TrueErase: Full-storage-data-path Per-file Secure Deletion

Sarah Diesburg • Christopher Meyers • Mark Stanovich
Michael Mitchell • Justin Marshall • Julia Gould
An-I Andy Wang
Florida State University

Geoff Kuenning
Harvey Mudd College

Overview

- Problem
  - Per-file secure-deletion is difficult to achieve
    - Important for expired data, statute of limitations, etc.
- Existing solutions tend to be
  - Limited to a segment of legacy storage data path
  - File-system- or storage-medium-specific
- TrueErase
  - Storage-data-path-wide solution
  - Works with common file systems & storage media

The Problem

- Most users believe that files are deleted once
  - Files are no longer visible
  - The trash can is emptied
  - The partition is formatted
- In reality
  - Actual data remains

OTTAWA CITIZEN

Division of Canwest Publishing Inc.

vancouver 2010 OFFICIAL SUPPLIER

REGIONAL NEWSPAPER PUBLISHER

Home | News | Opinion | Business | Sports | Entertainment | Life | Health | Technolog

'Sensitive' data found on returned hard drive

Retired Carleton professor says Staples should have deleted files before reselling it

BY SARAH SCHMIDT, CANWEST NEWS SERVICE MARCH 23, 2009

COMPUTERWORLD

Storage

IT Careers

es bought on eBay hold

porate

Twitter ShareThis

The Tech H

Home Business Hardware Software

Man finds U.S. military secrets on secc

by Rich Bowden - Jan 27 2009, 04:43

Newspost Online

HOME NATIONAL WORLD NEWS FEATURES BUSINESS ENTERTAINMENT

OCT 7 2008

Mobile phones can never be totally wiped clean of data

Posted by admin in Sci-Tech

1 Comment

GIZMODO THE GADGET BLOG

Display Condensed search Most recent

iPHONE APPS

BESTMODO

Ads by Google

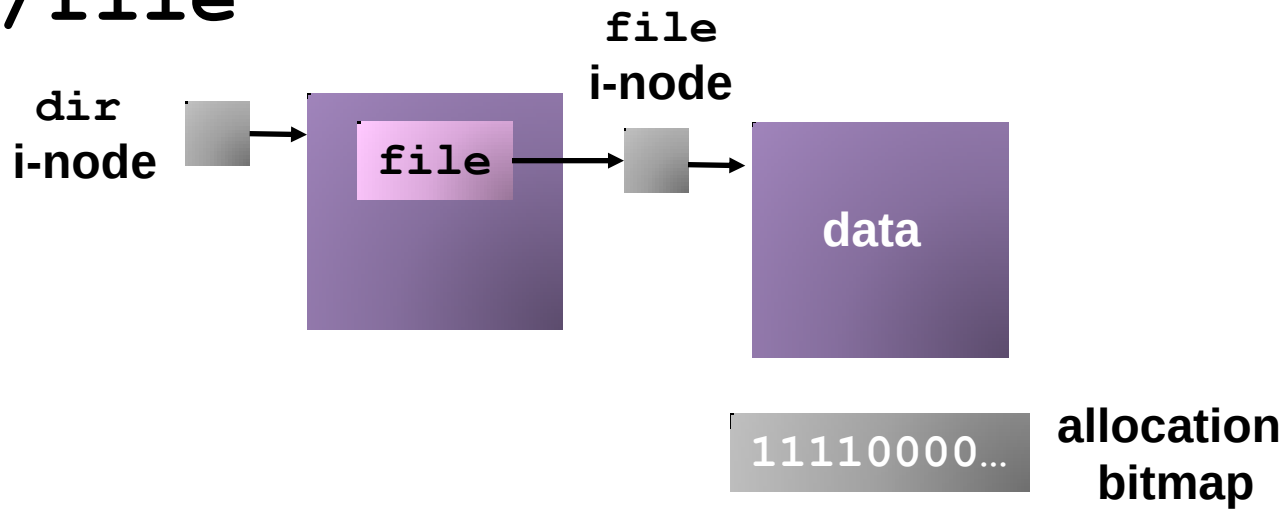
IPHONE

Refurbished iPhones Might Still Have Previous Owners' Personal Data, No Way to Erase It

By matt buchanan, 11:20 AM on Tue May 20 2008, 21,636 views

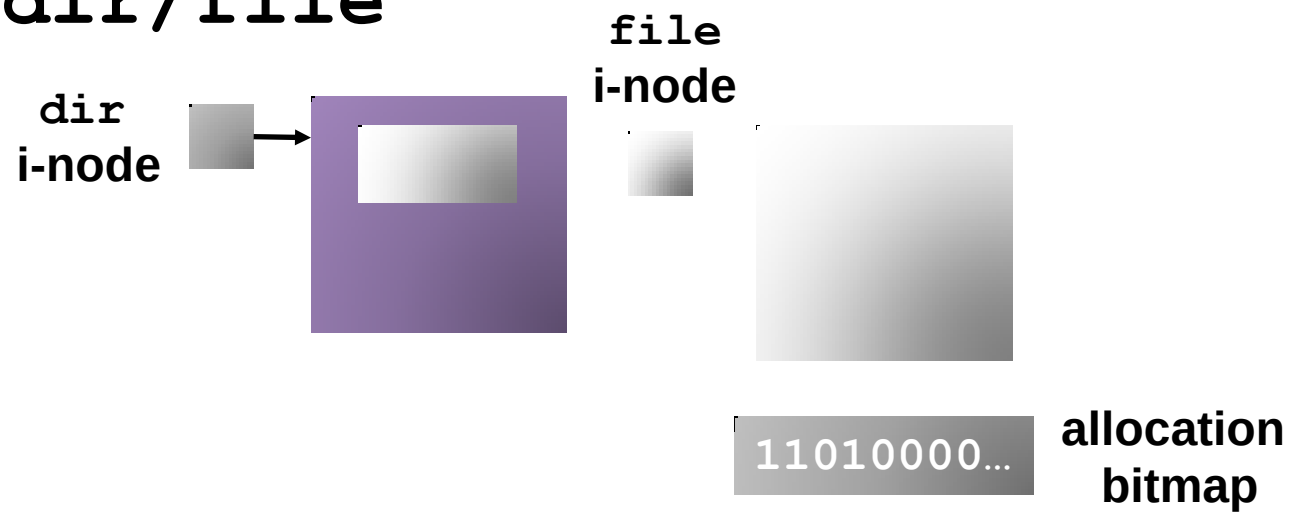
What is *secure deletion*?

- Rendering a file's deleted content and metadata (e.g., name) irrecoverable
- **`/dir/file`**



What is *secure deletion*?

- Rendering a file's deleted content and metadata (e.g., name) irrecoverable
- **`rm /dir/file`**



How hard can this be?

- Diverse threat models
  - Attacks on backups, live systems, cold boot attacks, covert channels, policy violations, etc.
- Our focus
  - Dead forensic attacks on local storage
    - Occur after the computer has been shut down properly

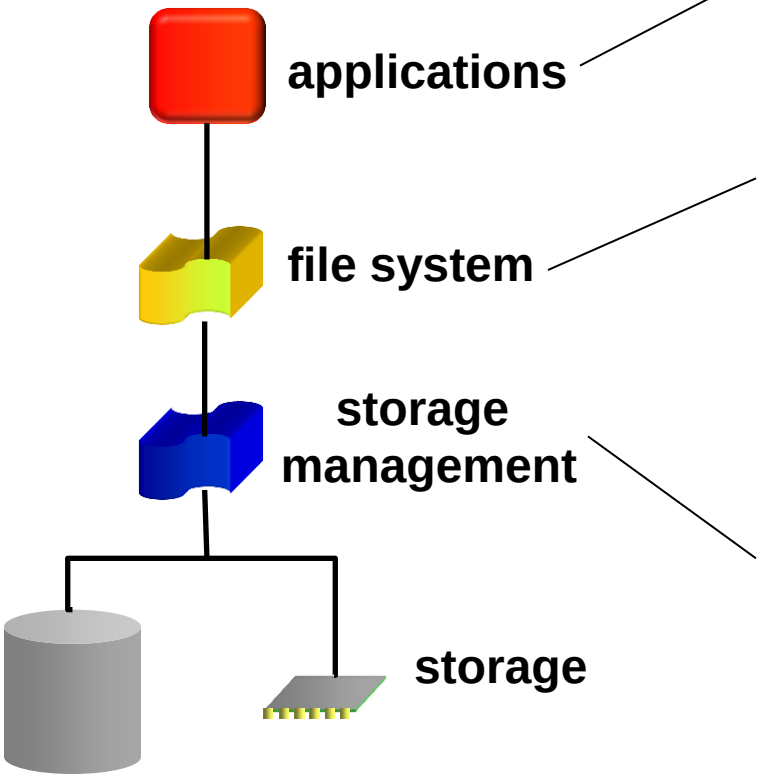
Basic Research Question

- Under the most benign environments
- What can we design and build to ensure that the secure deletion of a file is honored?
  - Throughout the legacy storage data path

TrueErase: A Storage-data-path-wide Framework

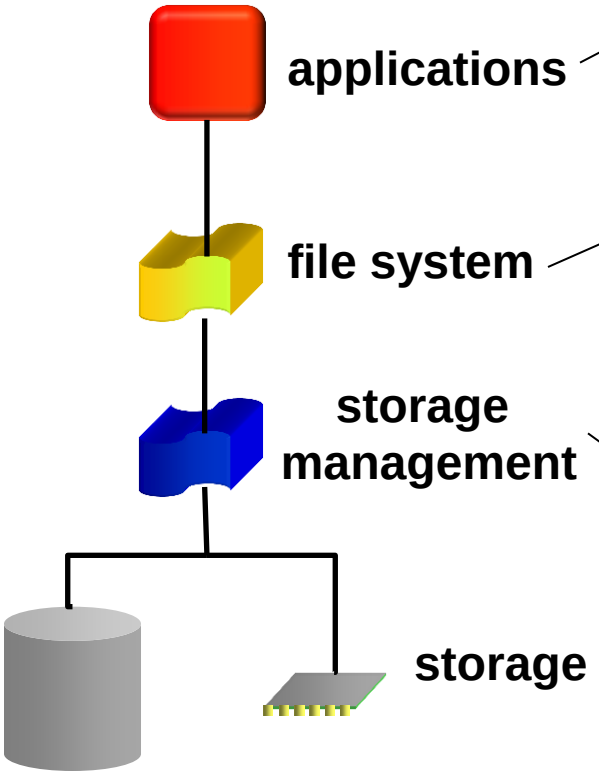
- Irrevocably deletes data and metadata
- Offers a unique combination of properties
  - Compatible with legacy apps, file systems, and storage media
  - Per-file deletion granularity
  - Solution covers the entire data path
  - Can survive common system failures
  - Core logic systemically verified

Legacy Storage Data Path



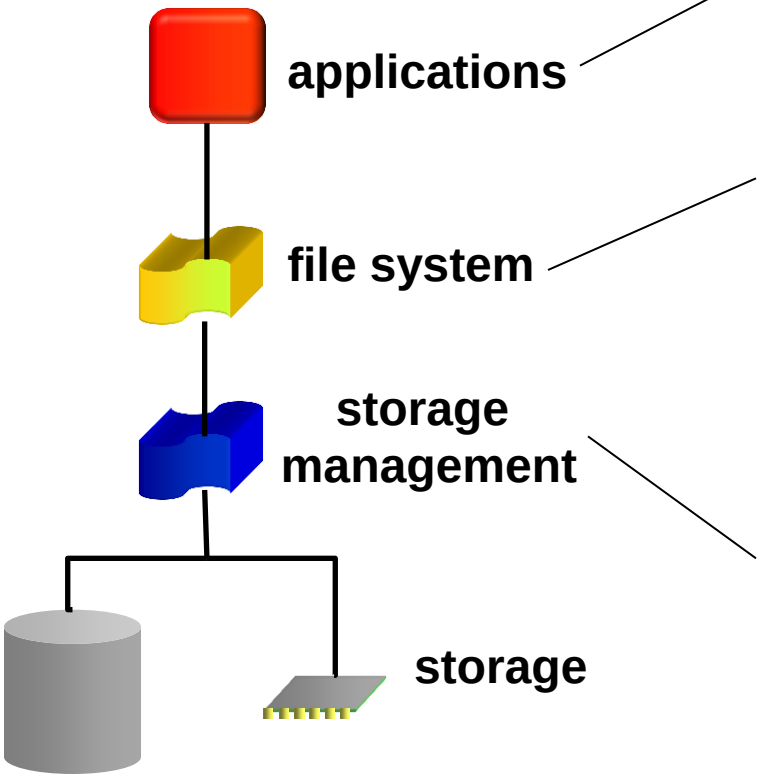
- Limited control over metadata
- Not aware of storage medium; limited control over storage locations
- No access to a block's type, file ownership, in-use status

Legacy Storage Data Path



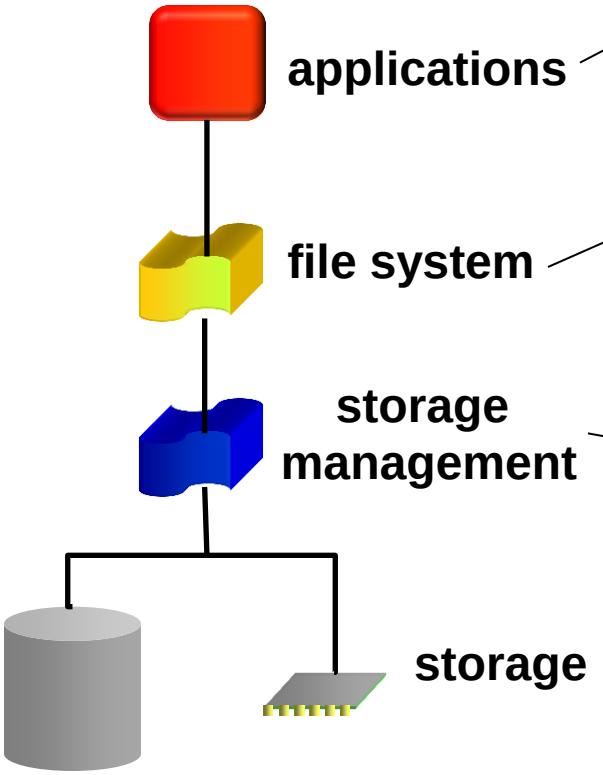
- Limited control over metadata
- Not aware of storage medium; limited control over storage locations
- No access to a block's type, file ownership, in-use status

Legacy Storage Data Path



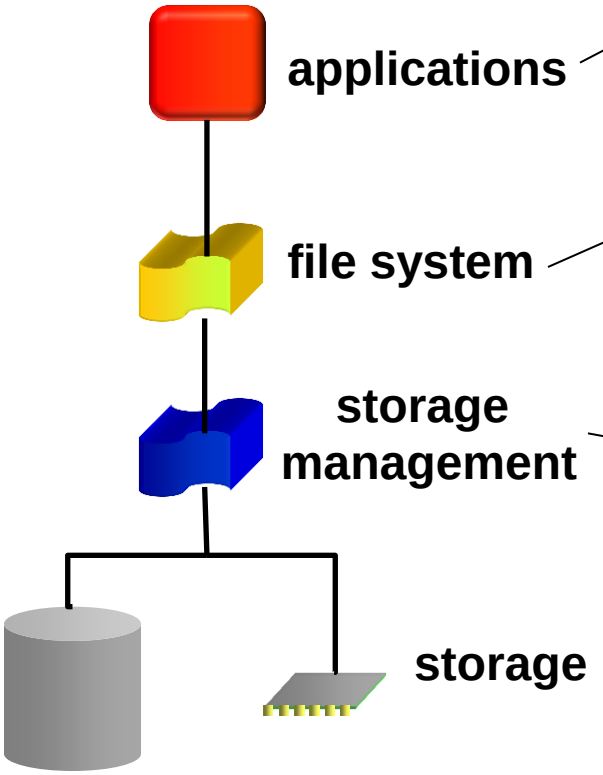
- Limited control over metadata
- Not aware of storage medium; limited control over storage locations
- No access to a block's type, file ownership, in-use status

