
  - Per phase data
  - Two metafiles
- Metadata tables have indexes
  - inode table
  - block bitmaps
  - inode bitmaps
  - descriptors
- Metadata and data in same pool

Metafiles

- Inode metafile
  - normal inode table blocks
  - initially normal position
- Group metafile
  - normal block bitmaps
  - packed inode bitmaps
  - group descriptors shrunk
    - free block, inodes, dir counts
    - total: 8 bytes (was 32)
- Metafiles embedded in metaroot
  - flattens the tree
- Equal-level index structure, nontraditional

How Ext2 became a tree



Metadata is mobile

No fixed metadata locations

Good news:

- common allocation pool
- inode blocks near data

Bad news:

- fsck can’t find the inodes
  - inode carries inode number, magic number
  - costs 1 byte/inode

Kinds of overhead

- Extra metadata writes
  - mainly on rewrite
  - Shrinks to 1% as phase gets long
- Noncontiguous Writes
  - Elevator helps
- Fragmentated Reads
  - more like "permutation"
- Best case: untar a big archive
- Worst case: untar the same archive again

Fragmentation

File growing slowly, one block per phase

```
[1][2][3][M][M][M]
[1][2][3][-][-][-][4][M][M][M]
[1][2][3][5][M][M][4][-][-][-][M]
[1][2][3][5][-][-][4][6][M][M][-][M]
[1][2][3][5][7][M][4][6][-][-][M][-][M]
[1][2][3][5][7][-][4][6][8][M][-][M][-][M]
[1][2][3][5][7][9][4][6][8][-][M][-][M][-][M]
```

Data stays together, but permuted

Fighting Fragmentation

- Distribute freespace throughout filesystem
  - Ext2 already does this
- Keep file blocks together, even if permuted
  - Ext2 allocation goal does this
- Have a track cache
  - Will benefit all filesystems
- Fill tracks - kill rotational latency
  - Need to know disk geometry
- When all else fails, on-the-fly defrag
- Turn off data branching per-file for db

What next?

- Development diffs for 2.4 next month
- Working version as Christmas present to me
- Live benchmarks - verify predictions
- Performance tuning
- The long road to production code
- Fsck - Yes! still needed
- Resizing

- What about Tux3??
  - Transactions
  - Checkpoints
  - Background defrag

Tux2 and High Availability

- Is tux2 an enabler for HA?
  - Write anywhere - move anywhere
    - Heirarchical storage management
  - Multiple trees
    - Each storage device has its own tree
    - Cross-tree transactions
  - What about redundant trees?
  - What about RAID?

Comparison to JFS

- Better
  - fewer writes
  - less seeking
  - fewer serializations
  - easier to audit
  - supports block migration
  - removable media

- Worse
  - requires block migration
    - permutation
    - adjacent file interleaving
    - large scale fragmentation possible
  - extra CPU to allocate
  - problems with linear allocation
  - commit latency

brought to you by Daniel
and innominate AG

Read more:

innominate.org/~phillips/tux2