Tux3 linux filesystem project

A Shiny New Filesystem for Linux

http://tux3.org

What is a next gen filesystem?

- Snapshots, writable and recursive
- Incremental backup, online Replication
- Good Extended Attribute support
- Online grow, shrink, check, repair
- Scale to Petabytes of data, Billions of files
- Can I run it on my cell phone too?

The modern user is Greedy

Status Quo of Filesystems

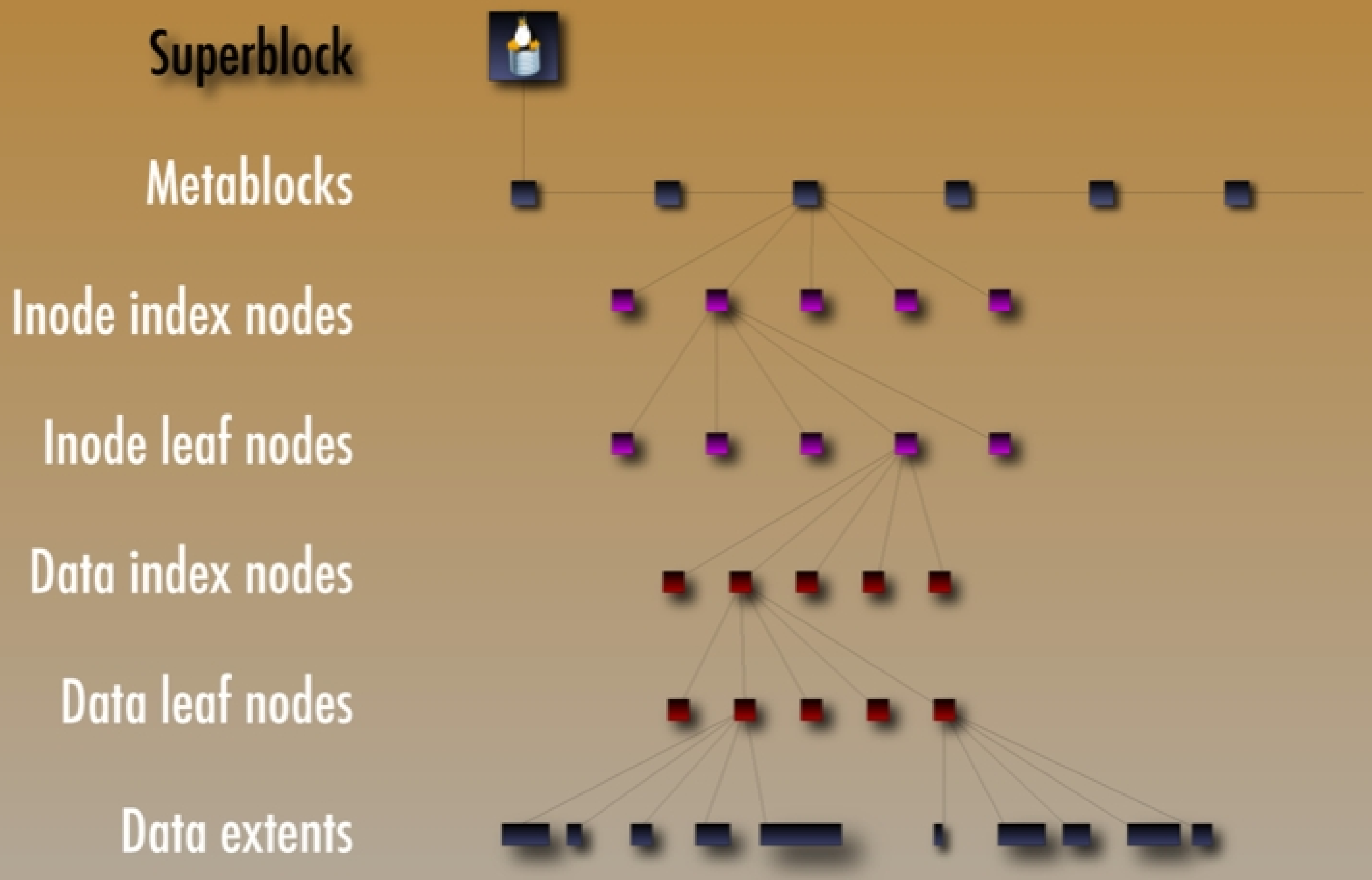
- Linux Ext2/3/4 descended from ancient UFS, others using 80's era journalling model
- Sun/Solaris leading the nextgen filesystem race with ZFS
- BSD ahead of us with Hammer, already stable
- Btrfs on the way, modelled on ZFS