Existing Secure-deletion Solutions



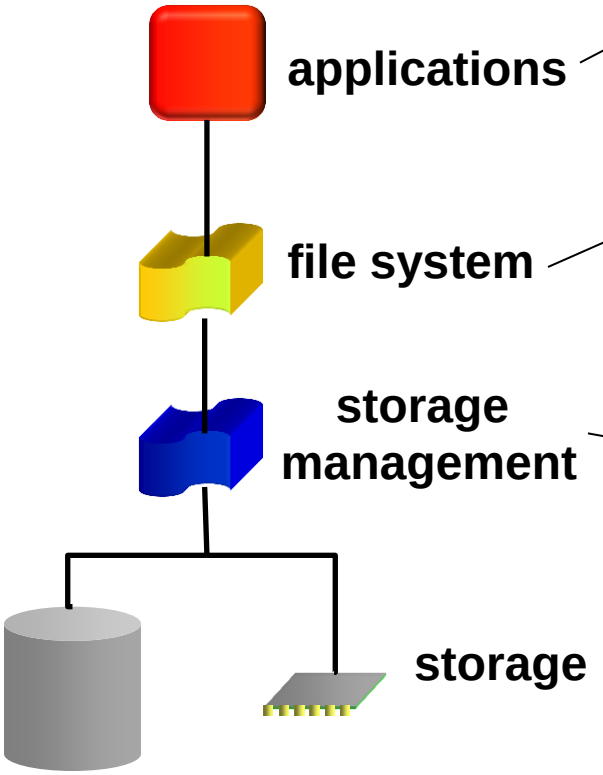
- May leak metadata information
- Cannot ensure ***in-place*** updates
  - Encryption will not help
- Hard to provide per-file solutions
- Cross-layer solutions tend to be file-system- and medium-specific

Existing Secure-deletion Solutions



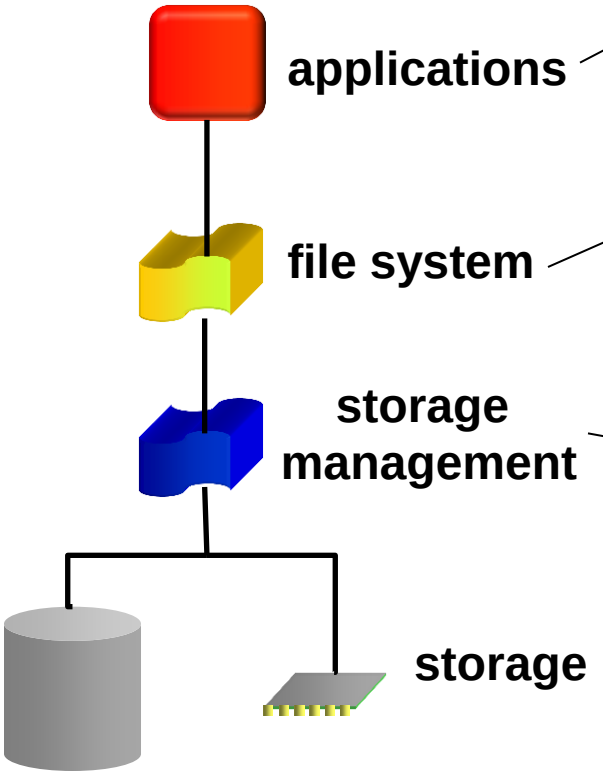
- May leak metadata information
- Cannot ensure ***in-place*** updates
  - Encryption will not help
- Hard to provide per-file solutions
- Cross-layer solutions tend to be file-system- and medium-specific

Existing Secure-deletion Solutions



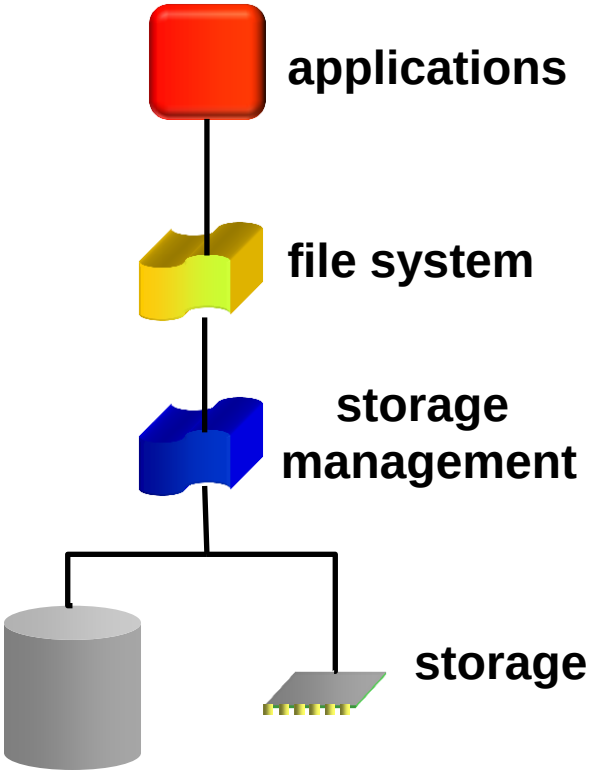
- May leak metadata information
- Cannot ensure ***in-place*** updates
  - Encryption will not help
- Hard to provide per-file solutions
- Cross-layer solutions tend to be file-system- and medium-specific

Existing Secure-deletion Solutions



- May leak metadata information
- Cannot ensure ***in-place*** updates
  - Encryption will not help
- Hard to provide per-file solutions
- Cross-layer solutions tend to be file-system- and medium-specific

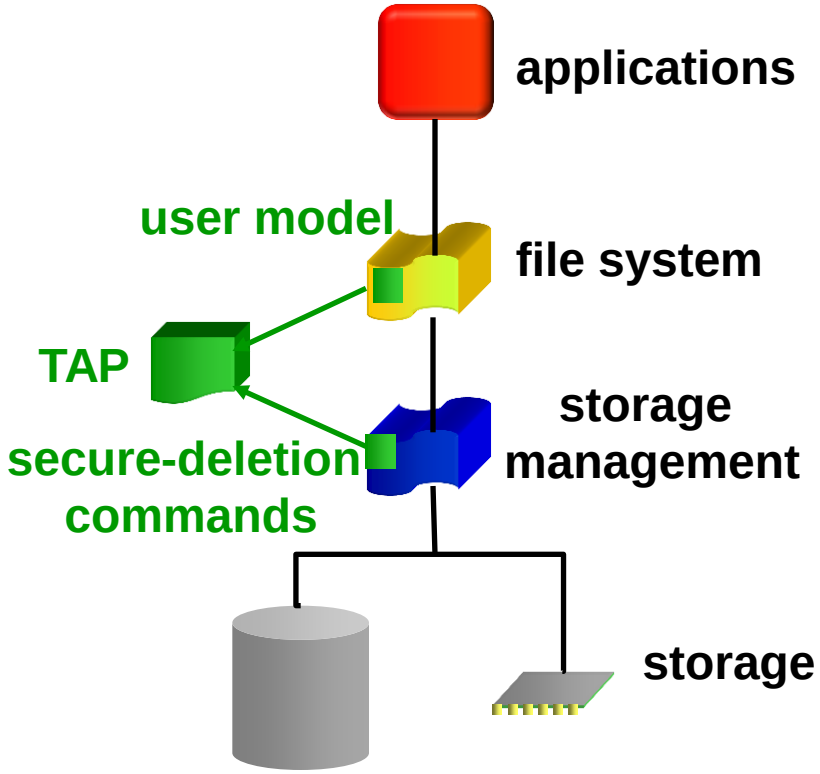
Other Secure-deletion Challenges



- No legacy requests to delete data blocks
  - For performance
- Legacy optimizations
  - Requests can be split, reordered, cancelled, consolidated, buffered, with versions in transit
- Lack of global IDs
- Crashes/verification

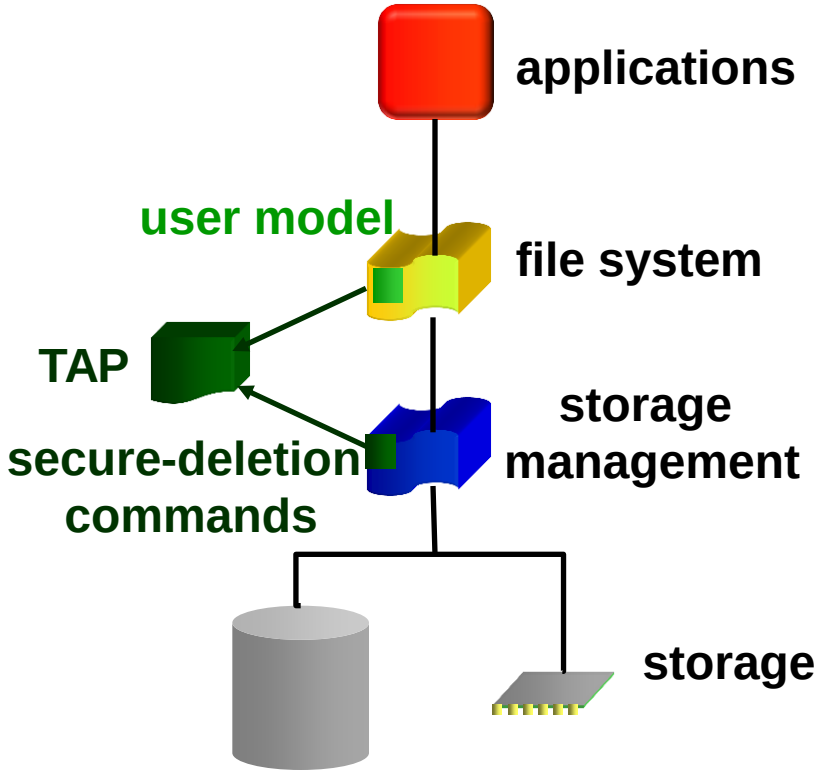
TrueErase Overview

- A centralized, per-file secure-deletion framework



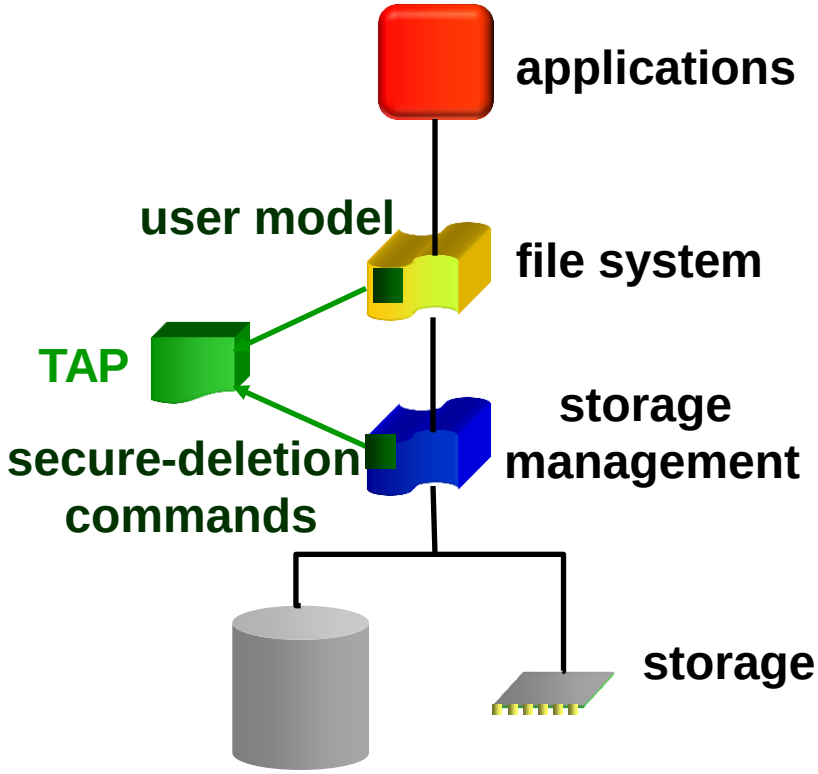
TrueErase Overview

- User model
  - Use extended attributes to specify files/dirs for secure deletion
  - Compatible to legacy applications



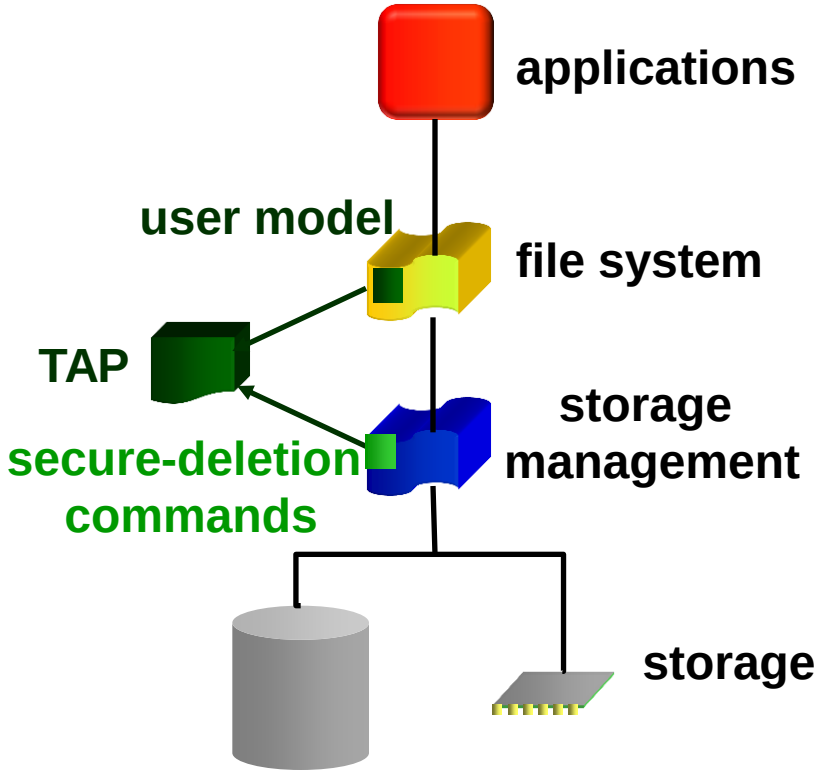
TrueErase Overview

- Type/attribute propagation module (***TAP***)
  - File system reports pending updates
    - Uses global unique IDs to track versions
  - Tracks only soft states
    - No need for mechanisms to recover states



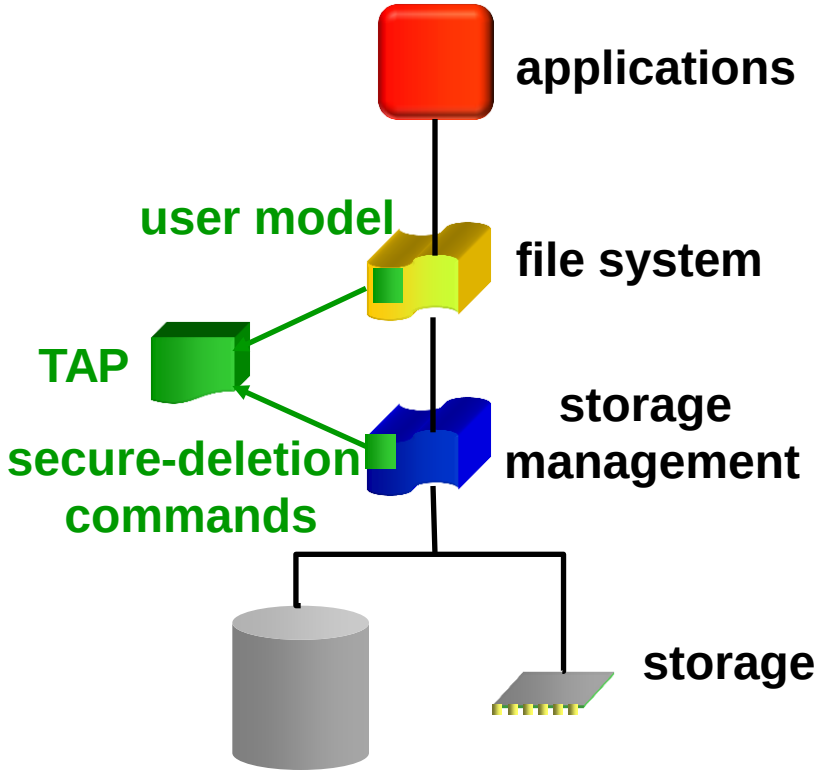
TrueErase Overview

- Enhanced storage-management layer
  - Can inquire about file-system-level info
  - Added secure-deletion commands for various storage media
  - Disabled some optimizations (e.g., storage-built-in cache)



