Building a File System from Scratch

Mark Gritter

TINTRÍ

Minnebar
May 7, 2001

Storage Tech in Minnesota

- Compellent (now owned by Dell)
- Isilon (now owned by EMC)
- Xiotech (headquarters moved to Colorado)
- LSC (bought by Sun in 2001, developed QFS)
- Cray, Unisys, IBM, Seagate...

Why build a new file system?

- Scale to larger capacities or I/O levels
  - Isilon, LUSTRE, QFS
- New storage features
  - Data Domain, WAFL
- Adapt to media characteristics
  - LFS, Compellent, Xiotech ISE
- Optimize for a particular workload
- Provide file-like access to services
  - Plan 9, httpfs

Example: Kealia

- Feed pre-packetized video data (1MB chunks) into DRAM-based Streaming Switch (Sun X4950)
- Efficient large-block storage, scale to multiple TB over multiple RAID groups, efficiently stream data to network
  - Fortunately, XFS was already present in Linux and satisfies *most* of the requirements
  - Added custom modifications to ensure 1MB allocation size, and direct-to-network data path

(P.S. How many companies make it to acquisition without ever getting a logo?)

Example: Tintri

- VM-aware Storage
  - Storage specialized for virtual machines
    - Per-VM management, snapshotting, QoS, replication, …
    - NFS-based storage appliance
- Flash/SATA hybrid file system
  - Inline compression and deduplication for Flash
  - Integrated file system, neither pure cache nor tiering