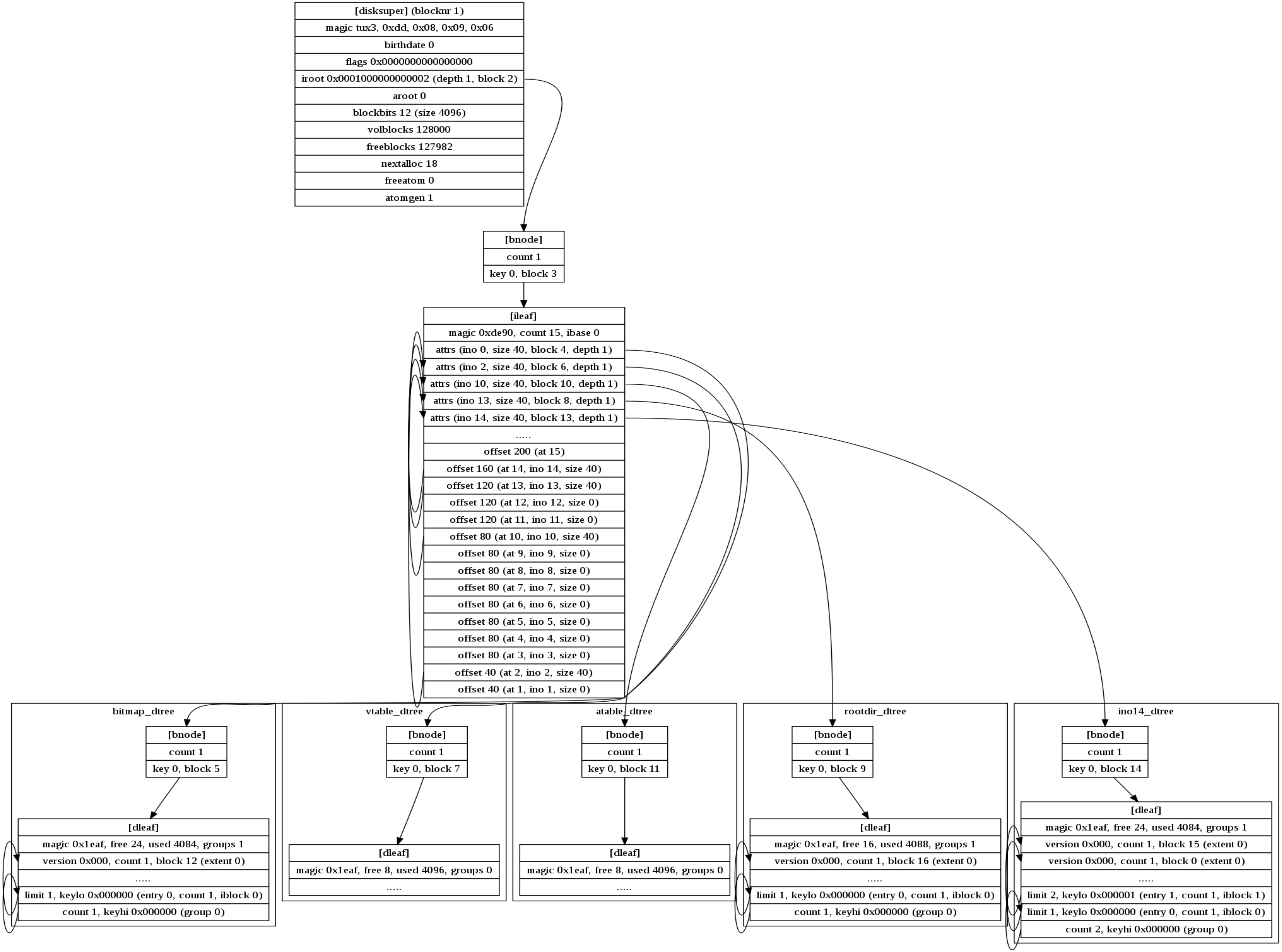
Non-Linux filesystems

Other Linux Filesystems

Tux3 is a Classic Design

Tux3 Filesystem Structure




[disksuper] (blocknr 1)
magic tux3, 0xdd, 0x08, 0x09, 0x06
birthdate 0
flags 0x0000000000000000
iroot 0x0001000000000002 (depth 1, block 2)
aroot 0
blockbits 12 (size 4096)
volblocks 128000
freeblocks 127982
nextalloc 18
freeatom 0
atomgen 1
[bnode]
count 1
key 0, block 3
[ileaf]
magic 0xde90, count 15, ibase 0
attrs (ino 0, size 40, block 4, depth 1)
attrs (ino 2, size 40, block 6, depth 1)
attrs (ino 10, size 40, block 10, depth 1)
attrs (ino 13, size 40, block 8, depth 1)
attrs (ino 14, size 40, block 13, depth 1)
.....
offset 200 (at 15)
offset 160 (at 14, ino 14, size 40)
offset 120 (at 13, ino 13, size 40)
offset 120 (at 12, ino 12, size 0)
offset 120 (at 11, ino 11, size 0)
offset 80 (at 10, ino 10, size 40)
offset 80 (at 9, ino 9, size 0)
offset 80 (at 8, ino 8, size 0)
offset 80 (at 7, ino 7, size 0)
offset 80 (at 6, ino 6, size 0)
offset 80 (at 5, ino 5, size 0)
offset 80 (at 4, ino 4, size 0)
offset 80 (at 3, ino 3, size 0)
offset 40 (at 2, ino 2, size 40)
offset 40 (at 1, ino 1, size 0)
bitmap_dtree
[bnode]
count 1
key 0, block 5
[dleaf]
magic 0x1eaf, free 24, used 4084, groups 1
version 0x000, count 1, block 12 (extent 0)
.....
limit 1, keylo 0x000000 (entry 0, count 1, iblock 0)
count 1, keyhi 0x000000 (group 0)
vtable_dtree
[bnode]
count 1
key 0, block 7
[dleaf]
magic 0x1eaf, free 8, used 4096, groups 0
.....
atable_dtree
[bnode]
count 1
key 0, block 11
[dleaf]
magic 0x1eaf, free 8, used 4096, groups 0
.....
rootdir_dtree
[bnode]
count 1
key 0, block 9
[dleaf]
magic 0x1eaf, free 16, used 4088, groups 1
version 0x000, count 1, block 16 (extent 0)
.....
limit 1, keylo 0x000000 (entry 0, count 1, iblock 0)
count 1, keyhi 0x000000 (group 0)
ino14_dtree
[bnode]
count 1