TrueErase Overview

- After a crash
  - All replayed and reissued deletions are done securely
  - All data/metadata in the storage data path from prior session will be securely deleted



TrueErase Assumptions

- Benign personal computing environment
  - Uncompromised, single-user, single-file-system, non-RAID, non-distributed system
- Dead forensics attacks
- Full control of storage data path
- Journaling file systems that adhere to the consistency properties specified in [SIVA05]
- All updates are reported
- Does not handle user copies (no tainting)

TrueErase Design

- User model
- TAP
- Enhanced storage-management layer

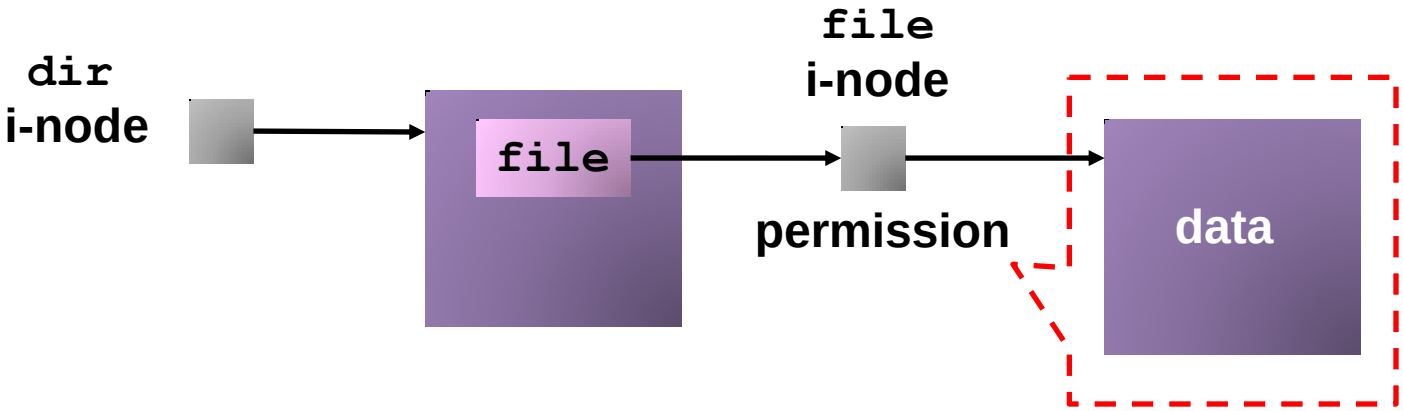
- Exploiting file-system-consistency properties to identify and handle corner cases

User Model

- Ideally, use traditional file-system permission semantics
  - Use extended-attribute-setting tools to mark files/dirs ***sensitive***
    - Which will be securely deleted from the entire storage data path
  - Legacy apps just operate on specified files/dirs

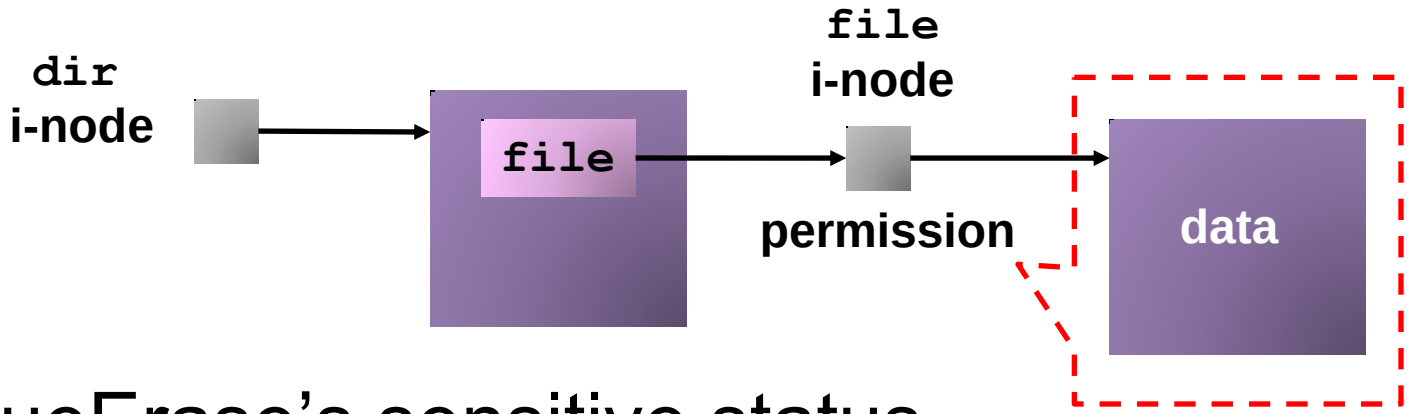
Name Handling

- Legacy file-permission semantics

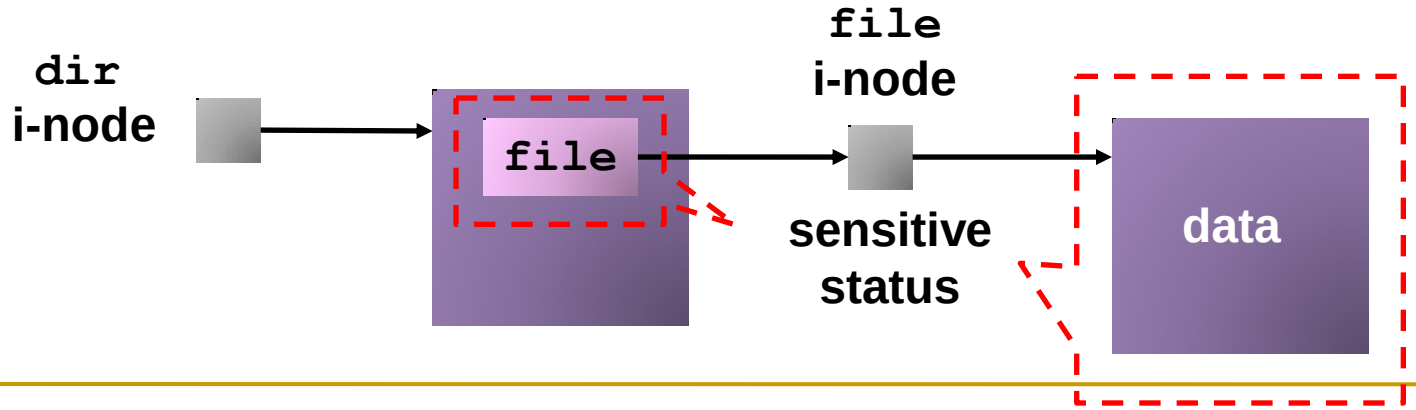


Name Handling

- Legacy file-permission semantics



- TrueErase's sensitive status



Toggling of the Sensitive Status

- Implications
  - Tracking update versions for all files at all times
  - Or, removing old versions for all files at all times
- TrueErase
  - Enforces secure deletions for files/dirs that have stayed sensitive since their creation

Name Handling

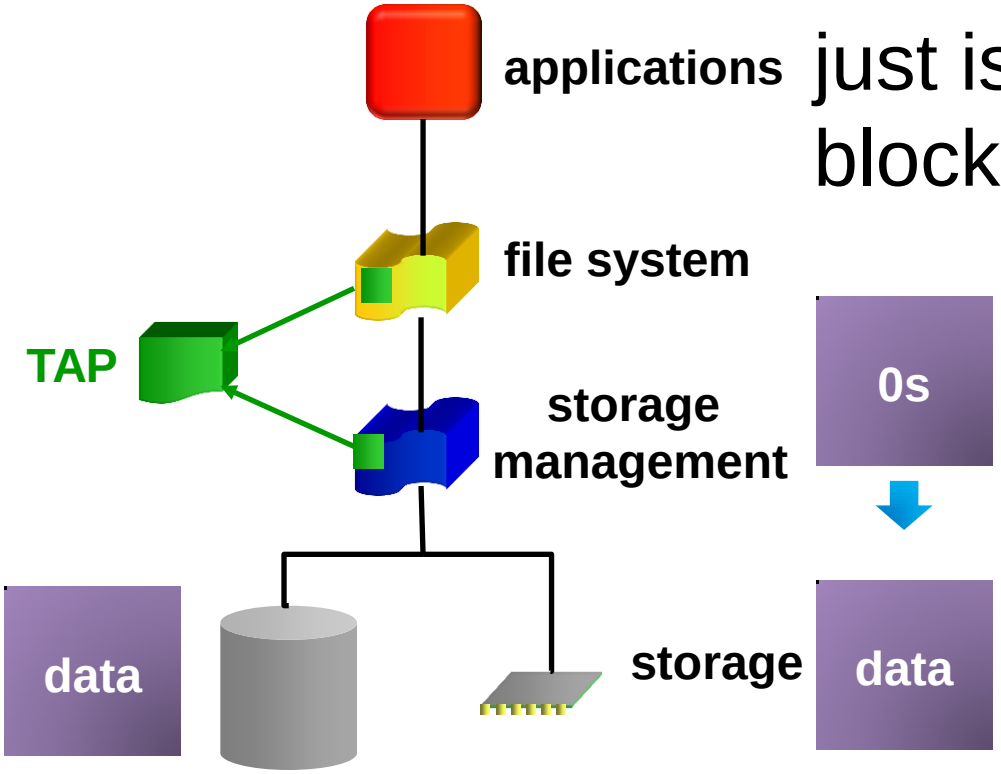
- By the time one can set attributes of a file
  - File name may already be stored non-sensitively
- Some remedies
  - Inherit the sensitive status
    - Creating a file under a sensitive directory
  - smkdir wrapper script
    - Creates a temporary name, marks it sensitive, and renames it to the sensitive name

TAP Module

- Tracks and propagates info from file-system layer to storage-management layer
- Challenges
  - Where to instantiate the deletion requests to file content?
  - What and how to track?
  - How to interact with TAP?

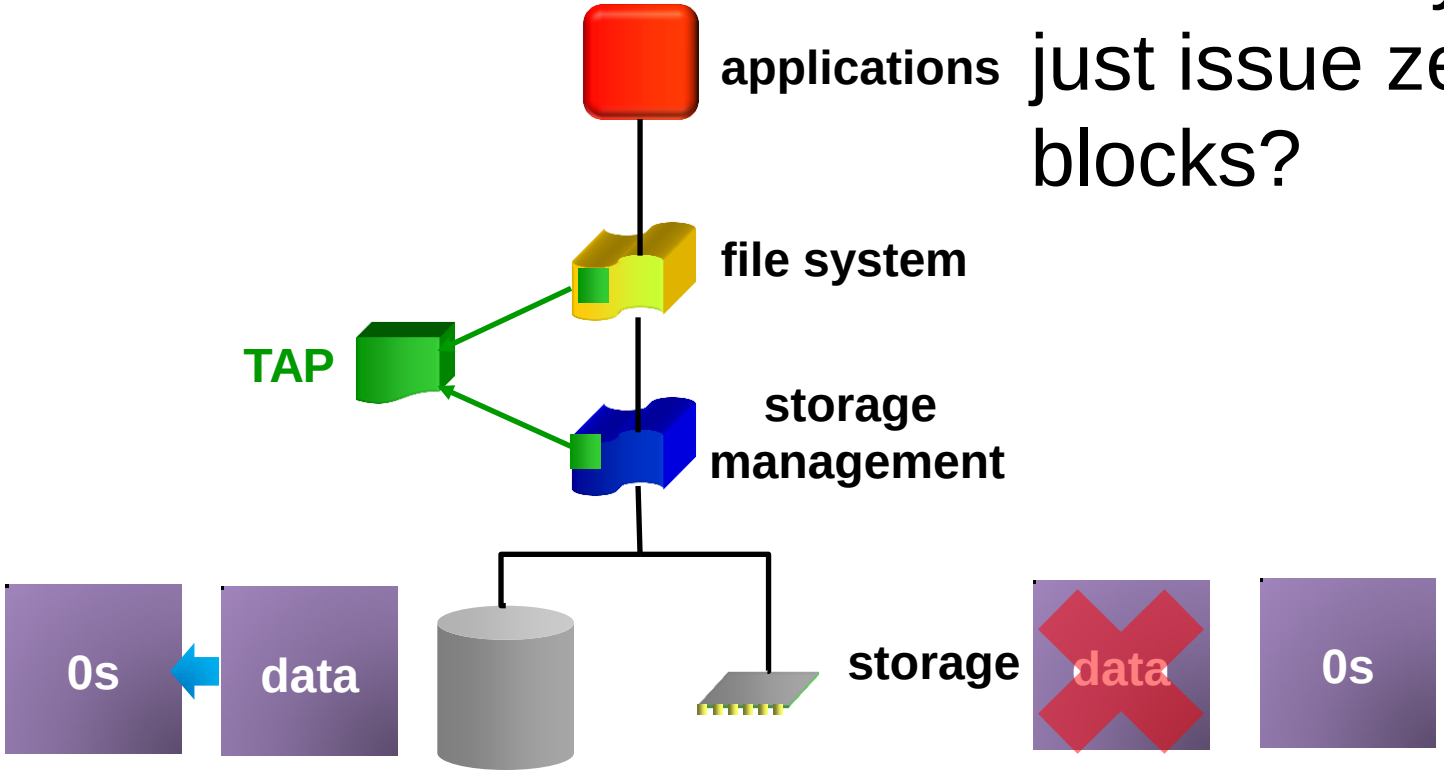
Where to instantiate deletion requests to file content?

- Can a file system just issue zeroed blocks?

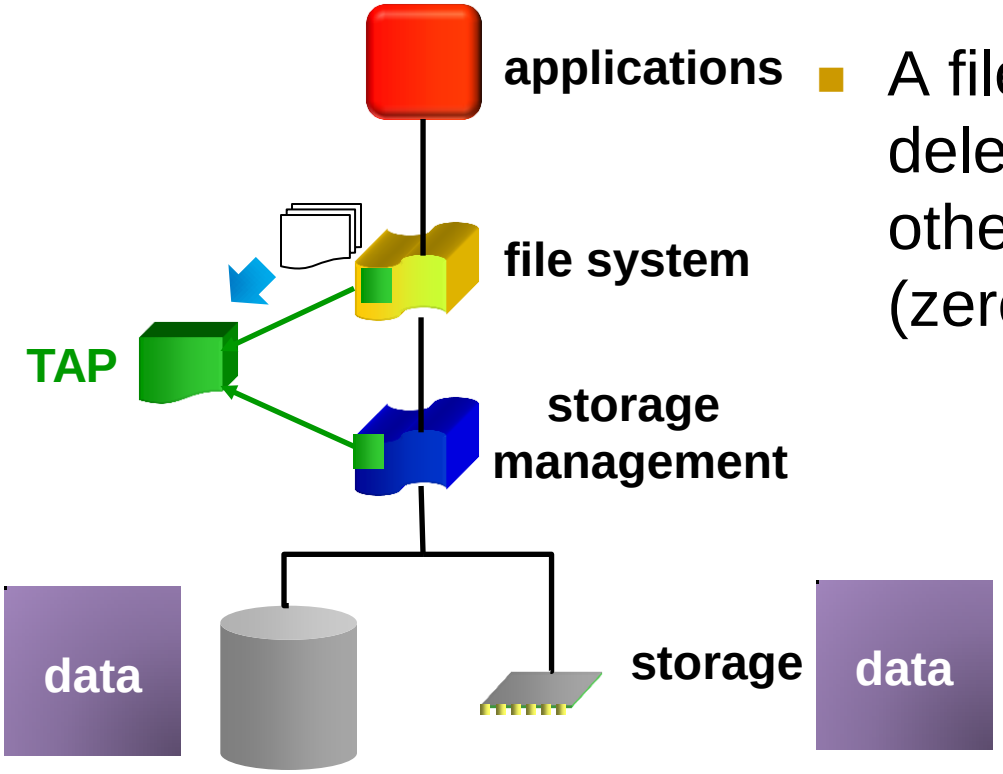


Where to instantiate deletion requests to file content?

- Can a file system just issue zeroed blocks?