TINTRÍ

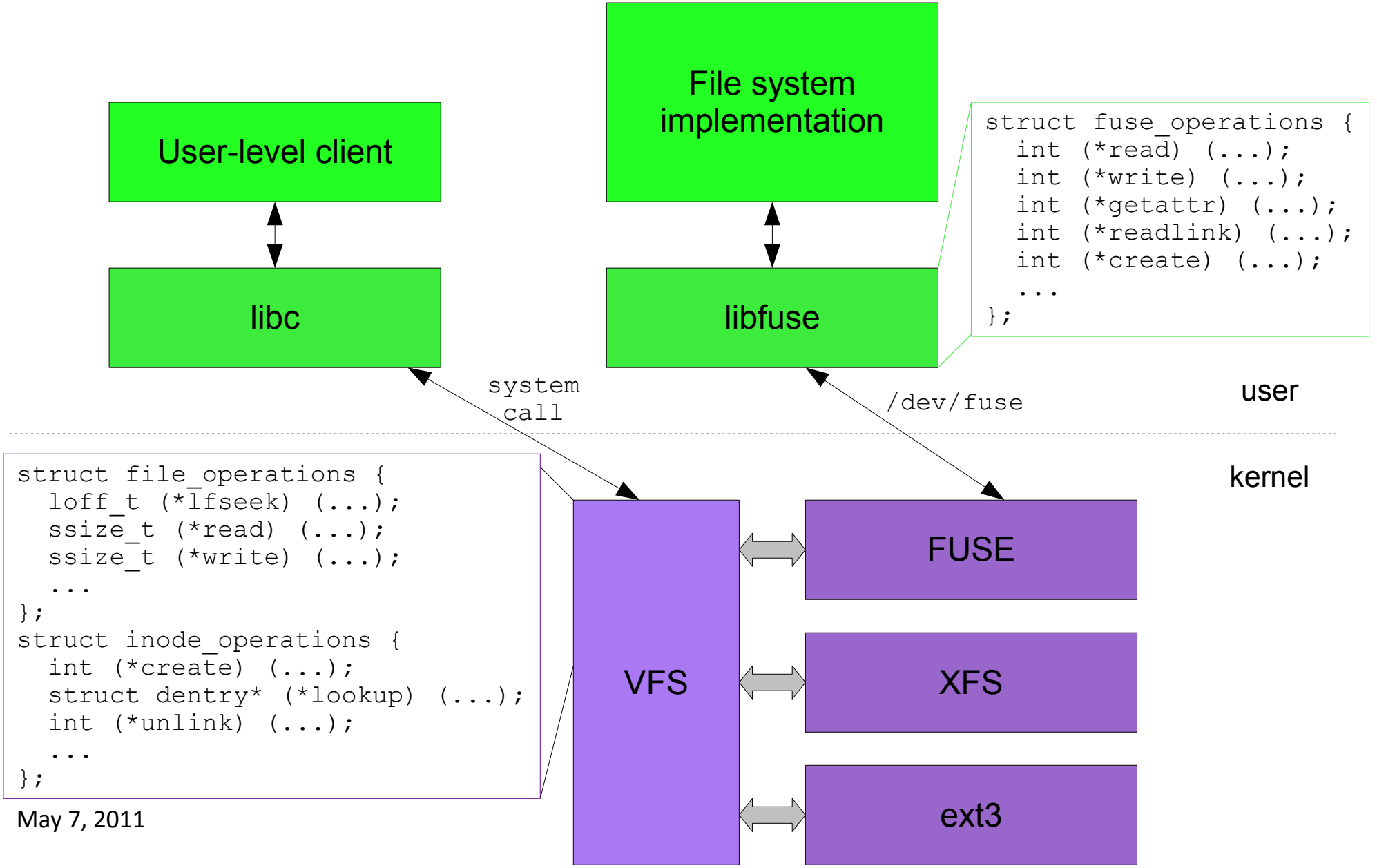
Bootstrapping a File System

- The wrong way:
  - Write a design document, split it up among a few teams, meet back in 9-12 months
  - Then spend another year or two on integration problems
- The right way:
  - Design for the long term but build piecewise
  - Identify ways to leverage existing systems to get started
  - Build incremental features and demonstrate a working system at every step