key 0, block 14
[dleaf]
magic 0x1eaf, free 24, used 4084, groups 1
version 0x000, count 1, block 15 (extent 0)
version 0x000, count 1, block 0 (extent 0)
.....
limit 2, keylo 0x000001 (entry 1, count 1, iblock 1)
limit 1, keylo 0x000000 (entry 0, count 1, iblock 0)
count 2, keyhi 0x000000 (group 0)

Versioned Pointers

- Actually: versioned extents and versioned attributes
- Each new write is labeled with the version in which it was written
- Follow the version inheritance graph to find data for a particular version
- Implements writable, recursive snapshots

Example version tree

```
.
`-- C '1003'
    `-- B '1002'
        |-- A '1001'
        `-- D
            |-- E '1005'
            `-- F
                |-- G '1007'
                `-- H '1006'
```

Version tree with exceptions

```
.
`-- C
    `-- B => p2
        |-- A => p1
        `-- D => p2
            |-- E => p2
            `-- F => p2
                |-- G => p2
                `-- H => p2
```

Exception list:

[[A, p1] [B, p2]]

Implied inheritance

```
.
`-- C
    `-- B [p2]
        |-- A [p1]
        `-- D
            |-- E
            `-- F
                |-- G
                `-- H
```

Exception list:

[[A, p1] [B, p2]]

Ghost Versions

Want to write to version A:

```
.
`-- A '1001'
    `-- B '1002'
```

- Cannot add exception to version A because it would be inherited by version B, violating isolation of snapshot 1002

Ghost Versions

Instead, add new version C to hold new exception [C, p1]

```
.
`-- A
    |-- B '1002'
    `-- C [p1] '1001'
```

- Move tag '1001' to version C
- Version A is now a ghost

Design Benefit of Versioned Pointers

- Versioning is done at a higher level, so does not require structural changes
- Can use a traditional structure where each allocated block is referenced exactly once
- Less metadata overall versus multiply rooted trees