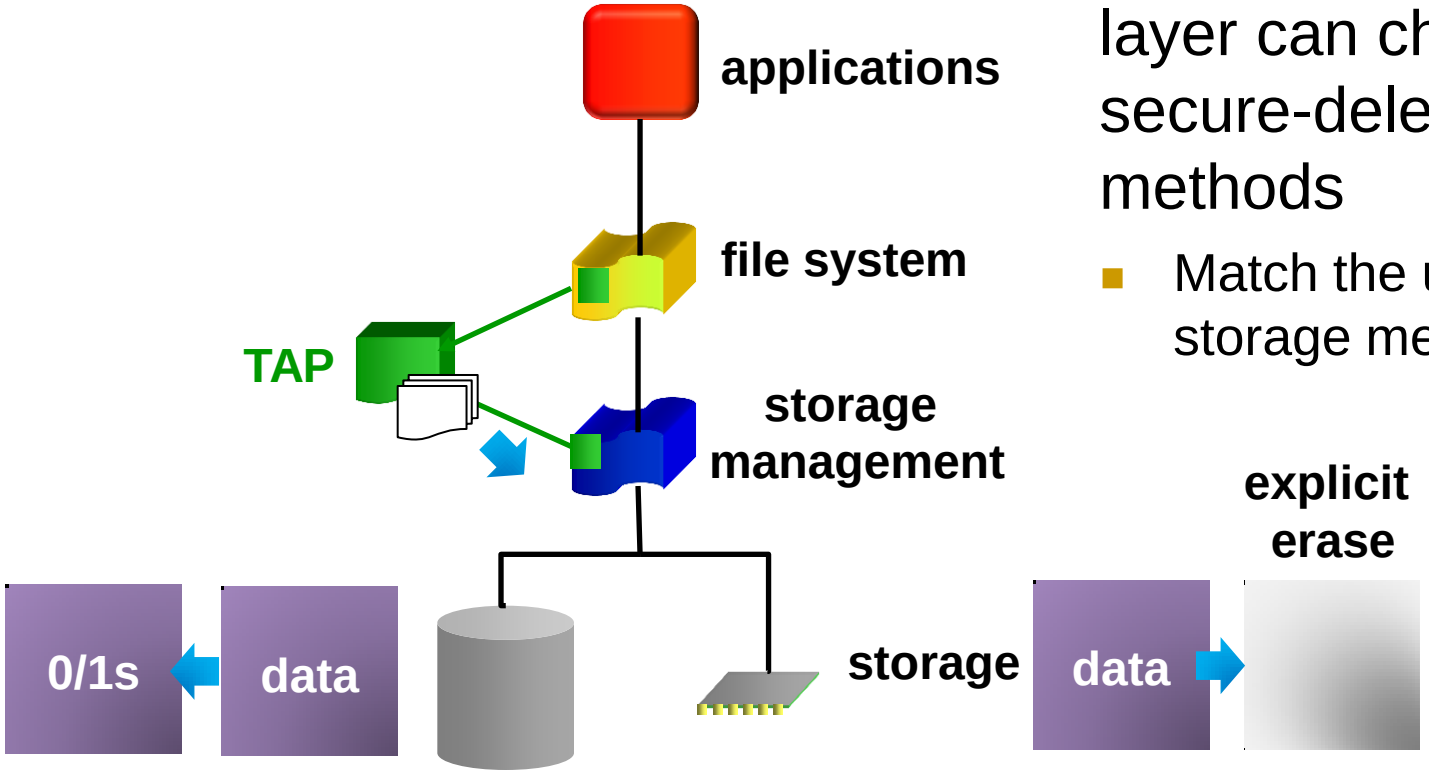


Where to instantiate deletion requests to file content?



- Instead
  - A file system attaches deletion reminders to other deletion requests (zeroing allocation bits)

Where to instantiate deletion requests to file content?



- Storage-management layer can choose secure-deletion methods
  - Match the underlying storage medium

What to track?

- Tracking deletion is not enough
  - At the secure-deletion time
    - Versions of a file's blocks may have been stored
    - Metadata may not reference to old versions
      - Need additional persistent states to track old versions
- TrueErase deletes old versions along the way
  - Overwriting a sensitive data

    = Secure deletion + update (***secure write***)
  - Tracks all in-transit sensitive updates

What to track?

- Tracking sensitive updates is still not enough
  - Metadata items are small
  - A metadata block can be shared by files with mixed sensitive status
    - A non-sensitive request can make sensitive metadata appear in the storage data path
- TrueErase tracks all in-transit updates
  - For simplicity and verification

How to track?

- Challenges
  - Reuse of name space (i-node number), data structures, memory addresses
  - Versions of requests in transit
- TrueErase
  - Global unique page ID per memory page

Tracking Granularity

- TrueErase tracks physical sector numbers (e.g., 512B)
  - Smallest update unit
  - GUID: global unique page ID + sector number

How to interact with TAP?

- Report_write() creates a per-sector tracking entry
- Report_delete() attaches deletion reminders to a tracking entry
- Report_copy() clones a tracking entry and transfers reminders
- Cleanup_write() deletes a tracking entry
- Check_info() retrieves the sensitive status of a sector and its reminders

Enhanced Storage-management Layer

- Decide which secure-deletion method to use
  - Based on the underlying storage medium
  - We used NAND flash for this demonstration

NAND Flash Basics

- Writing is slower than reading
  - Erasure can be much slower
- NAND reads/writes in ***flash pages***
  - Deletes in ***flash blocks***
    - Consisting of contiguous pages

NAND Flash Basics

- In-place updates are not allowed
  - Flash block containing the page needs to be erased before being written again
    - In-use pages are migrated elsewhere
- Each location can be erased 10K -1M times

Flash Translation Layer (FTL)

- To optimize performance
  - ***FTL*** remaps an overwrite request to an erased empty page
- To prolong the lifespan
  - ***Wear leveling*** evenly spreads the number of erasures across storage locations

Added NAND Secure-deletion Commands

- Secure_delete(pages)
  - Copies other in-use pages from the current flash block to elsewhere
  - Issue erase command on the current block
- Secure_write(page)
  - Write the new page
  - Call Secure_delete() on the old (if applicable)

Crash Handling

- A crash may occur during a secure operation
  - Page migration may not complete
- Since copies are done first
  - No data loss; but potential duplicates
  - Journal recovery mechanisms will reissue the request, and secure operations will continue

Wear Leveling

- When flash runs low on space
  - Wear leveling compacts in-use pages into fewer flash blocks
- Problem: internal storage reorganization
  - No respect for file boundaries, sensitive status

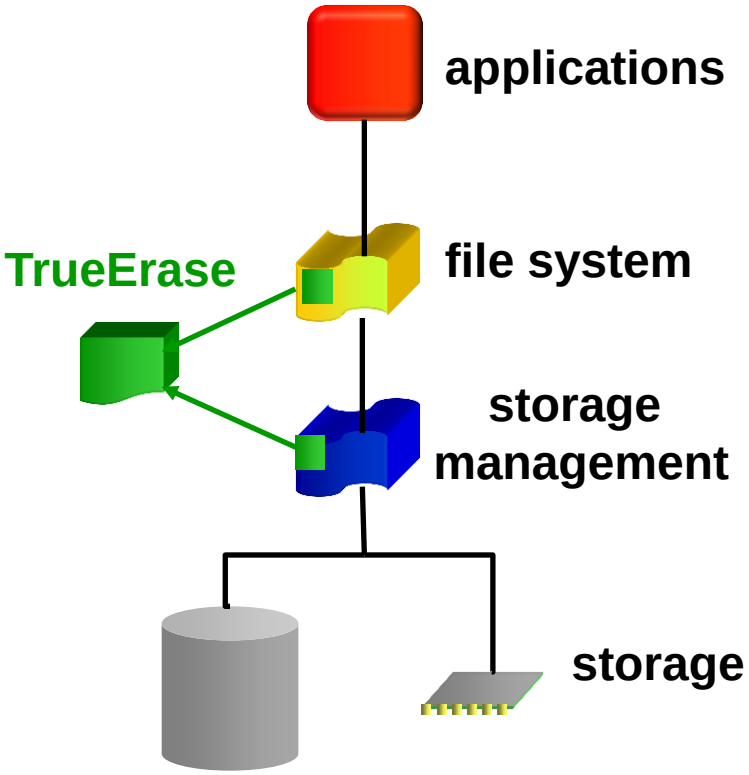
Wear Leveling

- TrueErase
  - Stores a sensitive-status bit in per-page control areas
    - Used to enforce secure-deletion semantics
  - May not always be in sync with the file-system-level sensitive status
    - E.g., short-lived files
    - When the bit disagrees with file system's secure status, mark the bit sensitive and treat it as such

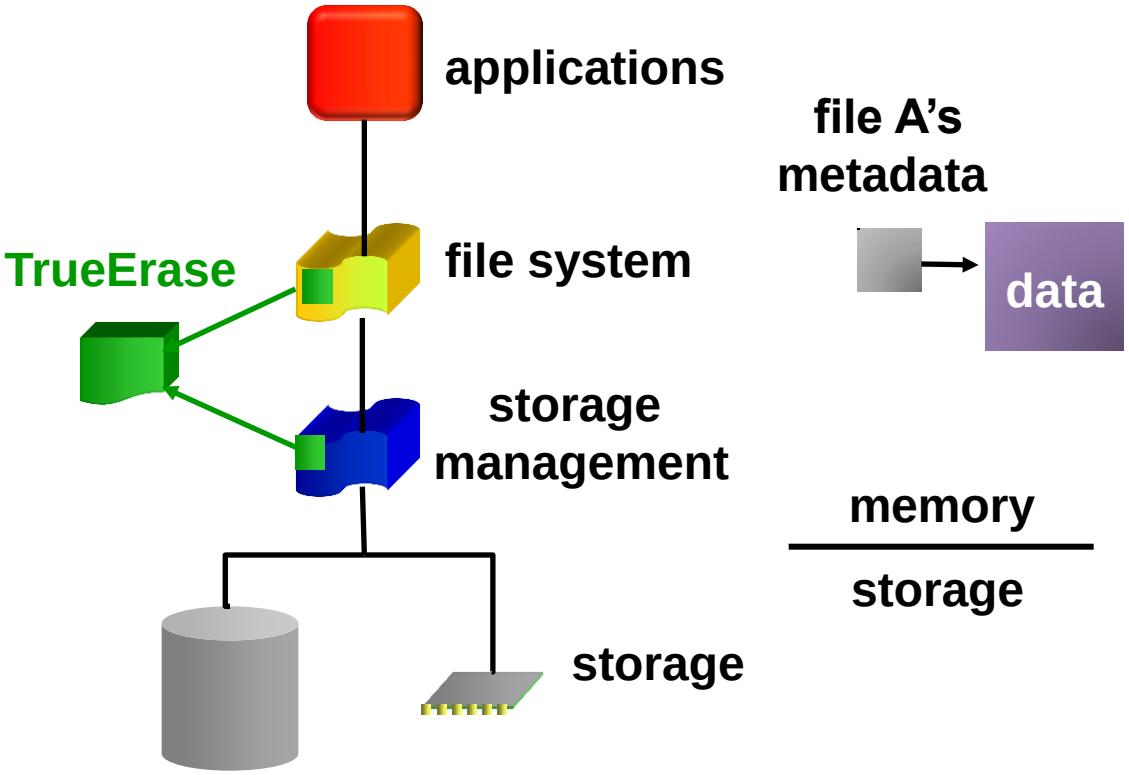
File-system-consistency Properties and Secure Deletion

- File-system-consistency properties
  - A file's metadata reference the right data and metadata versions throughout the data path
- For non-journaling file systems
  - ***Reuse-ordering*** & ***pointer-ordering properties***
  - Without both (e.g., ext2), a file may end up with blocks from another file
- For journaling file systems
  - ***Non-rollback property***