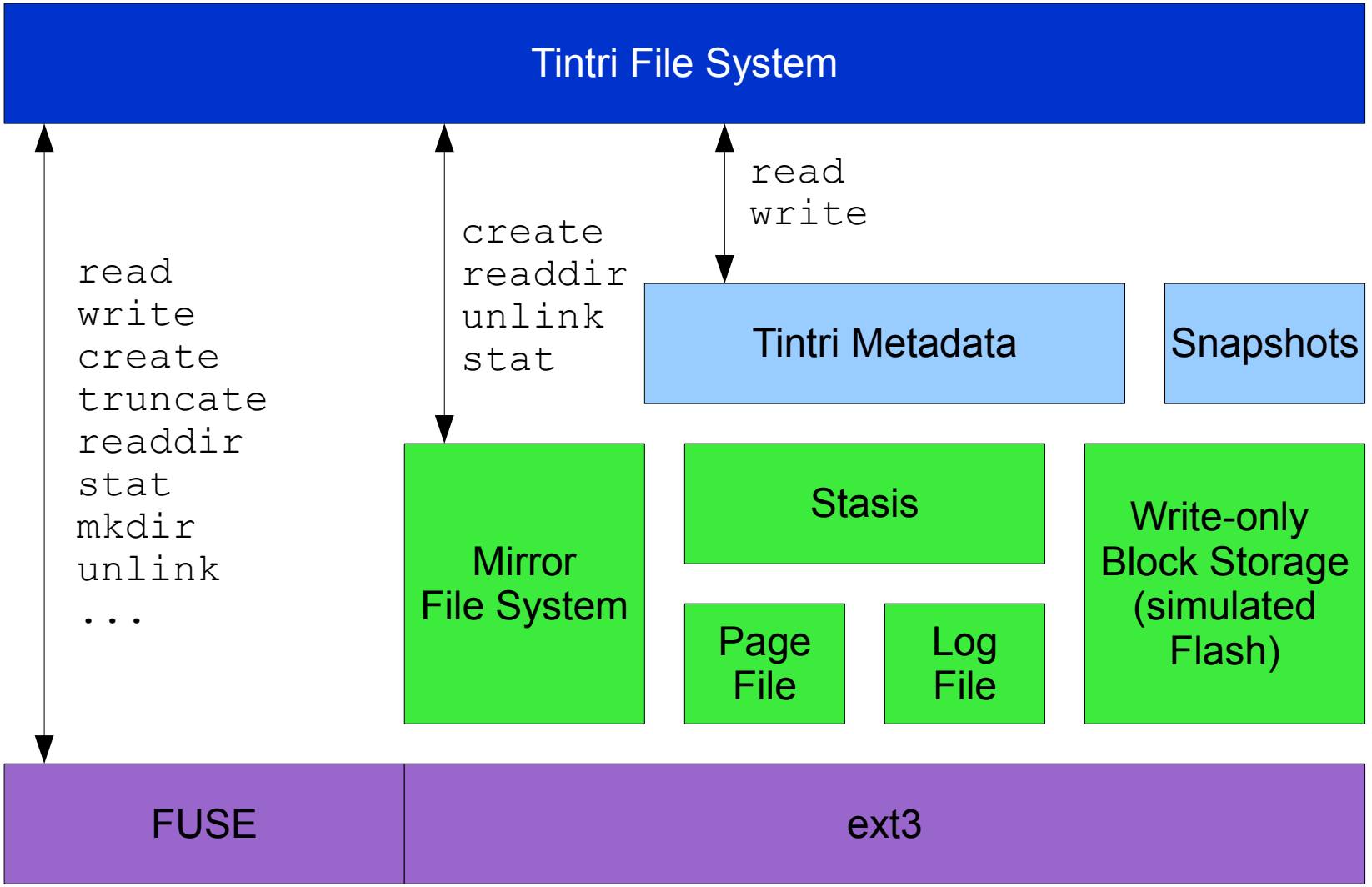
Bootstrapping a File System

- FUSE: File system in USEr space
  - Mainstream Linux kernel since 2.6.14, also available for Mac OS X, *BSD, OpenSolaris
  - http://fuse.sourceforge.net/
  - Lets user-space programs (and even non-privileged users) implement a file system, with no kernel modifications
  - libfuse provides a filename-oriented API that makes it very easy to get started--- no inodes, buffers, locking, etc.

FUSE in Operation



Tintri File System -- August 2009



FUSE + NFS

- Exporting a FUSE filesystem over NFS mostly works
  - File handles may not be persistent
  - But, allowed us to get real experience with VMware NFS traffic very early (using Linux Kernel NFS implementation)
- Read and write path turned out to be horribly inefficient, though:
  - NFS read = FUSE open, read, release, getattr
  - NFS write = FUSE open, getxattr, write, release, getattr

Other bootstrapping ideas

- Start using VFS directly (kernel module), but delegate to existing file system implementation
- Implement as network file system
  - NFS is reasonable, but only user-level open-source implementation I could find was single threaded and sort of old
  - CIFS is a beast but open-source implementations exist

Bootstrapping other components

- Emulate hardware dependencies with software modules or files on conventional storage
  - Developers can run on their own machines or VMs
  - Useful for unit-testing too
  - Tintri “desktop mode”: Flash, SATA disks, NVRAM as disk files
- Implement no-op or no-reclamation modes to start exercising internal interfaces

Design

- Storage architecture is hard:
  - Data must be persistent over scale of months and years. (Can't just drop a packet if we have a problem.)
  - Complex interaction of features
  - Performance trade-offs everywhere
  - Systems must scale beyond what can be easily tested
  - Storage protocols generally aren't end-to-end reliable systems, but storage products are expected to behave as though they were

Example: Tintri Design Decisions

- Block size? Multiple block sizes? Checksumming?
- Metadata format: Radix trees? B-trees? Skip lists?
- Snapshots: Redo log? Undo log? Copy-on-write?
- Deduplication: Reference counting? Garbage collection? Partial sharing? Identification of dups?
- Multi-threading, Locking, Queueing
- Consistency: Transactions? Intent logging?
- Data movement: Mechanisms, Policy?