Complexity Pushed to the Leaves

- Dleaf format, a mini btree
  - 8-12 bytes per extent including versioning
  - Has its own index
  - Tricky to update
- Extents add more complexity
- Versioning adds more complexity
- But the complexity is local, not distributed through the system

Taxonomy of Filesystems by Btree Structure

- Single Btree
  - Reiser, Btrfs, Hammer
- Multiple Btree
  - XFS, Ext4, Tux3

Taxonomy of Filesystems by Commit Method

- Journalling
  - Ext3, Ext4, XFS, JFS
- Copy on write
  - Reiser, ZFS, Btrfs, WAFL
- Logging
  - Logfs, Nilfs, Hammer
- Tux3 (something new)

Tux3 Atomic Commit Strategy

- Hybrid of logging and copy on write
- Log “promises” to update btree nodes
  - Dirty metadata index nodes are pinned in cache
  - Log enough data to reconstruct pinned cache on replay
  - Log blocks are written inline near data
  - Avoid seeking to far away places and writing metadata out of place

Tux3 Cache Model

- Physical Cache
  - Physical address given by cache index
  - Btree node and leaf blocks
- Logical Cache
  - Physical address stored in btree
  - Data files, directories, allocation bitmaps
  - Extended attribute atom tables
- All are mapped in page cache

Tux3 Cache Pipeline

- Frontend cache is operated on by user processes
- Backend cache is owned by Tux3
- Transfer dirty backend cache to disk while frontend cache changes asynchronously
- Introduce concept of buffer forking

Buffer Forking

- Copy on write cache block
  - Make a snapshot of dirty cache for transfer to disk
- Pull an in flight page out of cache, replace with copy...
- BUT multiple blocks share same page
  - Multiple tasks may read or write blocks on same page in parallel

Metadata Redirect

- Clean physically mapped cache blocks are always remapped when written
  - No need for forking
- Copy is done in cache, not read from disk
- Change parent in cache but not on disk
  - Log a promise instead

Delta Cycles

- Group changed blocks together in batches to take the filesystem from one consistent state to another
- Delta pipeline allows several deltas in flight simultaneously
  - Active, staging, writing
- Do not reuse freed blocks until delta has completed

Flush Cycles

- Periodic log fush writes “actual” metadata blocks
  - Redirect metadata block to new physical location, log “promise” to update parent
    - Avoids recursive copy to root
    - Consolidates multiple writes to same block in different deltas

Pinned Metadata

- Flush creates more pinned metadata
  - On-disk image is never “real” in normal operation
    - A part of the filesystem structure is always defined by the log
    - Except for special, optional flush on unmount
- For now, never flush log completely

Other Goodies

- Atom encoding of extended attributes
  - Long xattr names cost very little
  - New requirement to refcount atoms
- New PHTree directory index planned
  - Successor to Ext3/4 Htree
  - Handles NFS Abuse better
- Mixed bitmap and extent allocation map planned

Tux3 in multiple flavors

1) Tux3 userspace utility can read, write and create Tux3 filesystems
2) Mountable Tux3 FUSE filesystem
3) Virtualized Kernel filesystem
4) Kernel filesystem on real hardware

Development Model

- Majority of development is done in user space
- Also developing under User Mode Linux, KVM and VMWare
- Only recently, run on real machines
- Unit tests are key to low bug count

Development to present

- Started life as a userspace prototype
  - Ported Buffer layer to userspace
  - Borrowed initial code from Zumastor/ddsnap
- First mounted as a FUSE filesystem
  - Ported to FUSE by Conrad Meyer
  - Ported to low level FUSE by Tero Roponen
- Ported to kernel by Hirofumi Ogawa
  - Basic SMP locking by Christmas 08

Performance

Copy root filesystem to new partition

Tux3

real	9m41.554s
user	0m2.268s
sys	0m29.242s

Ext3

real	9m58.910s
user	0m3.040s
sys	0m31.086s

Next Steps

- In order of priority:
  - Atomic Commit
  - Begin review cycle
  - Allocation policy
  - Versioning
  - Directory Index
  - Extent allocation
  - Replication

Thanks to...

- Timothy Huber for cheerleading, graphics and extreme roller blading
- Shapor Naghibzahdeh for early hacking, slick web site, moral support
- Hirofumi Ogawa for amazing skill dedication and great code
- Many other members of the Tux3 Hall of Fame

Get Involved!

http://tux3.org

irc.oftc.net #tux3