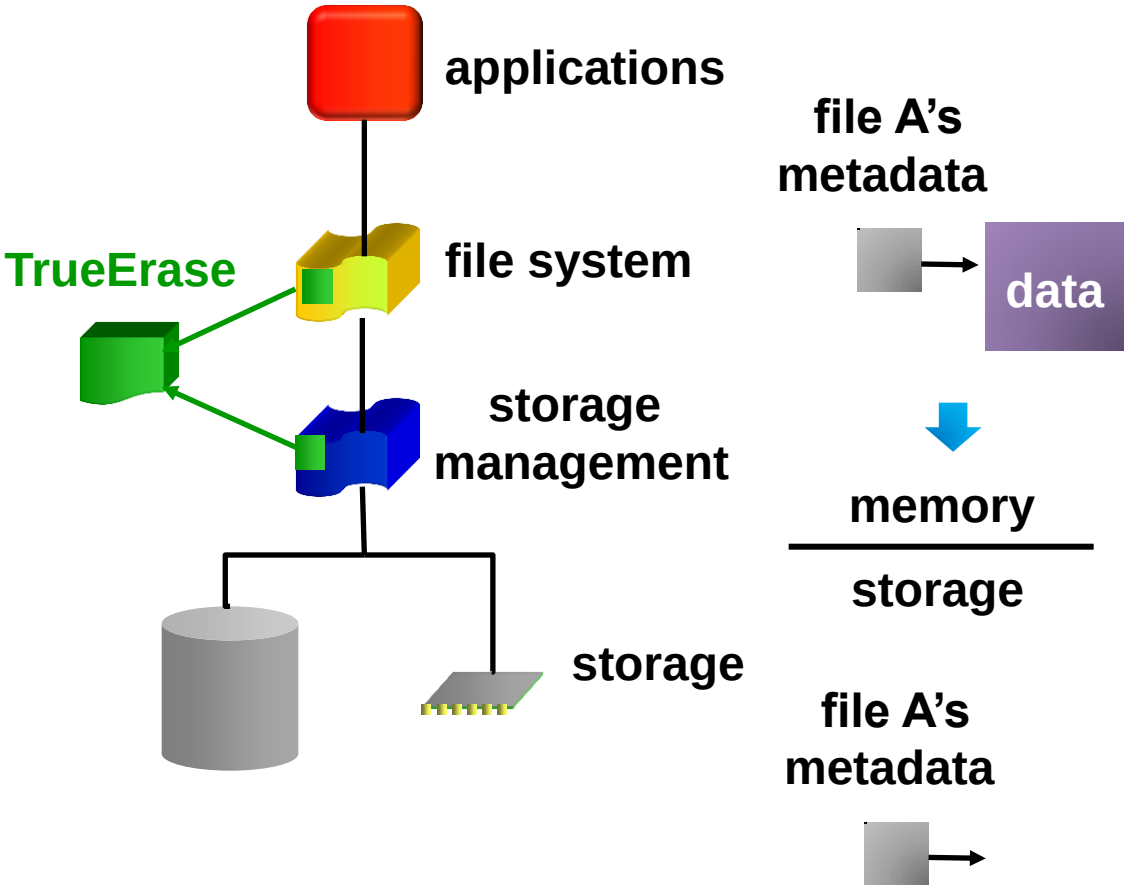
Without Pointer-ordering Property



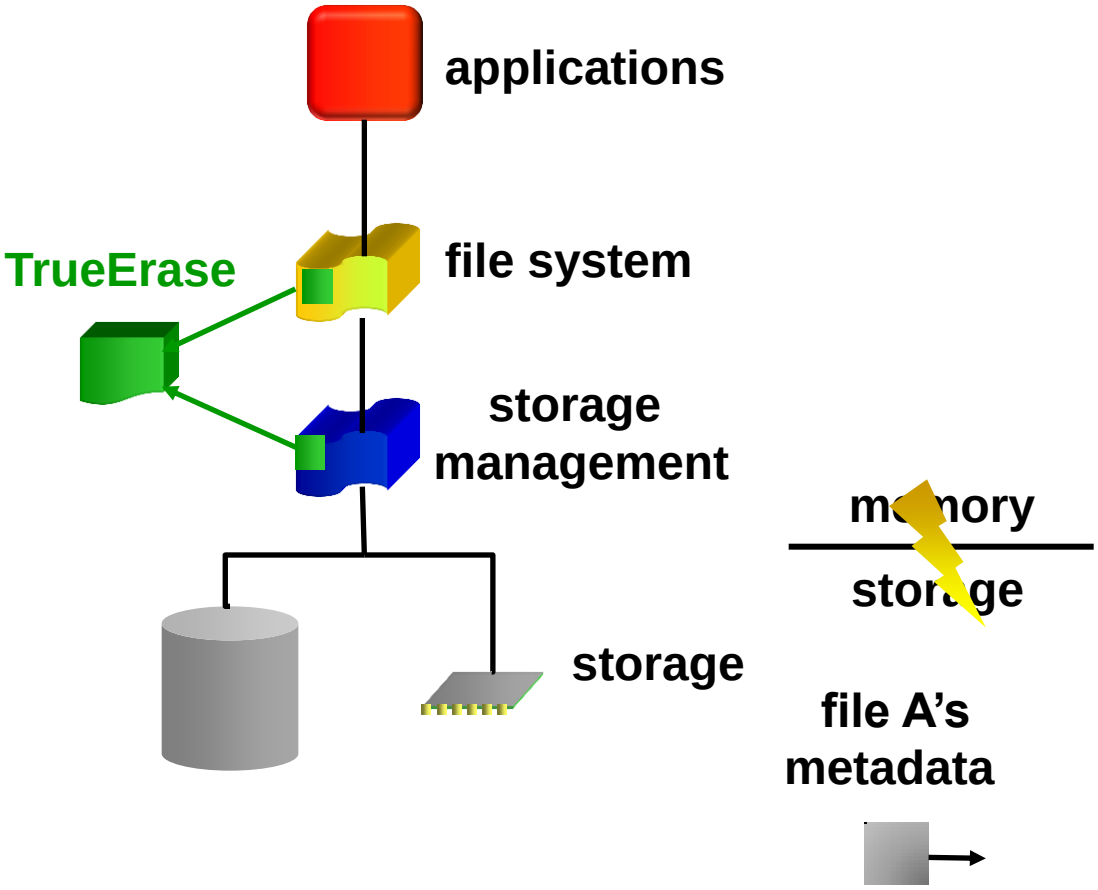
Without Pointer-ordering Property



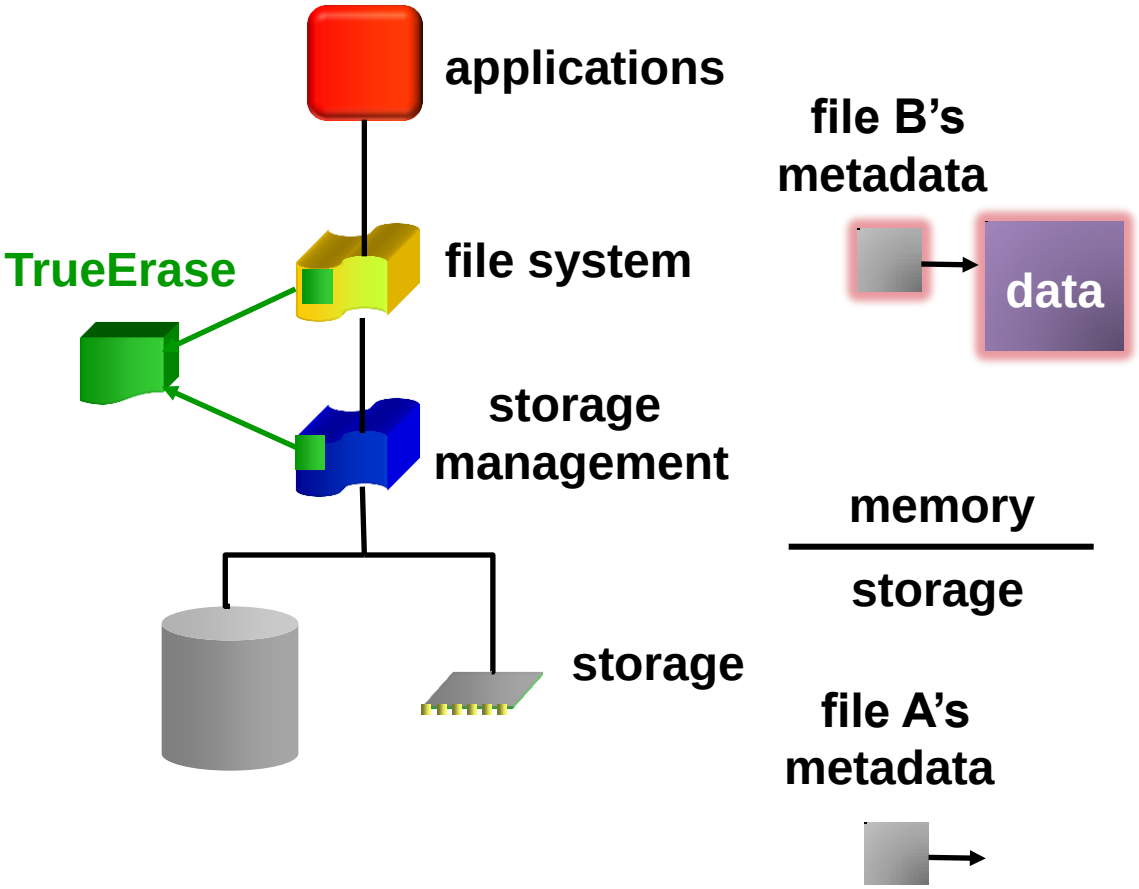
Without Pointer-ordering Property



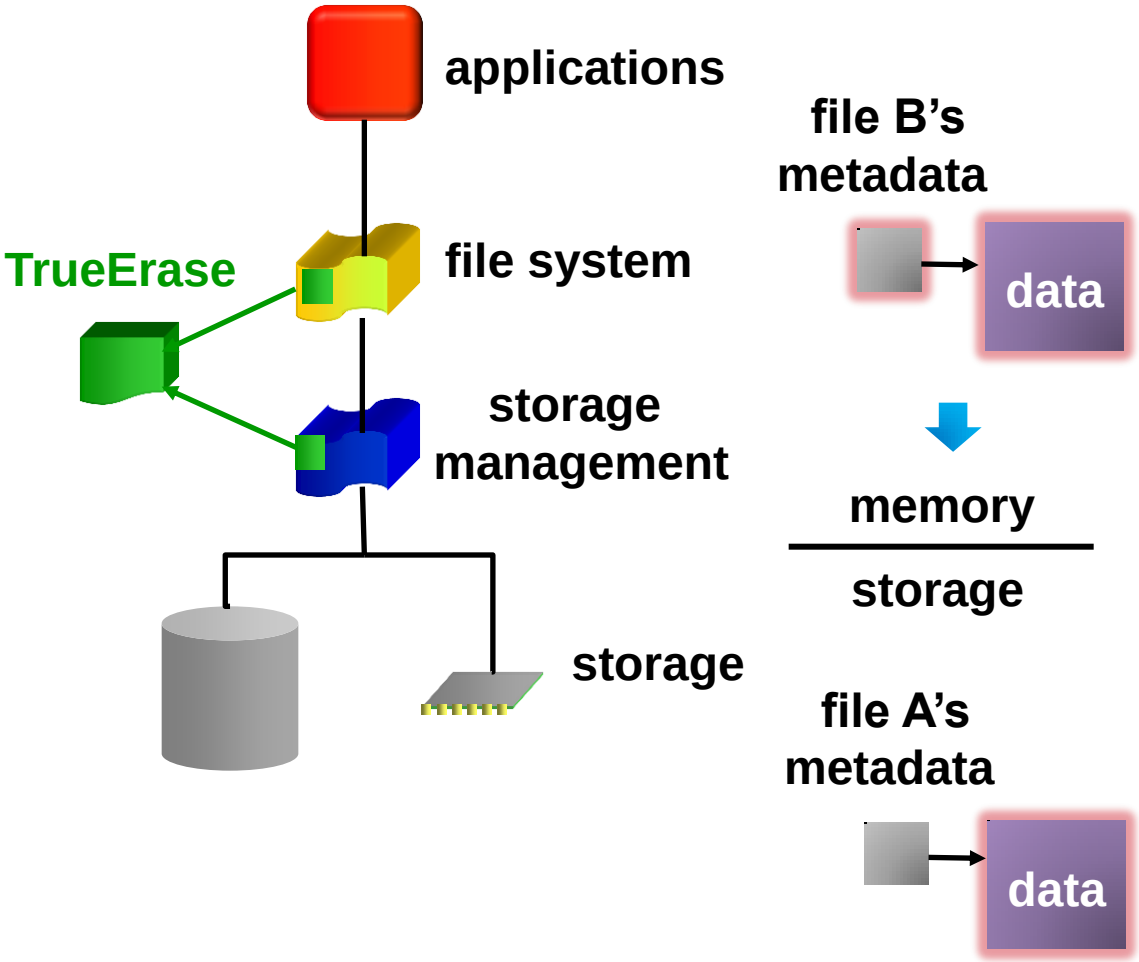
Without Pointer-ordering Property



Without Pointer-ordering Property

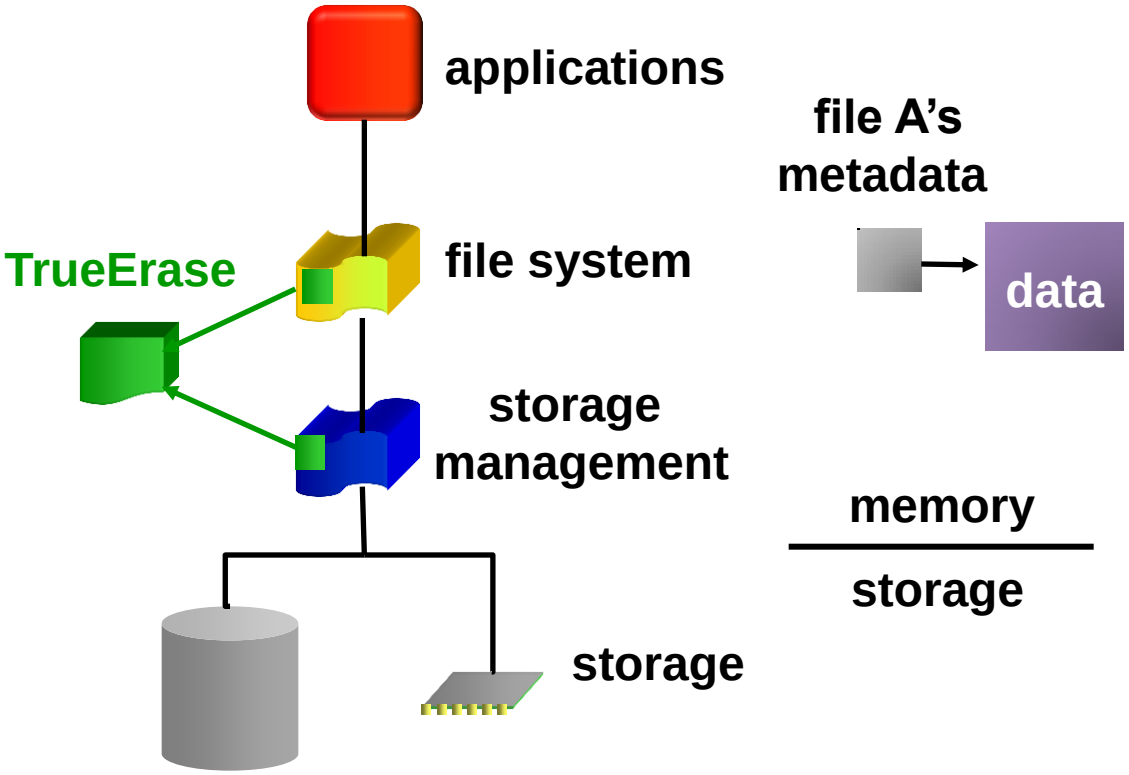


Without Pointer-ordering Property

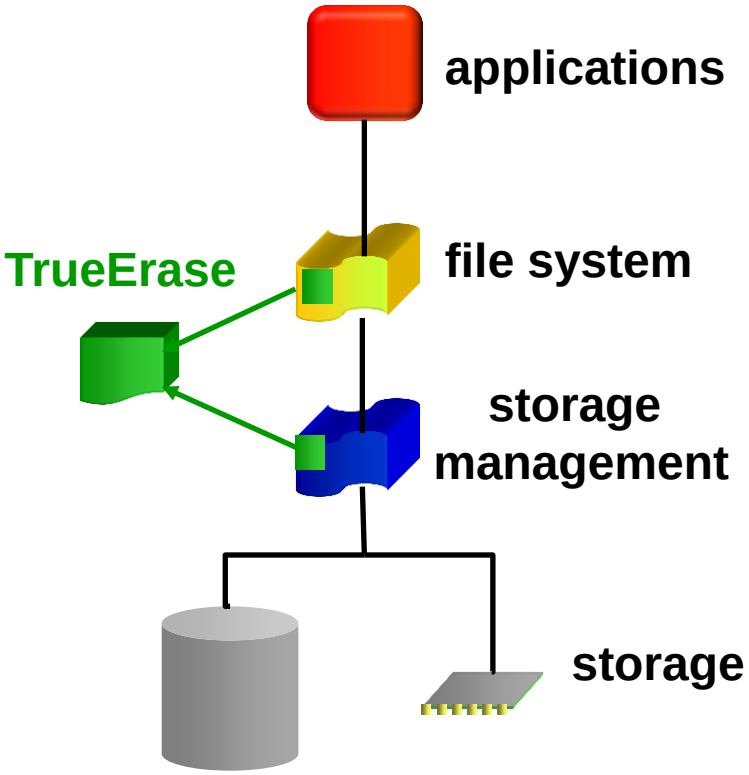


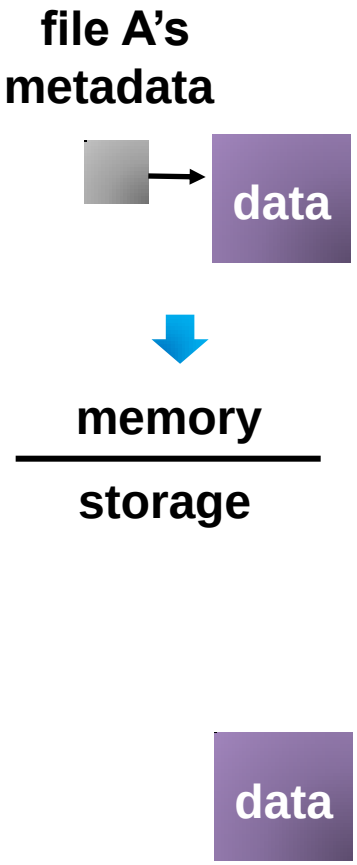
- Secure deletion of A can end up deleting B's block