Tintri and Stasis

- One early decision was to use a 3rd-party library called Stasis to access and update our metadata
  - Transactional support, paging, logging but a flexible approach to data model and locking
  - Allowed us to bootstrap very quickly, replace modules later to use real storage
  - See my talk from last Minnebar: http://bit.ly/m5FSb5

Tintri Low-level Storage Design

- 4KB blocks in Flash
  - Match common file system and application size
  - Minimize read-modify-write cycles for compressed data
  - Efficiently store just hot data in expensive flash
- Larger blocks on SATA storage
  - Get better throughput, less mapping overhead
  - Store disk metadata in Flash
    - Avoid multiple disk operations on a “cache miss”
    - Simplify design: no need to invent location-aware free lists, etc.
    - We had looked at using XFS or other file system for disk, but didn't want metadata overhead, nor reliance on Linux md RAID-6 implementation

Verifying a Design

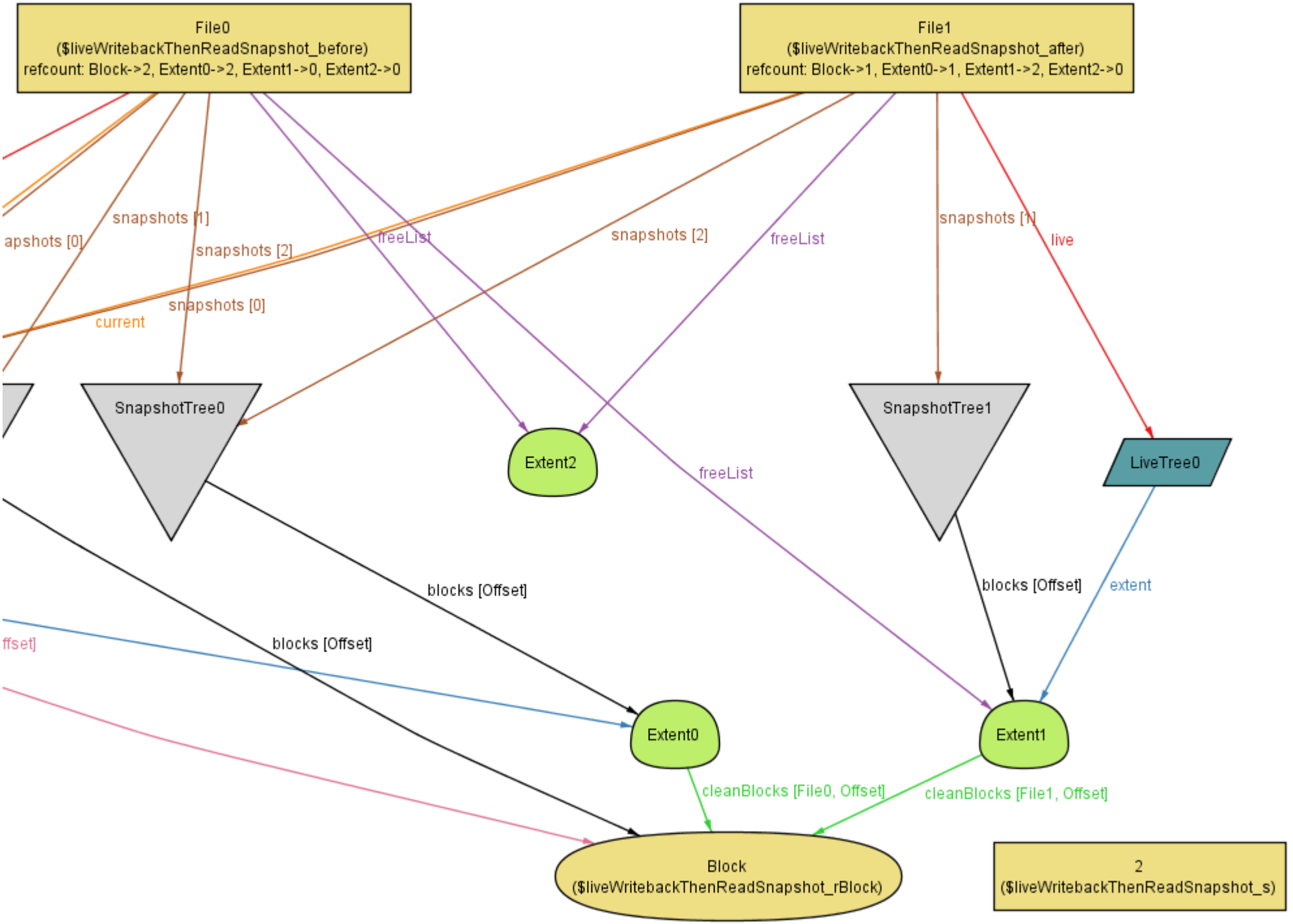
- Spreadsheet-ware: sizing, data rates, lifetime
- Slide-ware: design specifications and reviews
- Simulation: look at behavior of components under (real or synthetic) traces
  - Finding realistic data for deduplication was hard!
- Model verification
- Code and run-time analysis (assertions, counters)
- Stress testing (late in the process!)