Pointer-ordering Property



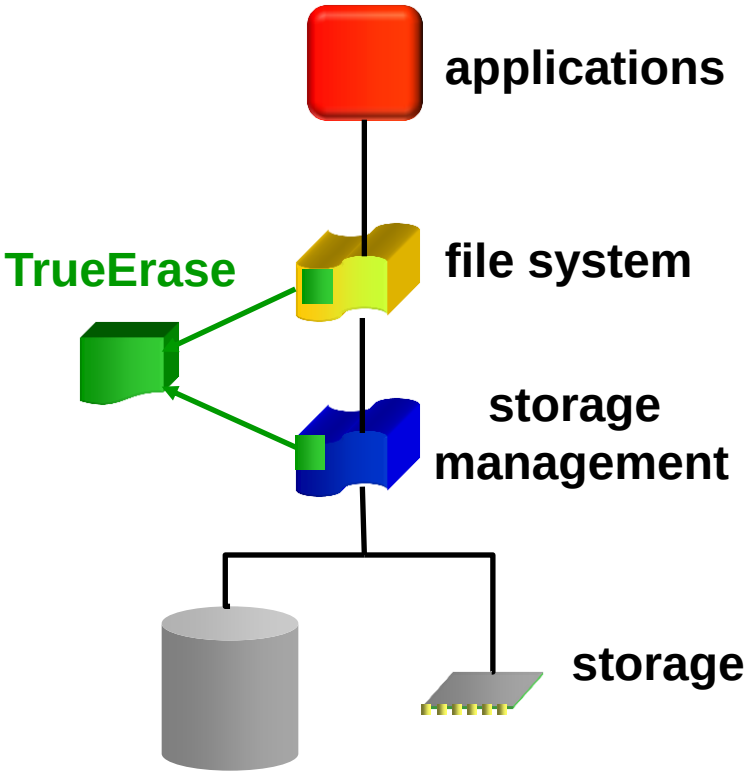
Pointer-ordering Property

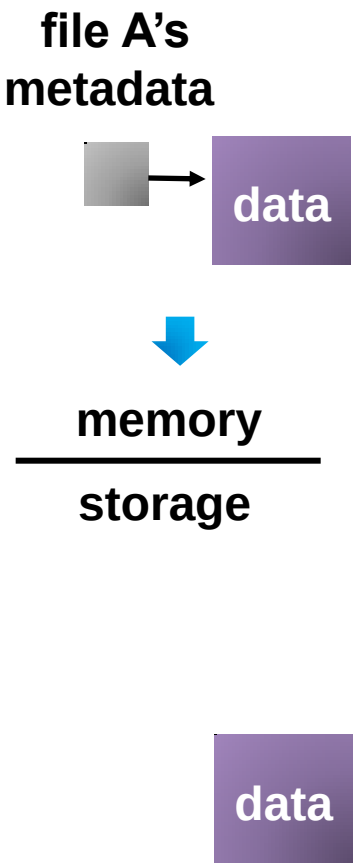




- Data blocks are propagated first

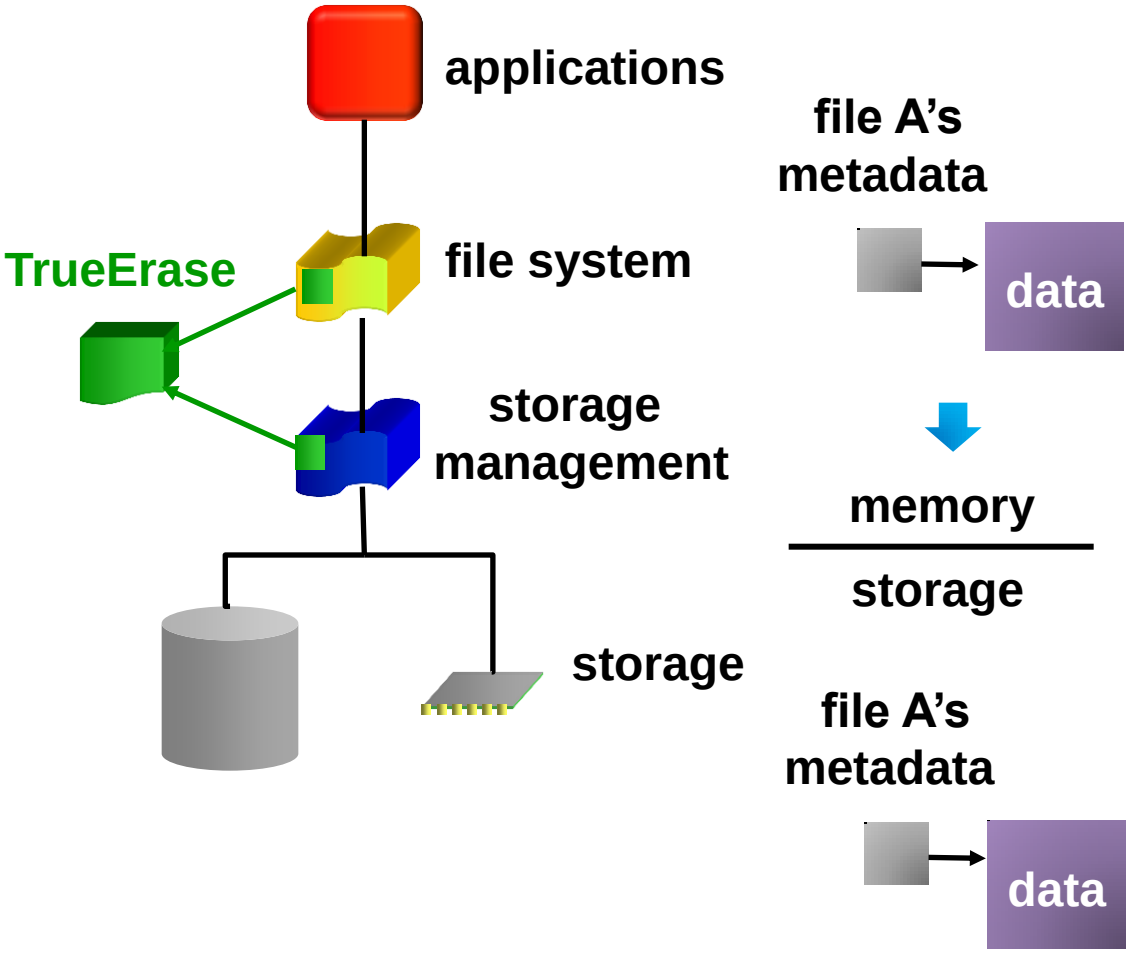
Pointer-ordering Property



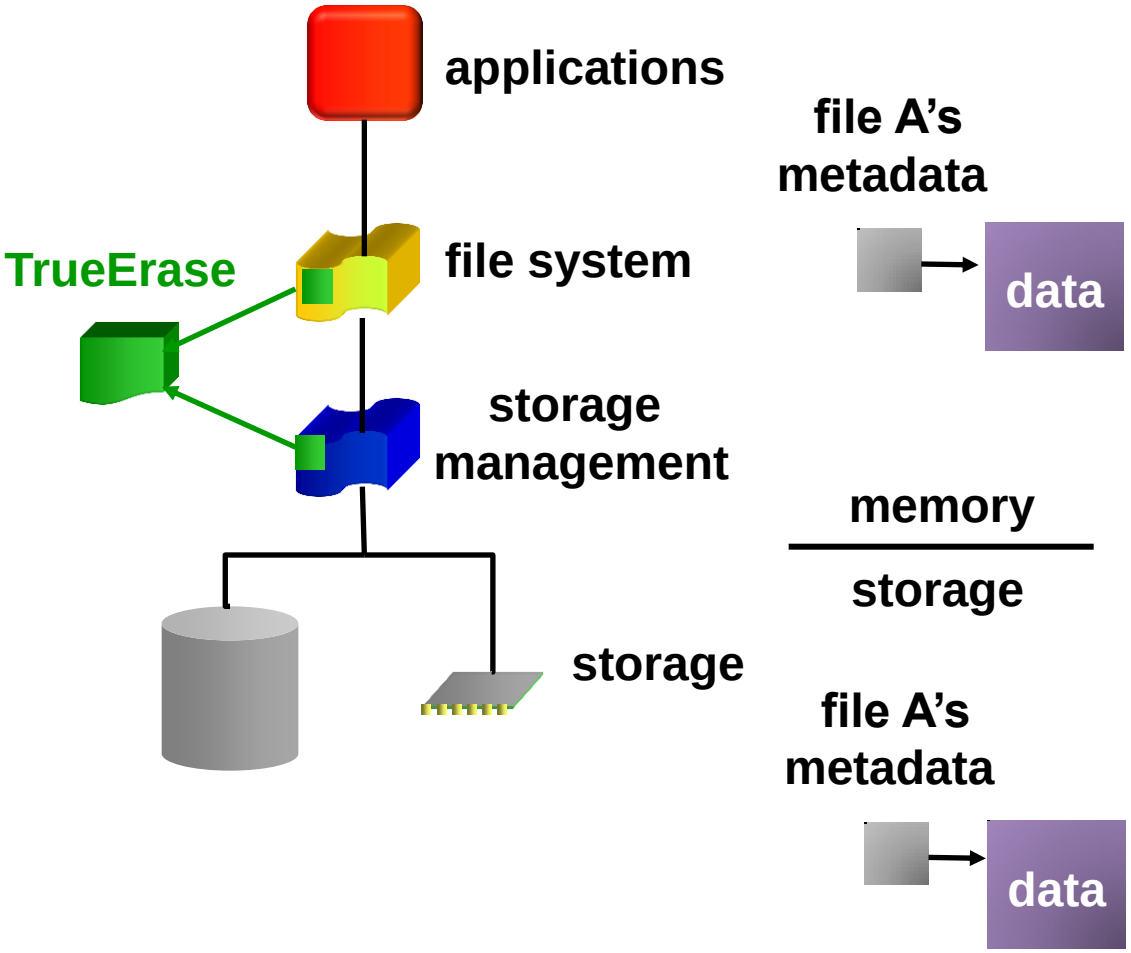


- May need to perform secure write
- Need to handle crash at this point (remove unreferenced sensitive blocks at recovery time)
- Need to ensure persistence (e.g., disabling storage-built-in caches)