Model checking

- Alloy: relational model with satisfiability solver
  - Best for “data structure” operations rather than network protocols: interleaved operations are harder to represent. (Look at something like SPIN instead.)
- Typical assertion looks something like:

```
all before, after : File |
        consistent[before] and
          someComplicatedOperation[before, after] =>
       consistent[after]
```

- Model checker finds counterexample, or proves up to limited number of objects


File0
($liveWritebackThenReadSnapshot_before)
refcount: Block->2, Extent0->2, Extent1->0, Extent2->0
File1
($liveWritebackThenReadSnapshot_after)
refcount: Block->1, Extent0->1, Extent1->2, Extent2->0
apshots [0]
snapshots [1]
snapshots [2]
snapshots [0]
current
freeList
snapshots [2]
freeList
snapshots [1]
live
SnapshotTree0
SnapshotTree1
LiveTree0
Extent2
freeList
blocks [Offset]
blocks [Offset]
blocks [Offset]
extent
ffset]
Extent0
Extent1
cleanBlocks [File0, Offset]
cleanBlocks [File1, Offset]
Block
($liveWritebackThenReadSnapshot_rBlock)
2
($liveWritebackThenReadSnapshot_s)

Future-Proofing Design

- Version numbers in structures to permit incremental change, polymorphism
- Extra space for new fields needed later
- Isolate layers as much as is feasible
- In implementation:
  - “Emergency reserve” space on storage array that can be released if the system is deadlocked
  - Mechanisms to identify and report bad behavior!

Testing

- Reproducibility is difficult
  - Traces, logging, counters, core analysis tools --- try to identify problems the first time
- Interaction of features usually where bugs crop up
  - Individual developers tend to be heads-down in a specific module
  - Performance is usually a multi-module problem
- Real-world load >> Synthetic load
  - All sorts of weird stuff happens when you hook up a real workload that you couldn't have predicted

TINTRÍ

- Founded in 2008, VC-funded
- First employee in January 2009, now ~35 people
- Started Beta in Q4 of 2010
- Launched in March, GA version released in April

Future File Systems?

- Interesting project by Brandon Salmon (now at Tintri): “Perspective: Semantic Data Management for the Home”
- Andrea and Remzi Arpaci-Dusseau's group at UW-Madison: lots of interesting work on reliable and smart storage
- New storage technologies: Shingled Magnetic Recording, NVRAM (of various sorts), Object Storage Devices

Thanks for Attending

Email: mgritter@gmail.com

Twitter: @markgritter

http://markgritter.livejournal.com/

http://www.tintri.com/