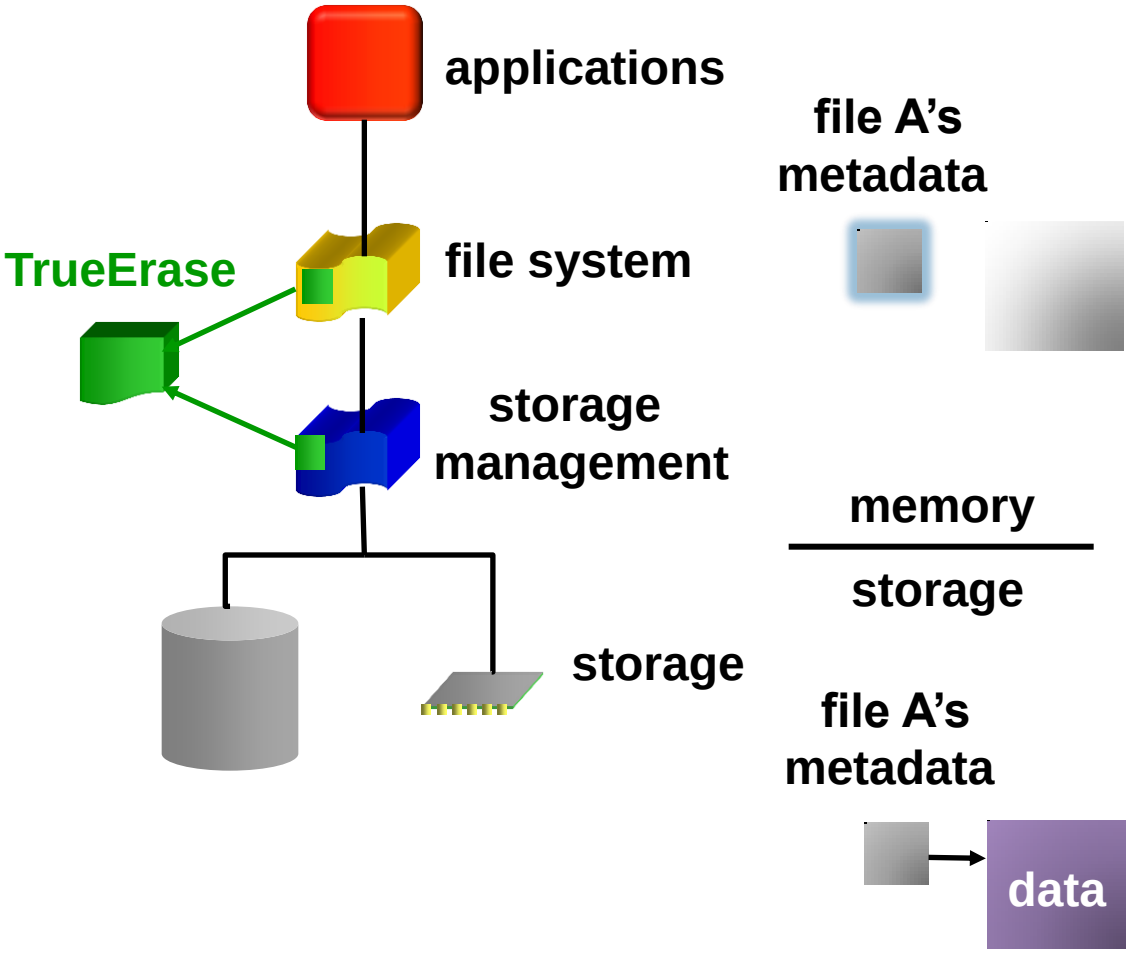
Pointer-ordering Property



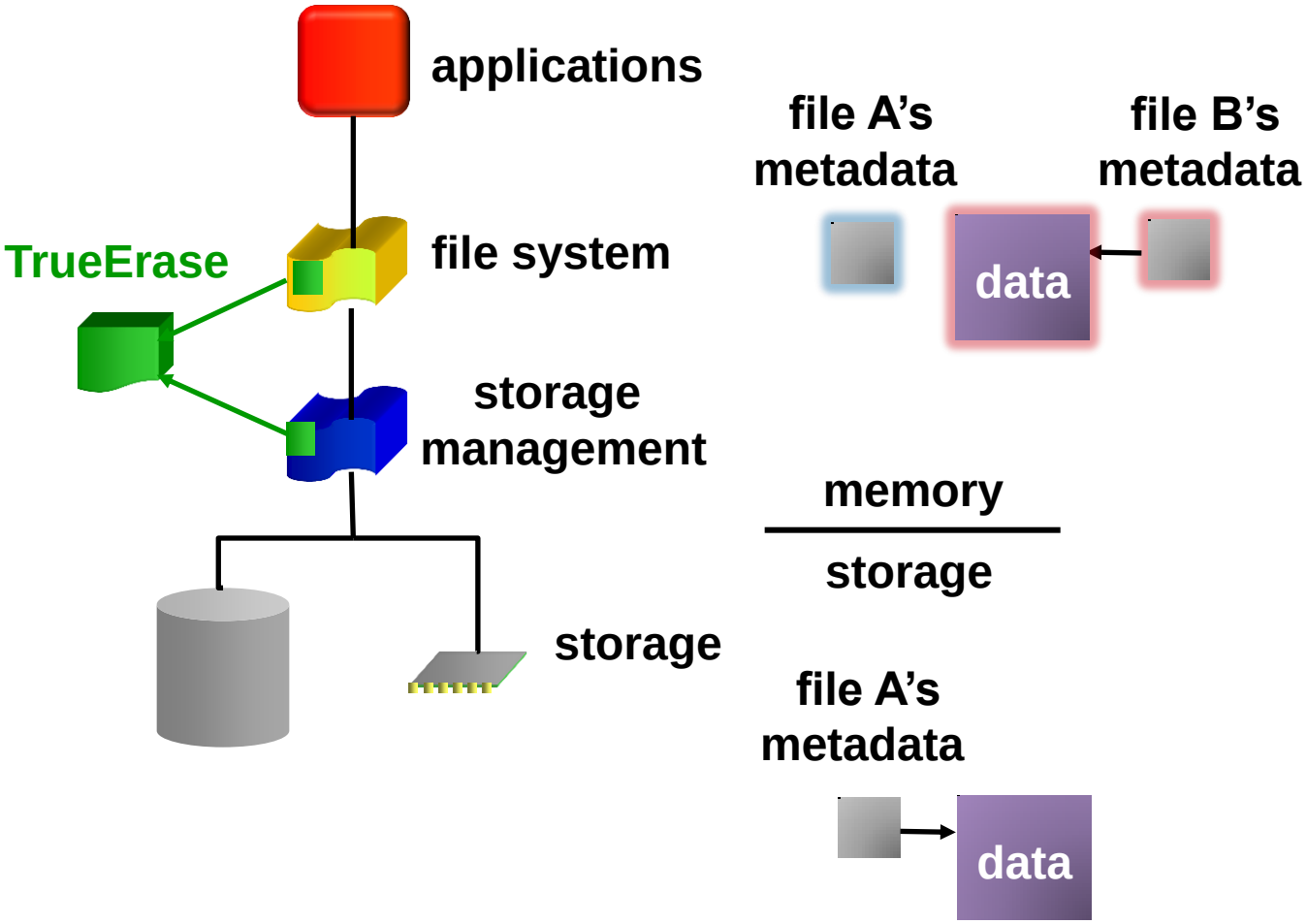
Without Reuse-ordering Property



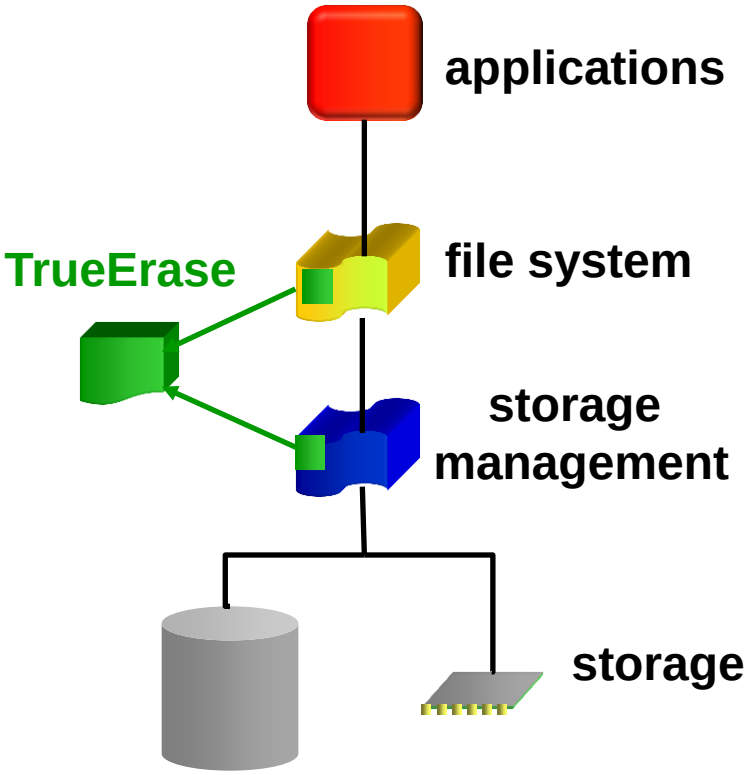
Without Reuse-ordering Property

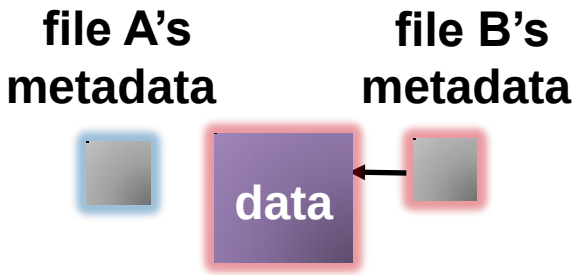


Without Reuse-ordering Property



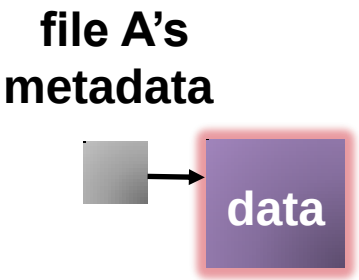
Without Reuse-ordering Property





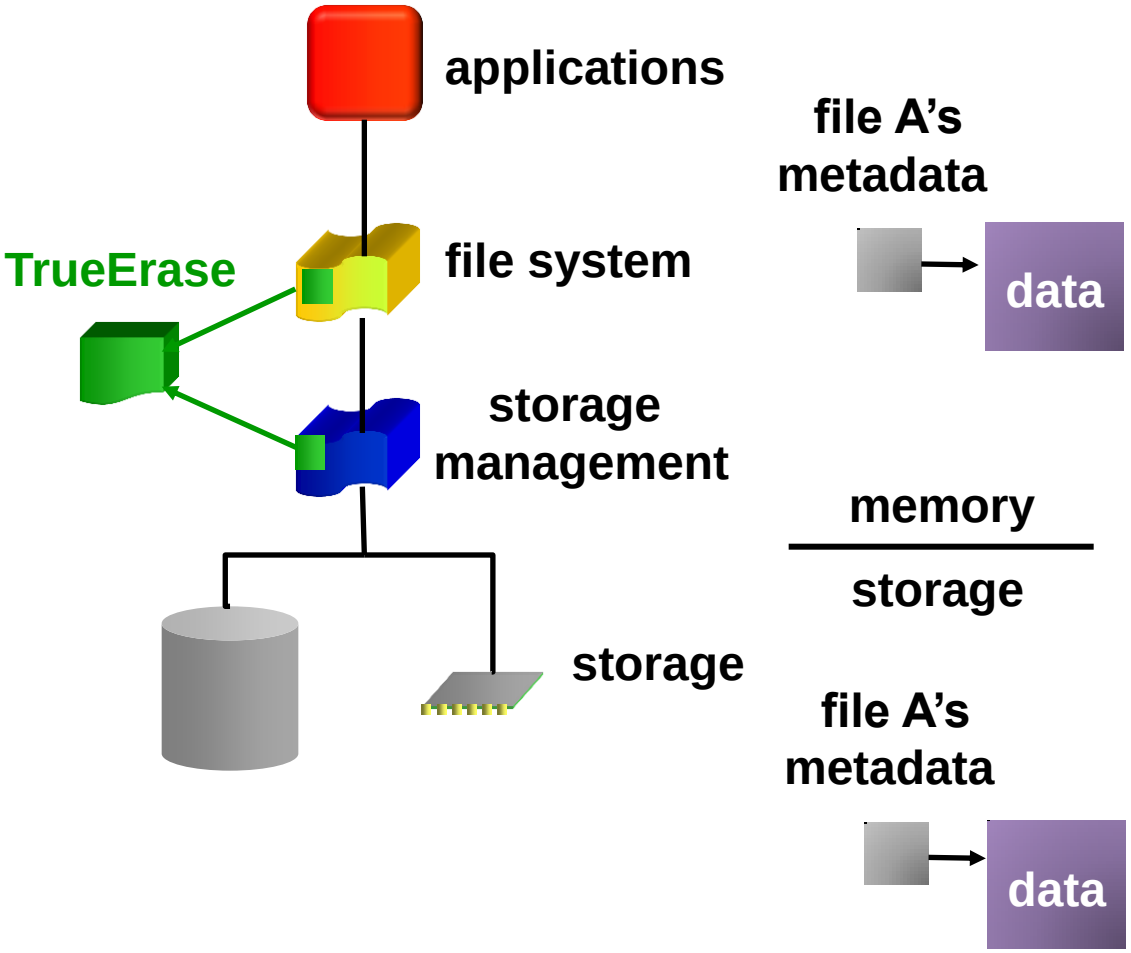
memory

storage

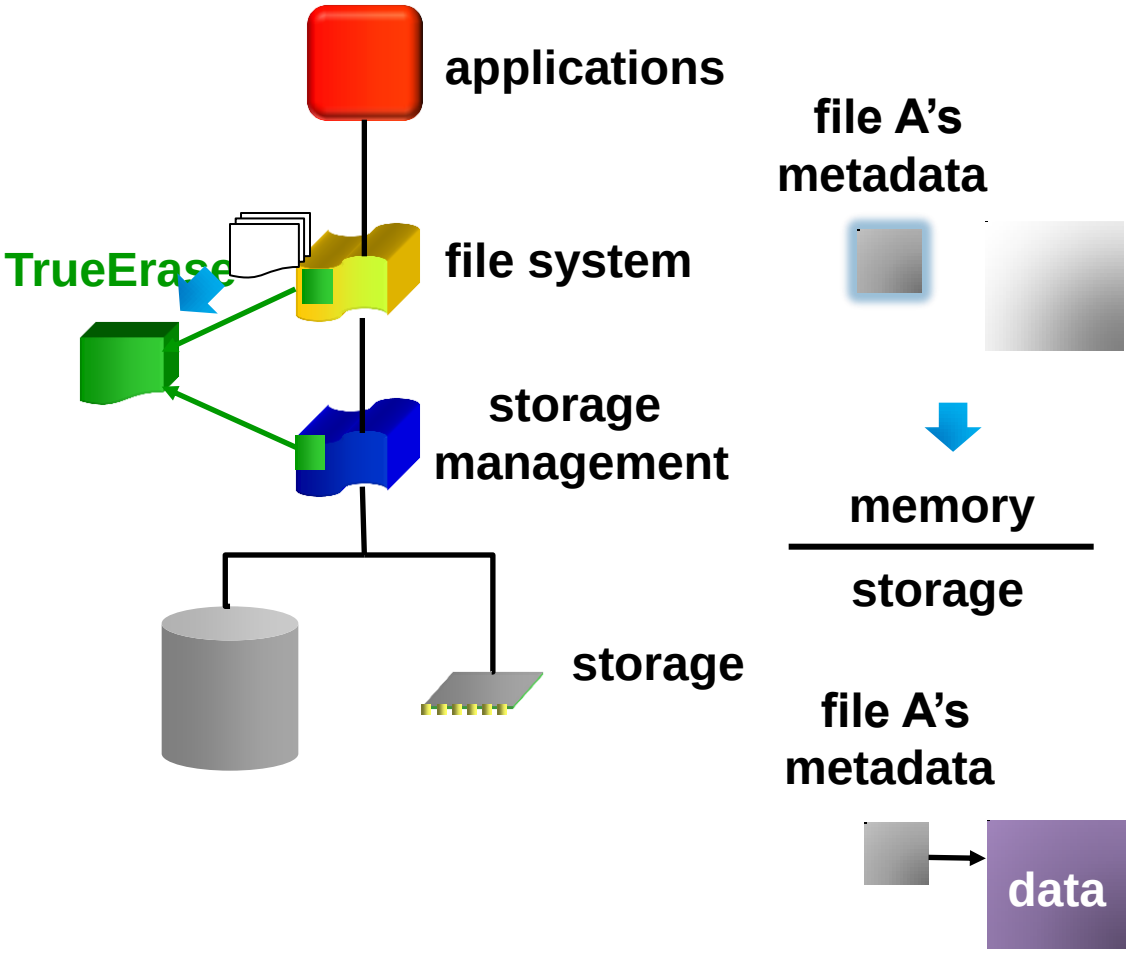


- Secure deletion of A can end up deleting B's block

Reuse-ordering Property

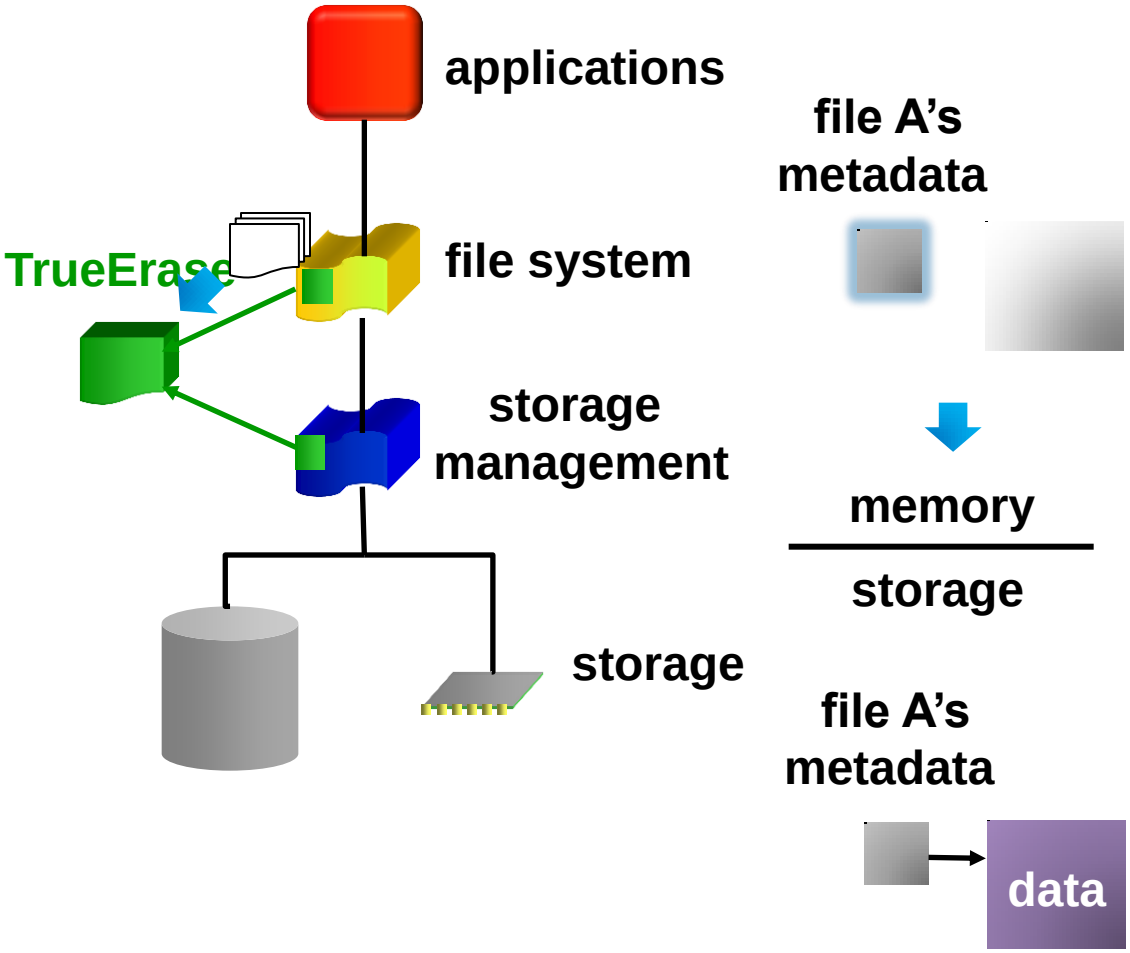


Reuse-ordering Property



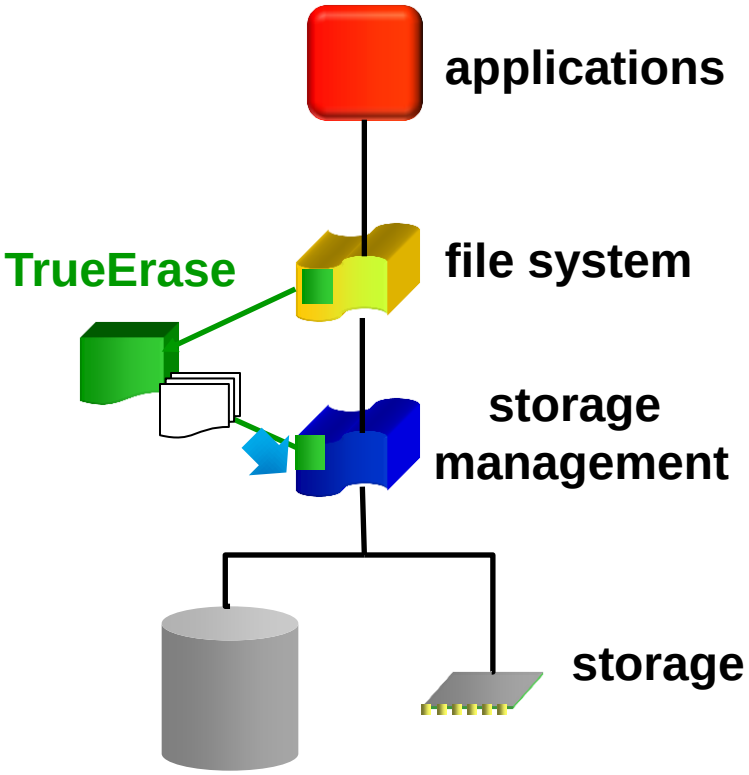
- A block cannot be reused until its free status is persistent

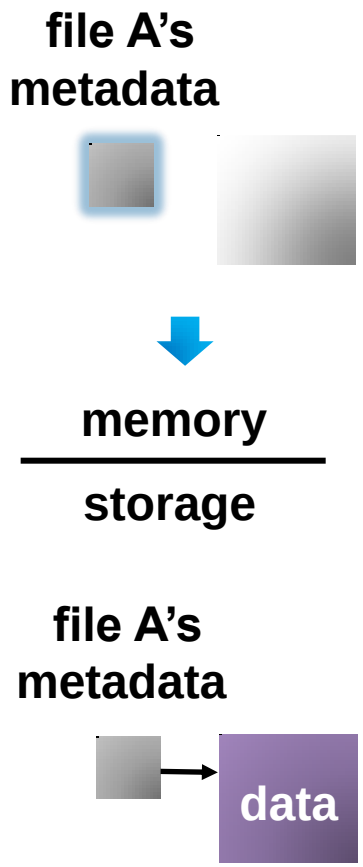
Reuse-ordering Property



- Pending updates to the unreferenced data block should not be written
- Unreferenced in-memory data blocks need to be wiped

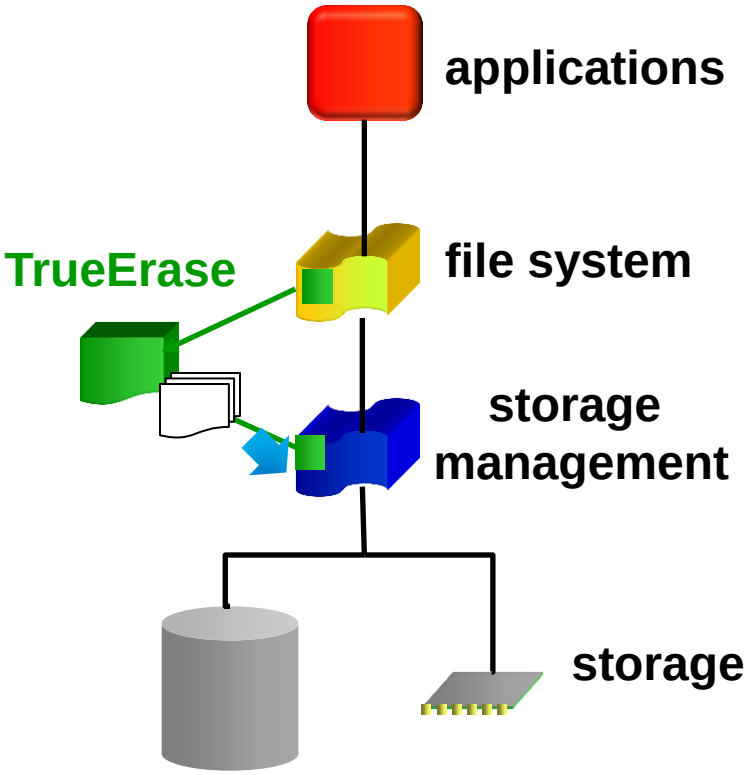
Reuse-ordering Property

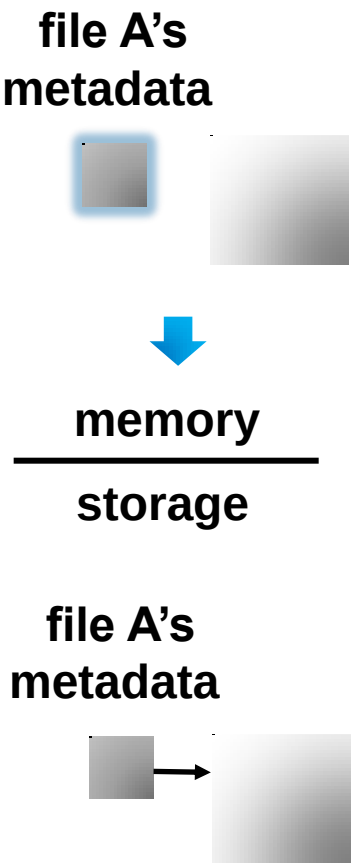




- By pointer ordering, all prior data updates are flushed
- Secure delete the data block before making its free status persistent

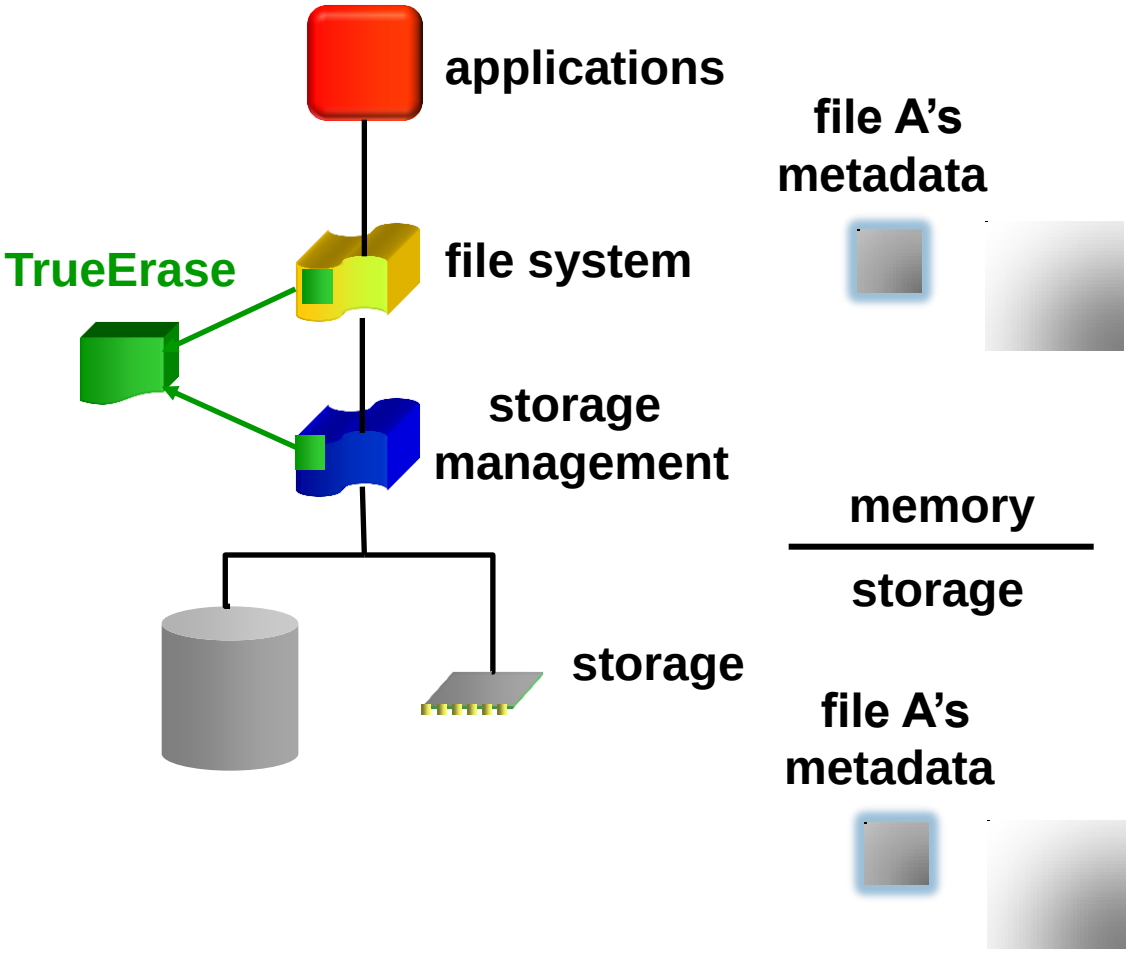
Reuse-ordering Property





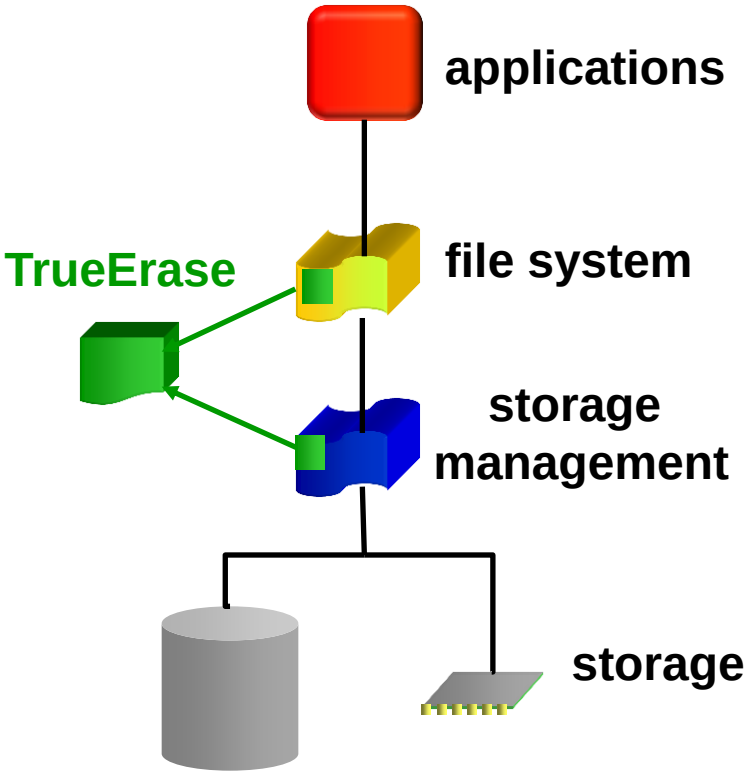
- A crash will show secure deletion in progress
- Recovery mechanism will reissue file deletion

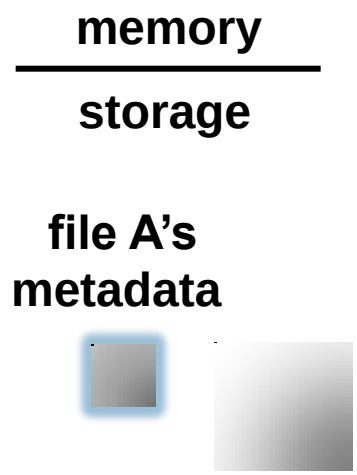
Reuse-ordering Property



- Need to ensure persistence (e.g., disabling storage-built-in caches)

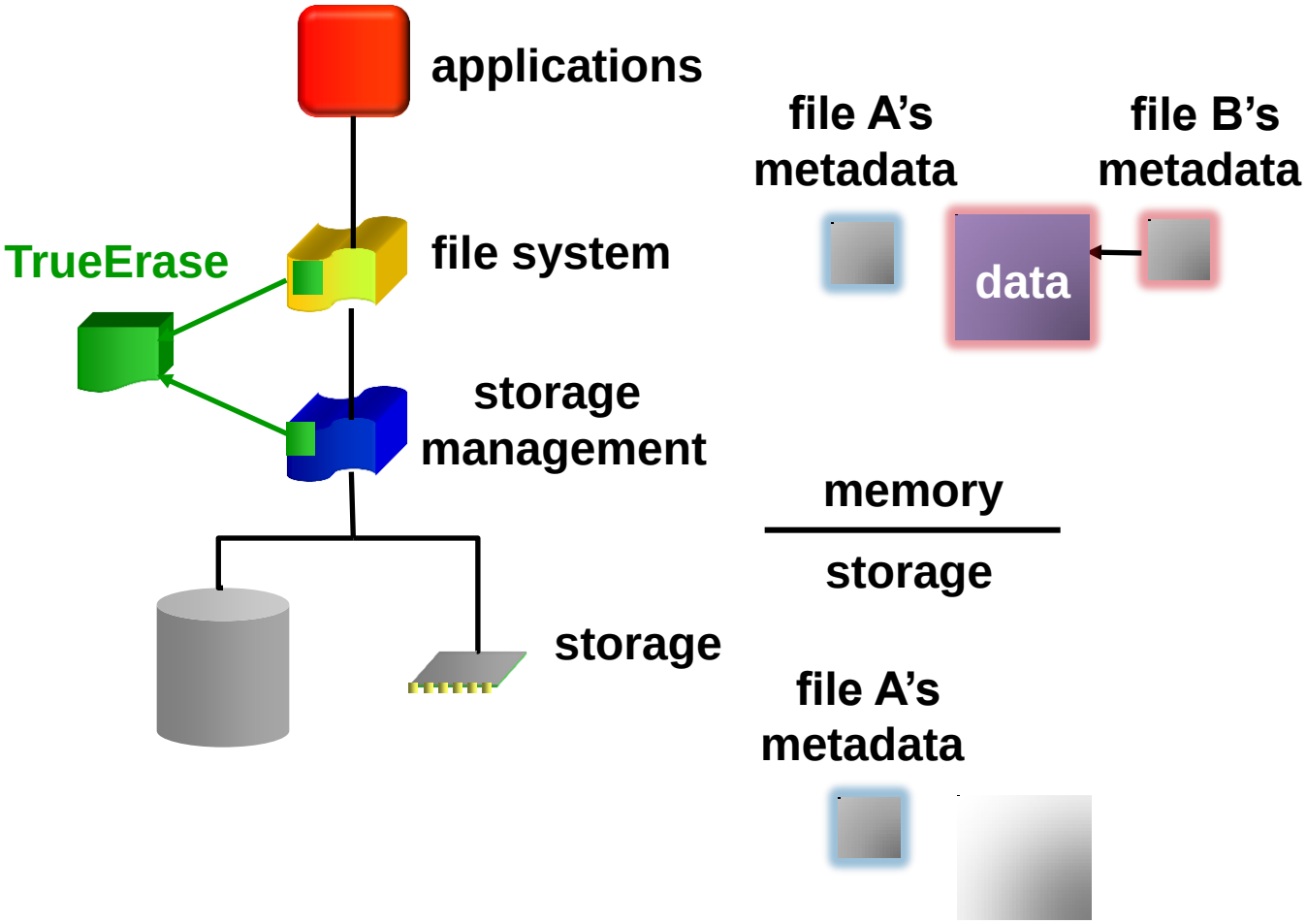
Reuse-ordering Property





- Static file types and ownerships for in-transit blocks
- Still need GUIDs to track versions
- Need to handle dynamic sensitive mode changes (once marked sensitive, always sensitive)

Reuse-ordering Property



Non-rollback Property

- Older versions of updates will not overwrite newer versions persistently
- Implications
  - An update followed by a secure deletion will be applied in the right order
  - Need to disable some optimizations at the storage-management layer (e.g., built-in cache)
  - Merging/splitting requests okay (we track sectors)
  - A consolidated update is sensitive, if one is sensitive

Structure of Corner Cases

- Ensuring that a secure deletion occurs before a block is persistently declared free
- Hunting down the persistent sensitive blocks left behind after a crash
- Making sure that secure deletion is not applied to the wrong file
- Making sure that a securely deleted block is not overwritten by a buffered unref block
- Handling versions of requests in transit

Crash Handling

- At recovery time
  - Replay journal and reissue incomplete deletion operations, with all operations handled securely
  - For flash, securely delete the journal and sensitive blocks not referenced by the file system
  - For disk, securely overwrite journal and all free space

TrueErase Implementation

- Linux 2.6.25
- File system: ext3 with its jbd journaling layer
  - Proven to adhere to the file-system-consistency properties [SIVA05]
- NAND flash: SanDisk's DiskOnChip
  - Lack of access to flash development environ.
  - Dated hardware, but the same design principle
- Storage-management layer: Inverse NAND File Translation Layer (INFTL)

Implementation-level Highlights

- Steps in deletion sequence can be expressed in secure write/delete data/metadata
- Exploited group-commit semantics
  - Reduced the number of secure operations
- Handled buffer/journal copies
- Handled consolidation within and across journal transactions

Verification

- Basic cases
  - Sanity checks
  - PostMark with 20% sensitive files
  - Reporting of all updates
  - File-system-consistency-based corner cases
- TAP state-space verification

TAP State-space Verification

- State-space enumeration
  - Tracked down ~10K unique reachable states, ~2.7M state transitions
  - Reached depth of 16 in the state-space tree
- Used two-version programming for verification
  - One based on conceptual rules
  - One based on the TAP kernel module
  - Identified 4 incorrect rules and 3 bugs

Empirical Evaluation

- Workloads
  - PostMark
    - Modified with up to 10% of sensitive files
      - Sensitive files can be chosen randomly
    - Each file operation takes < 0.17 seconds
      - Good enough for interactive use
  - OpenSSH **`make`** + sync with 27% of files that are newly created marked sensitive
    - Overhead within a factor of two

Related Work

- TRIM command
- FADED
- Type-safe disk
- Modified YAFFS with secure-deletion support

- TrueErase
  - Legacy-compatible, persistent-state-light, centralized info-propagation channel

Lessons Learned

- Retrofitting security features is more complex than we thought
- The general lack of raw flash access and development environments
  - Vendors try to hide complexities
  - File-system consistency and secure deletion rely on exposed controls/details for data layout/removal

Lessons Learned

- A holistic solution would not be possible
  - Without expertise across layers and research fields
- Highlights the importance of knowledge integration

Conclusion

- We have presented the design, implementation, evaluation, and verification of TrueErase
  - Legacy-compatible, per-file, secure-deletion framework
- A secure-deletion solution that can withstand diverse threats remains elusive
  - TrueErase is a promising step toward this goal

Acknowledgements

- National Science Foundation
- Department of Education
- Philanthropic Educational Organization
- Florida State University Research Foundation

Questions?

- Google keyword: TrueErase

Thank you for your attention!